Techniques for Supporting Dynamic and Adaptive Workflow

Peter J. Kammer
Gregory Alan Bolcer
Richard N. Taylor
Mark Bergman

Department of Information and Computer Science
University of California, Irvine, CA 92697-3425

Technical Report 99-03

January 14, 1999

Abstract

The unpredictability of business processes requires that workflow systems support exception handling with the ability to dynamically adapt to the changing environment. Traditional approaches to handling this problem have fallen short, providing little support for change, particularly once the process has begun execution. Further, exceptions vary widely in their character and significance, challenging the application of any single approach to handling them. We briefly discuss the classification of exceptions, highlighting differing impacts on the workflow model. Based on this discussion we suggest principal goals to address in the development of adaptive workflow support, including strategies for avoiding exceptions, detecting them when they occur, and handling them at various levels of impact. We then identify a number of specific approaches to supporting these goals within the design of a workflow system infrastructure. Finally, we describe the implementation of many of these approaches in the Endeavors workflow support system.

Techniques for Supporting Dynamic and Adaptive Workflow

Peter J. Kammer
Gregory Alan Bolcer
Richard N. Taylor
Mark Bergman

Information and Computer Science
University of California, Irvine
Irvine, CA 92697-3425 USA
+1 949 824 8438
{pkammer,taylor,mbergman}@ics.uci.edu, gbolcer@endtech.com
http://www.ics.uci.edu/pub/endeavors/

Effort sponsored by the Defense Advanced Research Projects Agency, and Air Force Research Laboratory, Air Force Materiel Command, USAF, under agreement number F30602-97-2-0021. The U.S. Government is authorized to reproduce and distribute reprints for Governmental purposes notwithstanding any copyright annotation thereon.
The views and conclusions contained herein are those of the authors and should not be interpreted as necessarily representing the official policies or endorsements, either expressed or implied, of the Defense Advanced Research Projects Agency, Air Force Research Laboratory or the U.S. Government.
Approved for Public Release - Distribution Unlimited.

Abstract

The unpredictability of business processes requires that workflow systems support exception handling with the ability to dynamically adapt to the changing environment. Traditional approaches to handling this problem have fallen short, providing little support for change, particularly once the process has begun execution. Further, exceptions vary widely in their character and significance, challenging the application of any single approach to handling them. We briefly discuss the classification of exceptions, highlighting differing impacts on the workflow model. Based on this discussion we suggest principal goals to address in the development of adaptive workflow support, including strategies for avoiding exceptions, detecting them when they occur, and handling them at various levels of impact. We then identify a number of specific approaches to supporting these goals within the design of a workflow system infrastructure. Finally, we describe the implementation of many of these approaches in the Endeavors workflow support system.

1 Introduction

The occurrence of exceptions is a fundamental part of organizational processes (Suchman, 1983). In order for workflow system to support these processes they must be able to support the handling of these inconsistencies and adapt to changes over time. Exceptions can result from such sources as inconsistent data (Cugola et al., 1996), divergence of tasks from the underlying workflow model (Bolcer, 1998), unexpected contingencies (Saastamoinen, 1994), and unmodeled changes in the environment (Tolone, 1996). Efforts to evolve, expand, and optimize the workflow process may also be sources of change that must be accommodated by the workflow system (Abbot and Sarin, 1994),(Nutt, 1996). Traditional approaches have utilized inflexible control policies that make reactive control and graceful exception handling difficult, if not impossible, tasks.

Exceptions impact a workflow model at varying levels of significance. Some exceptions cause only

minor perturbations to the work process. Others may affect only the current running instance of the workflow. The most significant require the process model itself to evolve to accommodate changes that have occurred in the environment. These different classes of exception require different approaches to support their handling and recovery as well as evolution within the workflow system.

Also of importance to exception management, however, is the ability to detect exceptions, or the possibility of an exception quickly as it occurs, improving the ability of participants to react and recover, minimizing lost time and wasted effort. In addition, some basic design approaches may help prevent exceptions, particularly those caused by incompatibility with elements of the work environment, poorly informed participants, or conflicts between workgroups.

Our research and development of the Endeavors dynamic web-based workflow system (Bolcer and Taylor, 1996), and its predecessor system (Young, 1991) have suggested a number of approaches to support these various goals.

In Section 2, we will provide some background and context for this work, briefly examining some ways of classifying exceptions and their impact. In Section 3 we discuss traditional approaches to adaptive workflow across several communities. Section 4 identifies and describes our principal goals in the support of adaptive workflow. Specific approaches to supporting those goals are identified in Section 5 and the implementation of many of those approaches in the Endeavors system is discussed in Section 6. Section 7 presents conclusions and future work.

2 Background and Context

In order to inform and motivate our discussion of workflow system design and functionality, we consider some of the work in studying and categorizing exceptions in organizational workflows. The term "exception" itself is used somewhat differently by different authors, with divergent emphases on organizational or system views. A number of authors express it as something that is undefined within the system model. For example, Auramäki and Leppänen (1989) describe an exception in terms of whether a complete set of rules is available to handle a condition. Strong and Miller (1995) use a similar concept, indicating exceptions are "cases that computer systems cannot process correctly without manual intervention."

Barthemess and Wainer (1995) define exceptions more organizationally, in terms of what is normal for particular process, as "departures of the history of a workcase from its prescribed (or normal) flow." This definition is somewhat more inclusive. In addition to events not modeled by the system, it also accounts for deviations from the normal flow for which models and behavior are defined, sometimes called *expected* exceptions (unmodeled events are *unexpected* exceptions.) Saastamoinen (1995) instead describes expected occurrences as *variations* and exceptions as events that cannot be handled by the main process flow *or the procedures that handle variations.*

For the purposes of our discussion, we are primarily concerned with the narrower class of "unexpected" exceptions. We are, of course, interested in mechanisms that assist in migrating some of these unexpected occurrences into expected variations, evolving the workflow model.

2.1 Perspectives on Sources of Exceptions

Strong and Miller (1995) identify a number of different perspectives on exceptions in information

processes. The *random event perspective* suggests that exceptions are random and uncommon occurrences. This perspective is, according to the authors, commonly assumed by managers and researchers. It is not, however, born out by research studies (e.g. Suchman, 1983).

The *error perspective* is that exceptions are errors to a normally executing process that may be targeted and eliminated. These can be further distinguished into subcategories describing the source of the error. *Operation errors* are those that result from mistakes in the execution of the process. *Design errors* result from problems in designing and implementing the system. Finally, *dynamic organizations* may also be a source of error, as the organization changes, becoming inconsistent with the system model. The solution approach associated with this perspective is Total Quality Management (TQM) which attempts to make continuous improvements in the process to eliminate these errors.

The *political system perspective* (see also Kling and Iacono, 1984) views workflow in terms of subunits in an organization with potentially conflicting goals. The need of varying organizational entities to accomplish their own goals, along with differences in influence over the specification of work processes, can generate exceptions in the workflow enaction. This perspective views exceptions as a normal part of organizational processes. The associated solution approach for this perspective lies in human centered system design that considers human as well as technological participation in the process.

These divisions provide insight into the general sources that might generate exceptions. Other views on the sources of exceptions may be found in (Saastamoinen, 1995) (more specific causes) and (Barthelmess and Wainer, 1995) (oriented toward the software system).

2.2 Scope of Exceptions

Of particular concern in handling the result of an exception is limiting its impact on the progression of the workflow and the organization. Saastamoinen, (1995) provides an extensive taxonomy of exceptions and divides potential impact into three levels:

- **Employee level**: Impact is limited to a single individual.
- **Group level**: Impact stretches across a group working on the same project, process, etc.
- **Organization level**: Impact is organization wide, covering more than one group.

The potential impact goes even beyond the organizational level described by Saastamoinen. Workflow is increasingly distributed and may not only represent operations within a single organization, but across boundaries between such entities, specifying such things as hand-off relationships and artifacts to be exchanged. (see, for example, Ben-Shaul and Kaiser (1994)). An exception in this scenario could easily have an impact stretching beyond a single organization. Recovery is a particular challenge across organizations as there is less constiatent control over work policies, products and procedures. In addition, existing internet protocols to not address these issues beyond disseminating information. For one effort to address the problem of coordinating distributed workflow systems see (Bolcer and Kaiser, 1999).

2.3 Consequences of Exceptions to the Work Model

The ramifications of an exception can be described not only in terms of how they impact the organization, but also how they impact the work model. We distinguish exceptions into three classes. Exceptions that can be tolerated by the process or safely ignored and still produce a satisfactory result

we refer to as *noise*. Some exceptions are relatively unique to a specific work instance or set of work instances yet still require changes be made to the process for those instances. These can be described as *idiosyncratic*. Finally, *evolutionary exceptions* require changes in the overarching workflow model (resulting, for example, from changes within organizational procedure). It is these exceptions that produce evolution over time, driving long term change in the work process. This classification corresponds closely to affects on office rules (the work model) described in (Auramäki and Leppänen, 1989). We summarize this relationship in the table below.

Table 1: Relationship Between Types of Exception and Changes to Workflow Model

Exception Type	Change in Workflow Model
Noise	None
Idiosyncratic Exceptions	Changes to specific instance of workflow, but the workflow type (the general model) remains the same
Evolutionary Exceptions	Evolution of general workflow model, affecting future instances of work process as well

The type of exception impacts the kind of support that should be provided by the underlying system. Depending on the classification, the needed functionality will vary. We will refer back to this classification when we discuss the goals of our work in Section 4.

3 Traditional Approaches

Traditional work solutions arise from a number of different areas of research. They possess overlapping constraints which limit their support for dynamic adaptation to change as well as the ease of their integration and adaptation to existing work environments. Even though the trend is toward convergence (Ellis and Nutt, 1996) we address issues of individual approaches here, highlighting obstacles to supporting exception handling and flexible process execution.

3.1 Process Technology

Process technologies typically assume closed world data consistency (Osterweil, 1998), This is usually enforced by limiting the actions of the user or agent to those only pre-specified by the process designer. In a dynamic, evolving real world environment, sometimes it is necessary to jump out of the process or out of the system to adhere to the specified process model. Process models that are over-specified are difficult to follow because their requirements for completion are too rigid. Process technologies that support evolution as a mechanism to address this problem require that changes be monotonic, i.e. each successive change to the workflow model is additive, incremental and always strongly dependent upon previous models for data integrity and consistency. Consistency sometimes must be violated to meet both functional and non-functional requirements of the process being followed or the product being produced. Coordination between dispersed process participants is sometimes difficult because a uniform representation of activities, artifacts, and resources may differ among people, groups, and organizations. Adding the complication that various participants possess differing skills, different levels of understanding of their obligations, and may require domain and context specific views, misrepresentation and miscommunication is often the result.

3.2 Workflow Technology

Traditional workflow technologies have achieved relative success in the workplace through simplification. These systems have been applied to problems of limited scope and scale. Workflow specifications may be ambiguous or contain inflexible semantics. Visual workflow programming languages sometimes lack representations for timing and execution constraints as well as complex relationship specification and management between process objects and people. Lack of relationship management can precipitate poor exception handling. Workflow processes that diverge from the intended series of planned activities or require some midstream changes may result in exceptions. Workflow technologies typically lack the infrastructure to query status and change values of their own processes, correct midstream deviations, or locate and reserve needed resources and artifacts. Often exceptions, whether requiring minor automated intervention or human participation to correct, require the reset and restart of a workflow process. Generally, workflow processes once they are deployed into their execution context are not changed. Significant changes may be difficult to accomplish in this environment as the workflow process model, the execution infrastructure, and the process presentation are likely to be tightly coupled.

3.3 Groupware and CSCW Technology

Groupware and CSCW are broad labels that describe a gamut of synchronous and asynchronous technologies including email, shared whiteboards, meeting schedulers, collaborative desktops, video conferencing, and other shared electronic media. While these technologies are useful for overcoming communication problems over time and collaboration problems over distance, they lack the guidance and automation mechanisms for performing structured tasks. While using these tools to accomplish unstructured tasks mimics some real world work environments, they do not guarantee consistency of results, maintain a standard of practice and procedure, nor lend themselves to optimization, improvement, and training. There is no explicit work model other than ad hoc utilization of the components at hand. There is no measurable status for determining completion, progress of a task, or even mechanisms for describing what work needs to be done other than capturing communication and collaboration relationships between participants. Adding a work model and management capabilities to a groupware or CSCW technology often leads to additional work on the part of the participant with no direct benefits such as guidance or automation. Synchronization between the proposed work and the actual work is often done by hand. This overhead of model maintenance may lead to the demotivation of the participants which is crucial to the success of an activity.

4 General Goals for an Adaptive Workflow System

In Section 2.3 we described a delineation of exceptions based on their impact on a workflow model. This categorization provides a guide for classes of functionality that should be provided by software systems supporting workflow. Further, we suggest two other needed emphases, detecting when a problem occurs and designing to avoid some of the causes of exceptions. While we briefly mention some here, we will discuss specific approaches to these goals in the next section.

4.1 Detecting Exceptions

The discussion of handling exceptions tends to proceed with the assumption that the anomaly has already been detected and fully described. A workflow system needs to support the discovery of

exceptions in a timely and useful manner. While some exceptions will be obvious to users, others will be more subtle. The initial error may not be the one that eventually derails the workflow. Data constraints or general rules about workflow progress may be used to provide forewarning of problems occuring, giving clearer indications of the source of the exception. Active data stores and event driven architectures may provide alerts when constraints are violated or inconsistencies arise. Multiple appropriate views of workflow execution can provide visualizations of where problems may be occurring. Reflexive process agents can provide analysis and feedback on workflow execution as it progresses.

4.2 Avoiding Exceptions

A workflow support system, integrated into an existing work environment, may itself be the cause of exceptional conditions. If a system is rigid in its use, involves significant adoption cost, or integrates poorly with existing tools and approaches, participants are likely to operate outside the system. Ownership of work procedures may be divided among subunits (or even across organizations). Open systems, support for incremental adoption, flexible execution approaches, reusable process components, and integrated support for communication between participants, may all help avoid conflicts that create exceptions.

Customized agents and event monitoring infrastructure are useful for gauging the effectiveness of these workflow components. Further, when a new process is introduced to a work environment and culture, there is often some pushback to its adoption. It is important for the process to be able to adapt in response to this pushback. In addition, process discovery tools are useful for comparing and validating the workflow model with the actual work being accomplished. Wide divergences can indicate technology mismatches or inapplicability and thus the need for evolution and optimization of the process.

4.3 Handling Exceptions

4.3.1 Tolerating Minor Deviations "Noise" in workflow execution describes minor deviations from the normal process that are not significant enough to require changes in the execution process model. Noise may be either accommodated within the tolerance of the model, or may produce consequences which may be ignored without unacceptable harm to the continued execution of the process.

An overly rigid workflow description, or one that requires overspecification of work activities results in workflow specifications that may be fragile, not accommodating reasonable minor deviations from the "ideal" process. Even requiring complete specification of process and resource dependencies prior to execution may contribute to this problem. While, the exact character of deviations from the main process may not be known, it may be possible to determine locations where added flexibility is required and permissible. Appropriate support for abstraction, flexibility in degree of specification, and a flexible execution model can provide some tolerance for minor deviations.

4.3.2 Handling Changes to the Process Instance Idiosyncratic exceptions are particular to a workflow instance or collection of instances. For example, if a regular participant is unavailable (e.g. sick or on vacation) then changes might be required to accommodate the absence. These would not be reflected in the model process however because the individual would resume the same role upon returning. This sort of change requires the ability to modify individual instances of the execution process without altering the overarching process type. A dynamic instance model of the workflow, created or modified

"on-the-fly" at run-time helps accommodate this sort of change (see, for example, Cugola (1998)).

An alternative approach to addressing this issue is not to specify a model with precise ordering of activities, but rather to provide constraints and present the user with multiple available tasks to perform. Freeflow (Dourish et al., 1996), for example, uses this approach.

4.3.3 Evolution and Optimization of the Process Model Evolution of process objects and workflows occurs over time as a result of changing tasks, priorities, responsibilities, and even people. Optimization occurs when the improvement of a previous work model results in a better way of doing things by adding, removing, or redefining process activities and their constraints. As a workflow process is repeatedly executed small changes may be made each time through. Eventually, a process may converge on a common practice and become institutionalized.

Unfortunately, a common practice and a best practice may not be the same thing. In order to determine this, metrics must be kept to evaluate one execution from another. This evaluation is subjective because the criteria underlying the metrics changes the same way the workflow does. Successful workflows, like successful software, will be applied in new situations that may have been unanticipated by the original creator, and unsuccessful workflows will be abandoned or changed. It is important in the workflow infrastructure to allow the change to occur both before and after deployment of the workflow, by both technical and non-technical participants. Some optimizations may even be performed by the system itself through agents, validation tools, or optimizers.

5 Functionality for Adaptive Workflow

Keeping in mind the principal goals we described in Section 4, we identify a number of functionalities to support managing exceptions and adaptation in workflow. These specific mechanisms arise from our experience in the research and development of the Endeavors workflow support system. Most of the techniques are implemented within the existing research system, with the remainder prototyped or targeted for future work. We provide an overview of Endeavors and a breakdown of implemented features in Section 6.

5.1 Dynamic Change and Composition

The ability to dynamically modify a process definition, the associated data, behaviors associated with objects, and the set of views into the process, at the time it is in progress, is crucial for workflow process execution and evolution over time. This allows the workflow to better fit changing requirements, availability of resources, and the applicability to the current work context.

While introduction of dynamic change to a process and its representation may require additional infrastructure to recognize and enforce consistency, limit access to change mechanisms as appropriate, or correct problems that impact other parts of the process, the ability to dynamically evolve in conjunction with the work environment, culture, and context is important to keeping the online description of work consistent with the actual work being performed. Long running, distributed processes involving multiple stakeholders will encounter a multitude of situations demanding change, escalation, and reorganization. A number of authors have addressed process dynamism as a fundamental component of handling exceptions. See, for example, (Ellis and Rozenberg, 1995) and (Cugola, 1998).

Late binding of resources in a workflow allows completion of activities using the resources at hand at the specific point in time the work is actually done. Planning and scheduling components should complement late binding to ensure that the required resources are available at the appropriate times to complete the tasks at hand. An example of late binding that provides a good mechanism for supporting dynamism in process object management is the separation an object's data from its behaviors. Object behaviors can be dynamically loaded and resolved as they are invoked. Object behaviors may be updated independent of the object's attributes.

5.1.1 On-the-fly Workflow Composition For some projects, even the principal workflow path cannot be completely specified prior to the start of execution. This occurs when downstream details are dependent on upstream results not available when the execution begins. Processes are dynamically composed as execution progresses. In this approach, sometimes referred to as "just in time" execution, the definition of the workflow is not created until needed. Examples of this sort of application include bug-tracking/resolution and experimental or exploratory workflow solutions.

5.2 Configurable Execution Models

A good recovery mechanism should permit execution of a disrupted workflow to resume, start at an arbitrary midpoint, or rollback to a previous point in the process using computer or human execution of reset, restart, undo, complete, abort, recover, ignore, or jump operations on the process interpreter (Kaiser et al., 1998). This allows participants or managers with appropriate authority to tailor invocation of work instances as needed for a particular occasion. This not only assists in recovering from exceptions, but also in avoiding them if they are anticipated ahead of time.

5.2.1 Partial Execution Partial execution supports dynamic composition by allowing the execution of fragments of an incomplete process. This approach is analogous to a cat's cradle, a child's game in which an intricately looped string is transferred from the hands of one player to the next, resulting in a succession of different loop patterns. It should be possible to pick up the execution of a process and continue it from any point specified including the dynamic reorganization of local relationships and constraints to fit the new work context. Partial execution supports multiple iterations of a process fragment or multiple alternate iterations of the same process fragment changing order, priority, focus, or completion criteria of the fragment. Support for resolving and integrating processes via pipe-and-filtering, re-stringing, rework, and amalgamation of diverse processes which include possibly competing priorities should be included into the workflow execution infrastructure. This may include temporal and spatial context awareness in addition to the possible execution space and control, coordination, and collaboration policies. For individuals or groups, this may include several partial executions, repeated until the artifact is good enough to post or hand-off to the next participant. While partial execution techniques may create ambiguity and diminish the ability to do global optimizations across all activities before execution, work specifications are generated on-the-fly and on-demand allowing many local optimizations based on discriminates available at execution time.

5.2.2 Guidance versus Enforcement A tailorable execution model also allows an appropriate level of process enforcement. Varying levels of prescriptiveness may be required at different points within the process. While it is often desirable to give participants a high degree of flexibility, standards of practice within an industry or policies within an organization may require that a series of activities are conducted without deviation in order for work to proceed. This may be true, for example, for reasons of security or safety. On other occasions, the workflow model ay serve only as a guide and assistant for participants who have a great deal of autonomy in their choice of activities. A workflow system should provide the

mechanisms to allow varying and appropriate controls throughout the workflow process.

5.3 Typed modeling of Activities, Artifacts, Resources, and Agents

By modeling entities in the process as typed objects, not only is information about the component available to the workflow model, equivalent classes of elements can be determined. An appropriate one may be selected based on information available at run time. For example, if any one of a number of meeting rooms may be appropriate for a review meeting, one of the appropriate type may selected based on information at run-time and associated with the activity.

Suitable typing of resources provides flexibility within a process. It allows users to select from a range of tools of the same class to perform an activity. The model of the given tool within the process can perform necessary manipulations of the result to incorporate it within the work model. Typed modeling allows choices to be made at run time with some assurance of minimal impact on the overall consistency of the work in progress. Description and modeling of equivalent components must be done carefully, of course, to assure that they will genuinely provide comparable behavior.

5.4 Reflexivity

Dynamic behavior in workflow execution is facilitated by a reflexive workflow process. A workflow process is reflexive if during execution it has the ability to remodel itself, either automatically by an agent or through the intervention of a human stakeholder, in response to status and feedback information about how the process is perceived to be progressing. A workflow process or infrastructure component should have the ability at any time to construct, query, and manipulate its own process fragments based on the running state and the target environment.

Based on information about the environment a process component may be able to integrate with new elements in the environment, optimize scheduling and constraints, and improve the organization of relationships and dependencies. Specific rules or strategies may provide a path for a reflexive workflow component to evolve. Such a component might be able to improve efficiency in systematic ways in response to quantitative measures collected over time.

Supporting reflexivity in a workflow infrastructure allows knowledge about a workflow's applicability to the context and the effectiveness of its deployment to be evaluated. Keeping track of a change history in addition to being able to derive change trees and programmatically form queries about them provides a mechanism for revisiting evolutionary branches. Catalysts for change, whether for better or for worse, are easier to isolate and recognize. In this way, reflexivity provides the foundation for continuous, at least partially automated, process monitoring and optimization. This activity might be hierarchically distributed throughout the workflow to provide for both localized and generalized optimization.

In addition to monitoring and manipulating existing processes, reflexivity is also useful for generative processes where, during the execution of the process, new process elements are created. This is applicable not only to the processes that are built as they execute (the dynamically composed workflows we described above), but also to workflows whose purpose is to create another process tailored to a specific application.

5.5 Logically Decomposable Process Models

The ability to decompose process models into components fragments provides a number of advantages in avoiding, containing, and recovering from exceptions. Abstractions, for example hierarchical decomposition, in workflow can enhance understandability and specify points of interaction between sub-processes. Process fragments may be assigned to activities dynamically at run-time.

Workflows often stretch across workgroups and even organizations, potentially creating conflicts in workflow specification and enaction (again, see (Kling and Iacono, 1984)). By decomposing the process into constituent fragments, process ownership, coordination, and hand-offs can be explicitly specified. Subprocesses associated with activities may be controlled by the appropriate subunits, diminishing conflict and separating responsibility. As with modularization in software, workflow abstractions can also be used to scope the appropriate detection and handling of exceptions.

5.6 Evolving Process Models

In Section 5.1, we discussed the on-the-fly composition of processes, letting the workflow model be built as needed to accommodate less defined work areas. Once composed, this instance may serve as the guide model for the next instance of the process. Process instances should be reusable, divisible, and synthesizable into overarching models. Combined with reflexive analysis and human assistance multiple iterations of process enactment can guide the refinement of the process model over time, highlight areas needing less stringent modeling or increased flexibility, and guide optimization.

5.7 Reusable Process Fragments and Component Libraries

To better support process evolution and optimization, process fragments should be easily reusable, divisible, understandable, and capable of being evaluated against some measurable criteria with respect to expected execution or anticipated behavior. A fragment that is successful in one context is likely to be applicable to another similar context. For instance, in a software testing process, the tester may require that the same outcome be reached over repeated executions. Different outcomes imply different qualities of the software. Similarly, a process can be used to develop the skill of a particular student in a training domain where the end-user's path through the process is dependent upon their skill. The inculcation results in the goal of visiting every activity or series of activities through repeated executions and increased skills. The process may change based on feedback, usage, level of interaction, and number of times executed. As with all changing components, change management techniques such as version control and transactions should be integrated with the system.

Guided by work such as that of Malone et al. (1997), which is oriented toward creating a handbook of organizational processes, it is possible to build libraries of reusable process components. Of particular interest are process fragments, sequences of activities with associated artifacts and resources, that can provide a standardized approach to a complex task. These could include configurable boilerplates of common processes, as well as standards of practice that specify a particular process to follow to assure the integrity of the result.

At a more general level, process workflow "patterns" would represent approaches that may be tailored to a particular application. Further,"templates" collect the various process components described into a toolkit oriented toward a particular application area.

5.8 Access to Organizational Work History and Expertise

By their nature, exceptions represent circumstances where information about appropriate practice or solutions is not close at hand. Handling exceptions may require a particular knowledge or historical perspective not immediately available to the workflow participant. Approaches to similar situations or expertise with a particular activity may inform an exceptional situation and speed recovery.

This sort of information and perspective may be integrated within the workflow system to provide appropriate information to assist in the recovery of exceptional cases. Historical information about previous instances of the workflow may provide insight including participants with applicable expertise. Locating expertise within the organization is itself a complex issue (see McDonald (1998) for one study), but may be augmented through integration with tools designed for storing and locating such information (e.g. Ackerman and McDonald, 1998).

5.9 Event Monitoring Architectures

Within an executing workflow instance, exceptions may manifest themselves through obvious occurrences (such as the appearance of error messages) or obvious inconsistencies with the actual work environment. Problems may be more subtle, however, and either not manifest themselves until significant progress has been made through the workflow, or until an unsatisfactory result is obtained.

Event driven architectures provide means for monitoring events as they take place within the workflow system and the executing workflow itself. An active object-store, i.e. one that reports on changes to process data as it occurs, can activate routines to monitor data and provide warnings when constraints are violated, inconsistencies arise, or time limits are exceeded. Reflexive components can monitor the performance of processes and provide information to autonomous agents or human-readable visualizations to locate potential problems before the workflow is derailed. For more on event driven architectures see, for example, (Taylor et al., 1996). Further, event monitoring on an internet scale can provide mechanisms for remote monitoring and analysis. (see Rosenblum and Wolf, 1997).

5.10 Integrated Support for Participant Communication

Managing work processes involves not only managing the people, products, activities and resources, but also the network of discussions that maintain coordination between participants. This is a particular issue when workflow is widely distributed geographically or organizationally. Often communication within the workflow centers around elements represented in the system, for example activities being performed or artifacts being acted upon. Associations between conversation and these entities can provide not only a focus for discussion, but may also capture historical information about rationale and provide insight into how the work process may be improved in the future.

Communication, and its relationship to other elements in the workflow, is particularly important when the unexpected happens. When the process encounters an exception, participants must work together to take corrective actions and resume the normal process. Close integration of tools for informal communication with a workflow support system not only expands the capability to model the process being enacted, but increases the ability of participants to recover from unexpected occurrences without having to "break out" of the workflow system to make corrections.

5.11 Partial Adoption and Integration

Many conflicts in workflow systems can result from the system being poorly integrated within existing

work tools and practices. Many systems require an "all-or-nothing" buy-in to make using the system worthwhile. Some of the practices we have already described, for example composing processes on-the-fly, evolving process models, and process component libraries, support this sort of incremental adoption. Further, a highly componentized system, one that allows users to adopt the system a piece at a time, can ease integration into existing environments, allowing functionality to be tailored to user needs. In addtion, support for multiple stakeholders, providing customized views into the work being performed, limits confusion, providing only appropriate and necessary information to participants.

In Section 5.3 we discussed modeling of external tools independently. The ability to integrate flexibly with existing resources also is an important factor in avoiding inconsistencies and conflicts in the workflow model. This flexibility allows a system to be integrated into already existing work practices without forcing participants to adopt new technologies. In addition to manipulating external tools, an open workflow system provides a mechanism for complete integration, allowing reflexive access to workflow components by the external systems. These sorts of integrations smooth adoption of a workflow system and provide mechanisms for adapting to changes in the environment over time. For full discussion of various kinds of tool integration (in the Endeavors system) see (Kammer, et al, (1998)).

5.12 Supporting the Goals of Adaptive Workflow

Table 2 provides an overview of how the approaches we have discussed in this section may be applied to the various goals we put forth in Section 4. While some approaches are targeted toward specific goals, others provide mechanisms to support several. Approaches that strengthen the ability to manipulate workflow tend to support both the handling of exceptions on an instance level and also the evolution of the process model over time. Many approaches that provide information or capability for recovering from exceptions can also help avoid them if taken advantage of prior to the exception occurring.

Table 2: Relating Functionalities to Supporting Adaptive Workflow Goals

	Tolerating Minor Deviations	Instance Level Exceptions	Evolving Process Models	Detecting Exceptions	Avoiding Exceptions
Run-time Dynamism (Data, Behavior, Process)		X	X		X
Configurable / Partial Execution	X	X			X
Typed Modeling of Artifacts, Activities, Resources		X	X		X
Reflexivity		X	X	X	X
Logically Decomposable Process Models	X	X	X		X
Evolutionary Process Models			X		
Reusable Process Fragments and Component Libraries			X		X
Access to Organizational Work History / Expertise		X	X		X
Event Monitoring Architectures				X	
Integrated Support for Participant Communication		X		X	X
Partial Adoption / Integration					X

6 Dynamic Adaptive Workflow Support in Endeavors

Endeavors is an open, distributed, extensible workflow support environment. It improves coordination and management by allowing flexible definition, modeling, and execution of workflow applications. Endeavors combines a sophisticated process modeling language with features designed for easy customization by both technical and non-technical users.

6.1 Dynamic Process Object Model

Endeavors uses a layered object model to provide for the object-oriented definition and specification of process artifacts, activities, and resources. Behavior of process objects is specified through the use of handlers: code invoked by the object in response to events received. Stored locally or loaded from a remote source, handlers are loaded and bound to objects at runtime, allowing them be changed dynamically in the course of process execution. Handlers themselves may reflexively access the state of the workflow through Endeavors interfaces, allowing for analysis and optimization by components of the process itself.

Activity networks associate activities by control-flow, data-flow, and resource-flow relationships. These objects and relationships may be changed dynamically to accommodate changes in the workflow and its environment. Abstraction and decomposition is achieved through hierarchically associating sub-networks with activities (see Figure 1 for an example). Process fragments from one network may be used in another through "cut-and-paste" style operations. Workflow components and processes may be extended and specialized through the usual object-oriented approaches.

6.2 Flexible Interpretation

Networks are executed by interpreters that traverse the network and send appropriate events to objects to invoke the objects' behaviors. A rich user interface provides for the dynamic specification of interpreters and control of their execution. Interpreters may be visually manipulated as they traverse activities, allowing them to be started, stopped, and redirected as needed. By altering the configuration of the interpreter or substituting an alternate interpreter component within the Endeavors system, the execution model may be tailored to the needs of a particular environment providing, for example, the appropriate level of guidance versus prescriptiveness at different points in the workflow.

Figure 1 shows one example view of an Endeavors activity network. The largest window shows the top level process. The "palette" on the left provides access to the basic language constructs and an extensible collection of activities. These activities are composed into the workflow process and tailored to specific needs.

The activity *Dept. Approval* is expanded into the sub-network shown below the larger network. Also visible is the main control panel and a dialog for editing the individual attributes of an activity.

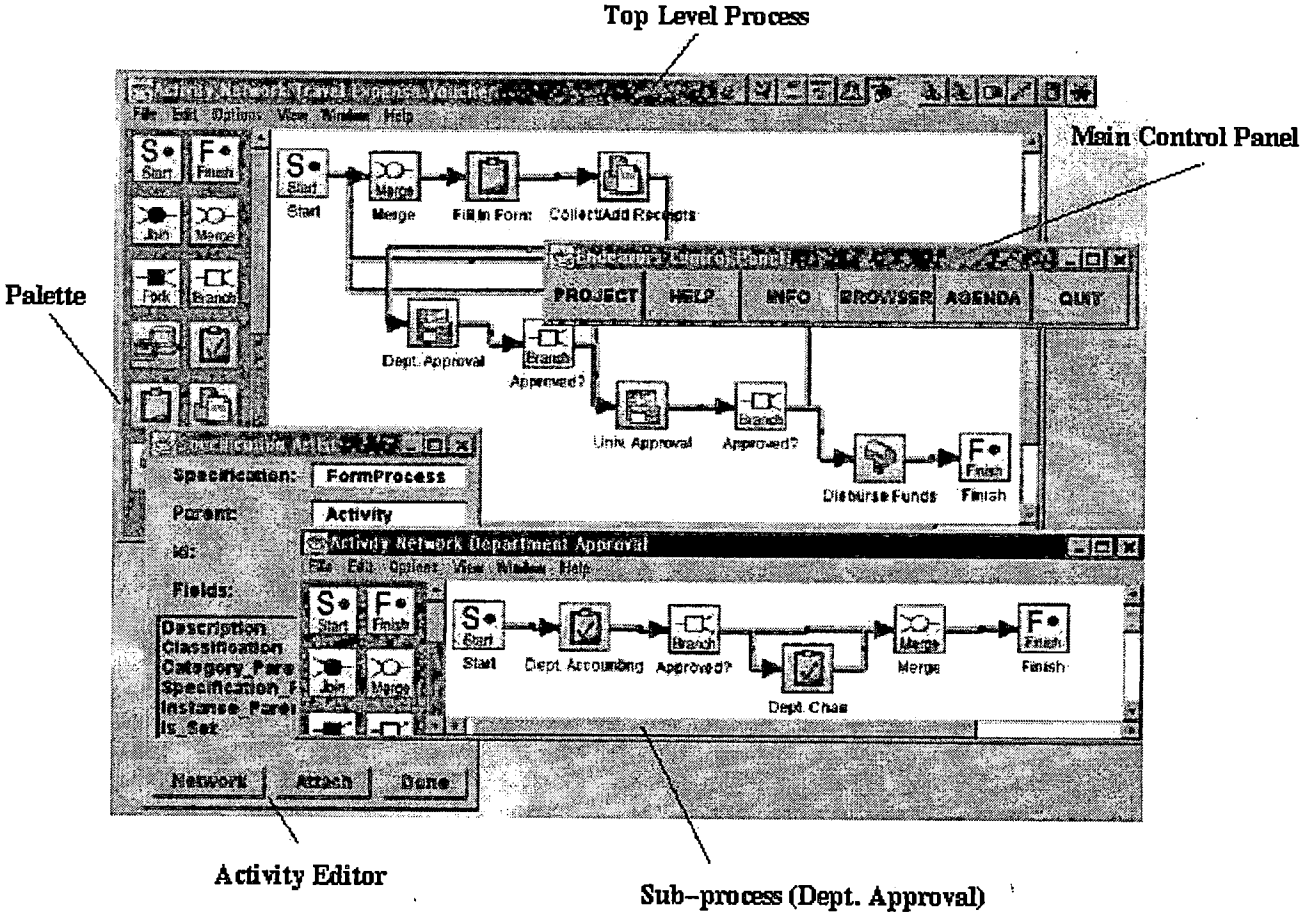

Figure 1: *One view of Endeavors activity networks*

6.3 Open Architecture

Figure 2 shows a high level view of the system architecture and functional breakdown. Endeavors has a three-tiered architecture. Each layer has its own object model and responsibilities. The user layer provides the interface for human interaction with the underlying system and processes. It monitors events from the underlying layers and maintains a coordinated view of the system and the processes specified. Beneath the user level, the system level provides the domain object model. At this layer, artifacts, activities, and resources may be programmatically created, manipulated, associated in networks, and executed by an interpreter. The foundation layer implements the *class-metaclass model* (Young, 1991), managing the loading of objects and handlers as well as their persistent storage. This layer triggers objects' handlers in response to received events.

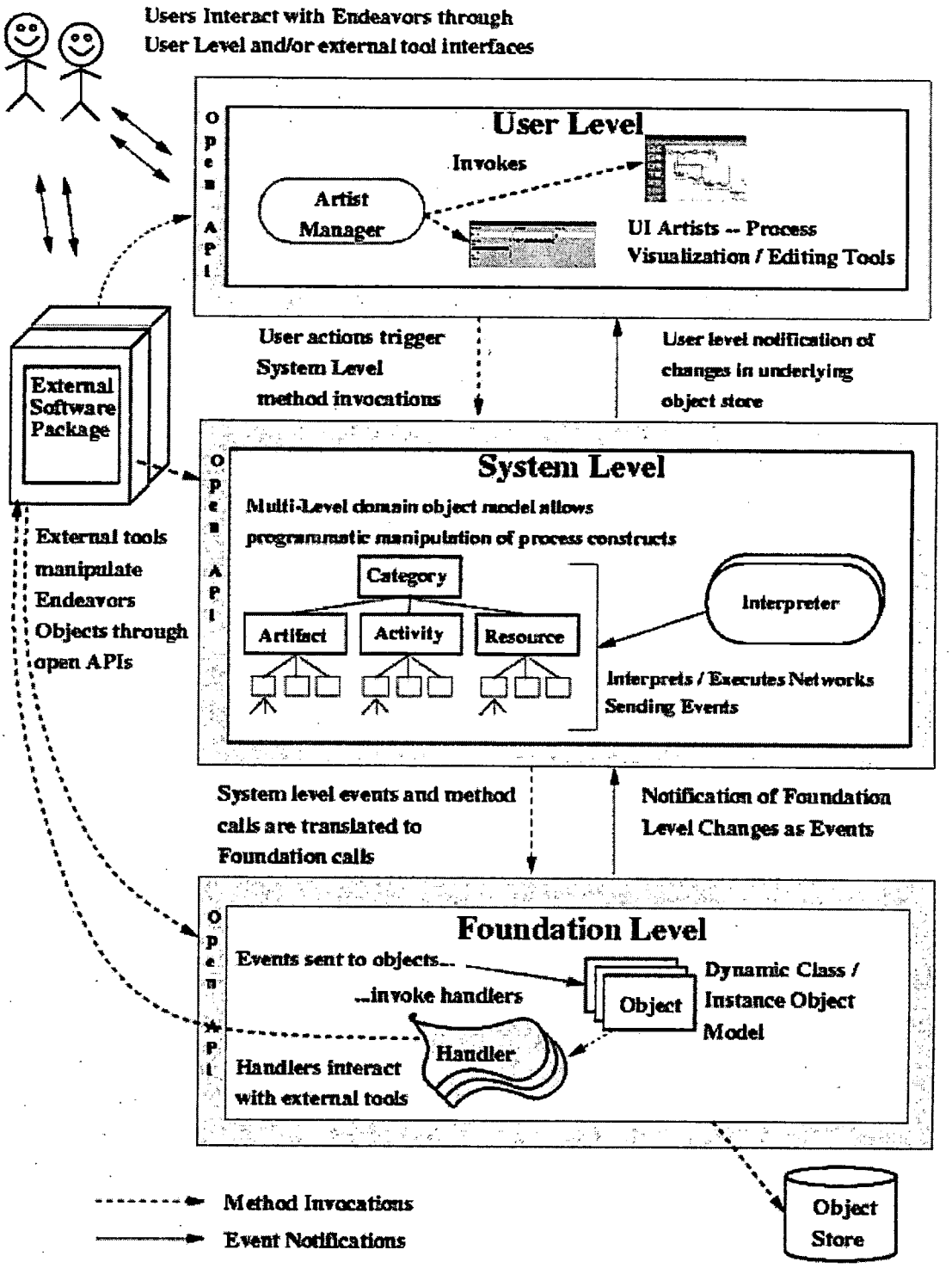

Figure 2: *Endeavors layered system architecture*

Endeavors provides an open architecture that simplifies integration with external tools, providing for two-way communication, and simplifying incorporation into preexisting work environments. Handlers may activate and manipulate external tools through existing or custom APIs. In a manner similar to reflexive workflow components, external entities may access Endeavors through its open interfaces

provided at each layer of the system's virtual machine architecture, manipulating process state or the processes themselves (subject, of course, to authorization controls).

Endeavors leverages off the capabilities of the Java Programming Language (Gosling, 1996) in which it is entirely written. The system is highly componentized to facilitate incremental adoption and reuse. System functionality may be downloaded to process execution sites as required without the need for explicit installation processes.

Leveraging off of standard protocols, Endeavors components may be distributed across multiple sites to allow shared access to resources and services. This extends capabilities for supporting participants across organizations, supports remote participation and management of workflow, and extends possibilities for integration. For a full discussion of these issues see (Kammer et al., 1998).

6.4 Summary of Functionalities Implemented in Endeavors

Development of Endeavors informed the identification of the approaches discussed in Section 5. Most are already implemented within the system. Others are prototyped or a focus of ongoing research. In brief, Endeavors improves on the current generation of workflow technologies by:

- using a dynamic extensible typing model,
- providing rich execution level semantics facilitating analysis and optimization,
- supporting incremental adoption and use,
- including configurable control policies for flexible approaches to workflow execution,
- providing a broadly applicable approach, rather than a point solution, leveraging off internet protocols to provide lightweight distribution.

Table 3 summarizes the level of implementation of these features in Endeavors, and areas of future research emphases. Additional detail about Endeavors may be found in (Bolcer, 1998) and (Bolcer and Taylor, 1996).

Table 3: Implementation of Adaptive Workflow Approaches in Endeavors

Approach	Endeavors Implementation
Run-time Dynamism (Data, Behavior, Process)	**Implemented**: Data model supports binding of data, behaviors and process elements at runtime, allowing on-the-fly composition and change of workflows.
Configurable / Partial Execution	**Implemented**: Execution model may be tailored to individual workflow needs. Execution may be reflexively controlled by workflow agents or authorized human participants
Typed Modeling of Artifacts, Activities, Resources	**Implemented**: Process entities are modeled with a layered object-oriented mechanism, supporting typing, generalization, and specialization of object categories.
Reflexivity	**Implemented**: Process components as well as authorized external tools may use Endeavors interfaces to access and manipulate workflow definitions and the current state of execution.

Logically Decomposable Process Models	**Implemented**: Process may be hierarchically decomposed into sub-processes, providing abstraction and separations of responsibilities and ownership. Process fragments may be manipulated and reused in other processes.
Evolutionary Process Models	**Implemented**: Using a layered object model, workflow instances may be evolved into models for later executions
Reusable Process Fragments and Component Libraries	**Supported**: Endeavors provides an infrastructure for libraries of components and process fragments. **Future Work**: Development of domain specific and generalized workflow components, standardized processes, and process fragments
Access to Organizational Work History / Expertise	**Future Work**: Provide integrated access to appropriate work history and expertise, most likely through rich integration with third-party tools.
Event Monitoring Architectures	**Implemented**: Endeavors is based on an event driven architecture. **Future Work**: Improved ability to distribute observable events to integrate with external tools. Improved support for event driven processes.
Integrated Support for Participant Communication	**Prototyped**: Close integration with a third-party conversation (chat) tool-kit. **Future Work**: Integration of additional CMC tools, incorporating communication artifacts within the workflow model.
Partial Adoption / Integration	**Implemented**: Highly componentized open system, supporting adoption in pieces and integration with tools in existing environments. Multiple views may be tailored to individual participants. Integration with standard protocols and tools for distribution limits buy-in cost.

7 Conclusion and Future Work

Process workflow systems are often called upon to adapt to a changing environment. Exceptions arising from such causes as inconsistencies with the actual process or unexpected occurrences often drive the need to adapt. Alternatively, the need to evolve or optimize the process may drive the changes in the process model. Traditional approaches to handling dynamism in workflow have generally fallen short. Some attempts have proven too rigid, not easily tolerating dynamic change once a process starts executing. Others lack the structure to provide a coherent process model, identify exceptions, guide responses, or manage evolution of processes over time.

Exceptions vary in their impact on the workflow model. This variance suggests the need for different approaches to support different ways of handling exceptions. Overall flexibility, integrated support for dynamic change, and appropriate information all contribute to the ability to handle unexpected occurrences as they develop. Systematic support for evolution can reduce (though not likely eliminate) unexpected exceptions over time.

No single approach will be applicable in all cases. Building on basic goals, we have highlighted a number of mechanisms that provide a range of information and capabilities to help build flexible solutions. In our work with Endeavors, we have attempted to provide an underlying infrastructure to support a breadth of alternative approaches, providing workflow designers and participants the mechanisms to tailor solutions to individual cases. Our work with (Fielding, 1998) has suggested these are important elements in future workflow systems.

We will be exploring a number of avenues of fertile research to further implement these mechanisms and explore additional capabilities to assist adaptation in workflow. Endeavors event-driven architecture may be further extended to provide richer integration with external tools through bi-directional event interactions. This will allow more reactive work activities and greater support for external monitoring of executing processes.

Rich integration with external tools also provides a number of possibilities for adding valuable functionalities to Endeavors. Integrating with tools for computer-mediated-communication (CMC) will allow the workflow system to coordinate informal communication between participants, improving their ability to handle problems that may arise or avoid problems that could derail process execution. Modeling communication artifacts and their relationships with other workflow elements may provide rich history that can inform evolution and capture rationale associated with exception handling and evolution. Integration with mechanisms for locating expertise and providing access to organizational history can inform workflow activities and reduce time required to find solutions when exceptions occur.

The development of libraries of reusable workflow components, process fragments, and standard approaches will expand the capability of workflow developers to quickly construct processes and tailor them as needed. Collections of components and approaches, tailored to a specific domain, may provide a rich way of codifying commonly used artifacts, standards of practice, and traditional problems solving approaches for a particular community.

Acknowledgments

The authors would like to recognize the hard work and effort in the design and implementation of Endeavors by Patrick Young, Clay Cover, Arthur S. Hitomi, Ed Kraemer, and Peyman Oreizy. In addition we would like to acknowledge the members of the C2, Chimera, and WebSoft projects at UCI for their exchange of ideas during the development of this system.

References

Abbott, K. and Sarin, S. (1994): "Experiences with Workflow Management: Issues for the Next Generation" Proceedings of the Conference on CSCW, Chapel Hill, NC, pp. 113-120, 1994.

Ackerman, M. and McDonald, D. (1996): "Answer Garden 2: Merging Organizational Memory with Collaborative Help", Proceedings of 1996 Conference on Computer Supported Cooperative Work (CSCW96), Cambridge, MA. November, 1996.

Auramäki, E. and Leppänen, M. (1988): Exceptions and office information systems. In Barbara Pernici and Alex Verrijn-Stuart, editors, *Office Information Systems: The Design Process*, pages 167-182, Linz, Austria, August 1988. IFIP, Elsevier Science Publishing Co. Inc.

Barthelmess, P. and Wainer, J. (1995): "Workflow systems: A few definitions and a few suggestions." In *Proceedings of Conference on Organizational Computing Systems*, pp. 138-147, Milipitas, CA, August 1995. ACM SIGOIS.

Ben-Shaul, I. and Kaiser, G. (1994): "A Paradigm for Decentralized Process Modeling and its Realization in the Oz Environment", Proceedings of the 16th International Conference on Software Engineering, pp. 179-188, 1994

Bolcer, G. and Kaiser, G. (1999): "Simple Workflow Access Protocol: An Introduction", Technical Report, Information and Computer Science, University of California, Irvine, January, 1999.

Bolcer, G. (1998): *Flexible and Customizable Workflow Execution on the WWW*, PhD Thesis, University of California, Irvine. September, 1998.

Bolcer, G. and Taylor, R. (1996): "Endeavors: A Process System Integration Infrastructure", 4th International Conference on Software Process, Brighton, UK, December, 1996.

Cugola, G. (1998): "Tolerating Deviations in Process Support Systems via Flexible Enactment of Process Models" IEEE Transactions on Software Engineering, vo. 24, no. 11, November, 1998.

Cugola, G., Di Nitto, E., Fuggetta, A., and Ghezzi, C. (1996): "A Framework for Formalizing Inconsistencies and Deviations in Human-Centered Systems" ACM Transactions on Software Engineering and Methodology (TOSEM), 5(3), ;pp.191-230, 1996.

Dourish, P. (1995): "Developing a Reflective Model of Collaborative Systems", ACM Transactions on Computer-Human Interactions, vo.2, no.1, pp.40-63, March, 1995.

Dourish, P. et al. (1996): "Freeflow: Mediating Between Representation and Action in Workflow Systems", Proceedings of 1996 Conference on Computer Supported Cooperative Work (CSCW96), Cambridge, MA. November, 1996.

Ellis, C. and Nutt, G. (1996): "Workflow: The Process Spectrum", NSF Workshop on Workflow and Process Automation in Information Systems: State-of-the-Art and Future Directions", Athens, Georgia, pp. 140-145, May, 1996.

Ellis, C. and Rozenberg, G. (1995): "Dynamic Change Within Workflow Systems", n *Proceedings of Conference on Organizational Computing Systems*, pp. 10-21, Milipitas, CA, August 1995. ACM SIGOIS.

Fielding, R. et al. (1998): "Web-Based Development of Complex Information Products", Communications of the ACM, vol. 41, no. 8, August, 1998.

Gosling, J. et al. (1996): *The Java Language Specification*. Addison-Wesley. August 1996. http://java.sun.com/docs/books/jls/html/

Kaiser, G. et al. (1998): "WWW-based Collaboration Environments with Distributed Tool Services", World Wide Web Journal, Baltzer Science Publishers, January, 1998.

Kammer, P. et al. (1998): "Supporting Distributed Workflow Using HTTP", 5th International Conference on Software Process, Chicago, June, 1998.

Kling, R. and Iacono S. (1984): "The Control of Information Systems Developments After Implementation", Communications of the ACM, vol. 27, no. 12, December, 1984.

Malone, T. et al. (1997): "Tools for Inventing Organizations: Toward a Handbook of Organizational Processes", Working Paper, Center for Coordination Science, MIT, 1997. http://ccs.mit.edu/CCSWP198/

McDonald, D. and Ackerman, M. (1998): "Just Talk to Me: A Field Study of Expertise Location", Proceedings of 1998 Conference on Computer Supported Cooperative Work (CSCW98), Seattle, WA. November, 1998.

Miller, J. et al. (1997): "The Future of Web-based Workflows" LDIS Department of Computer Science, University of Georgia, Athens, Research Directions in Process Technology Workshop, Nancy, France, July, 1997.

Nutt, G. (1996); "The Evolutions Toward Flexible Workflow Systems" Distributed Systems Engineering, vol. 3, no. 4, pp. 276-294, December, 1996.

Osterweil, L. (1998) "Software Processes are Software Too, Revisited", In Proceedings of the International Conference on Software Engineering, Boston, MA., pp. 540-548, May, 1998.

Rosenblum, D. and Wolf, A. (1997): "A Design Framework for Internet-Scale Event Observation and Notification", Proc. Sixth European Software Engineering Conf./ACM SIGSOFT Fifth Symposium on the Foundations of Software Engineering, Zurich, Switzerland, Sep. 1997, pp. 344-360.

Saastamoinen, H. et al. (1994): "Survey on Exceptions in Office Information Systems" Technical Report CU-CS-712-95, Department of Computer Science, University of Colorado, Boulder, 1994.

Saastamoinen, H. (1995): *On the Handling of Exceptions in Information Systems*. PhD thesis, University of Jyväskylä, 1995.

Strong, D. and Miller, S. (1995): "Exceptions and exception handling in computerized information processes." *ACM Transactions on Information Systems*, 13(2):206-233, April 1995.

Suchman, L. (1983): "Office Procedure as Practical Action: Models of Work and System Design." ACM Transactions on Information Systems, 1(4), October 1983.

Taylor, R. (1997): "Dynamic, Invisible, and on the Web", Research Direction in Process Technology Workshop, Nancy, France, July, 1997.

Taylor, R. et al. (1996): "A Component- and Message-Based Architectural Style for GUI Software", IEEE Transactions on Software Engineering, June 1996.

Tolone, W. (1996): "Introspect: a Meta-Level Specification Framework for Dynamic Evolvable Collaboration Support", PhD Thesis. University of Illinois at Urbana-Champaign, 1996.

Young, P. (1991): *Customizable Process Specification for Technical and Non-technical Users*. Ph.D. thesis, University of California, Irvine. August, 1991.