Freeflow: Mediating Between Representation and Action in Workflow Systems

Paul Dourish, Jim Holmes, Allan MacLean, Pernille Marqvardsen[†] and Alex Zbyslaw

Rank Xerox Research Centre, Cambridge Lab (EuroPARC), UK

[†]Also Dept of Information Science, Aarhus University, Denmark

[surname]@cambridge.rxrc.xerox.com

Abstract

In order to understand some problems associated with workflow, we set out an analysis of workflow systems, identifying a number of basic issues in the underlying technology. This points to the conflation of temporal and dependency information as the source of a number of these problems.

We describe Freeflow, a prototype which addresses these problems using a variety of technical innovations, including a rich constraint-based process modelling formalism, and the use of declarative dependency relationships. Its focus is on *mediation* between process and action, rather than the *enactment* of a process. We outline the system and its design principles, and illustrate the features of our approach with examples from ongoing work.

Keywords: workflow, process support, process description, constraints, dependencies, temporal organisation.

1 Introduction

Workflow systems, in one form or another, have been with us for over ten years. Workflow systems embody representations of working processes, as a basis for supporting those processes, potentially distributed in time and across multiple people. Workflow systems offer to "relieve users of the burden of coordination", by managing task coordination within the system, so that users can focus on the constituent work activities.

The development and introduction of workflow systems has not been unproblematic, to say the least. Early experiences led to considerable problems with user communities, and terms such as "naziware" are testament to some of these troubles. Studies of work have increasingly observed that the "coordination" of work *is, itself, work*; but this coordination is taken as unproblematic in workflow process representations, so that control over coordination can be passed from users to systems. Similarly, studies of people following both paper-based and technologically-represented processes have highlighted the flexibility with which they interpret the processes in order to get their work done [Orr, 1991; Bowers et al., 1995]. Observations such as these have led to the identification of various issues to be addressed in future workflow systems [Abbott and Sarin, 1994].

Recently, and most obviously, a debate between Lucy Suchman and Terry Winograd in the pages of the CSCW Journal [Suchman, 1994; Winograd, 1994] and subsequently a host of responses and commentaries from other researchers in the area [CSCW, 1995] have again brought some of these issues to the fore, concentrating not only on the appropriateness of particular formulations of multi-person behaviour, but also on wider political issues in categorisation and the formalisation of work. None the less, as these debates continue in the research community, workflow technologies are increasingly becoming part of everyday life for many people, going hand-in-hand with the popularity of Business Process Reengineering.

What should be concluded from these debates? Should we, perhaps, decide that the undeniable popularity of workflow technologies as business tools adequately demonstrates their validity, and that their opponents are arguing abstract, theoretical points, too distant from practice to be of relevance? Or should we work to halt the seemingly inexorable progress of technologies which seem to deny and defeat the very working practices by which organisational life and work proceeds?

In this paper, we will present the preliminary results of work conducted at RXRC over the past eighteen months. Our starting point is not to reject the potential value of process representations in collaborative work. Instead, drawing on a variety of approaches including ongoing ethnographic investigations of the use of process technologies, we concentrate primarily on the *mediation* between working activity and process representations.

In the next section, we will set out a conceptual framework for the various sorts of transformations which we might introduce into traditional process management technologies—the dimensions of workflow. Of these, we will highlight one particular area on which we have been concentrating, and describe how it is approached. We will describe two particular technical developments which have supported this work: first, an enriched process model which allows us to separate information about the *dependencies* between task activities from information about the *temporal sequence* of their performance; and second, a declarative model of typed inter-activity dependencies which addresses a number of problems in handling complex activity sequences.

Based on these, we will move on to set out the architecture we have developed in our prototypes. Finally, we will discuss user interface features and a third technical design principle (delegation); and then illustrate the principles we have developed with examples from some preliminary experiments.

2 Dimensions of Workflow

Our starting point, then, is a conceptual framework surrounding workflow technologies—how they are derived and how they are used. The basic principles are illustrated in figure 1.

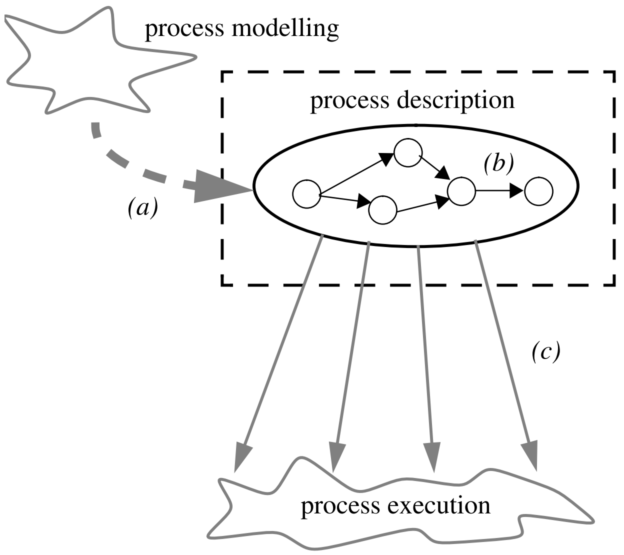


Figure 1: Teasing out the dimensions of workflow.

As outlined earlier, process description is at the heart of workflow technology. In the traditional view, a process modeler[1] will build a description of a working process, which is then embedded in a process management system. The process model describes (in general) the sequences of actions corresponding to the correct execution of the process. Subsequently the system will drive the sequences of user actions required to execute the process, according to the sequence of tasks described by the process model.

There are three particulars relationships between stages or components of this model highlighted in figure 1 by the italic letters *(a)*, *(b)* and *(c)*. As indicated by the arrows, the traditional model casts each of these relationships as being one-way or "directional". However, each can be thought of as running in either direction, with systems varying between each extreme. So part of the design space for workflow systems is set up by these three dimensions.

2.1 The Definition Dimension

The first of these is the "definition" dimension, focussing on the relationship marked by *(a)* in figure 1. The traditional approach puts the definition of the model before its use, that is, before the work. The creation of the model occurs prior to the use of the model in the system.

A critical feature of this is that it places the description of work *outside* its enactment. Although the eventual users might be involved in the definition process, the roles of process modeler and process participant are often separate; but certainly, "theorising" about the model and "doing the work" are different, separate steps. We can't do the modelling by doing the work.

However, this approach to model definition can be reversed. We can turn this into a dimension of the design space, rather than always focussing on one particular point. Moving along this dimension would begin to allow users to engage in the task of creating and modifying their own process descriptions, in or alongside the execution of their work. Coming from users in the enactment of working activities, such descriptions might serve various purposes; not only the traditional purposes of process descriptions in workflow systems, but also for individual and group reflection on process, for documentation or organisational learning purposes, and so forth.

2.2 The Sequence Dimension

The second dimension, at the point labelled *(b)* in figure 1, concerns the way in which the sequential structure of the process representation is exploited.

Traditionally, workflow systems are used to guide and structure activity, and in particular activity which might involve coordination or communication as part of the process. It ensures a flow of activity across an organisation, from the starting state to the goal states, to achieve some specific end, produce or transform an artifact, and so forth. So the relationship between the component tasks of a process is not simply one of dependency, but one of sequential execution; and the system steps through the model from beginning to end, ensuring that each point on the path is completed.

What would it mean for this not to be true? What would it mean to work this in the other direction? The result would be similar to the notions of forward-chaining and backward-chaining in AI and planning systems. Rather than always starting at the current state and determining what steps are allowable at this point ("forward-chaining"), it would start at the goal state and work backwards, to determine which sequences of operations, or paths of activation, would successfully lead to the completion of the task. This approach might be used for fault-finding, or for investigating alternative strategies for accomplishing activity.

2.3 The Activity Dimension

The activity dimension, labelled *(c)* in figure 1, covers the relationship between the process description embodied in the workflow system, and the user activities involved in executing this process. The traditional approach is to drive user action from the process description. Typically, user actions are informed or constrained by the valid "moves" from the current state defined by the process model. In this way, the system ensures that the sequential structure of user activity

1. Here, "process modeler" denotes a role that someone plays, rather than a professional category.

accords to the process definition's model of legal transitions and correct execution of the process.

The inverse of this relationship would be to move in the other direction, from action to process description. This opens up a new potential role for process descriptions. Rather than driving user action, they can be used to present, organise or explain the activities in which users might engage. So sequences of activity in the world could be rationalised or presented in terms of the process description; and so the process system is concerned with relating user action to a process description, rather than driving action from it.

2.4 Exploring the Dimensions

Since we can use these dimensions to characterise the approaches of various workflow systems, we can also see earlier research on novel workflow developments as explorations of this space. Fujitsu's Regatta system [Swenson et al., 1994] provides support for the collaborative development of process models in the course of doing the work; it opens up what we have called the "definition" dimension. The goal-based approach to workflow explored by Ellis and Wainer [1994] introduces higher-level goals as well as lower-level activities into process descriptions, and can use these goals to represent otherwise unstructured activity; this addresses aspects of the "activity" dimension.

The focus of our attention has primarily been on the second and third dimensions. We are especially concerned with the relationship between user action and the process description ('activity"), which then leads us to exploring how process models embody commitments to particular sequences of action ("sequence").

As well as looking at how process-based work is conducted in a range of settings, we have also been looking at new technological approaches to workflow which can be used to support this more flexible approach. The next few sections will outline the approach we have adopted, and opportunities it presents.

3 Constraint-Based Workflow

Our concern is with how process representations can be exploited, for the purposes of structuring, describing, coordinating and explaining activity, without introducing the stringent restrictions on that activity which have traditionally been associated with workflow technologies. In mid-1994, we developed an early prototype, called Contraflow, to explore the use of *constraints* to address this problem.

3.1 Constraints in Programming Languages

Constraints are a mechanism available in some programming languages, which set up persistent relationships between objects over time. Imagine that a traditional programming language executes the statements:

```
a := 10;
b := a* 2;
a := 6;
```

Their effect is, first, to set assign the value 10 to the variable `a`; second, to assign twice the value of `a` (i.e., 20) to the variable `b`; and then to reset the value of variable `a` to 6. The result is that `a` holds the value 6 and `b` holds the value 20. Now imagine a similar (but not identical) set of operations in a programming language with constraints:

```
a := 10;
b <- a * 2;
a := 6;
```

The second statement is not at assignment; instead, it sets up a constraint which states that the value of variable `b` is constrained to be twice the value of variable `a` at any subsequent point in the program (or until the constraint is changed). At this point, the variable b will hold the value 20, as before. However, when the third statement is executed and the value of variable `a` changes, *so will the value of variable* `b`. The constraint still holds; `b` should always be twice `a`; and so when the value of variable `a` becomes 6, the value of variable `b` changes to 12. Furthermore, variable `b` will continue to track changes to the value of variable `a` through the execution of the program.

3.2 Constraints in Contraflow

In Contraflow, the relationships between the individual tasks which make up a process description are modelled using constraints which operate in much the same way. That is, instead of describing the *instantaneous transitions* between task states, they describe the *continuous relationships* between those states. For example, "format the document once the text has been approved" states an instantaneous transition, moving activity from one task to another in the course of process execution. In contrast, "document formatting cannot be performed unless the text has been approved" describes an ongoing, continuous relationship.

This reconstruction of the process model has a number of consequences. First, it separates two relationships between tasks which are typically conflated into a single relationship in a traditional workflow system—a *temporal* relationship concerning the ordering of activities, and a *dependency* relationship between work activities. Constraint-based process models in Contraflow declaratively describe the dependency relationships between tasks, rather than procedurally describing sequences of action. Second, and consequently, it frees users to engage in the component tasks of a process as appropriate to their circumstances at any given moment, rather than sequentially as defined by particular, absolute paths through the process model. Third, constraint maintenance is an active, ongoing process which continuously relates the overall state of the process instance to the user activities which are going on, rather than having to restrict the scope of potential action in order to achieve a coordination between activity and the model.

As a result, then, users of Contraflow can engage in tasks opportunistically, relying on the constraint mechanism to coordinate activity across tasks and maintain the overall process state. Contraflow actively maintains the correspondence between ongoing user activity and the dependencies in the process description, allowing the sequential organisation of action to be created by the users in the course of their work.

This theme—the active mediation of the relationship between process description and process execution—has been a key principle in our subsequent work. Contraflow was designed as a proof-of-concept, and as an exploration of the use of constraints and related mechanisms in process management systems. More recently, these ideas have been expanded in our Freeflow prototype. Before describing the architecture we have developed, we will discuss two particular aspects of Freeflow's design which follow on from the Contraflow work. The first is Freeflow's *extended process model*, building on the separation of task dependency and execution sequence; and the second is the use of *generic activity relationships*, rather than raw constraints, to simplify process modelling.

4 Freeflow's Expanded Task Model

The Contraflow prototype introduced the idea of separating the temporal and dependency relationships between the tasks which make up a process description. It did this through the use of constraints, which were used to relate task *states*—that is, the various states through which any task progressed. Contraflow embodied an extended task state model (not discussed here) to support this constraint mechanism. In our subsequent work, we have developed and refined this extended model into that embodied in Freeflow.

In a typical workflow system, activation flows through the process description, usually from beginning to end, with different tasks active at any given time. Each individual task, then, proceeds through three different states: *inactive*, before it has become the active component of the process; *active*, when it is being worked on; and *done*, when the work associated with that task is complete and activation has moved onto the next stage of the process.

Freeflow employs a richer six-state model (see figure 2). This model is based on a separation between the representation of user and system states. The separation between user and system states is a reflection of the separation between temporal and dependency relationships. The three positions on the left-hand side correspond to the three "user states". An activity is "inactive" before any work has been done on it; when a user begins work on the activity, it becomes "active", and once the work is complete, it becomes "ready". So these user states reflect potential user action.

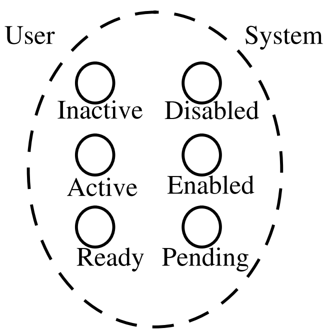


Figure 2: Freeflow's extended six-state task model.

The three right-hand states reflect the relationship of this activity to other activities within the process—the pattern of enablements and constraints between the individual activities, or causal dependencies. So, an activity is "disabled" when an activity on which it depends has not yet been completed. It moves to the "enabled" state when these preconditions are met; that is, there are now no dependencies to prevent this activity from beginning. The final state, "pending", is for tasks which are prevented from *completing* by some external dependencies (in contrast with the disabled state, when a dependency prevents a task from beginning).

The various activities which make up a process are related to each other by constraints between the states of those activities. So, for instance, the statement that "activity B cannot start until activity A has been started" is represented at this level with two constraints, which tie the "enabled" state of activity B to the "enabled" and "active" states of activity A. By placing these constraints between the activities, the modeler states that activity B cannot enter the "enabled" state until activity A is both "enabled" (ready to run) and "active" (currently receiving user attention).

The right-hand (system) states reflect activity dependency information. They capture the information which is typically represented in traditional process modelling systems; that is, the patterns of enablement between activities. However, they do not imply temporal ordering. Since we no longer rely on the same three states to record both dependency information and activation information, we can allow users to engage in activities opportunistically, and to organise their own work around the particular circumstances in which they find themselves.

The work of Freeflow, then, is not only to provide a richer process language and to model tasks in those terms, but also to maintain the correspondence between the two sets of states, system and user.

5 Generic Activity Relationships

Any number of the system states of one activity may be linked by constraints to any number of states in any number of other activities. Clearly, then, there is a huge number of possible permutations of constraints. However, we rapidly found a number of regularly-occurring patterns of constraints. They represent particular, idiomatic patterns of relationship between two activities which occur commonly in process descriptions.

Freeflow makes these commonly-occurring patterns available directly as activity dependences. A number occur in our prototype. For instance, the "after" dependency states that one activity should be performed after another has been completed, while the "concurrent" dependency states that two activities are performed concurrently (that is, one can start as soon as the other has started).

These dependencies are not new forms of relationship which we introduce into the model. Instead, they are a higher-level shorthand for particular sets of activity-state constraints. At this higher level, the dependencies are of particular "types" (e.g. after, concurrent), unlike the simply, untyped constraints which underlie them. These generic typed dependencies are more convenient than the collections of constraints for describing common activity relationships in the process definition language.

Since the dependencies are implemented in terms of activity-state constraints, they share with those constraints a declarative, rather than procedural, status. They say, "such-and-such a relationship exists between these two activities", rather than saying, "when this happens, do that". One advantage of this declarative style is that it greatly simplifies more complex patterns of activity sequencing, such as looping, branching, and iteration. The "validation" dependency illustrates this particularly clearly.

5.1 Control Flow: the Validation Dependency

Validation is a common process idiom. It refers to the case where the work performed in one activity must be validated or approved before the process progresses. Common cases might be a form which has to be approved by a manager, or a report which must checked, corrected and authorised. Validation introduces a loop into the process description, since approval might be denied, in which case the activity which had to be validated (filling in the form, or writing the report) must be restarted; and once it has completed the second time, the validation activity will begin again.

A number of existing process description languages model simple information flows and have no support for loops in process descriptions at all. Others may support looping, but must encode it procedurally (perform the base activity, then perform validation, then either proceed or perform the base activity again). In these cases, depending on the richness of the language, it may be hard to encode the conditions under which the loop will terminate, and depending on the process, it might even be inappropriate to formulate such conditions. At the same time, it can also be difficult to specify properties which are variable but hold constant across the validation loop (for example, the case where one of a number of people can approve the form, but if it's not approved, the same person should be called upon to re-examine it after further work).

In Freeflow's modelling approach, validation is not expressed as a procedural description of activity progression, but rather as a *relationship* between two activities. Freeflow uses a validation relationship to express that activity A validates activity B. With this approach, the process modeler does not have to detail precisely how the validation activities will proceed. The system is then free to interpret that as appropriate in different circumstances, and manage the activation of these two activities in context. Again, remember that high-level activity dependency relationships such as validation are simply shorthands for an underlying pattern of activity-state constraints, which are themselves both declarative and open to interpretation in differing circumstances. So using this approach to these dependencies overcomes a number of these problems with sequence management in process description languages.

6 The Freeflow Architecture

The previous sections have introduced the main ideas which motivate and underpin the design of our Freeflow prototype.

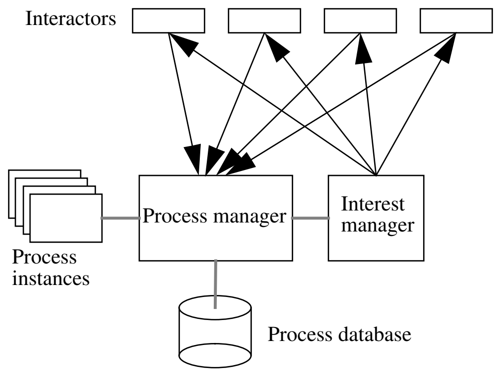


Figure 3: The structure of Freeflow's conceptual architecture.

In this section, we will introduce the conceptual architecture against which this prototype has been designed, to illustrate how it takes advantage of the open approach to activity management which Freeflow embodies.

We will discuss three principal components of the architecture here: the process manager, interactors and events.

6.1 Process Manager

The Freeflow process model itself is embodied in a *process engine*, which provides the semantics for the constraint language. It is responsible for progressing activities from one state to another, and managing the dependencies between states. The process engine is embodied in a *process manager* which uses the engine to control a series of *process instances.* Each process instance represents a single running instance of a business process; the representations of these processes are stored in a process database. So, there may be many process instances with the same process description running at any one time, corresponding to multiple concurrent instances of the same process. For example, in a financial services system, there might be a process description for the mortgage application process. The description of this process is stored once in the process database; but there will be a separate process instance for each mortgage application which is currently in progress.

6.2 Interactors

The process manager acts like a database system, managing the records of current process instances and performing process-specific manipulations. It is not responsible for user interaction or the presentation of process descriptions or process instances to users. These are managed by *interactors.*

An interactor is any component of the system which deals with interaction and process representation issues. There are two sorts of interactors—*specialised* and *appropriated.* A specialised interactor is one which has been designed specifically to operate as part of the Freeflow environment (in much the same way as other workflow systems provide user interfaces for task management, process descriptions, activity lists and so on). Using an event mechanism (described below), we also have the opportunity to create appropriated interactors, which are existing applications in our users' environments which can be "Freeflow-enabled". For example, we can place a wrapper around a word processor or text editor so that, when a file is edited, this activity will be registered with Freeflow so that it can relate it to current process activations which are associated with that file.

6.3 Events

The glue which links these components together is a generalised event mechanism. Rather than directly define the patterns of communication between system components—the process engine, interactors and so forth—we use an event mechanism for coordination. This means that we can add new interactors and new relationships between components much more easily.

Activities within the system generate events. For instance, creating a new instance or moving an activity from one state will generate an event. System components can register their interest in particular events with the interest manager. This interest manager is responsible for distributing notifications of events to those system components which have expressed an interest. So patterns of communication are specified indirectly, in terms of the processes themselves. So, for instance, an interactor which is being used to control a particular process instance might register interest in any event associated with that instance; meanwhile, another interactor which provides an overview of particular activities throughout the system might register an interest in all events related to processes of a particular type. The event mechanism helps "decouple" interactors from the process manager, and so interactors can be started and stopped at any time. This decoupling also leads to another important aspect of Freeflow's approach—delegation—which will be illustrated in our example.

7 An Extended Example

We have been developing and experimenting with our prototype implementation of Freeflow since early 1995. To demonstrate the principles which we have introduced in the previous sections, we will now look at an application which we have been using in-house as an early example.

To aid cross-platform development, we have been using the World Wide Web to provide interfaces. We have developed specialised interactors which are accessed through CGI scripts (essentially, active Web pages), organised into sets of activities which correspond to the various roles in our example process.

The process itself is the management of public technical reports. This is a well-established process in our laboratory, which involves five or six different people in different roles, and provides us with an excellent opportunity to compare the process as supported by Freeflow with the process as it was previously managed. Before the introduction of the Freeflow prototype, the technical report process was entirely paper-based (that is, there was no automated process support). We have been working with the various parties in this process to develop a support system using our Freeflow prototype.

7.1 The Technical Report Process

The technical report process, loosely, runs as follows. Individual authors submit papers to be admitted to the technical report series, which is managed by two editors. Either or both editors might be involved in the processing of any particular technical report. The editors have 4 primary responsibilities:

1. ensuring that the report has been cleared for publication. Authors are required to submit their clearance form, which the editors will check.
2. ensuring that publication as a technical report would not violate any copyright regulations. Authors are required to submit any copyright assignment which they have made for published papers.
3. ensuring that the report is of adequate quality for publication (both in form and content). The editors will copy-edit the paper to check the form. In the case of papers published externally, there is no further check on the content; for unpublished papers, two peer reviewers are assigned.
4. ensuring that the electronic copy of the report is printable. The editors check the document prints satisfactorily on various printers.

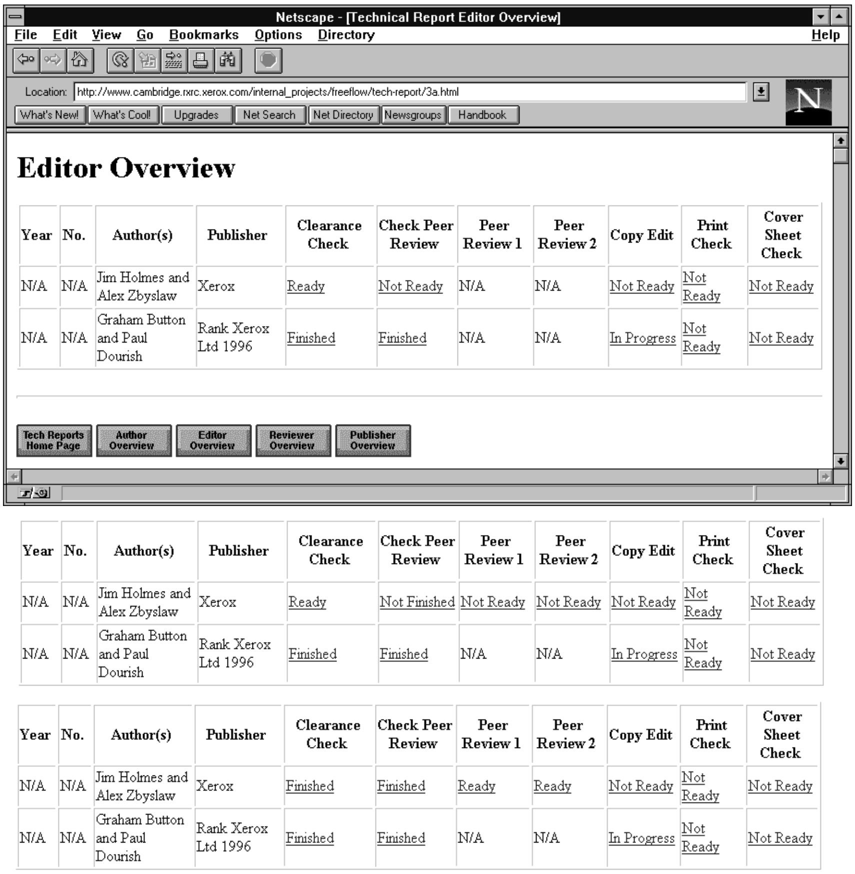
When these checks have been made, the report can be assigned a number, entered into the official lists, and made available electronically inside and outside the laboratory (via the World Wide Web).

7.2 Performing the Process

Let's look at an example of part of this process being performed using our prototype. First, the author of a paper or report submits the document for inclusion in the technical report series by filling out a form on a WWW page. The form describes the document, its authors and status, etc. When this is submitted, Freeflow creates a new process instance for the technical report process.

The technical report series editors, who are responsible for the administration of this process, have an "overview" page shown in figure 4 (a). This shows two documents currently being processed, including the example document we will follow here, a paper by Jim Holmes and Alex Zbyslaw. The labels associated with this document in the editor overview table show the current state of the various activities in this process instance. They are also WWW links to the pages which control the execution of those activities.

The formal process states that a corporate clearance must be obtained before the process goes any further. However, in our example scenario, imagine that the managers authorised to clear the paper for publication are unavailable. The activities which follow the clearance task, such as copy-editing and peer review, are time-consuming, and can sensibly proceed without the clearance for the moment, even though this is a deviation from the official process. In this case, the editors can begin the peer review process anyway. When they select the select the activity label for "Check Peer Review", the dialogue shown in figure 5 appears. This dialogue warns the editor that the action which they have just taken breaks a



(a)

Year	No.	Author(s)	Publisher	Clearance Check	Check Peer Review	Peer Review 1	Peer Review 2	Copy Edit	Print Check	Cover Sheet Check
N/A	N/A	Jim Holmes and Alex Zbyslaw	Xerox	Ready	Not Ready	N/A	N/A	Not Ready	Not Ready	Not Ready
N/A	N/A	Graham Button and Paul Dourish	Rank Xerox Ltd 1996	Finished	Finished	N/A	N/A	In Progress	Not Ready	Not Ready

(b)

Year	No.	Author(s)	Publisher	Clearance Check	Check Peer Review	Peer Review 1	Peer Review 2	Copy Edit	Print Check	Cover Sheet Check
N/A	N/A	Jim Holmes and Alex Zbyslaw	Xerox	Ready	Not Finished	Not Ready	Not Ready	Not Ready	Not Ready	Not Ready
N/A	N/A	Graham Button and Paul Dourish	Rank Xerox Ltd 1996	Finished	Finished	N/A	N/A	In Progress	Not Ready	Not Ready

(c)

Year	No.	Author(s)	Publisher	Clearance Check	Check Peer Review	Peer Review 1	Peer Review 2	Copy Edit	Print Check	Cover Sheet Check
N/A	N/A	Jim Holmes and Alex Zbyslaw	Xerox	Finished	Finished	Ready	Ready	Not Ready	Not Ready	Not Ready
N/A	N/A	Graham Button and Paul Dourish	Rank Xerox Ltd 1996	Finished	Finished	N/A	N/A	In Progress	Not Ready	Not Ready

Figure 4: The “editor overview” provides editors with a view of current process instances.

constraint; in particular, the constraint which specifies that clearance should be obtained before reviewing begins. In this case, believing that the clearance will be granted, the editor chooses to note this and proceed.

The peer review process involves assigning two reviewers from inside the lab who will review the paper informally. Reviewing is a validation activity, and so the review process may involve a loop of review and correction. However, at some point the reviewers are satisfied and approve the paper for publication.

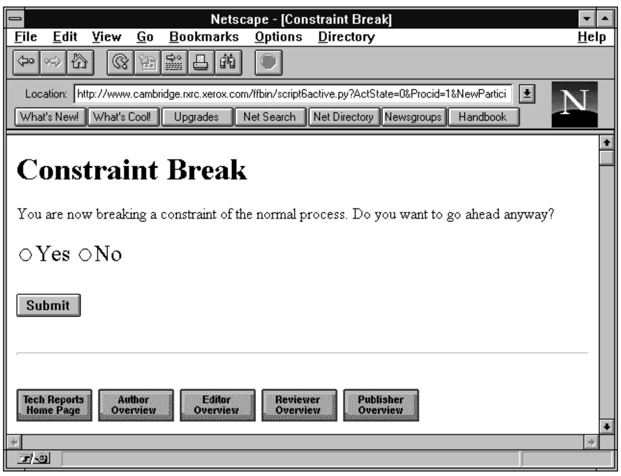


Figure 5: Dialogues inform users when constraints are broken.

At this point, the “check peer review” activity should move to “Finished”, indicating that the activities have been performed. However, the broken constraint prevents this. Instead, this activity is marked as “Not Finished”, which indicates that the work has been done, but a previously broken constraint is still outstanding (figure 4 (b)). In this case, it is the constraint relating this to the clearance activity; the report has still to be cleared for publication. In our example, now, an appropriate manager is available, and the paper is cleared for publication. When this activity is performed, the “check peer review” activity can be marked as finally finished (figure 4(c)).

7.3 User Interface Delegation

This short example illustrated one other design principle which has been implied by what has come before, but deserves explicit mention—*user interface delegation*.

There are a number of points where decisions about action can be made. Particularly when we introduce the separation of user and system states, and the opportunity to break constraints, then the number of such points is greatly increased. In traditional systems, decisions about activity sequencing—what task to perform next—are made within the workflow system itself. By contrast, in Freeflow, these decisions are *delegated* to the user, and appear at the user interface. The dialogue in the example above, where the editor decides to proceed with the peer reviewing despite the lack of a clearance, is an example of this sort. At such a point, the user can bring to bear contextual information (such as, for instance, that this report deals with material previously cleared for publication in other forms, and therefore is likely to be approved for publication) to the resolution of the decision about progress. Clearly, of course, this could happen in other systems simply by stepping outside the bounds of the system; this is exactly the sort of strategy which Bowers et al [1995] report in the context of a workflow system for print shop operation. In these cases, where the system rigorously enforces some internal notion of task sequence, users simply step outside the system to perform the actions which they know are appropriate in particular circumstances. However, the traditional approach renders those activities invisible to the system; and so, if the system is also used for monitoring and accounting purposes, then there is time and effort unaccounted for. The need to step outside the system forces a breakdown in the process model; once an activity has moved outside the process system, there is little incentive to bring it back in again.

Freeflow's flexible approach allows users to control the way in which sequencing operations are carried out. This allows these "hidden" activities not only to proceed, but also to be recognised within the system. Freeflow flexibly maintains the correspondence between user action and process representation. This active maintenance, mediating between working activity and the process representation, gives users control over the way in which their work is organised, but retains many essential benefits of process-based approaches, including the use of the process to guide action, and the rationalisation and explanation of activity with respect to a process model.

8 Conclusions

The procedure of deciding, before the actual occasion of choice, the conditions under which one among a set of possible alternative courses of action will be elected, is one definition of a rational strategy. It is worth noting that this rational property of the decision-making process in managing everyday affairs is conspicuous by its absence.
—Harold Garfinkel [1967].

While the development of workflow systems since their development in the early 1980s has yielded considerable improvements in the richness of their models and their usability, some fundamental problems remain. Not least among these is a problem frequently cited by critics of the workflow approach—that the process models which workflow systems embody are descriptive accounts of working activity, reused as prescriptive regimentations of that activity. Once the work has been described by a process model, and this model has been embodied in a system, the computer takes over the process. The result is first, that users lose control over their work; and second, that the work loses the benefit of the insights with users bring to bear on how their work should be organised for the particular circumstances in which any instance of the work is carried out.

Our approach attempts to address this problem by effecting a separation between the description of the working process and the actualities of working activity. The emphasis of this work, then, is not on how to build richer, more descriptive languages for process definition, or how to arrange for working activity to follow these descriptions. Rather, we *actively mediate* between abstract process descriptions and the concrete activities in particular circumstances. The work is in how this correspondence can be maintained.

We have described a prototype system which demonstrates and develops some of these ideas. The roots of Freeflow are in an activity model which separates "system" information (recording dependencies between activities) from "user" information (recording work over those activities). This, in turn, allows us to separate the constraints of process structure from the sequential organisation of working activity. Freeflow further embodies a number of principles which aid this separation—the use of declarative, typed activity dependencies, and the delegation of conflict handling to the user interface.

In this paper, we have drawn on examples based around our work with a separate user community within our own organisation. Ongoing work is looking into the use of these same principles in a large financial services company in the UK. The development and use of these design principles provides us with a means to retain some of the organisational benefits of process support systems, while acknowledging and supporting the inherent openness of work practice—a new development for workflow.

Acknowledgments

We would like to thank a number of colleagues for their contributions to the development of this work, especially at the early stages. In particular, we thank Bob Anderson, John Bowers, Graham Button and Remo Pareschi for their insights.

References

[Abbott and Sarin, 1994] Ken Abbott and Sunil Sarin, *"Experiences with Workflow Management: Issues for the Next Generation"*, Proc. ACM Conference on Computer-Supported Cooperative Work CSCW'94, Chapel Hill, North Carolina, October 1994.

[Bowers et al., 1995] John Bowers, Graham Button and Wes Sharrock, *"Workflow from Within and Without"*, Proc. European Conf. Computer-Supported Cooperative Work ECSCW'95, Stockholm, Sweden, September 1995.

[CSCW, 1995] Special Issues of *Computer Supported Cooperative Work*, 3(1), 1995.

[Garfinkel, 1967] Harold Garfinkel, "Studies in Ethnomethodology", Prentice-Hall, New York, 1967.

[Ellis and Wainer, 1994] Clarence Ellis and Jacques Wainer, *"Goal-based Models of Collaboration"*, Collaborative Computing, 1(1), pp. 61–86, 1994.

[Orr, 1991] Julian Orr, *"Talking about Machines: An Ethnography of a Modern Job"*, Technical Report SSL-91-07, Xerox PARC, Palo Alto, California, 1991.

[Suchman, 1994] Lucy Suchman, *"Do Categories have Politics? The language/action perspective reconsidered"*, Computer-Supported Cooperative Work, 2(3), pp. 177–190, 1994.

[Swensen et al., 1994] Keith Swenson, Robin Maxwell, Toshikazu Matsumoto, Bahram Saghari and Kent Irwin, *"A Business Process Environment Supporting Collaborative Planning"*, Collaborative Computing, 1(1), pp. 15–34, 1994.

[Winograd, 1994] Terry Winograd, *"Categories, Disciplines and Social Coordination"*, Computer-Supported Cooperative Work, 2(3), pp. 191–198, 1994.