Computer Supported Cooperative Work **9:** 399–412, 2000.

A Knowledge-based Approach to Handling Exceptions in Workflow Systems

MARK KLEIN[1] & CHRYSANTHOS DELLAROCAS[2]
[1]*Center for Coordination Science, Massachusetts Institute of Technology, Cambridge, MA, U.S.A. (E-mail: m_klein@mit.edu);* [2]*Sloan School of Management, Massachusetts Institute of Technology, Cambridge, MA, U.S.A. (E-mail: dell@mit.edu)*

(Received 26 May 1999)

Abstract. This paper describes a novel knowledge-based approach for helping workflow process designers and participants better manage the exceptions (deviations from an ideal collaborative work process caused by errors, failures, resource or requirements changes etc.) that can occur during the enactment of a workflow. This approach is based on exploiting a generic and reusable body of knowledge concerning what kinds of exceptions can occur in collaborative work processes, and how these exceptions can handled (detected, diagnosed and resolved). This work builds upon previous efforts from the MIT Process Handbook project and from research on conflict management in collaborative design.

Key words: adaptation, failure handling, process exception, re-design

1. The challenge

Workflow systems are currently ill-suited to dealing with exceptions. These systems typically institutionalize a more or less idealized preferred process. When exceptions do occur we are often forced to "go behind the workflow system's back", making the system more of a liability than an asset. Workflow models can, of course, include conditional branches to deal with anticipated exceptions. Current process modeling methodologies and tools (Harrington, 1991; Davenport, 1993; Hammer and Champy, 1993; Grover and Kettinger, 1995; Kettinger et al., 1995) do not, however, make any provision for describing exception handling procedures separately from "main-line" processing. Inclusion of exception handling branches, therefore, can greatly complicate process models and obscure the "preferred" process, making it difficult to define, understand, and modify. Up-front prescription of exception handling can also reduce or eliminate the discretion workflow participants in precisely the cases most likely to profit from individual attention. Current workflow technologies provide, in addition, no support for uncovering what kinds of exceptions can occur in a given process model, and how they can be resolved. Process designers are then forced to rely on their own, probably incomplete, experience and intuitions about possible exceptions and how they can be managed.

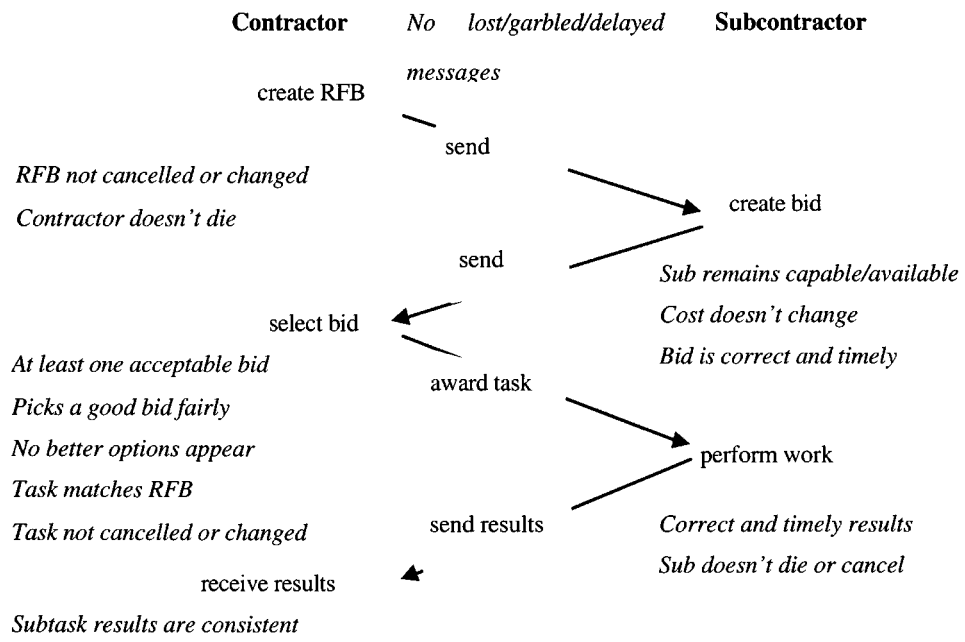

Figure 1. Implicit assumptions for subcontracting.

This paper describes a knowledge-based approach to meeting these challenges. This approach provides design tools that allow designers to systematically analyze "normal" process models, anticipate possible exceptions and suggest ways in which the "normal" process can be instrumented in order to detect or even anticipate them. When exception manifestations occur, enactment-time tools are provided to help diagnose their underlying causes and suggest specific interventions for avoiding or resolving them. The remainder of the paper will discuss how this approach works, what it contributes to previous efforts in this area, and some directions for future work.

2. A knowledge-based approach to exception handling

2.1. WHAT IS AN EXCEPTION?

We define an exception as any deviation from an "ideal" collaborative process that uses the available resources to achieve the task requirements in an optimal way. An exception can thus include errors in enacting a task or communicating results between agents, inadequate responses to changes in tasks or resources, missed opportunities and so on. To make this more concrete consider the possible exceptions for the generic coordination process known as "subcontracting". Subcontracting can be used whenever one wants to share agents who perform some service among requestors of that service. The requestor for a service (the

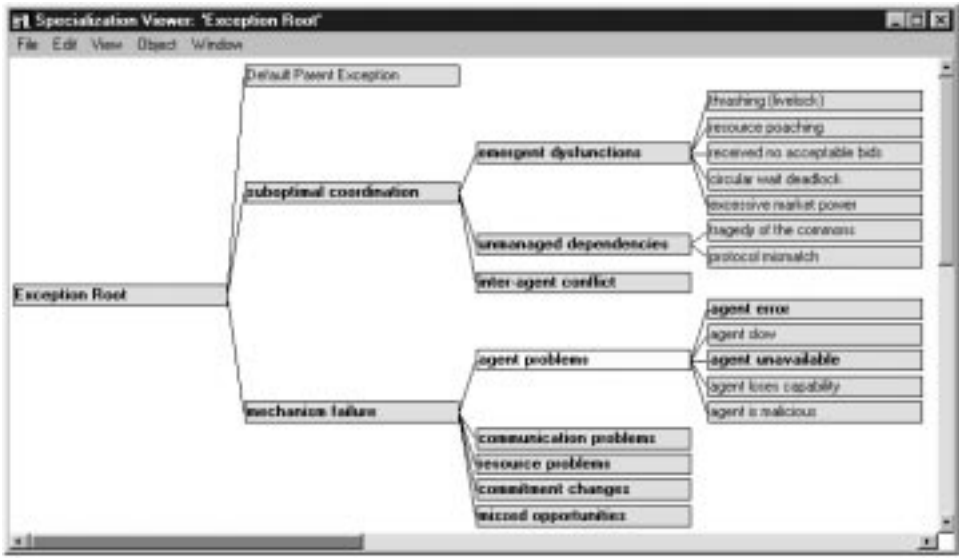

Figure 2. A subset of the exception type taxonomy.

"Contractor") sends out a request for bids (RFB), asking for agents (potential "Subcontractors") to perform a given task, the interested subcontractors respond with bids, the Contractor awards the task to the Subcontractor with the best bid (based, for example, on anticipated cost, quality or timeliness), at which point the Subcontractor performs the task and returns the results to the Contractor.

This mechanism makes many implicit assumptions; violations of any one of them can lead to exceptions (Figure 1). It assumes, for example, that the task required by the Contractor does not change after it has sent out the RFB, that the Subcontractor does not renege on the task it was assigned, become incapable of doing it, make a mistake etc. Any assumption violation represents a possible exception type. In addition to that, some exceptions take the form of dysfunctional systemic behavior that may result even when the mechanism is followed perfectly. Thrashing (where coordination, such as resource swapping, occurs to the exclusion of actual work), deadlock (where several agents are each waiting for another one to do something) and resource poaching (wherein high-priority tasks are unable to access needed resources because these resources have already been reserved by lower priority tasks) are all examples of systemic dysfunctions.

We have developed, as a result of analyses like that shown above, a growing taxonomy of exception types, a subset of which is show in Figure 2. As we shall see, the essence of our work is developing a knowledge base that captures such exceptions and relates them to (1) the processes that they can occur in, and (2) the processes that can be used to handle them.

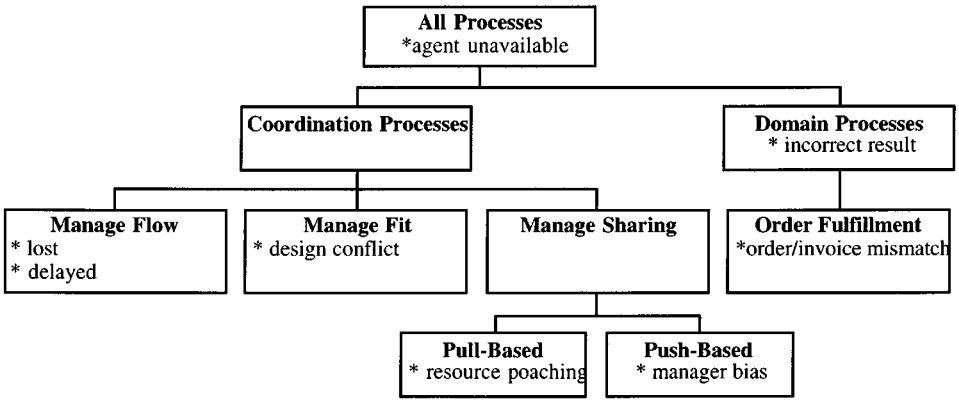

Figure 3. An example of a generic process taxonomy annotated with exceptions.

2.2. PREPARING FOR EXCEPTIONS

The first step is for a workflow designer to determine, for a given "ideal" workflow, the ways that the process may fail and then "instrument" the workflow so that these exceptions can be detected or even anticipated. The principal idea here is to compare a process model against a taxonomy of generic process templates annotated with possible exception modes.

A process taxonomy can be defined as a hierarchy of process templates, with very generic processes at the top and increasingly *specialized* processes below. Each process can have attributes that define the challenges for which it is well-suited. Note that process *specialization* is different from *decomposition*, which involves breaking a process down (i.e. "decomposing it") into subactivities. While a subactivity represents a part of a process, a specialization represents a "subtype" or "way of" doing the process (Malone et al., 1997).

Generic process templates are annotated, in our approach, with the ways in which they can fail, i.e. with their characteristic *exception types* (see Figure 3) which can be uncovered using failure mode analysis (Raheja, 1990). Each process template in a taxonomy inherits all characteristic exceptions of its parent (generalization) and may contain additional exceptions which are specific to it.

Given an "ideal" process model like that shown above in Figure 4, to identify possible exceptions we need only identify the generic process templates that match (components of) the process. The potentially applicable exception types will then consist of the union of the exceptions inherited from the matching generic templates. The workflow above consists of a subprocess for allocating design tasks (performed in this case by subcontracting), a subprocess for allocating shared resources such as mainframe computer time to design groups (allocated first-come first-served based on requests), followed by subprocesses where the different subcomponent designs are consolidated and then sent on to be manufactured, delivered, and inspected. We can see, for example, that the "distribute shared

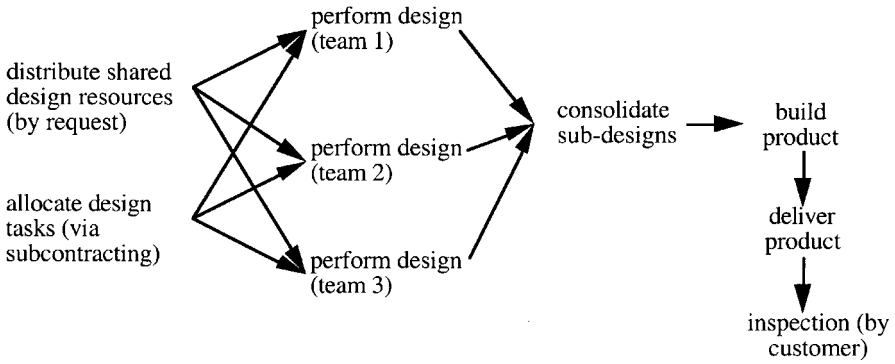

Figure 4. An example "ideal" workflow process.

design resources" subprocess in our example is a subtype of the generic "pull-based sharing" process template in Figure 3, since the resources are "pulled" by their consumers rather than "pushed" (i.e. allocated) by their producers or managers. This template includes "resource poaching", which we have already encountered, among its characteristic exceptions. The "consolidate sub-designs" subprocess is a specialization of the "manage fit" template and thereby inherits the "design conflict" exception. The "deliver product" subprocess is a specialization of the "manage flow" template, with characteristic exceptions such as "item delayed", "item lost" and so on. All the subprocesses also inherit exception modes, such as "agent unavailable", from the generalizations of these matching templates.

Every exception type has an associated knowledge base entry that gives its definition, what situations it is known to be especially critical in, and how it can be handled. The entry for the "resource poaching" exception (Figure 5).

The workflow designer can use the definition and criticality information to help select, from the 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 to handle the "lost item" exception.

For each exception type of interest, the workflow designer can then decide how to augment the workflow models to prepare for (anticipate or detect) these exceptions. Every exception type includes pointers to "exception detection" process templates in the Handbook repository that specify how to detect the symptoms manifested by that exception type. These templates, once incorporated into the workflow by the workflow designer, play the role of "sentinels" that check for signs of actual or impending exceptions. The template for *detecting* the "resource poaching" exception, for example, operates by comparing the priority of the tasks that have the resources they need vs the priority of those that do not. The template for *anticipating* the resource poaching exception, by contrast, looks for situations where the potential subcontractor population is busy and a set of high-priority tasks is expected or at hand.

Exception	resource poaching
For Process	subcontracting
Definition	One or more high-priority tasks are unable to access needed subcontractors because they already have been 'grabbed' by lower priority tasks
Criticality	This exception can have a high impact when there is a significant variation in task priority and the available subcontractors population can be oversubscribed.
Anticipation Processes	**Anticipation-Process-1** (The potential subcontractor population is currently busy with low priority tasks, and a set of high-priority tasks is expected)
Detection Processes	**Detection-Process-1** (The priority of the tasks that have the resources they need is less than the priority of those that do not) **Detection-Process-2** (A contractor does not get any bids for an offered task within time-out period)
Avoidance Processes	**Avoidance-Process-1** (Require that subcontractors collect several request-for-bids before bidding, and respond preferentially to higher-priority bids)
Resolution Processes	**Resolution-Process-1** (Allow subcontractors to suspend lower-priority tasks and bid on later higher-priority tasks)

Figure 5. The knowledge base entry for the 'resource poaching' exception.

An exception can have more than one candidate handler process for anticipation, detection and so on. All handler processes are annotated with their characteristic preconditions and performance properties to help the workflow designer select among them. Figure 6 gives, as an example, the information captured for the two different detection processes listed in Figure 5 for resource poaching.

A workflow designer can use process design tools already included in the Process Handbook, such as tradeoff tables (Malone et al., 1997) and recombinant design techniques (Herman et al., 1998) to help find existing exception handlers or even generate new variants customized for particular contexts. Note that, currently, exception handler processes are captured informally as textual descriptions sometimes augmented with task decompositions and data/control flow diagrams. The workflow participant is responsible for implementing the ideas these represent into the current workflow process.

2.3. DIAGNOSING EXCEPTIONS

The next step is to figure out how to react when an exception is anticipated or detected during the enactment of the workflow process. A key challenge here is that the symptoms revealed by the exception detection/anticipation processes can often result from a wide variety of possible underlying causes. Many different exceptions (e.g. "agent not available", "item misrouted", "resource poaching" etc.) typically manifest themselves, for example, as missed deadlines. Just as in medical domains,

Handler	Detection-process-1
Definition	Look for cases where the priority of the tasks that have the resources they need is less than the priority of those that do not
Preconditions	Task priority info is used and available
Overhead	Relatively high
False Positives	Zero
Best For	Low volume processes

Handler	Detection-process-2
Definition	Look for cases where a contractor does not get any bids for an offered task within time-out period
Preconditions	There are deadlines for request for bid responses
Overhead	Relatively low
False Positives	> Zero
Best For	High volume processes

Figure 6. Knowledge base entries for handler (detection) processes.

selecting an appropriate intervention requires diagnosing the underlying cause of the presenting symptoms.

Our approach for diagnosing exception causes is based on heuristic classification (Clancey, 1984). This approach works by traversing the exception taxonomy (introduced in Figure 2 above). Every exception, as we have seen, includes defining characteristics that need to be true in order to make that diagnosis potentially applicable to the current situation. When an exception is detected, the responsible workflow participant traverses the exception type taxonomy top-down like a decision tree, iteratively refining the specificity of the diagnoses by eliminating exception types whose defining characteristics are not satisfied. Distinguishing among candidate diagnosis 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 in the "allocate design tasks" step. The exceptions that can manifest this way include "agent unavailable" (e.g. the agent responsible for contracting is unavailable), "item misrouted/delayed/lost" (the request for bid or actual bids are lost, misrouted or delayed), "resource poaching" (as defined above) and so on. We can distinguish among these possibilities by such means as contacting the contractor agent, checking whether the subcontractors received the request for bids and sent any responses, checking task priorities and so on. In the current implementation of our approach, the workflow participant is responsible for making these queries by his or herself; the exception taxonomy serves as a guide to what kinds of problems can occur and how they can be distinguished.

Heuristic classification is a "shallow model" (Chandrasekaran and Mittal, 1999) form of diagnosis because it is based on compiled empirical and heuristic exper-

tise rather than first principles. This approach is appropriate for domains, such as medical diagnosis, where complete and consistent behavioral models do not exist. This, we would argue, is also true for workflows with human and complex software agents. An important characteristic of heuristic classification is that the diagnoses represent *hypotheses* rather than guaranteed *deductions*: multiple diagnoses may be suggested by the same symptoms, and often the only way to verify a diagnosis is to see if the associated prescriptions are effective.

2.4. RESOLVING EXCEPTIONS

Once an exception has been detected, and at least tentatively diagnosed, one is ready to define an *prescription* that responds appropriately to the anticipated or actual exception. This can be achieved, in our approach, by selecting and enacting one of the exception avoidance/resolution handler processes associated with the exception. Imagine, for example, that we have diagnosed either the actual or impending occurrence of a resource poaching exception. If the exception is *anticipated*, the workflow user can implement Avoidance-process-1 (i.e. that requires that subcontractors collect several request-for-bids before bidding, and respond preferentially to higher-priority bids). If the exception is actual, the workflow user can implement Resolution-process-1 (instruct the subcontractors to suspend lower-priority tasks and bid on the current higher-priority tasks). Some other examples of exception handling strategies captured in our knowledge base include:

- IF a subprocess 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. releasing partial results to allow later tasks to start earlier) or parallelization to increase concurrency.
- IF an agent receives garbled data, THEN trace back to the original source of the faulty data, eliminate all decisions that were corrupted by this error and start again.
- 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.
- 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.
- IF an agent in a serial production line fails to perform a task, THEN re-allocate the task to an appropriately skilled agent further down the line.

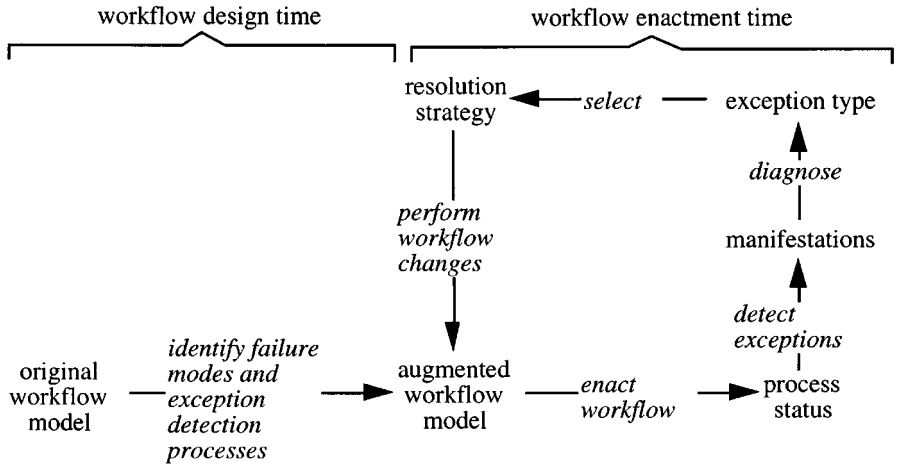

Figure 7. Summary of exception management approach.

As with the anticipation/detection handler processes, one can select among multiple candidate avoidance/resolution handler processes using performance and precondition information stored with the handler processes. Also as above, in the current implementation a human workflow participant is responsible for actually changing the workflow process in a way that implements the suggested avoidance/resolution strategy.

2.5. SUMMARY

Our exception handling approach can be summarized as Figure 7. An "ideal" workflow model is checked at design time, using a process taxonomy augmented with exception mode information, to see the ways it can fail. It is then augmented by the workflow designer with "sentinels" that check for anticipatory/actual manifestations of these exceptions. When the process is enacted, these sentinels flag any exception manifestations that they encounter. The notified workflow participant can then use the Handbook's knowledge base of exception types and associated exception handler processes to uncover the underlying cause for the problem, and select the appropriate handler. The user then modifies the workflow process as suggested by the handler, allowing the process to continue.

Our project has captured, to date, roughly 100 exception types and 200 exception handler processes in the Process Handbook repository. We have focused the bulk of our effort on acquiring exception handling information relevant to generic "coordination processes" (i.e. processes used for managing the flow of resources among tasks and agents) rather than "core processes" (i.e. domain-specific tasks such as filling out an invoice) because information about coordination exceptions can be highly leveraged; it is our experience that a relatively small set of coordi-

nation mechanisms is used in a very wide range of real-world collaborative work processes.

3. Contributions to previous work

The approach described here builds upon two long-standing lines of MIT 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 (Malone et al., 1993; Dellarocas et al., 1994; Malone and Crowston, 1994; Malone et al., 1997). The goal of this project is to produce a repository of process knowledge and associated tools/techniques that help people to (among other things) better redesign organizational processes, learn about organizations, and automatically generate software. The Handbook database continues to grow and currently includes over 4000 models covering a broad range of business processes. We have developed a mature Windows-based tool for editing the Handbook database contents, as well as a Web-based tool for read-only access. Both are being actively used by a highly distributed set of scientists, students and sponsors from government and industry. A key insight from this work is that a repository of business process templates, structured as a taxonomy, can help people design qualitatively more innovative processes more quickly by allowing them to retrieve, contrast and customize interesting examples, make "distant analogies", and utilize "recombinant" (mix-and-match) design techniques (Herman et al., 1998).

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 (Klein, 1989, 1991, 1993) and collaborative requirements capture (Klein, 1997). 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. This integration is based on the insights that (1) the exception handling expertise applicable to a given process can be identified via annotations on generic process templates in a process taxonomy, and (2) diagnostic techniques based on those applicable to collaborative

design conflict management can be applied to workflow exceptions (a conflict, after all, is just one type of exception).

This work also constitutes, we believe, a substantive and novel contribution to previous efforts on exception handling, which have been pursued in the context of workflow (Kunin, 1982; Kreifelts and Woetzel, 1987; Auramaki and Leppanen, 1989; Karbe and Ramsberger, 1990; Strong, 1992; Mi and Scacchi, 1993; Ellis et al., 1995; Klein, 1998), manufacturing control (Parthasarathy, 1989; Katz, 1993; Visser, 1995), model-based fault diagnosis (DeKleer and Williams, 1986; Krishnamurthi and Underbrink Jr., 1989; Birnbaum et al., 1990; Friedrich et al., 1990), planning (Sussman, 1973; Goldstein, 1975; Broverman and Croft, 1987; Birnbaum et al., 1990), and failure mode analysis research (Raheja, 1990). Most workflow research has focused on languages for expressing correctness-preserving transforms on workflow models, providing no guidance, however, concerning *which* transforms to use for a given situation. There has been some manufacturing and workflow research on providing guidance for how to handle exceptions, but this has been applied to a few narrow domains (mainly software engineering and flexible manufacturing cell control) and/or has addressed a small handful of exception types. The planning work, by contrast, has developed powerful computational models but they are only applicable if the planning technology was used to develop the original work process. This is typically not the case for workflow settings where processes are defined by people rather than automated planning tools. Model-based fault diagnosis approaches use a single generic algorithm to uncover the causes of faults in a system without the need for a knowledge base of failure modes and resolution heuristics. This approach is predicated, however, on the availability of a complete and correct model of the system's behavior, which is possible for some domains (e.g. the analysis of electrical circuits), but not for many others including, I would argue, most collaborative work settings that include human beings and/or complex computer systems as participants. Model-based fault diagnosis also typically assumes that resolution, once a fault has been diagnosed, is trivial (e.g. just replace the faulty component) and thus does not provide context-specific suggestions for how to resolve the problem. Current work on failure mode analysis describes a systematic process, but the actual work must be done by people based on their experience and intuitions. This is potentially quite expensive, to the extent that this analysis is rarely done, and can miss important failure modes due to limitations in the experience of the analyst(s) (Raheja, 1990).

The contribution of the work described herein, therefore, is that it provides a systematic approach for helping workflow designers and users decide *which* exceptions to be concerned about, and *how* to anticipate/detect/avoid/resolve them, in a way that is realistic for workflow settings, building on a substantive and growing knowledge base of generic exception handling expertise.

4. Future work

Our efforts to date have focused on developing the appropriate schemas for capture of exception handling expertise, accumulating a substantive knowledge base of this expertise, and developing methodologies for applying this expertise to workflow design. Our goal is to create increasingly capable *exception handling engines* that interface directly with workflow controllers, and (at least semi-) automatically diagnosing and fixing routine exceptions (e.g. by automating the data-gathering and precondition-checking activities), thereby reducing the exception handling burden on human workflow participants. The key challenges here include formalizing and standardizing (1) the queries that must be made in order to identify and instantiate exception handler processes, as well as (2) the transforms that modify the workflow process model as appropriate to respond to the exception. We have made significant progress towards the first challenge. We have found, in particular, that a relatively small set of question types get used again and again when describing the preconditions for different exception types and handlers. Examples include questions about the status of a task, the status of a resource, the rationale for a task (e.g. its underlying goals) and so on. We are formalizing this set of questions into what we call the "query language" (Klein, 1989, 1993), with the goal of defining a fully capable query language that will be simple enough to allow substantial automation of the exception diagnosis process, i.e. where most or all questions are answerable by software systems. We plan to take advantage of the substantial emerging body of research of specification languages for correctness-preserving workflow transformation (Ellis et al., 1995) to address the second challenge.

For further information about our work, 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/*

Acknowledgements

The authors gratefully acknowledge the support of the DARPA CoABS program (contract F30602-98-2-0099) and the NSF CSS Program (contract IIS-9803251) while preparing this paper.

References

Auramaki, E. and M. Leppanen (1989): Exceptions and Office Information Systems. *Proceedings of the IFIP WG 8.4 Working Conference on Office Information Systems: The Design Process., Linz, Austria.* Amsterdam: North Holland.

Birnbaum, L., G. Collins, et al. (1990): Model-Based Diagnosis of Planning Failures. *Proceedings of the National Conference on Artificial Intelligence (AAAI-90)*. Cambridge, MA: MIT Press.

Broverman, C.A. and W.B. Croft (1987): Reasoning About Exceptions During Plan Execution Monitoring. *Proceedings of the National Conference on Artificial Intelligence (AAAI-87)*.

Chandrasekaran, B. and S. Mittal (1999): Deep Versus Compiled Knowledge Approaches To Diagnostic Problem Solving. *International Journal of Human Computer Studies*, vol. 51, no. 2, pp. 357–368.

Clancey, W.J. (1984): Classification Problem Solving. *Proceedings of the National Conference on Artificial Intelligence (AAAI-84)*.

Davenport, T. (1993): *Process Innovation: Reengineering Work through Information Technology*. Boston, MA: Harvard Business School Press.

deKleer, J. and B. Williams (1986): Reasoning About Multiple Faults. *Proceedings of the National Conference on Artificial Intelligence (AAAI-86). Philadelphia, PA*.

Dellarocas, C., J. Lee, et al. (1994): Using a Process Handbook to Design Organizational Processes. *Proceedings of the AAAI 1994 Spring Symposium on Computational Organization Design. Stanford, CA*. Menlo Park, CA: AAAI Press.

Ellis, C.A., K. Keddara, et al. (1995): Dynamic Change within Workflow Systems. *Proceedings of the Conference on Organizational Computing Systems*. New York: ACM Press.

Friedrich, G., G. Gottlob, et al. (1990): Physical Impossibility Instead of Fault Models. *Proceedings of the National Conference on Artificial Intelligence (AAAI-90)*.

Goldstein, I. (1975): Bargaining Between Goals. *Proceedings of the International Joint Conference on Artificial Intelligence (IJCAI-75)*.

Grover, V. and W.J. Kettinger (eds.) (1995): *Business Process Change: Concepts, Methodologies and Technologies*. Harrisburg: Idea Group.

Hammer, M. and J. Champy (1993): *Reengineering the Corporation: A Manifesto for Business Revolution*. New York: Harper Business.

Harrington, H.J. (1991): *Business Process Improvement: The Breakthrough Strategy for Total Quality, Productivity, and Competetiveness*. New York: McGraw-Hill.

Herman, G., M. Klein, et al. (1998): A Template-Based Process Redesign Methodology Based on the Process Handbook. Unpublished discussion paper. Cambridge MA, Center for Coordination Science, Sloan School of Management, Massachussetts Institute of Technology.

Karbe, B.H. and N.G. Ramsberger (1990): Influence of Exception Handling on the Support of Cooperative Office Work. In S. Gibbs and A.A. Verrijin-Stuart (eds.): *Multi-User Interfaces and Applications*. Amsterdam: Elsevier Science Publishers, pp. 355–370.

Katz, D.M.S. (1993): Exception Management on a Shop Floor Using Online Simulation. *Proceedings of 1993 Winter Simulation Conference – (WSC'93), Los Angeles, CA, USA*. New York: IEEE.

Kettinger, W.J., S. Guha, et al. (1995): The Process Reengineering Life Cycle Methodology: A Case Study. In V. Grover and W.J. Kettinger (eds.): *Business Process Change: Concepts, Methodologies and Technologies*. Hershey, USA: Idea Group, pp. 211–244.

Klein, M. (1989): *Conflict Resolution in Cooperative Design*. PhD thesis. Computer Science. Urbana-Champaign, IL, University of Illinois.

Klein, M. (1991): Supporting Conflict Resolution in Cooperative Design Systems. *IEEE Systems Man and Cybernetics*, vol. 21, no. 6, pp. 1379–1390.

Klein, M. (1993): Supporting Conflict Management in Cooperative Design Teams. *Journal on Group Decision and Negotiation*, vol. 2, pp. 259–278.

Klein, M. (1997): An Exception Handling Approach to Enhancing Consistency, Completeness and Correctness in Collaborative Requirements Capture. *Concurrent Engineering Research and Applications* (March).

Klein, M. (1998): Toward Adaptive Workflow Systems. *Workshop at the ACM CSCW-98 Conference, Seattle, WA*. Available at http://ccs.mit.edu/klein/cscw98/.

Kreifelts, T. and G. Woetzel (1987): Distribution and Error Handling in an Office Procedure System. *IFIP WF 8.4 Working Conference on Methods and Tools for Office Systems, Pisa, Italy.*

Krishnamurthi, M. and A.J. Underbrink Jr. (1989): Knowledge Acquisition in a Machine Fault Diagnosis Shell. *SIGART Newsletter – Knowledge Acquisition Special Issue* vol. 108, pp. 84–92.

Kunin, J.S. (1982): Analysis and Specification of Office Procedures. In *Department of Electrical Engineering and Computer Science*. Cambridge MA: MIT Press, pp. 232.

Malone, T.W., K. Crowston, et al. (1993): Tools for Inventing Organizations: Toward a Handbook of Organizational Processes. *Proceedings of the 2nd IEEE Workshop on Enabling Technologies Infrastructure for Collaborative Enterprises (WET ICE), Morgantown, WV, USA*. Los Alamitos, CA: IEEE Computer Society Press.

Malone, T.W., K. Crowston, et al. (1997): Toward a Handbook of Organizational Processes. Cambridge, MA, CCS Working Paper 198. MIT Center for Coordination Science.

Malone, T.W. and K.G. Crowston (1994): The Interdisciplinary Study of Coordination. *ACM Computing Surveys*, vol. 26, no. 1, pp. 87–119.

Mi, P. and W. Scacchi (1993): Articulation: An Integrated Approach to the Diagnosis, Replanning and Rescheduling of Software Process Failures. *Proceedings of 8th Knowledge-Based Software Engineering Conference, Chicago, IL, USA*. Los Alamitos, CA: IEEE Computer Society Press.

Parthasarathy, S. (1989): Generalised Process Exceptions-a Knowledge Representation Paradigm for Expert Control. *Proceedings of the Fourth International Conference on the Applications of Artificial Intelligence in Engineering, Cambridge, UK*. Southampton, UK: Comput. Mech. Publications.

Raheja, D. (1990): Software System Failure Mode and Effects Analysis (SSFMEA)-a Tool for Reliability Growth. *Proceedings of the International Symposium on Reliability and Maintainability (ISRM-90), Tokyo, Japan*. Tokyo, Japan: Union of Japanese Sci. & Eng.

Strong, D.M. (1992): Decision Support for Exception Handling and Quality Control in Office Operations. *Decision Support Systems*, vol. 8, no. 3, pp. 217–227.

Sussman, G.J. (1973): *A Computational Model Of Skill Acquistion.* PhD thesis. AI Lab. Cambridge, MA: MIT Press.

Visser, A. (1995): An Exception-handling Framework. *International Journal of Computer Integrated Manufacturing*, vol. 8, no. 3, pp. 197–203.