Balancing Generality and Specificity in Document Management Systems

W. Keith Edwards and Anthony LaMarca
Xerox Palo Alto Research Center
Palo Alto, CA 94304 USA
+1 (650) 812-4405
{kedwards, lamarca}@parc.xerox.com

Abstract. This paper describes an experiment to extend the reach of the document-centric metaphors of computing to new physical and virtual objects in the workplace. By bringing these entities into the sphere of electronic documents, we leverage users' knowledge and expectations about how documents work, and also leverage existing tools and applications that understand and manipulate documents. Being able to use general-purpose tools on semantically-loaded content types can be useful, but there are potential pitfalls with the loss of the functionality provided by special-purpose applications. We investigate a solution to this problem that strikes a balance between generality and specificity by allowing application functionality and user interface components to be associated with the document and used by general-purpose applications that operate on that document.

Keywords: document management, user interfaces, extensible software architectures, Placeless Documents.

1. INTRODUCTION

Much of contemporary computer use is characterized by document-centered tasks: we edit, create, mail, print, and file electronic documents. We commonly see file and folder icons on our desktops, and the web has brought a document-centered metaphor to network computing tasks. The prevalence of electronic documents on our virtual desktops mirrors the use of physical documents in the workaday world [1][2]. The result is that the metaphors of document-centered computing—opening and closing, reading and writing, searching and printing—have become part of the lingua franca of computing.

Our desktop tools are specialized to match our document management tasks and have increased in sophistication to support the various uses to which documents are put. Modern tools support such techniques as indexing, hyperlinking, collaborative document filtering and recommendation [9], organization and browsing of document collections [7], automatic identification of document genre [6], and so on. The diversity of document management tools is indicative of the prevalence of electronic document management in our online work.

But there is still a large portion of our day-to-day work which is not document-centric. We commonly perform system administration tasks such as managing printers or installing software; control processes such as databases or other applications; collaborate with users by initiating on-line conferences or managing work flow tasks; and control external devices such as attached cameras or speakers. While these tasks may use documents—a work flow task is typically focused around some shared set of documents for instance—the commonly used and understood metaphors of document-based computing do not apply to these tasks. In short, while they may *involve* document use, creation, and management, the tasks themselves are not directly *represented as* documents, with the same affordances and utility of existing electronic documents.

At first glance, many of these tasks seem qualitatively different than typical document management: they involve working with processes, users, and devices that have no innate and *a priori* existence as readable and writable content. On closer examination, however, one can see similarities between these currently non-document centered tasks and document management. In particular, many of these tasks involve the creation and manipulation

of externally-represented state: system management involves retrieving and updating system status. Work flow involves determining the steps a task has passed through, and updating some external record of the task to move it to new states. Many devices are readable and writable and can both provide and accept what may be thought of as content.

With these similarities in mind, we have begun an investigation into whether a document-centered model could be expanded to include both physical and virtual entities not previously represented as documents. Essentially, the objects at the focus of this work would be reified as documents, allowing the metaphors of document creation and manipulation to be applied to new tasks. And—perhaps more importantly—such an extension of the document metaphor would leverage the wide body of *existing* applications, tools, and services that already understand how to work with documents.

In such a system, users would "live" in a document-oriented world and would be able to use familiar tools such as editors, and techniques such as cut and paste or drag and drop, to interact with a much wider range of computational and physical entities than before. In essence, electronic documents become a metaphor for the interactive objects in both the virtual and the physical worlds, rather than simply a metaphor for physical documents.

1.1. Goals

Building a system to support a document-centric view of the world has two major challenges. First we need a document management infrastructure that can seamlessly support the mapping from physical and computational entities to documents in such a way that these new documents appear as first-class peers of their existing cousins. Our investigation is part of a larger project called Placeless Documents [4] which provides this needed functionality; an overview of Placeless Documents is given in Section 3.

Second, we need to understand how the concept of the document as a user interface metaphor be mapped to these other entities. We believe that by creating such a mapping, we can achieve a number of benefits. There are the obvious benefits of leveraging users' knowledge of document systems in new domains, and leveraging existing document tools to apply them to new types of objects. But beyond these benefits, we believe that two particular effects will come into play in such a system.

The first is *multiplicity*. Multiplicity is the combinatorial effect which occurs when a document is accessible by any number of tools, users, and protocols. Multiplicity allows users to select their preferred tools for interacting with our new document types, rather than having to rely on customized, special-purpose tools. It is multiplicity which makes these new entities seem like "real" documents by allowing users to interact with them using, say, Microsoft Word or the File Manager, like they would any other file on their computer. Multiplicity also encourages adoption by supporting users' desired tools, rather than forcing them on to new applications.

The second effect is *knitting*. Knitting is the ability use a document as a representation of a live and up-to-date view of the world. The content for these live documents can come from any number of sources: it can be composed from pieces of traditional static content, formed from query results, or read from on-line devices. By having live content, knitting allows documents to know the context and history of their use, and respond in appropriate ways to changes in their environment. Knitting not only supports documents that can provide an instantaneous, updating view of some state, but also supports documents which carry their own behaviors, semantics, and perhaps even interfaces with them. In our discussion of the new document types we have created, we shall see several examples of multiplicity and knitting in action.

1.2. Striking a Balance

There is a delicate balance to be struck in this work. The reason that these computational objects have not been represented as documents is that they often carry with them such a great deal of domain-specific semantics, and the functionality that encodes these semantics, that general-purpose document tools are an inappropriate medium for working with them. Clearly, specialized functionality for a camera would not be accessible through tools like Word or Excel.

To expose this functionality, such computational entities are usually manipulated via special-purpose applications. For example, a cellular telephone may come with a special purpose application to allow the editing of its internal phone book from a desktop computer. This application reflects the functionality supported by the phone, and yet may allow its content—the actual names and numbers in the phone book—to be imported or exported in some common formats. The fact that the content may be imported or exported is an acknowledgment of the "documentness" of the phone book.

This approach to using specialized applications for accessing and manipulating specialized sorts of data is, of course, very common, and represents one endpoint in the spectrum of possibilities. At this endpoint, the content is so laden with domain semantics that it is best manipulated via specialized tools rather than general purpose applications. Video games are a prime example: they may contain traditional content in the form of images and sound samples, but without the game specific functions, the experience is quite different.

At the other end of the spectrum we see purely general purpose tools such as word processors,

spreadsheets, and web browsers. These tools, while they may operate on only a small number of types of data, still impose very few restrictions on the content they manipulate—the domain semantics, if any, in the content they manipulate are not reflected in the applications themselves. Although, just as specialized applications may attempt to support general purpose tools through importing and exporting, general purpose applications may attempt to reflect application semantics through the use of extensions such as ActiveX controls, browser plug-ins, or macros carried with the content.

Traditionally, these two endpoints of the spectrum have been the only options available. In our desire to leverage the power of general-purpose tools and take advantage of the combinatorial effects of multiplicity, we wish to strike a new balance. We want to be able to expose content—even semantically-loaded, domain-specific content—as documents. And yet we need to provide ways for documents to carry enough of their application behavior that they remain useful in this new setting.

2. RELATED WORK

Some work has been done in the past to expose computational, and sometimes physical, objects as documents. On the Unix system, the /proc filesystem [3] is a specific example of making one particular type of entity—a Unix process—appear as a file. These files can be read to determine the state of a process, and written to effect changes in the state of a process. Operating systems such as Mach [8] provide a generalization of this feature by supporting easily-loadable filesystem code that can be used to provide filesystem interfaces for arbitrary entities. As opposed to these example, we have available to us a general framework for easily extending *any* type of entity into the realm of documents. Further, Placeless documents are more fully realized as first-class documents than the files in such filesystems: they can be freely created, grouped, and renamed. The /proc filesystem in particular imposes sufficient restrictions that process files are noticeably different than "normal" files in the Unix filesystem.

On the web, cgi scripts [5] are used to dynamically create web pages based on some external state—an online catalog may use a cgi script to create a page with product listings from a database. An essential distinction between such scripts and our work, however, is that Placeless documents are full-fledged documents with their own innate identity. The URLs that refer to cgi scripts, on the other hand, are simply the *names* of documents. The "documents" that they refer to actually have no inherent identity as documents—they cannot be renamed, foldered, sent to other users, and so on. As opposed to simply providing a naming system which can refer to new types of content, Placeless provides a system in which entirely new first-class documents can be created.

3. ARCHITECTURE

This section presents an overview of the architecture of our system. We begin with a discussion of the basic facilities of the Placeless architecture, highlighting those features that were necessary to build our document-centric metaphor.

3.1. Overview of Placeless Documents

Traditional document management systems and filesystems present a hierarchical organizational 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 its context of use: the document is a budget; it's shared with my workgroup, and so on.

3.1.1. 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 bits of code that can support 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 to interested parties.

It is these active properties in the system, particularly the bit providers, which accomplish the "knitting" together of dynamic information described in the introduction. Since property code can perform arbitrary actions when it is invoked, properties can return arbitrary results based on the context of their use, and the state of the document management system at the time they are invoked. We shall see some examples of these in Section 4.

3.1.2. 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, *serve* document collections where they see fit. So, for example, a user who is often disconnected can run an instance of the Placeless Documents system on a laptop to provide full

Placeless functionality, even when disconnected from the rest of the world. Such a user could not, of course, see documents that reside on other, unreachable systems with full consistency.

A final feature of Placeless Documents is to provide compatible interfaces for supporting off-the-shelf applications. 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. 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.

For example, we have built a Network File System (NFS) server layer atop Placeless. This layer lets existing applications, 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, 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.

As mentioned in the introduction, one of the key benefits of our model is in its multiplicity of access: by exposing arbitrary entities as documents, and then making these documents widely available through arbitrary interfaces, we gain great leverage from existing tools and applications. We shall see some examples of multiplicity in use in the next section.

4. DOCUMENTS AS A METAPHOR FOR THE WORLD

Within the context of Placeless, we have built the infrastructure that allows us to experiment with document-centric computing. Our work has followed two main paths. First, we implemented a set of novel document types within the Placeless framework. These new document types reflect objects and tasks commonly used in office work. So far, we have focused on documents as representations for:

- Physical devices
- People
- Computational resources, including processes, machines, and the Placeless system itself
- Abstract tasks and processes such as work flow

The challenge here was to implement such disparate concepts within a document metaphor—that they would *act* like documents in all important ways. By leveraging users' knowledge we can build on their *expectations* of document systems, so that our documents will behave in predictable ways. We believed that if we could achieve this goal, then the Placeless infrastructure would allow existing document management tools to be brought to bear on these new document types.

The second path of our research was to use the facilities provided by Placeless to augment and enhance the semantics and behaviors of these new documents in powerful ways. While pre-existing applications can use these documents without understanding their special qualities, we felt that there was great utility to be had by providing a means through which tools *specifically written* for the Placeless environment could interact in a *general way* with these new document types in novel ways. In particular, the manifestation of this idea we have focused on has been to allow documents to provide portions of their own user interface and application behavior. Through this mechanism, applications which are open to runtime extensibility can essentially be configured by the documents they are working with to have new behaviors and interfaces.

The following sections describe a number of case study experiments we have done.

4.1. Physical Devices as Documents

The first—and perhaps most obvious—extension of the document model was to represent physical devices as documents. In some ways, the mapping of devices to documents has been accomplished before; note the prevalence of live cameras pointed at coffee pots on the web, for instance.

There is a subtle but important difference between this work and "web cams" however. While web cams provide live content from a device, they are not true first-class documents that support organizing, sharing, annotating and customizing as our Placeless documents do. In addition, web pages do not offer multiplicity as the content is only available through a limited set of interfaces. A URL pointing to a web cam could be part of the implementation of a first-class document, but without the rest of the Placeless functionality it is just a pointer to a set of content.

Still, in the tradition of the web cam, one of the first physical devices we modeled as a document was a camera. This document's bit provider generates a stream of JPEG images, and appears in the filesystem as a JPEG document. The document is not writable.

A more interesting example of a document representing a physical device is our printer document. A printer document is instantiated by creating a new document with the PrinterBitProvider attached to it. This bit provider looks for a property called "Printer" that designates the name of the

printer it represents. Reading from this file generates content representing the current state of the print queue at the moment in time at which it is read. Any content which is written to the document is printed.

Our final device document is the most complicated. We have built an interface to a digital cellular telephone. Our particular telephone (a Nokia 6190 GSM phone) affords two possible document representations. The first is the telephone's role as a carrier of a personalized phone book. The second represents the telephone's role as a sender and receiver of SMS (Short Message Service) messages.

In our system, both of these capacities are represented as documents. Reading from a phonebook document downloads the contents of the phone into the system for viewing and editing. Writing this document updates the stored information on the telephone itself. Since the phonebook content is a true Placeless Document, it can be indexed and edited using existing tools on the desktop. With this particular device, information can only be read or written if the phone is physically docked with the computer. Therefore, the Placeless system binds the "readable" or "writable" attributes of the phonebook document to the phone's connection to the computer.

In its second capacity, the phone can send and receive Short Message Service (SMS) messages. This role is represented as a separate document which, when read, downloads the list of stored messages from the phone and presents them as the document's content. Any text written to this document will be sent via an SMS gateway to the physical phone represented by this document. In the case of this SMS document, messages can be read only when the phone is docked, but messages can be written anytime. So one can message a remote user by opening his or her phone document in Word or other tool, creating some content, and them simply saving the file.

While specialized tools can provide access to all of these features of the phone, multiplicity allows us to leverage existing tools in an extremely powerful way. Both the phonebook and SMS documents can be indexed by existing tools such as AltaVista Personal, edited via existing tools such as MS Word, and grouped and managed in collections just as any other document on the desktop.

4.2. People as Documents

Clearly one of the entities that people most often work with on-line is other people. Placeless has a cryptographically secure notion of a principal, with a one-to-one correspondence to physical users. We have constructed an on-line representation of the users of the system as "person documents." These documents are more than simple home pages for users of the system. First, we use the document for a principal as the locus of interaction with that person. All references to a principal are established as references to that principal's document. Second, we use the idea of knitting to build a representation of the state and context of the person in the world at the time the document is viewed.

The person document for an individual is created as a collection of properties that contain personal preferences and settings, and a bit provider which can assemble this information, and other information from the state of the world, into a cohesive form. These documents can be annotated by their owners (or, in some cases, others) with properties for telephone numbers, email addresses, and so forth. These properties are also used by the bit provider in constructing its content when the document is read. Figure 1 shows an example of a person document. Contact information is retrieved from the properties attached to the document and displayed as HTML.

Figure 1: A Person Document

More interestingly, however, is the "live" content the bit provider constructs based on the context of the user in the world. The person document for a given user will contact the Placeless Documents system and retrieve a list of all currently open documents the user is working with. These documents are represented as HTML links to the actual documents. Here we see both an example of knitting—the construction of dynamic, live content from multiple sources—and multiplicity—since these documents are available as web documents, we can access them through a web browser or other tool which understands HTML.

Writing to a person document transmits the written information to the person represented by the document. A property on the document, settable by its owner, denotes the "preferred" contact method—email, SMS message, etc. The bit provider for the document will consult this property to see how to transmit the information. It may also use the type and size of the information in its process (it probably makes no sense to send a JPEG file over SMS to the

user's telephone, for instance). As we shall see in Section 4.5, applications which are aware of Placeless Documents can be written to interact with person documents in a more intelligent way.

4.3. Computation as Documents

A third class of entity we have experimented with is computational processes. To this end we have created document types for two computational processes: the Java Remote Method Invocation (RMI) registry, and the Placeless kernel itself.

The Java RMI system uses a nameserver, called the registry, on each host on which a remote object runs. The RMI registry document provides a means for determining the objects named in the registry by reading it. Currently the registry document is not writable, although one could imagine a process by which writing to the registry caused new objects to be registered, or changed the state of the registry.

More interesting is the Placeless kernel document. This document represents the instantaneous state of a user's Placeless kernel. When read, the bit provider creates a page that contains information about the type of the kernel on which it is running (the version number, database implementation being used, etc.), and information about kernel status (whether certain internal threads are active, and which documents are currently in the kernel's internal cache). See Figure 2. The kernel document is updated live as the kernel changes states. While the kernel document is not writable, it does use properties as a way to provide mechanisms for controlling it from Placeless-aware applications (see Section 4.5).

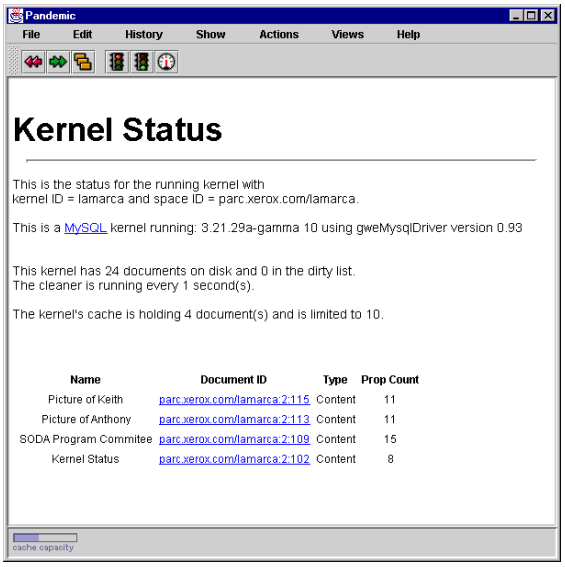

Figure 2: A Kernel Document in a Placeless-aware Content Viewer Application

4.4. Tasks as Documents

The final type of entity we have brought into the document domain is abstract tasks. In particular, we have created a workflow process that is reflected in documents and properties.

In a traditional workflow system, a tool will be used to shepherd some document through a set of required steps (usually approvals by managers) until it reaches a final state. This workflow application understands the semantics of the particular tasks it is administering, perhaps via some rule set that encodes the company's policies. The workflow application provides a centralized point of focus for the process; the documents that are being approved are the things which move through this process under the control of the workflow tool.

We wanted to turn the traditional arrangement on its head. Rather than using a single workflow tool that operates on particular types of form documents, we wanted to be able to use *any* type of document as the object of a workflow. Further, rather than using a specific application that manages a given workflow interaction, we use a document that represents the abstract *process* of a particular workflow situation.

Our primary example of this is a travel approval document, used to obtain management permission to take a trip. Rather than representing a travel approval form or request, this document represents the *abstract process* of travel approvals. Opening the document causes its bit provider to scan the system, retrieving a set of documents flagged via properties as being used in the travel approval process. The bit provider then generates an instantaneous summary of the state of the world from the perspective of the travel approval process—which requests are outstanding, which have been fully approved, which have been denied.

A document is marked as being "part of the travel approval process" by having a specific property attached to it. Thus, any document (not just a particular form) can be the source of a travel request. By attaching the travel approval property to it, it *becomes* a travel request and makes its way through the system. Opening the process document retrieves the complete state of all travel approvals involving the current user; dropping a new document on the process document causes that document to become a travel request, and begins the approval process.

When a document that is involved in the process is viewed by a user who needs to make a decision about that approval, the document can actually communicate its new affordances—acceptance or denial of the travel request—through extensions to its interface. While this process is described more fully in the next section, this ability for documents to carry their own behaviors, and be aware of the context of their use, is one of the key benefits of our system.

The travel approval workflow is an example of knitting: documents remain the focus of activity, but their behavior is affected by the context of their use, and by the states of other documents in the system.

The travel approval process also introduces an interesting quandary. Specialized forms and applications for managing a particular process exist because they encode the semantics of a particular task. Once we've removed both the forms and the applications, we've entered a world of generality in which we are no longer constrained by the semantics of the tasks at hand. The next section presents some thoughts at a solution to this problem.

4.5. Documents that Carry their Interfaces

In the current world, specialized forms of content are operated on by specialized interfaces. To revisit some of the examples in this paper, the phonebook of my phone can be updated by specialized software that comes with the device. The corporate travel approval process uses a set of forms that represent the steps and signatures required for the process.

These specialized applications exist because the data they operate on are not general-purpose, free-form documents: they have restrictions on them by virtue of the semantics of the applications in which they are used. The applications for these data present specialized interfaces which reflect these restrictions. By moving to the lowest common denominator of the document we gain a certain power (reuse of existing tools, leverage of existing knowledge), but we also lose the application-specific support which makes specific tools valuable. Thus, there is a tension between the desire for generality and the need for application-specific knowledge to be involved in the editing of these documents.

We have investigated a solution to this problem in which we use the property system provided by Placeless to allow documents to carry active behavior and even portions of their own user interfaces. In essence, "atoms" of application code are broken out and, rather than existing solely in a centralized application, travel with the documents they are attached to. Obviously, existing tools and systems will have no way to know about Placeless' properties on documents; this facility is useful for applications specifically written for the Placeless setting.

We have established a number of conventions by which Placeless-aware applications can have portions of their behavior and interfaces dynamically loaded from the documents they are operating on. For this investigation, we have defined three classes of new document behaviors: *actions*, *views* and *components*.

Actions are simple argumentless commands that documents can be given. The actions are free to do whatever computation and produce whatever side effects they desire, but provide no return result. Actions provide a way for documents to export simple operations. As examples, the Placeless kernel document has an action to flush its internal caches, the people document has an action to bring up a window for composing email to the person, and the camera document has actions to adjust the color, size and timestamps on the image from the camera.

Views are like actions in that they are simple argumentless commands, but differ in that they return another document as the result of their execution. Views provide an extensible way for documents to point to or themselves produce other documents. As an example, people documents have views that both return the resume and homepage of the person.

The last class of behavior extension is the *component* which, when evaluated, returns a UI component to help display some aspect of the document's state. Component extensions provide the ability to extend the UI beyond the chosen rendition of the document's content type. The kernel document, for example, has a component extension which returns a progress bar widget which shows how full the system's internal caches are.

Placeless provides a way to programmatically discover and access the extensions that a document is offering. To demonstrate the use of this, we have built a general purpose document viewer/editor which introspects the displayed document and makes the extensions available in a sensible way: Actions are put both in an menu named "Action" and on a tool bar. Selecting an action causes it to be executed. View extensions are put in a menu called "View", and selecting it causes its execution and the resulting document to be viewed. Component extensions are all executed and the resulting widgets are placed in a status bar at the bottom of the viewer. (See Figure 2.) In this way, we have populated a new point in the spectrum between general-purpose and domain-specific applications. Our document viewer is general purpose in that it is free of application specific semantics and can view many different content types, yet it offers some application specific functionality via action, view and component extensions.

5. CONCLUSIONS

This work has investigated a model in which the notion of an electronic document is extended to new types of physical and virtual entities. Our system, built atop the Placeless infrastructure, provides a way to represent novel entities as documents. Further, the system can present these documents in such a way that they can be used by existing applications, including tools that read and write to the filesystem, and that read and write to the web.

Unlike related work on loadable filesystems, cgi scripts, and the like, Placeless takes a holistic approach to document management: our system is extensible in how it exposes underlying content, and extensible in how it makes this content available to applications. The resulting multiplicity—the ability to make most anything look like a document to any

most application—is a key source of power in this system.

Likewise, the ability to dynamically create content which exists nowhere in the physical world and is based on the state of world and the context of the use of the document is of key importance. This ability to take multiple information sources and knit them together based on context is essential in turning isolated sets of documents into a coherent whole.

6. STATUS AND FUTURE WORK

While the implementation of the Placeless system is ongoing, the core functionality has been developed to the point that we were able to build all of the document types described here. The core Placeless system is approximately 100,000 lines of Java code. The properties needed to support particular document types, however, are only about 300 lines of code each. We have been encouraged by our experiments in mapping physical and virtual objects into the document space. Our work has validated the usefulness of the facilities provided by Placeless as an infrastructure on which to base this metaphor.

As discussed, one of the problems with representing arbitrary entities as documents—and making them usable via general purpose tools—is that these entities may lose some of the semantics which are the basis of their utility. To this end, we have investigated a set of conventions for allowing documents to carry with them fragments of application behavior that are made manifest when the document is used by a Placeless-aware application. This solution represents a middle ground between using existing general-purpose document tools (and hence losing all semantics associated with the document), and using specialized applications (and losing the ability to use existing tools on the application content). This approach deserves more work to determine if it is a truly viable solution.

7. REFERENCES

[1] Adler, A., et al. "A Diary Study of Work-Related Reading: Design Implications for Digital Reading Devices." *Proceedings, ACM CHI Conference*, 1998.

[2] Bellotti, V,. Rogers, Y., "From Web Press to Web Pressure: Multimedia Representations and Multimedia Publishing." *Proceedings, ACM CHI Conference*, 1997.

[3] Curry, D., *UNIX Systems Programming*. O'Reilly Associates, 1996.

[4] Dourish, P., et al. "Extending Document Management Systems with Active Properties." Submitted, *Transactions on Information Systems*.

[5] Gundavaram, S. *CGI Programming on the World Wide Web*. O'Reilly Associates, 1996.

[6] Kessler, et al., "Automatic Detection of Text Genre." *Proceedings ACL/EACL*, 1997.

[7] Lamping, J., et al., "A Focus+Context Technique Based on Hyperbolic Geometry for Visualizing Large Hierarchies." *Proceedings, ACM CHI Conference*, 1995.

[8] Rashid, R., et al., "Mach: A Foundation for Open Systems." *Proceedings, Second Workshop on Workstation Operating Systems* (WWOS2), 1989.

[9] Shardanand, U. Maes, P. "Social Information Filtering: Algorithms for Automating Word of Mouth." *Proceedings, ACM CHI Conference*, 1995.