S. Bødker, M. Kyng, and K. Schmidt (eds.). *Proceedings of the Sixth European Conference on Computer-Supported Cooperative Work, 12-16 September 1999, Copenhagen, Denmark*
©1999 Kluwer Academic Publishers. Printed in the Netherlands

Taking the Work out of Workflow: Mechanisms for Document-Centered Collaboration

Anthony LaMarca, W. Keith Edwards, Paul Dourish, John Lamping, Ian Smith and Jim Thornton

Xerox Palo Alto Research Center, USA
lamarca@parc xerox com

Abstract: There are two aspects to technical support for collaborative activity; support for content work and support for coordination. The design of CSCW systems must typically address both of these, combining them in a collaborative application This approach, however, suffers from a variety of well-known problems, not least the compatibility between collaborative and single-user applications, working styles and practices In this paper, we describe an alternative approach that makes coordination and collaborative functionality an aspect of the collaborative artifact rather than a collaborative application. We present an infrastructure and a series of application examples to illustrate the idea of document-centered collaboration, in which coordination and collaboration are separated from and independent of applications.

Introduction

Most computer applications focus on the manipulation of content They gather information from the user, record it, control access to it, organize it, act on it, and present it on demand. Content includes user-readable forms of information, such as is handled by a word processor, presentation package or software development environment, as well as internal machine-readable forms such as database records, index structures and object code formats.

Collaborative applications add a second concern to this focus on content a concern with coordination. Schmidt and Bannon (1992) draw on the work of Anselm Strauss to argue that the central focus in CSCW is "articulation work"—the means by which people "divide, allocate, coordinate, schedule, mesh, interrelate, etc" their concerted activities.

Since CSCW systems must be able to deal with both content and coordination, a fundamental question for the development of CSCW technologies is the nature of the relationship between the two. It has been a troublesome one. Some CSCW applications have attempted to take on the entire problem single-handedly; this approach has led to the development of collaborative text editors, drawing tools, and so on. Other systems have focussed on the coordination aspects and left application development alone; this approach has led to the development of collaboration harnesses and run-time frameworks for transparent application sharing. Each of these approaches has problems, as we will discuss.

In this paper, we present a new approach to the relationship between the content and coordination facets of collaborative systems. In our approach, we focus on collaboration as a feature of the artifact rather than the application In doing so, we free ourselves from a set of restrictions on the interoperability of content that come from an application focus. Our model adopts electronic documents as the primary means of content exchange Electronic documents can be opened, read, manipulated, changed, clustered and sorted with a variety of familiar tools; but they also, critically, carry their semantics with them, rather than delegating them to an application.

Applications, Content, and Encodings

Applications manage content. They store, represent, render and manipulate content in a wide variety of formats

Often, this content is encoded in application-specific ways, so that it can only be used or interpreted by specific applications imbued with the ability to understand the meaning of that particular encoding The content types used by these applications may not be shareable with other applications, and indeed, they may not even be readily exposed to other applications at all. For example, a custom database application, such as an insurance claims processing tool, makes semantics assumptions about the validity of the information it will store and use The data stored by this application will not be understandable by other applications In fact, these other applications would likely be unable to even access the information, stored as it is in a custom database rather than a simple file.

In other words, applications contain encodings and representations of their content that are tailored to their specific needs, and the ability to interpret and manipulate these representations is encapsulated within the application itself. Although this approach is successful most of the time, it presents problem when we attempt to extend the functionality of the application into new areas

Content and Coordination

On top of their concern with content, collaborative applications must also be concerned with support for the coordination of activities between individuals A variety of mechanisms have been introduced to support coordination, such as the use of awareness within a shared workspace (Dourish and Bellotti, 1992) or the use of explicit representations of conversational action as a basis for managing patterns of interaction (Winograd and Flores, 1986). Support for coordination has been a primary research focus with CSCW. However, this has often been to the detriment of support for content in prototype collaborative tools, often, the latest research prototype shared text editor may well be shared, but isn't a very good text editor The problem is how to combine application features—the interpretation and management of specialised content—with new features for collaboration and coordination.

There have been two common approaches to the relationship between content and coordination in shared applications. The first is to combine them in a single, monolithic application, in which the needs of collaboration can be fully integrated with the manipulation and presentation of content; the second is to use a collaboration harness to share single-user applications. Although the second approach has proved useful in specific scenarios, such as remote presentations, most deployed systems use the first approach.

Take a workflow application as an example. As discussed above, such an application will have an innate understanding both of the content and the process of the workflow When a user updates one bit of content in the system—say, by digitally "signing" an expense request—the system notices this change in state and will move the process along to its next logical state. This change may involve displaying forms on other users' screens or in their work inboxes, removing documents from the queues of other participants, and so on. The workflow tool is able to do this because it is tightly integrated with the content it manages, can detect changes to that information (because changes come through the application itself), and can dynamically update the information on users' screens. The downside of this integration is that the workflow tool must take on all the responsibilities.

Document-Centered Collaboration

We have been exploring a new approach to the integration of content and coordination in collaborative systems We call our approach *document-centered collaboration* In our approach, we move coordination functionality out of the application and onto the documents themselves.

To achieve this, we exploit a novel document infrastructure which gives to *documents* the resources to maintain application integrity As we will explain, in our infrastructure, all operations on a document, including reading, writing, moving, and deleting, can be observed by active code associated with the document This active code can then take appropriate action, such as making notifications, performing the operation itself, or vetoing the operation. This ability to associate computation with content gives the ability to tightly bind application semantics in with the content that those semantics constrain. At the same time, by using active code to exploit knowledge about the external state of the world, documents can, in effect, become "situationally aware" and be responsive to changes in their use and in their users.

We believe that such a system can offer a novel approach to workflow, where the state of the workflow process is exposed as a set of documents that can be read, written, or operated on using standard, existing tools. Users need never directly see or use a workflow tool. Instead, computation attached to the documents enforces the coordination conventions of a workflow process, while still allowing the documents to be used in all the same ways as conventional electronic documents.

To enable the construction of these active documents, we have built a middleware layer that sits between ordinary applications and existing document repositories such as file systems. To general purpose applications, it can look like a file system, and to file systems, it can look like a general purpose application. The infrastructure can also maintain "extra" information about the documents that might not be provided by the underlying repository, and provides the ability to execute arbitrary code when documents are accessed. By sitting between files and the applications that use them, the infrastructure can add new capabilities to electronic documents, while allowing existing applications to continue to work with them.

We believe that this approach will have a number of tangible benefits. First, it does not force a choice between doing workflow and using a favorite set of document-based applications such as word processors and the like, workflow is brought to these applications, rather than creating standalone applications to do workflow Coordination and the interpretation of content have been separated.

Second, we can still use all of the existing tools that are available for working with documents. Indeed, the ability to bind computation into document content can ensure that these general-purpose tools operate on our documents in semantically constrained ways that do not violate the requirements that workflow may place on them Essentially, we enable the documents to be used in standard, general ways without corrupting workflow semantics; new, constrained functionality is layered on top of existing document functionality.

Third, for designers of workflow processes, we allow a rapid pace of development. A workflow process builder just has to write the pieces of computation that are attached to documents to implement the workflow process, not the surrounding tools and interfaces for mediating and visualizing the workflow process.

The work described here has been done in the context of the Placeless Documents Project at Xerox PARC (Dourish et al., 1999) Placeless Documents is an effort to build a "next generation" document management system that provides radical extensibility, as well as smooth integration with existing document repositories and tools We shall describe the Placeless substrate on which our workflow experiments were founded, as well as investigate three separate collaborative applications we have built using our document-centered model.

Comparison with Existing Approaches

The relationship between content and coordination functionality is a fundamental issue in the design of CSCW systems. One approach has been to embed collaborative functionality in new tools. Users have been offered new sorts of drawing tools (e.g. Brinck and Hill, 1993), new sorts of spreadsheets (e g. Palmer and Cormack, 1998), new sorts of presentation tools (e.g. Isaacs et al., 1994), etc., incorporating functionality for information sharing and collaboration. However, it has been regularly observed (e.g. Bullen and Bennett, 1990; Olson and Teasley, 1996) that the introduction of new tools for simple, everyday tasks has been a barrier to the introduction of collaborative tools. Even in the case of single-user systems, Johnson and Nardi (1996) attest to the inertia surrounding the adoption of new tools, even when the new applications are tailored to specific domain needs.

Another approach to this problem is the development of "collaboration-transparent" systems, and harnesses to make existing tools support collaborative functionality Examples include Matrix (Jeffay et al., 1992), DistView (Prakash and Shim, 1994) and JAM (Begole et al., 1997) The collaboration transparent approach suffers from the problem that, since the applications are unchanged, their semantics cannot be revised to incorporate the needs and effects of collaborative use. However, it has the advantage that users can carry on using the tools with which they are familiar and in which they have an often significant investment.

Although collaboration transparency allows group members to carry on using existing single-user tools, another problem arises in that they may not all be using the *same* single-user tools. People use different incompatible software systems, and even incompatible versions of the same software system; and forcing people to upgrade their system to use your new collaborative system is no better than asking them to switch to a new word processor What is more, if different group members like to use different word processors, then a collaboration-transparent approach will not be sufficient to let them work on a paper together.

Our approach is to focus not on *applications*, but on *artifacts*—in this case, documents. Our goal is to design a system in which one user can use Microsoft FrontPage on a PC, another can use Adobe PageMill on a Mac, while a third can uses Netscape Composer on a UNIX machine, and yet they can all three work seamlessly on the same web documents. We achieve collaboration by augmenting the documents themselves.

Coordination languages have been used to separate application functionality from that supporting collaboration, and so move coordination into the infrastructure Examples include Linda (Gelernter, 1985) in the domain of parallel systems and DCWPL (Cortes and Mishra, 1996) in the domain of collaborative ones. Our approach differs, though, in that our artifacts themselves take on an active role in managing coordination.

Perhaps the closest related approach is that of Olsen et al. (1998), who explore the use of collaboration and coordination through "surface representations." Like us, they want to move beyond the model in which each application encapsulates a fixed semantic interpretation of data that is kept separate from the actual document contents. Their approach is to abandon encoded semantics and, instead, to operate at the level of "human-consumable surface representations," or the simple perceptible graphical patterns in application displays. Our approach is similar in that we do not rely on semantics in the application, but different in that we move those semantics closer to the document itself rather than its representation.

Abbott and Sarin (1994) suggested that the next generation of workflow tools would be "simply another invisible capability that permeates all (or most) applications." By decoupling collaboration functionality from the application, and making it a middleware component, our approach has brought us closer to this model, as we will demonstrate.

We will begin by introducing the Placeless Documents system, a new infrastructure for document management which provides the basis for our approach. In the main body of the paper, we will introduce and illustrate the use of our approach by describing three example systems that have been built on top of Placeless Documents, and which serve to explain how the document-centered approach operates in practice. Finally, we will consider how this approach relates to current and potential future practice in the development of collaboration technologies.

The Placeless Documents Project

This section presents an overview of our system. We begin with a discussion of the basic facilities of the Placeless architecture, and look in particular at the mechanisms in Placeless that were necessary to build our document-centered view of workflow.

Overview of Placeless Documents

Traditional document management systems and filesystems present a hierarchical organization to their users. documents are contained in folders; folders can be nested within other folders. This structure, while easy to understand, is limiting. For example, should an Excel document for a project's budget be contained in the Excel folder, the budget folder, or the project's folder?

The goal of the Placeless Documents project is to build a more flexible system for organizing a document space. In the Placeless model, organization is based on *properties* that convey information about context of use: the document is a budget, it's shared with my workgroup, and so on. Properties are *metadata* that can describe and annotate the document and can facilitate its use in various settings.

Active Properties

While many systems support the association of extensible metadata with files and documents, properties in Placeless can be *active* entities that can augment and extend the behavior of the documents they are attached to. That is, rather than being simple inert tags, extensionally used to describe *already-extant* states of the document, properties can also be live code fragments that can implement the user's *desired intentions* about the state of the document.

These active properties can affect the behavior of the document in multiple ways: they can add new operations to a document as well as govern how a document interacts with other documents and the document management system in which it exists For example, in Placeless, active properties are used to implement access control, to handle reading and writing of document content from repositories (such properties are called "bit providers"), and to perform notifications of document changes to interested parties

It is these active properties, particularly the bit providers, which provide the ability to associate computation with content. Since property code can perform arbitrary actions when it is invoked, properties can return results based on the context of their use, and the state of the document management system at the time they are invoked. Active properties are the heart of the Placeless Document System; we shall see how they are used to implement document-centered workflow in our particular examples.

Distribution and Compatibility

Placeless Documents was architected to be a robust distributed system. Our design allows users to access document collections across the network and, more importantly, to *serve* document collections where they see fit.

The attributes of the Placeless system described above—active properties and robust distribution—enable the construction of novel applications and document services. To be truly useful, however, the system must also work with *existing* document- and file-based applications This is crucial to our desire to support workflow using arbitrary off-the-shelf applications. To this end, we architected a number of "shims" which map from existing document- and file-management interfaces and protocols into the concepts provided by Placeless. Examples of such shims might include file system interfaces, HTTP, FTP, IMAP and POP3 protocol interfaces, WebDAV, and so on. Existing applications "expect" to access files and electronic documents through these comment interfaces, and so Placeless provides these interfaces to its documents

For example, we have built a Network File System (NFS) server layer atop Placeless. This layer lets existing applications—including such tools as Word and Powerpoint—which are only written to understand files, work with live Placeless documents. Existing applications do not have to change to work with Placeless, although there is a loss of power since many of the Placeless concepts do not find an easy expression in a more traditional file system model.

For the purposes of this paper, the Placeless infrastructure can be thought of as a middleware layer—essentially a multiplexor—capable of reusing or generating content from varied sources, creating a uniform notion of "documents" from that content, and then exposing the resulting documents via a multiplicity of interfaces By exposing arbitrary entities as documents through the bit provider mechanism, and then making these documents widely available through arbitrary interfaces, we gain great leverage from existing tools and applications.

We have presented an architectural foundation that can allow computation to be tightly bound in with document content. The Placeless system can not only integrate existing sources of information from filesystems and web repositories, but can expose all of its documents through interfaces that make those documents accessible through existing applications. Now, we will present three experiments in workflow that we have built around the Placeless system. The systems described here provide a document-centered approach to workflow that allow their users to break away from closed, specialized applications, and bring the power of general-purpose computing tools to workflow processes.

A Simple Example: Travel Approval

The first and simplest of our applications is called *Maui* and manages a corporate travel approval process The actual process description is simple: employees submit trip itineraries for approval, which requires consent from both the employee's manager and department head. It was our goal to allow employees to easily submit trip requests, check on trip status as well give managers an easy way to check for and access travel requests requiring their attention. The rest of this section describes how we implemented this in a document-centered style by building two new active properties on top of our flexible property-based document infrastructure.

User's View

In Maui, users can construct their itineraries any way they wish and are free to choose the application and document format of their choice As an example a user might write a note in FrameMaker, or submit an Excel spreadsheet with detailed cost information, or simply scan in a paper copy of a conference invitation. This differs significantly from traditional workflow systems where relevant data must be manipulated with proprietary integrated tools.

To enter this new itinerary into the travel approval process, users open a standard document browser (like the Windows explorer) and drag the itinerary document onto the *trip status document*. The trip status document is special in that it serves as a central point of coordination for the approval process. Once an itinerary has been dragged onto the trip status document, the approval process in underway, and the employee's task is done, short of checking on the status of the trip. When a trip has been approved or denied by the relevant people, the employee is sent an email notification of the result.

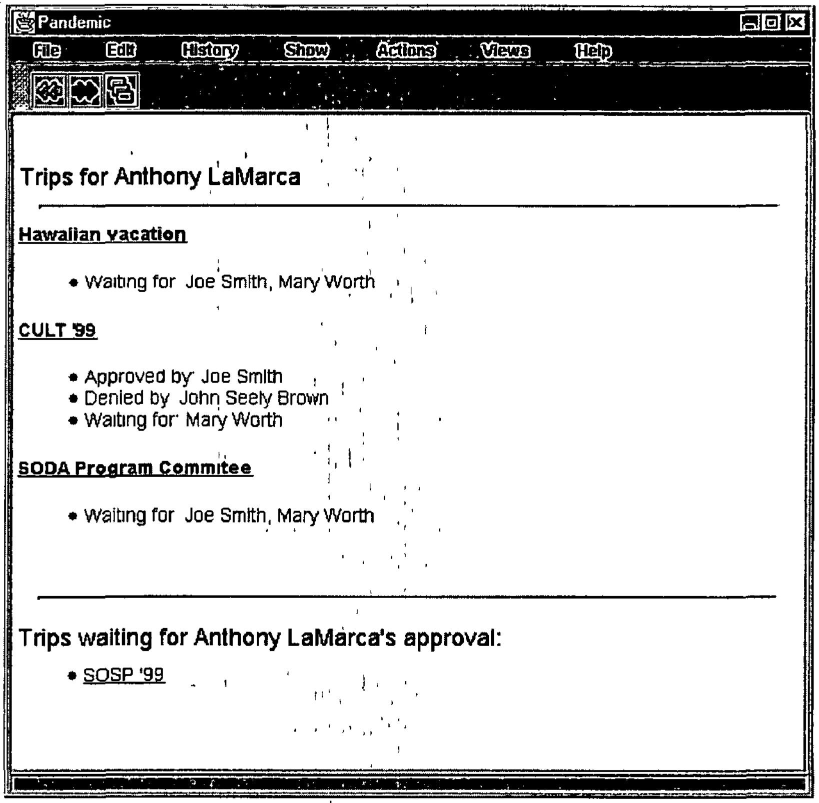
As well as serving as a drop target for new trip itineraries, the trip status document contains content that summarizes the state of the user's travel plans. The content is in HTML format and contains a summary of all of the trips for which an employee has submitted requests (see Figure 1). In this way, an employee can run any application that understands HTML (such as Netscape, Word, or FrameMaker) and view this document to check on the status of their pending trips. The contents of the trip status document also help managers by giving them a list of the itineraries that require their attention. The trip status document serves as a nexus of coordination for those both taking trips and approving trips; and its content is dynamically updated as the states of the pending and processed travel approvals change.

The actual approval or denial of a trip is performed on the itinerary document itself. When a manager opens a travel itinerary that requires their vote they view the document as usual, but something else happens as well: they are presented with a Yes/No voting box, created by an active property, which allows them to decide to approve or deny the trip.

Note how the arrangement differs from classic workflow: users in our system never explicitly run any workflow software. In this case, a manager would open the document in whatever way they normally would to view or edit it, and the system augments their normal interaction to include access to the user interface components needed to drive the workflow process.

How It Works

Maui is made up of two new active properties. The first is the *StatusBitProvider*, which is attached to the trip status document. This property has two functions. First, it listens for drops of other documents and, when it receives them, starts those documents in the travel approval process. It does this by determining the user's manager and department head from organizational information stored in other documents that represent the users of the system (Edwards and LaMarca, 1999), and attaching copies of the second property, described below, to the dropped document. The dropped document becomes—in addition to whatever other roles it is performing—a trip request. Second, the StatusBitProvider serves up the HTML content which summarizes the state of the user's trips. This is



Trips for Anthony LaMarca

Hawaiian vacation

- Waiting for Joe Smith, Mary Worth

CULT '99

- Approved by Joe Smith
- Denied by John Seely Brown
- Waiting for Mary Worth

SODA Program Commitee

- Waiting for Joe Smith, Mary Worth

Trips waiting for Anthony LaMarca's approval:

- SOSP '99

Figure 1: The Travel Status Document

largely a straightforward task of querying for travel documents and formatting the results in HTML; since the bit provider is invoked whenever content is required, it can dynamically generate new information based on the state of the world at the time it is invoked.

The second new property is the *Approve/DenyProperty*, which is what managers interact with when casting their votes on a trip. This property can determine if the user currently viewing the document it is attached to is a manager who's decision is needed for this particular travel request When appropriate, the property can create and display a GUI component with a Yes/No button for voting. Clicking on one of these buttons will record the manager's vote on the document and send the employee notification if appropriate. Applications which are "Placeless-aware" can check for the existence of these components and display them alongside the document content. But the Approve/Deny property can also create a separate, standalone GUI control panel that would appear whenever a travel request is viewed by any application

The knowledge and state of our travel approval process is distributed between these two properties. The status bit provider knows how to add and configure properties in order to turn an ordinary document into a pending itinerary, but does not understand how votes are applied. Any one instance of the Approve/Deny property knows about a single manager's vote, but knows nothing about how any other managers voted. In the next section we describe a more complex process, and we will see that the distribution of knowledge and state increases as more behaviors and users are included in the process.

Managing a Complex Process: Hiring Support

Here at our research lab we have a hiring process which involves a number of different steps and people. We chose to implement a hiring process application as our second document-centered workflow application as it potentially offers significant benefits to us and also tests our model of interaction.[1] As with the travel approval application, we have implemented this on top of Placeless Documents using properties to hold both the logic and state of the process; the collection of properties that comprise the hiring application is called *Carlos*.

The Hiring Process

An illustration of the hiring process in Carlos is shown in Figure 2. In this process, candidates submit their application in the form of a resume and a set of supporting documents such as articles and papers. Upon determining that the application is in order, reference letters are requested for the candidate. Once at least three letters have been received for the candidate, the materials are reviewed by the *screening committee* It is the job of the screening committee to decide whether or not an interview should be scheduled. In Carlos, the screening committee can be of arbitrary size, but we designed our policy for a small group where every member votes. If an interview is approved, the candidate is contacted and a date is chosen for the interview through traditional administrative procedures. At this point, the candidate is brought in for the interview and the process moves into the general voting stage and all members of the lab are invited to submit a hiring proxy and vote on the candidate as described below In Carlos, there are no rigid quantitative rules governing the number of votes that must be cast or what the rejection/acceptance thresholds are Rather, votes and statistics accumulate for the review of the lab manager who makes the final hiring decision.

The User's View

In Carlos, users interact with a number of different document types throughout the hiring process. Some of these documents exist on a per-candidate basis and some

1 The process we describe is similar to, but not the same as the one we use in our lab We have made changes to the process in order to both simplify and illustrate interesting things in the system

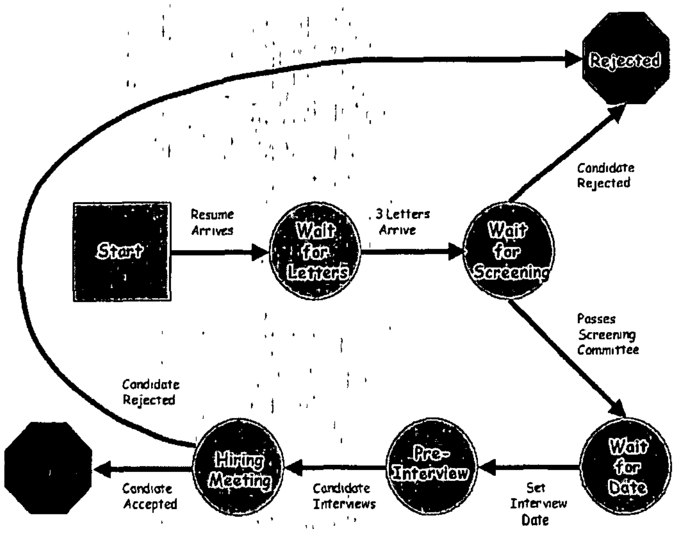


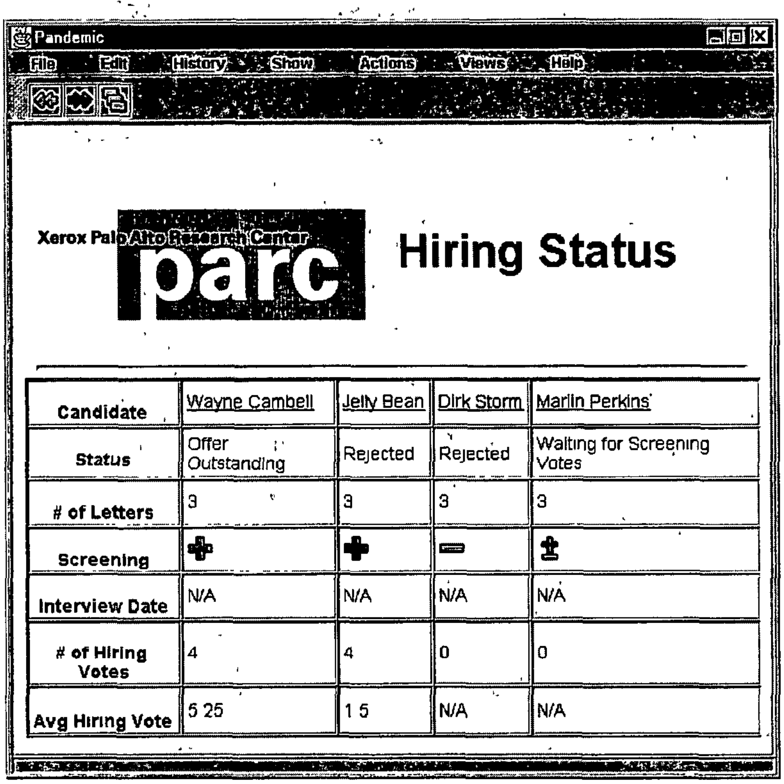
Figure 2: The Hiring Process

are shared. The most important shared document is the *hiring status* document which contains a live up-to-date summary of the status of all of the candidates in the system. A user, using any tool that understands HTML content, can open the status document and be apprised of where any candidate is in the process (See Figure 3). In this overview users can view candidate letters, jump to supporting documents, and see compilations of both screening and hiring votes that have been cast

The status document also serves as the mechanism for adding new candidates to the system. A candidate can be entered into the hiring process by dragging a link to their resume onto the hiring status document. We again see the departure from traditional workflow· in Carlos resume documents can be composed in any application and can be saved in any format. This is especially useful in the hiring process where resumes and letters arrive in a number of different forms including PostScript, simple ASCII, and TIFF images from scanned paper documents.

Upon dragging a resume onto the status document, a new *candidate document* is created This document serves three important functions. First, it contains HTML content that gives a detailed view of a candidate's status. The content is similar to that given in the hiring status document, but provides greater detail

The candidate document also functions as the mechanism for adding reference letters and supporting documents for a candidate. When users drag documents onto a candidate document, they are presented with a choice of what type of document is being dropped (letter or supporting document); the system records their choice Transitions between states in the hiring process take place automatically and user intervention is not required. upon dropping the third reference letter onto a candi-



Hiring Status

Candidate	Wayne Cambell	Jelly Bean	Dirk Storm	Marlin Perkins
Status	Offer Outstanding	Rejected	Rejected	Waiting for Screening Votes
# of Letters	3	3	3	3
Screening	+	+	−	±
Interview Date	N/A	N/A	N/A	N/A
# of Hiring Votes	4	4	0	0
Avg Hiring Vote	5 25	1 5	N/A	N/A

Figure 3: The Hiring Status Document

date, for instance, the candidate's status is automatically changed from "waiting for letters" to "requiring screening decision."

Finally, the candidate document is used to cast both screening and hiring votes in the system. In Carlos, a vote is not just a simple yes/no or a number. Rather, votes in our system have a quantitative portion plus a related document called the *proxy*. This gives users of our system considerably more flexibility to express what they are thinking and why they voted the way they did. To cast a vote for either screening or hiring, users compose their proxy however they desire and then drag this document onto the candidate. At this point, the user is presented with a small GUI to allow them to enter the quantitative portion of the vote. In the case of a screening vote, the quantitative portion is a simple yes or no, in the case of a hiring vote, candidates are judged on a scale from 1 to 7.

In our research lab, hiring votes are often cast in a number of ways. Roughly half the people in the lab attend a formal hiring meeting to discuss the candidate, some people send in email proxies, while others leave voice mail proxies Due to its flexibility, our system can accommodate proxies in all of these forms Email and voice mail are easy turned into documents and attached to the candidate document. Our digital video infrastructure makes it possible to record the entire hiring meeting and break it into different documents, each containing an individual's proxy.

Since Carlos is not a centralized application, but rather a set of coordinated properties, we make an effort to provide the user with as coherent an experience as possible. It is for this reason that we chose HTML as the format for the overall status and candidate documents. The hyperlinking in HTML makes it easy for users to smoothly move from the overall status to a single candidate's status and from there to one of the candidate's letters or proxies.

How It Works

Like our travel approval application, the functionality of Carlos is divided across a number of active properties. The functionality for the overall status document is provided by the *HiringStatusBitProvider* which both provides up-to-date status for all of the candidates and can create new candidate documents given a resume The bulk of the logic for the hiring process lives in the *CandidateBitProvider*. Like the status bit provider, it knows how to produce HTML content describing the candidate's status. It also understands how to receive drops of supporting documents, reference letters and both screening and hiring votes. To do these things, it needs to understand the various states of the hiring process and how and when transitions take place As an example, this property knows that if a candidate is in the "waiting for letters" state and has a third letter dropped on it, it should advance to the to "requiring screening decision" state. Finally, Carlos uses the *RelatedDocument-Property* which gives a document the ability to refer to another relevant document Carlos adds this property to every supporting document, reference letter and proxy vote and configures it to point at the relevant candidate document. In this way, users can quickly jump across linked clusters of related documents

Extending the Document Focus to Other Domains: Software Engineering

The final application we have built using our document-centered model is a tool to support the software development process, called *Shamus* Software development, since it is such a fluid and often chaotic process, needs strong support from workflow tools. These tools try to provide clean task boundaries and separation of responsibilities that can enhance the software development process

Software development *is* a collaborative process and has many of the same attributes of more "traditional" workflow processes. the task is shared by many people, all of whom have different, but perhaps overlapping, responsibilities for the result; the shared artifact (the code) moves from state to state as it progresses toward readiness

One factor that complicates the software process greatly is the presence of multiple *versions* of the artifact—each developer may have his or her own copy of the software that differs from that of the others This divergence is one of the things that distinguishes software development from more traditional workflow. In most workflow situations, there is *one* "version" of the artifact—either the travel is approved or it isn't, the candidate is being interviewed or not. The different participants in the workflow share what is a more-or-less fully consistent view of the state of the process. Contrast this to software where the developers have their own snapshots of the code state that may vary widely from one another

We wanted to see if we could expand the reach of our document-centered model into the extremely fluid domain of software engineering. Our goals were two-fold. First, we wanted to support *awareness* of the overall state of the development process. Second, we wanted to support *automation* of the subtasks in the development process.

Providing a true and useful sense of awareness is typically a hard problem in collaborative development—I may know that you have a piece of code checked out, but I have no way of knowing at a finer granularity what you are doing with it, or even if you're actually changing it at all Workflow systems are *designed* to provide awareness in the context of the task at hand. a manager is aware of the state of travel approvals in terms that "make sense" for the travel approval task, for example. Unfortunately, software engineers typically have only very coarse forms of machine-supported awareness available to them—only at the level of knowing who has an individual file checked out, for instance. We believed that we could support better facilities for fine-grained and task-focused awareness in our model.

We also wished to explore automation of the development process by embodying individual tasks as active properties attached to program content. These properties would serve to off-load some of the cumbersome chores of development, such as running tests, generating documentation, and ensuring that builds finish correctly

Shamus provides a collection of properties to support the development process by enhancing awareness and automation. With its document-centered focus, it also brings with it the benefits that accrue in our other applications—the ability to use the tools at hand to operate on content, and the unification of disparate task models into a single document metaphor.

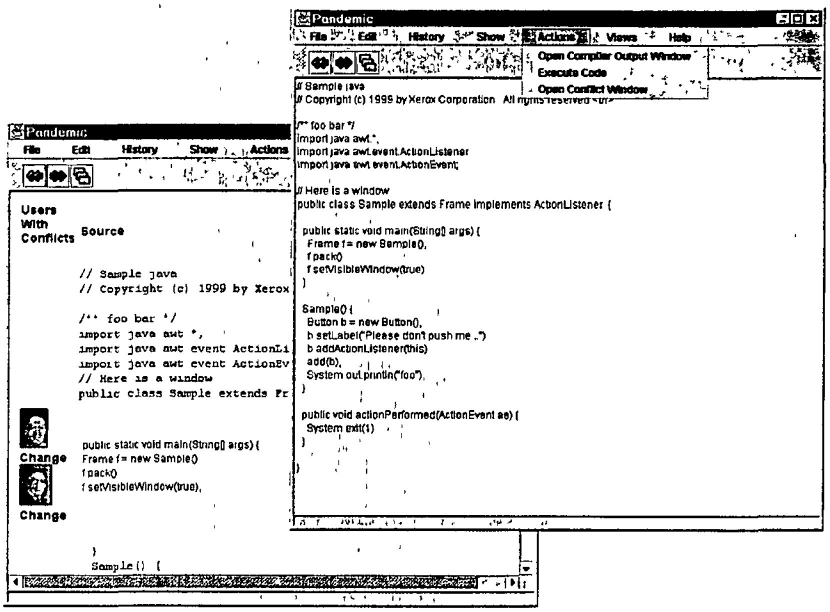
What Shamus Does

The Shamus "application" is actually a collection of properties attached to source code files. These properties control the behavior of the files, and ensure that they interact with one another to support the development process in a smooth way

As mentioned, one of the common awareness problems in collaborative code development is knowing who is working in which source files. Typically the process of reconciling working versions is done when users "check in" the code to a centralized repository that represents the true view of the project.

Our desire was to lessen the potentially cumbersome and error-prone process of reconciling changes at check-in time by providing awareness of the changes others are making in the code base in real-time. The system shows details about where others are working in the code snapshots they have currently in use—regardless of whether this code has been checked in. The system also understands that users are working with program files and knows the basic structure of source code so it can provide awareness information in a way that is semantically meaningful for the task at hand. In our Java example, the system shows which methods in which classes have been changed by users. This form of awareness can allow users to know if they are working in a portion of the code that another user is—or has—worked in regardless of check-in status. The system doesn't prevent users from making overlapping changes—that is what the weightier check-in process is for—but it can provide information about what colleagues are doing.

Figure 4 shows an example of a Java source file being edited by a user The display provides fine-grained information identifying areas that have been changed by other users, even though those changes may not have been checked in yet.



```
// Sample java
// Copyright (c) 1999 by Xerox

/** foo bar */
import java awt *,
import java awt event ActionLi
import java awt event ActionEv
// Here is a window
public class Sample extends Fr


 public static void main(String[] args) {
 Frame f = new Sample()
 f pack()
 f setVisibleWindow(true),


   }
   Sample() {
```

```
// Sample java
// Copyright (c) 1999 by Xerox Corporation All rights reserved <br>

/** foo bar */
import java awt.*,
import java awt.event.ActionListener
import java awt event.ActionEvent;

// Here is a window
public class Sample extends Frame implements ActionListener {

 public static void main(String[] args) {
  Frame f = new Sample(),
  f pack()
  f setVisibleWindow(true)
 }

 Sample() {
  Button b = new Button(),
  b setLabel("Please don't push me ..")
  b addActionListener(this)
  add(b),
  System out.println("foo"),
 }

 public void actionPerformed(ActionEvent ae) {
  System exit(1)
 }

}
```

Figure 4: Shamus supports collaborative development

Shamus also supports the automation of common tasks in software development. Another set of active properties attached to source files can automatically compile and run the programs defined by the source files, and automatically generate documentation. Refer again to Figure 4 Shamus will automatically compile the program and generate documentation when the code is changed. The properties that comprise Shamus can generate new user interface components that will be presented to the user. Here we see a number of "views" that can display the up-to-date documentation for the currently-edited source file, and look at the state of a compilation of the file. An extra control is enabled if the code compiled without errors, allowing the user to easily run the application.

All of these tasks happen automatically without user intervention, and without requiring user attention to the results. These steps are typically a part of the build process that is automated by Makefiles or other build tools. Shamus allows new build tasks to be added incrementally and attached directly to code documents.

We should note that many of our developers work in integrated development environments (IDEs). While the properties that comprise Shamus cannot add new UI components to these applications, or change their innate controls for displaying code, they can still assist in the development process. For example, all of the developers can remain aware of what the IDE-using developer is doing, because the active properties that detect changes are still live and noticing updates to the document, even though it is being edited in an IDE. Second, the properties that provide new UI components for controlling Shamus and displaying conflicting updates can pop up as stand-alone windows next to the IDE's windows. And finally, the automation aspects of Shamus are still active, even though the user is running the IDE. Documentation and object files are being generated in the background, and the results of these operations are available as separate Placeless documents.

How Shamus Works

The awareness aspects of Shamus are driven through a property attached to code source files, called the *CodeAwarenessProperty*. This property awakens periodically and computes hash code signatures of the text of individual methods. These signatures are stored on the document as properties identifying the state of the content in terms of the source code structure.

With the signatures computed, the property performs a query to find all versions of the same source file that are in use by other Placeless Documents users. Since these documents also have the CodeAwarenessProperty attached to them, they will have up-to-date lists of their signatures attached to them as well These signature lists can be compared to see if users are making potentially conflicting changes.

The CodeAwarenessProperty can provide a UI widget which presents these potential conflicts.

The automation tasks are implemented by a separate property, called the *CompilerProperty*. This property performs its tasks whenever new content is written, by invoking external tools to do their jobs. The CompilerProperty will run the *javac* compiler to compile Java code when needed, and the *javadoc* documentation generation system. The property captures the output of the external compilation process and creates a new Placeless document for it The object files that result from compilation are imported into Placeless, as are the error and warning messages from the compiler. The javadoc-generated HTML documentation is also imported into Placeless.

These properties expose these derived documents as "views" of the original source, which are directly available from property-provided UI controls (they can, of course, also be accessed through a browser or via query from other applications) When the compilation executes successfully, the CompilerProperty also generates a "run" control that applications can use to invoke the code.

Our experiment with Shamus has convinced us that the facilities provided by the Placeless Documents system are applicable to a range of application domains Shamus provides support for two of the most time-consuming tasks in collaborative development—coordination with peer developers, and automation of aspects of the build process. Shamus adds value to the development process while allowing users to use the tools they are comfortable with. It enhances the functionality of existing tools without getting in the way of commonly-used development processes.

Conclusions and Future Directions

Our work has examined a model of workflow which separates monolithic workflow applications into individual component behaviors which can be attached directly to the content which is at the heart of many work activities. By exposing the state of a workflow process as a document, we can leverage the wealth of general-purpose document management tools that already exist in our environments. Via the integration of computation and content, we hope to move closer to the model envisioned by Abbott and Sarin (1998) of workflow being an "invisible capability that permeates" *all* applications.

The ability to use these tools comes with a price, however—by using general-purpose tools we lose the tight semantic binding that specialized applications provide. Custom, stand-alone workflow tools can embody the semantics of the tasks they manage in ways that general-purpose tools cannot. This is a tension in our work: in the design of the workflow processes in our model, we must strike a balance between the use of general tools and the specific requirements of the task at hand. Managing this balance effectively is a direction for future research.

In addition to the pragmatic benefits of allowing standard tools to integrate with workflow practice in a novel way, our model of active content also can be used to provide more fluid expressions of workflow tasks For example, a single piece of content can participate in multiple workflows via a "layering" of active properties New behaviors can easily be added to documents without disrupting the behaviors already in place

While our primary focus has been on enabling the types of tasks that workflow applications have typically managed, we believe that our infrastructure has implications and uses for other types of tasks as well. The software engineering example presented in this paper is one of these—it represents a class of task that, while collaborative, is very different from the rigorously structured and "globally consistent" views that traditional workflow processes enhance. The same active property mechanisms that support workflow can be used by this task to great benefit, though We believe that the foundation provided by the Placeless Documents infrastructure can support the construction of "applications"—or more accurately, clusters of active property sets—for a range of document organization, management, and use tasks

The systems described in this paper have been implemented atop the Placeless Document system. While Placeless itself is approximately 100,000 lines of Java code, the amount of code needed to implement the active properties we use is quite small—on the order of 300 lines of code each. We believe that Placeless provides an excellent substrate for constructing and experimenting with new document services and tools, as it allows easy development of behavior in a piecemeal fashion

This paper is primarily an exposition of the "systems" aspects of Placeless and how they can be applied to novel applications; indeed, our original motivation for doing this work was to see if the facilities provided by Placeless could be applied to a demanding domain such as workflow. We believe that the primary value of this work is in the collection of systems concepts that allow the construction of new forms of document-based interaction, more than any particular expressions of these systems concepts in actual workflow examples. Our desire is to put forward a novel model of workflow and a description of the infrastructure that would be required to support this model.

We are currently investigating new applications for Placeless, including electronic mail, general-purpose organizational tools, and new "vertical" application domains including more workflow examples We plan to continue to refine and extend our infrastructure throughout the next year based on the lessons we learn from these applications.

Acknowledgments

The Placeless Documents project at PARC is a large effort that has benefited from the involvement and support of many people In particular, we'd like to thank Karin Petersen and Douglas B Terry, both of whom contributed to the Placeless design and implementation, but were not directly involved in the workflow applications described here

References

Abbott, K and Sarin, S (1994) Experiences with Workflow Management Issues for the Next Generation Proc ACM Conf Computer-Supported Cooperative Work CSCW'94 (Chapel Hill, NC) New York ACM

Begole, J., Struble, C , and Smith, R (1997) Transparent Sharing of Java Applets A Replicated Approach Proc ACM Symp User Interface Software and Techology UIST'97 (Banff, Alberta) New York. ACM

Brinck, T and Hill, R (1993) Building Shared Graphical Editors Using the Abstraction-Link-View Architecture Proc European Conf Computer-Supported Cooperative Work ECSCW'93 (Milano, Italy) Dordrecht Kluwer

Bullen, C and Bennett, J (1990) Learning from Users' Experiences with Groupware Proc ACM Conf Computer-Supported Cooperative Work CSCW'90 (Los Angeles, CA) New York ACM

Cortes, M and Mishra, P (1996) DCWPL A Programming Langauge for Describing Collaborative Work Proc ACM Conf Computer-Supported Cooperative Work CSCW'96 (Boston, MA) New York ACM

Dourish, P. and Bellotti, V (1992) Awareness and Coordination in Shared Workspaces. Proc ACM Conf Computer-Supported Cooperative Work CSCW'92 (Toronto, Ontario) New York ACM

Dourish, P , Edwards, K., LaMarca, A , Lamping, J , Petersen, K , Salisbury, M , Terry, D and Thonton, J (1999) "Extending Document Management Systems with Active Properties " Submitted, *Transactions on Information Systems.*

Edwards, K , LaMarca, A (1999) "Balancing Generality and Specificity in Document Management Systems " Submitted, Interact'99

Gelernter, D (1985) Generative Communication in Linda ACM Trans Programming Lanaguages and Systems, 7(1), 80-112

Grudin, J (1988) Why Groupware Applications Fail Problems in the Design and Evaluation of Organizational Interfaces Proc. ACM Conf Computer-Supported Cooperative Work CSCW'88 (Portland, OR) New York ACM

Isaacs, E , Morris, T and Rodriguez, T (1994). A Forum for Supporting Interactive Presentations to Distributed Audiences Proc. ACM Conf Computer-Supported Cooperative Work CSCW'94 (Chapel Hill, NC) New York. ACM

Jeffay, K , Lin, J , Menges, J , Smith, F and Smith, J (1992) Architecture of the Artifact-Based Collaboration System Matrix Proc ACM Conf Computer-Supported Cooperative Work CSCW'92 (Toronto, Ontario) New York ACM

Johnson, J and Nardi, B (1996) Creating Presentation Slides A Study of User preferences for Task-Specific versus Generic Application Software ACM Trans Computer-Human Interaction, 3(1), 38-65

Olsen, D , Hudson, S , Phelps, M , Heiner, J , and Verratti, T (1998) Ubiquitous Collaboration via Surface Representations Proc ACM Conf Computer-Supported Cooperative Work CSCW'98 (Seattle, WA) New York ACM

Olson, J and Teasley, S (1996) Groupware in the Wild Lessons Learned from a Year of Virtual Collocation Proc ACM Conf Computer-Supported Cooperative Work CSCW'96 (Boston, MA) New York ACM

Palmer, C and Cormack, G (1998) Operation Transforms for a Distributed Shared Spreadsheet Proc ACM Conf Computer-Supported Cooperative Work CSCW'98 (Seattle, WA) New York ACM

Prakash, A and Shim, H (1994) DistView Support for Building Efficient Collaborative Applications using Replicated Objects. Proc ACM Conf Computer-Supported Cooperative Work CSCW'94 (Chapel Hill, NC) New York ACM

Schmidt, K and Bannon, L (1992) Taking CSCW Seriously Supporting Articulation Work Computer-Supported Cooperative Work, 1(1-2), 7-40.

Winograd, T and Flores, F (1986) Understanding Computers and Cognition A new foundation for design Norwood, NJ Ablex