DoKumobility: A Document-based Service Architecture

Gene Golovchinsky, Gerry Filby

FX Palo Alto Laboratory, Inc.
3400 Hillview Ave., Bldg. 4
Palo Alto, California, 94304
{gene, filby}@fxpal.com

***Abstract*—Mobile users often require access to their documents while away from the office. While pre-loading documents in a repository can make those documents available remotely, people need to know in advance which documents they might need. Furthermore, it may be difficult to view, print, or share the document through a portable device such as cell phone. We describe DoKumobility, a network of web services for mobile users for managing, printing, and sharing documents. In this paper, we describe the infrastructure and illustrate its use with several applications. We conclude with a discussion of lessons learned and future work.**

***Index Terms*—**Document services, printing, service orchestration.

I. Introduction

The ever increasing pervasiveness of the Internet has promised changes in the work-style of the mobile worker. No longer do professionals need to be constrained to the physical limitations of their laptops, only accessing the information and applications on the local hard-drive.

Many Internet systems have strived to provide access to necessary or important information to people while mobile, away from their desktop. Web repository applications such as BSCW [2] and DocuShare [7] provide web interfaces to file systems that users could access from anywhere (provided they had first uploaded the files to the web site). Xerox's MobileDoc [6] provided users with specialized services to access their organizations' documents through Blackberry devices or mobile phones and then email or fax them. Other Internet systems have attempted to provide services that act on documents, rather than provide access. Internet fax services like eFax [13] enable users to receive and send faxes using email or a web site.

However, each of these solutions remains an island. Different web repositories provide their own interfaces to their own file systems. Accessing these files on a cell phone remains a frustrating exercise – what do you do with a 30Mb PowerPoint on a cell phone?

The emergence of web services and the growing provision of remote API's for using these services offers an opportunity to start to close the gap between these different document providers and document services.

In this paper we present DoKumobility, a small service oriented architecture that enables uniform access to any type of file source provider and the ability to apply a pluggable set of web services to those files.

II. Motivation

Perry *et al*. [5] described four aspects of mobile work: the role of planning, working in "dead time," accessing remote technological and informational resources, and monitoring activities of remote colleagues. In this work, we focused on reducing the need for planning and streamlining access to remote documents and services.

As part of our investigation into mobile workers' document needs, we interviewed several people who engaged in mobile work, and looked at how hotel business centers and Kinko-FedEx stores support such people. This investigation motivated some of the design features of our system, and led to the creation of a new service for document sharing.

We conducted some informal interviews with several people who worked while away from the office, and with a few people responsible for providing mobile workers with computer and printing facilities, such as business center operators in big hotels. We were focusing on people traveling without much support equipment (perhaps not even a networked laptop) who need to rely on unfamiliar networks and hardware to do their work. This is in contrast to other kinds of mobile work, such as people traveling between fixed offices, or working out of the trunks of their cars [3].

We asked people about their document use practices, and about situations in which they found their support infrastructure lacking. Several people reported using VPN to access to documents on the home network, but this solution has several limitations. One person we interviewed made long trips to a satellite office to print documents because her home printer didn't work well enough. Another reported that his customers' networks often preclude VPN access, and he has to either request that someone at work email documents to his personal account, or he has to go back to his hotel (which has a less restrictive network) at lunch to get the needed documents.

We also interviewed several business center operators and staff about patterns of use of their equipment and services by people traveling on business. One hotel business center

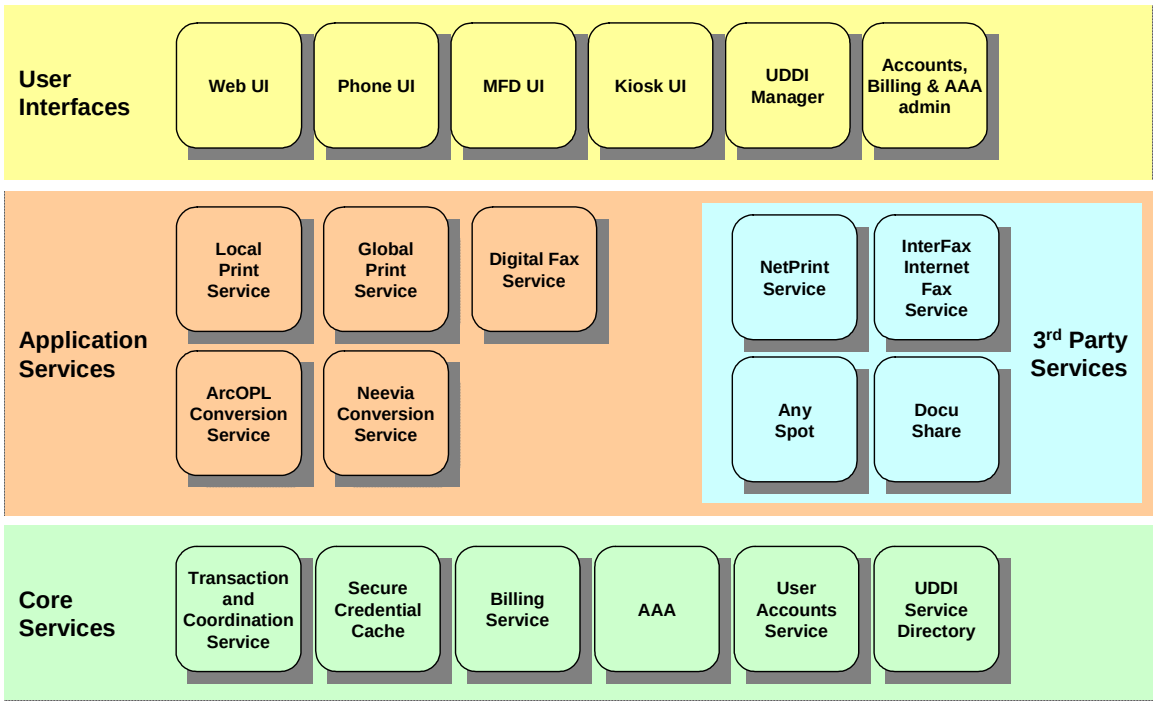

Figure 1. System Architecture.

operator we talked to reported that lawyers would often receive contract information by fax or e-mail, sign the hard-copy, and send it back via courier. One of the factors that make it less likely that they fax back the signed document is the poor quality of a twice-faxed document, and, in some cases, the size of the document.

Business center staff reported, and we observed first-hand, increasing use of e-mail to receive documents for printing. Some hotel business centers operate their own e-mail address to which guests can send their documents; a hotel employee prints the attached documents manually, and the guest can pick them up later, or have them delivered to the room. E-mailed attached documents are becoming a more common replacement for faxing, as the document quality is considerably better and as printers become more widely available.

A. *Comparative Scenarios*

We illustrate the capabilities of our system through the following scenario, described in terms of Fax, e-mail, and DoKumobility technology. Consider Hubert, who is traveling on business: Sally, his real estate agent, needs to get a document to him for his signature by the end of the day.

Facsimile. She faxes the document to his home fax machine, but misses him, as he's already left for the airport. She then decides to fax it to the airport lounge, and leaves a message on his cell phone to that effect. Unfortunately, due to longer than usual lines at the security checkpoint, Hubert doesn't get a chance to stop at the lounge, leaving the document stranded yet again, this time in a potentially insecure location. Finally she sends the document again to the hotel at which Hubert will be staying, where it sits for several hours while Hubert is changing planes in Dallas.

E-mail. Sally sends the MS Word document (which includes MS Visio illustrations of the proposed layout) to Hubert's personal e-mail address. He checks his e-mail during a layover in Dallas, but cannot view the diagrams because MS Visio is not installed on that machine. He sends a reply to Sally to create a PDF for him, but the conversion takes longer than expected, and he has to leave to catch his connecting flight. Finally, at the hotel, he logs onto the public internet terminal (despite his concerns about typing his e-mail password into a machine that may have keystroke logging or other malware) and is able to print out the PDF document.

DoKumobility. Sally sends the document to him by Confirmed Digital Fax, and the service sends an SMS message to Hubert's cell phone informing him of a waiting document. Hubert receives the notification when his plane lands, heads over to the airline lounge during his layover in Dallas, and uses his cell phone to route the incoming document to the club's regular fax machine for a preview copy. He reviews the document on his connecting flight. When he arrives at his hotel, again using his cell phone, he routes the Word document to a DoKumobility-aware MFD[1] at the hotel's business center to print out a high-quality color copy of the document, signs it, and uses the MFD to send a Confirmed Digital Fax back to Sally.

This example illustrates some of the capabilities of the DoKumobility system – including the use of multiple devices, integration of 3rd party software, document routing, and single sign-on – compared with conventional document transmission technologies. Current use of the fax machine, while common for travelers is fraught with difficulties ranging from wrong

[1] Multi-Function Device, a copier/printer/scanner combination

numbers to unreliable transmission to poor quality to issues of poor timing between the sender and the receiver.

Although e-mail solves some of the authentication and location-independence issues, it has some problems as well. Business office staff reported several problems people had in accessing e-mail, including overzealous spam filters, mistyped e-mail addresses, and incompatibility of attached documents with software installed on the business center machine. Some of these observations and experiences with document sharing on the road led us to create the Confirmed Digital Fax Service described detail in section VI.D.

III. System overview

We created the DoKumobility system that allows people to access documents and services remotely without need for up-front planning typical of document repository solutions. The DoKumobility system includes core services, a number of application services, and several user interfaces (see Figure 1). Core services are those services that are required for the system to function; removing any one of them prevents the system from working. Application services are web services that perform useful tasks for users, whether selected explicitly by a user or through composition with other services. Finally, the user interfaces – web, phone, printer, and a windows application – allow the user to interact with the services.

Services and applications use UDDI to discover the locations of other services. A user authenticates himself through an application user interface, which causes the AAA service to establish a login session. The application then triggers an orchestration that corresponds to the requested action. The Coordination Service monitors the orchestration through a transaction mechanism and debits the billing system appropriately.

A consequence of DoKumobility's document-centric service oriented architecture is that many pieces of functionality overlap, and indeed leverage, existing services. On-line services like PrintMe [10] or eFax [13] provide alternatives for the printing and faxing capabilities in DoKumobility, and could be incorporated into the architecture (by adding appropriate wrappers) to further increase the availability of printers to the system. Used natively, however, these types of document-oriented internet services are limited to the documents either uploaded before-hand, or available at the user's device for uploading. This sort of planning may not be possible or desirable in many situations. In contrast, DoKumobility facilitates the coupling of arbitrary document sources – including files stored on computers in the home or office – with these types of services.

Conversely, systems that act as document sources such as BSCW[2] or GotoMyPC [8] provide the ability to access documents wherever you go, but only provide a very fixed number of supplementary services (if any), or can only be accessed through desktop devices, something unsuited to a mobile workers' requirements. Wrapping such systems with DoKumobility's common WSDL interfaces enables users to access these different sources in a unified manner, providing services on-demand on any device.

The actual document source and directory interfaces provide similar functionality to the WebDAV [9] protocol, but provide it at a web service level (enabling orchestration) rather than as a lower-level network protocol. Similarly, we are operating at a level of abstraction above JINI [14], and integrate heterogeneous applications and services without regard for platforms and with no requirements on networks other than the availability of HTTPS connections.

IV. Service infrastructure

This section describes some of the basic infrastructure that underlies each service and mediates communication among services and with applications. Although the .NET IDE generates WSDL for a service from a specific host's implementation, we did not want to hard-code host names in our services. Not only would that make it hard to deploy the system to an arbitrary network, but also it would be more difficult to add new services to an existing system.

A. UDDI

Run-time composition and orchestration is accomplished by retrieving service locations through the UDDI. All services are registered with the UDDI, and each service uses UDDI to find other services in the system prior to invoking methods on them. Service types are identified through a *Software IDENT*, a string that corresponds to a particular set of WSDL declarations. For example, when an application wants to find the location of the AAA service, it contacts the UDDI with a request for the *UserAccount.1.0* service. The UDDI server returns a structure that describes the requested service. The structure includes such elements as the service GUID, its textual description, its public key, its access point, and any other attributes that may be defined for the service. The application can then use the returned access point to execute web methods on the service.

Every service must be registered with the UDDI to be usable on the system because services and applications use the UDDI server to discover other services. The base class from which all services in our system inherit defines the mechanisms for searching the UDDI for requested service descriptions.

B. Reusing library code

We also wanted to reuse utility library code between services without worrying about the mapping between WSDL and .NET namespaces. The .NET IDE creates a new namespace for each web service reference, and creates stubs for classes that are passed as parameters and return values in that namespace. For example, class Common.C1 in the service definition becomes Service1.C1 in web reference code generated for service Service1, making it impossible for the same library code handle class C1 on both the client and on the server.

To solve this problem, we created a special proxy library for each service. The proxy carries the web reference, implements an API that exposes some of the key functions of the services (to improve readability and to hide data conversion), and

manages data conversion between the service and its client (which could be another service).

Data conversion is handled through reflection: we created a utility method that takes an arbitrary source instance and a Type of a target class, creates a new instance of target and sets its public properties and fields with correspondingly-named properties and fields from the source. This technique is applied recursively to all public writable fields and properties in the target. Although this would fail for cyclical structures, in practice we apply these transformations to data that is handled by the XML serializer that writes out the SOAP body. Since the serializer cannot handle cyclical structures, our restriction does not impose further constraints. This combination of proxies and type conversion allows us to reuse code between services and clients, and streamlines web service invocation code.

V. Core Services

Core services are the glue that allows other (application) services to perform meaningful work on the user's behalf. They facilitate user authentication, orchestration, and billing. The specific services are described in the following sections.

A. AAA

Users authenticate themselves to the AAA service. We allow multiple credentials for the same account, so a user can use different authentication schemes depending on the device of choice. For example, a user may use the cell phone hardware id as the login, and a pin as a password. This service also implements a secure credential cache for storing credentials for third-party services. When a user associates a new service that contains private data with his account, the system elicits a userid and password for that service. For security purposes, credentials are encrypted with the public key of the corresponding wrapper service. When a wrapper service needs to sign into a 3rd-party service, it retrieves the credentials for the signed-in user, decrypts them with its private key, and signs in on behalf of the current user. The logic of each particular wrapped service dictates when credentials are presented.

B. Transactions

We implemented light-weight transactions to coordinate composite operations. A transaction specifies which services must perform which operations to complete a given task. The Coordination Service creates transactions at another service's request and collects notifications from participating services regarding the progress of a transaction. Transactions can have sub-transactions. The Coordination Service interacts with the Billing service to debit the user's account as operations are performed. If a transaction fails, the debit operations are reversed, and each participating service is notified to reverse the operation. Services may implement appropriate compensating operations.

C. Billing

The Billing Service manages customers' accounts. A customer is the entity that owns the account; it may be an individual or a corporation. A user is a person who interacts with system services through some application. Services create transactions to perform billable operations. A transaction consists of one or more billable operations, each of which specifies the type and amount of billable resources for which the service is charging the customer. The Billing Service maintains a list of billable resources for each service, and maintains service plans that specify the costs of each resource for each customer.

The Billing system is designed as a separate application that can handle many real-time micro-payments for document services. It has two account types, prepaid and credit: prepaid users can execute services as long as the account balance stays above zero; credit users can execute services up to the credit limit. The billing system has two customer types: individuals and organizations.

Billable resources and rates are stored in a database. Service billing rates depend on selected Service Plans. For example, printing services can be charged per page, while translation services can be charged per page, per word and/or per document setup. A discount can be applied when a transaction involves printing and translation services at the same time.

VI. Application Services

The DoKumobility system allows people to apply services to documents. A variety of document sources are available in the system, including document repositories and documents stored on particular machines. In addition, many services maintain their own document collections that represent recent use histories. A user interacts with the system by selecting a document from a source and then by applying a service to that document. The sections below describe the services involved in more details, and the User Interfaces section that follows describes the application interfaces we've developed that mediate access to the services.

A. Document sources

A Document Source is a web service that represents a collection of documents. We have implemented Document Sources for document repositories such as Fuji Xerox's DocuShare [7], for file systems on local machines by using AnySpot,[2] and for other web services. The Document Source API includes methods for listing the contents of a collection or folder, for getting the metadata corresponding to a particular document, for adding a document to a collection, and for retrieving a document. Document Sources can charge the user for storing and retrieving documents, depending on the active service plan.

There are two kinds of Document Sources in the system, public and subscription. Public Document Sources represent unique services on the network (such as the Printer Directory) that manage their own access control. Thus the Printer

[2] AnySpot is a remote file access utility developed at FXPAL. It allows, among other things, for users to access documents stored within the FXPAL firewall while away from the office over HTTPS connections.

Directory Document Source will show each user's print history when that user logs in. Since there is only one Printer Directory service in the system, there is no need for users to configure access. Subscription Document Sources, on the other hand, require each user to specify login credentials that will authenticate that user to the 3rd party document source service.

While these wrapper services give access to documents and to metadata, they do not, in general give full access to the 3rd party service being wrapped. For example, users need to use the existing AnySpot interface to configure the Spots.

B. Printing

Printing is one of the key considerations that motivated the design of this system. We wanted to create a system through which a mobile user could print any document on any available printer. As described above, documents could originate on any document source available to the user.

Printing a document involves the orchestration of three services: the Printer Directory service, a Document Conversion service, and a Local Print service. Unlike normal web service orchestration, in which all services are visible to each other, printing poses a particular challenge because Local Print services typically run behind firewalls, and are not accessible directly by other services. We get around this limitation by having the Local Print service poll the Printer Directory service to request print jobs.

Thus, the printing process consists of the following steps: a user selects a document to print, and a Local Print Service that corresponds to the desired print location, then selects a specific printer managed by that Local Print service, and submits the job. The Printer Directory service identifies a Conversion Service (see below) that can convert the input document type to the print description language (e.g., PostScript) and invokes that service to convert the document. When conversion completes, the Printer Directory service queues the converted document for printing.

At some point, the Local Print service contacts the Printer Directory service to request pending print jobs. For each queued job, the Local Print service transfers the document to be printed from the Conversion service, prints it, and notifies the Coordination Service when the print succeeds (or fails).

This separation of job queuing, conversion, and actual printing, coupled with a distributed network of document sources, allows DoKumobility users to print documents stored behind one firewall on a printer behind another firewall without modifications to either organization's security policies. It also charges the user for the print job only if the job actually succeeds, and can remunerate the Local Print service provider for the consumed resources.

C. Conversion

Document Conversion services are used to transform documents from one format to another. Although there is nothing print-specific about the definition of the Document Conversion service, the only existing implementations of this service are designed to convert documents for printing.

We employ two conversion services in the system: one that generates PostScript, and another that generates the ART-X format used by some Fuji Xerox printers. In both cases, the services are wrappers for existing conversion software; the wrappers create web services interfaces that implement our document transport and transaction services.

Document Conversion services are selected based on the MIME type of the document to be converted and on the MIME type of the Print Description Language (PDL) that should be produced. Service selection is based on metadata stored in the UDDI: the first document conversion web service that matches the input and output document types returned by the UDDI is used.

D. Confirmed Fax

The fax machine is a ubiquitous technology in the business world. For all its popularity, however, it suffers from several important problems:

1. Faxing is tied to devices, not to people. This means that to send a document to a person, you must know the phone number of the machine that is accessible by the intended recipient. If that person is traveling, determining the right fax number may be quite difficult.

2. Fax machines lack adequate feedback regarding the success of the transmission. The receiving fax machine may not print the sent document (because it is out of toner, for example) or may skip printing pages, without alerting the sender. Similarly, if the sender's machine mis-feeds the document, it may not be transmitted as intended.

3. Faxing is not secure. In many organizations, fax machines are housed in public spaces where they are accessible to many people. One person may inadvertently (or intentionally) pick up someone else's fax, and the intended recipient will not know that a document had arrived.

4. The recipient of the fax may deny its existence. Because the sender lacks adequate feedback on the outcome of the process, it is common for recipients to deny having received the document.

Despite these shortcomings, the notion of sending a document to another location is invaluable to many business processes. We introduced a new service into our system that is similar in its uses to fax transmission, but one that sends documents to a particular person rather than to a device, that notifies the sender that a document has been sent and when it has been read, and one that requires the recipient to authenticate himself prior to receiving the document. The approach is similar to sending e-mail attachments, but, in contrast to e-mail, notifies the sender when the recipient has accessed the document as part of a transaction. Unlike e-mail where the recipient's software agent has a choice about sending a read acknowledgement to the recipient, the Confirmed Digital Fax service guarantees notification.

Finally, the fax-like nature of the transaction is reinforced by the integration of this service with an MFD such as Fuji Xerox's f450 printer/copier. A mobile worker can scan a paper document, and have the Confirmed Digital Fax service deliver

an electronic copy of it to the recipient; the recipient can view it on the screen, print it to paper, or route it to some other service (including an analog fax!).

E. Third-party services

One of the strengths of a web services-based infrastructure is the ease with which 3rd party services can be incorporated. We built wrappers for two "printing" services for mobile workers: NetPrint [12] and InterFax [11].

NetPrint. NetPrint is a joint venture between Fuji Xerox and 7-11 Japan that allows a person to upload a document to an online repository, and then to print that document from any 7-11 by typing in the six digit code for the document and paying ¥30-120 (depending on the size of the paper and options such as color prints).

We connected the NetPrint into our system with a service wrapper that receives a document from a Document Source, submits it to NetPrint, and sends a notification with the document code to the user's cell phone or e-mail.

InterFax. InterFax is a company that provides a web service through which electronic documents can be sent via analog fax to any fax machine in the world. Our wrapper service submits documents to InterFax, and debits the user's account for the cost of faxing the document.

Using these 3rd party services allows a mobile user to use the existing printing infrastructure while on the go.

We can illustrate the interaction of the various services by stepping through the sequence of service calls required to print a document through the NetPrint 3rd party service; other application services are integrated in an analogous manner. Figure 2 shows the sequence of interactions between DoKumobility components and the NetPrint application. NetPrintWrService is a wrapper around the NetPrint component that takes documents from a DoKumobility document source, transmits it to the NetPrint web site and processes the response to obtain the id for the printed document. The Coordination Service manages the transaction and causes the billing service to post pending charges to the user's account. When the transaction completes, the Coordination Service invokes the Billing Service to debit the customer's account.

VII. User Interfaces

Mobile workers may use their laptops, PDAs, or cell phones to access electronic information, as the situation permits. One of the design goals for DoKumobility was to provide access to documents and services through a variety of devices to support this opportunistic work practice. Here, we describe three interfaces that make up the DoKumobility suite. Through these

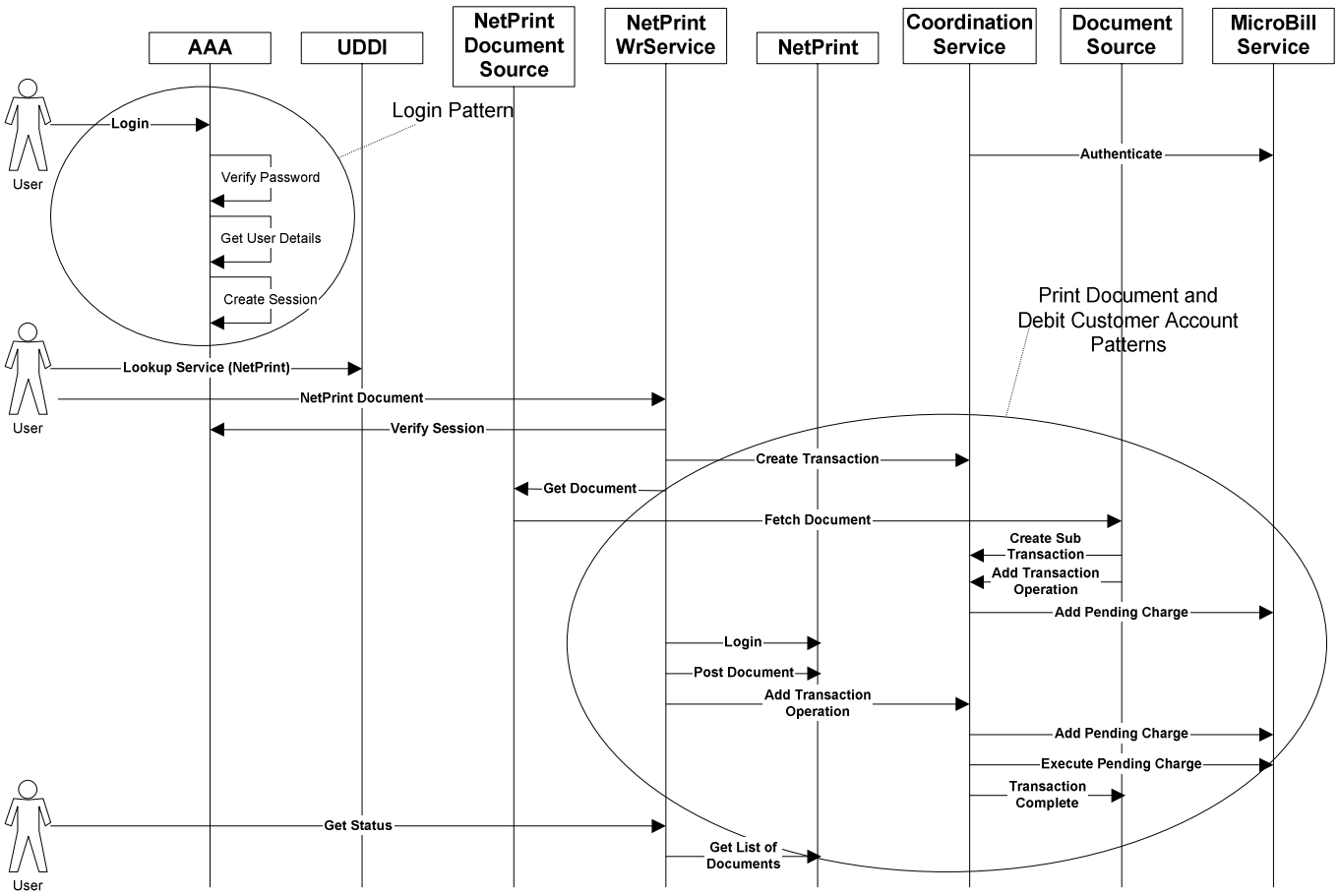

Figure 2. NetPrint Interaction diagram.

interfaces, users are able to retrieve their documents, and print, fax, share, or view them.

A. Web

When a web browser is available, mobile workers can access documents and services through the web UI. This interface (Figure 3) lists the various document sources on the left, shows the contents of the selected source in the middle frame, and the details of a selected document in the bottom pane.

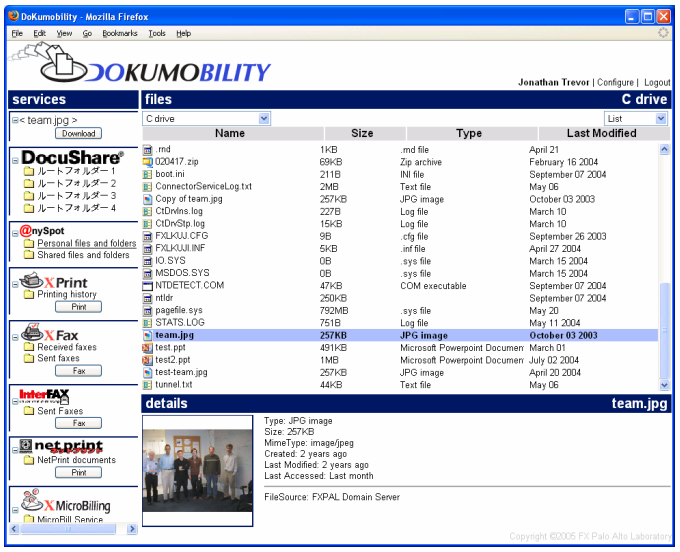

Figure 3. DoKumobility on the web.

The left column ("Services") in example screenshot shows several DocuShare collections to which the user has subscribed, two AnySpot locations, the printing service (XPrint), the Confirmed Digital Fax (XFax), the InterFax service, the NetPrint service, and the Billing service. XPrint, XFax, InterFax, and NetPrint services show command buttons that will act on the document selected in the "files" pane. The "details" pane shows a preview of the selected document along with some metadata.

When a document is selected, services that can act on that document display buttons (e.g., "Print", "Fax", etc.) that trigger the corresponding actions on the selected document. Interaction takes place through dialog boxes (e.g., Figure 4).

B. Printer

We extended the interface of Fuji Xerox's DocuCentre f450 printer/copier to connect to our web services, and to perform printing and scanning functions on documents. By selecting appropriate job templates from the printer console, users can perform the following functions: they can scan a paper document and route it to themselves or to another person, they can scan a document and fax it, they can print any document, including incoming confirmed digital faxes. The integration of the device's scanning and printing capabilities into the DoKumobility framework allows users to treat paper and electronic documents interchangeably, switching from one to the other as appropriate to the circumstances. This not only streamlines interaction with the system, but also allows the DoKumobility user to work with others who may not have access to the DoKumobility platform.

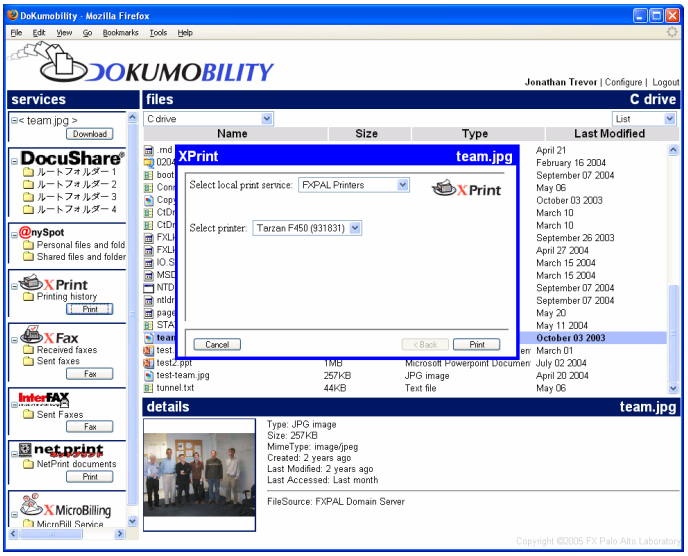

Figure 4. Printing a file.

C. Phone

When traveling, a cell-phone based interface to DoKumobility is a viable and essential substitute for the more capable infrastructure available in the office and desktop-oriented UIs. The phone interface provides access to the same documents and services through a web-based interface which is more suitable for such a small form-factor.

Figure 5. Cell phone interface.

Using the cell phone interface, users can select a document from any source and then apply the same services – faxing, printing, sharing, *etc.* – available in the web interface to the selected document (Figure 5).

D. Kiosk UI

We also built a WinForms user interface to DoKumobility that replicated much of the web-based user interface and interaction style, but augmented it in one important way. We were concerned about security when logging into third-party

computers. Whereas DoKumobility uses HTPS to transmit data between clients and services, and uses public key encryption to store passwords for 3rd party services, it does not provide control over keystroke logging on machines that a user may have access to while traveling.

We decided to create a more secure interface for accessing DoKumobility services by authenticating through a trusted device (e.g., a user's cell phone), and then sharing the created session with the less trusted public terminal. Session sharing is implemented as follows:

1. The user authenticates himself to the cell phone application by providing a PIN.
2. The cell phone application authenticates the user, creates a session object for the user.
3. The user requests a session barcode, and the cell phone application creates a two-dimensional barcode representing the session.
4. The user then shows the barcode image to the kiosk webcam, which reads the session ID and uses that ID to connect to web services on the user's behalf.

The advantages of this approach are that the user uses a trusted device for authentication, reducing the security requirements (PIN vs. user id and password) and increasing convenience. At the same time, he does not expose his credentials to malware potentially present on public machines, but is able to take advantage of the bigger screen and the less modal interface.

VIII. Lessons Learned

A. Optimizing document processing

The nature of DocuMobility promoted "documents" as one of the main types of data to flow between services – typically in the range of a few kilobytes to several megabytes. It became apparent very early in our development that moving this data between services would cause significant delays in both the user experience of using the suite of services, as well as individual operations provided by the services themselves. It also became apparent that while many of the services (e.g. the print directory service) consumed or took documents as input parameters, they did not process the document data itself – merely requiring the document's meta-data (id, name, mime-type, size, modified date, and so on).

Therefore we created a mechanism that separated the document content from the documents meta-data, only fetching the actual content if the service required it. Documents are represented using XML. This XML structure contains a set of pre-defined common document fields (name, MIME type, size and so on), a name-value bag that could be used by any service to store additional information about that document in addition to the common fields (such as the number of pages in a paginated document), and a reference to which document source service actually has the document content.

Web services receiving such a document description can perform processing using this meta-data and fetch the actual contents locally (to disk) by simply looking up the URL of the document source service and invoking a method.

In our current design this actual transfer is performed using a simple HTTP request, although a DIME based method would also be appropriate.

B. Programming interfaces

As we developed services, it became clear that all services had some common methods and properties, including the service ID, public and private keys, a description, a service type, etc. Furthermore, some services shared functionality such as mediating access to collections of documents. Our initial attempt at using the C# inheritance mechanism to factor the code proved ineffective due to problems with inheritance of web method attributes. To get around this limitation, we implemented the following design pattern.

For each Web Service Interface (WSI) that we wanted to share among web services, we created a regular (non-web service) interface (I) that specified appropriate methods, and a regular C# class (C) that implemented them. The C# class also assigned WebMethod attributes to appropriate methods. Then, for each WSI, we created a C# web service project that specified the corresponding WSDL. Finally, we added a new web service to each non-interface Web Service (WS) project that needed to implement a WSI, and changed the inheritance of the newly-created class to C (the non-web service class), as shown in Figure 6.

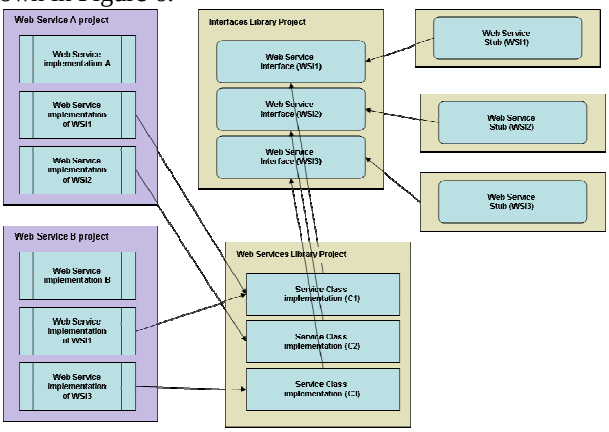

Figure 6. Web Service interface architecture.

We created these interfaces for those services in our system that had multiple implementations of the same interface. These include the Generic Service, the Document Source Service, the Document Directory Service, and the Conversion Service, as shown in Table 1. *Table 2* shows services with the interfaces they implement.

Generic Service. Each web service implements the Generic Service interface. This WSDL specifies methods and structures for accessing service descriptions and for registering the service with UDDI. All web services in our system implement this interface – it is in effect the base class of our service hierarchy. This service consists of four methods: GetRegistrationInfo, TransactionCompleted, GetServiceAccessKey, and VerifyServiceAccessKey.

Web Service Interface (WSI)	Interface (I) Testbed.Service.Interface namespace	Class (C) Testbed.Common.UtilService namespace
Testbed.Service.Interface.IGenericTestbedService	IGenericService	GenericService
Testbed.Proxy.Core.IDocumentSource	IDocumentSource	DocumentSource
Testbed.Proxy.Core.IDocumentDirectory.IDocumentDirectory	IDocumentDirectory	DocumentDirectoryService
Testbed.Proxy.Application.IDocumentConversion.IDocumentConversion	IDocumentConversion	DocumentConversion

Table 1. Web service interfaces

GetRegistrationInfo returns a structure that characterizes the service in the UDDI. It is called by the administrative user interface through which services are registered with the system. The method returns the software ident, its name, public key, and other metadata. It is called by the administrative interface when the service is added to the system.

TransactionCompleted is called by the Coordination Service to notify services about the outcome of the transaction in which those services participated. The method takes two parameters: a transaction ID and flag indicating success or failure. If the flag is true, the service should finish up any processing for that transaction; if it is false, it should execute a compensating action if required.

GetServiceAccessKey and VerifySericeAccessKey implement a security mechanism by which a service being invoked by another service can verify the identity of the invoking service. The caller service requests a secret (newly-generated GUID) from the called service by calling GetServiceAccessKey with the caller's unique public ID. The called service uses that ID to find the caller's public key in the UDDI, and uses that key to encrypt a secret that it returns to the caller. The caller decrypts the secret, and calls VerifyServiceAccessKey() on the called service. The called service compares the returned secret with its copy, and either marks that service's HTTP session as authenticated or not. Subsequent communication during that HTTP session does not require additional authentication.

Document Source. The Document Source interface specifies methods for obtaining document metadata and for fetching and storing document contents. Contents are retrieved using HTTP GET requests and stored using HTTP POST requests.

Document Source implements four methods: GetToken to return the metadata for a document given an ID, GetIcon to return a PNG file that represents the file type, GetThumbnail to return a thumbnail image of the document, and SetTransaction to set the ID of the transaction for the given HTTP session. Different web services implement this interface in different ways. The AnySpot wrapper, for example, merely remaps our web methods onto the (approximately) corresponding AnySpot Service methods and performs parameter type conversions. Services wrappers for services such as NetPrint and InterFax maintain a document cache of documents transmitted to the wrapped service for auditing and history purposes.

Note that a document source doesn't implement a directory structure, but rather stores all documents in a flat space. It maintains a database entry for the metadata for each document, but does not represent any hierarchical relationships among documents. This function is handled by the Document Directory Service, as described below.

Document Directory. The Document Directory interface defines the API for a hierarchical directory structure that is used in conjunction with a Document Source. It implements GetRootFolder to return the root for the given service, GetParent to return the Token representing the container of a document or folder, GetParents to return an array of Token that represents the containment hierarchy for a given document, GetFiles to return an array of Token that is the contents of a folder, and GetFilesEx to return an array of files that match certain conditions.

Directories are used by some services to segment documents

Service	Type	IGeneric Service	IDocument Source	IDocument Directory	IDocument Conv.
AnySpotWrapper	App	•	•	•	
ArcOPLConversion	App	•	•		•
Coordination	Core	•			
ConfirmedDigiFax	App	•	•	•	
DocuShareWrapper	App	•	•	•	
InterFaxWrapper	App	•	•	•	
LocalPrint	App	•	•		
MicroBill	Core	•	•	•	
NetPrintWrapper	App	•	•	•	
Notification	App	•			
PrintConversion	App	•	•		•
UserAccount	Core	•			
BPELInterface	Core	•	•		

Table 2. Services and their web service interfaces.

by user. The Confirmed Digital Fax Service, for example, has two sub-folders in its root folder. One sub-folder holds the incoming documents, and the other holds the outgoing documents. The contents of each folder are filtered by the identity of the logged-in user. This scheme allows the user interface to maintain the familiar illusion of a file system without the need to manage individual folders. Services that do not have an explicit manifestation in the user interface, such as the Conversion Service, may not implement a Directory Service at all.

Conversion Service. The Conversion Service WSDL defines an API for converting documents from one MIME type to another. Each service that implements this interface also registers with the UDDI the pairs of MIME types that it can convert. It is the caller's responsibility to find the right service instance for the required conversion. The conversion service is used by the Printer Directory Service to produce the appropriate Print Description Language document. One service in our network is used to produce Postscript, another is used to produce Art-X, a format used by some Fuji Xerox printers.

C. HTTP cookies and proxies

The IIS web server uses session IDs transmitted as HTTP cookies to identify data that must persist across web service invocations. The server maintains a hash table based on the session ID and allows services to store and retrieve data appropriate to their session. Since an application service may implement multiple WSDL interfaces (as described above), it requires the corresponding number of proxies. Thus we have the GenericServiceProxy class, the DocumentDirectoryProxy class, etc. This means that a client may potentially hold onto several proxies to the same service, one for each relevant aspect. The problem with this approach is that unless the various proxies to the same logical service use the same session cookie, the various instances of the invoked service will not be able to share data because their state will be associated with different sessions. Thus we needed to implement a proxy manager that caches session cookies and assigns them to proxies based on the URL of the service being accessed.

IX. BPEL

Our initial system architecture was essentially a peer-to-peer service network with distributed, coordinated transactions that relied on each participating service to create transactions or to perform actions within the scope of a transaction and to bill for computation.

This architecture required each service to be aware of the transaction and billing system. Thus it was not possible to include an arbitrary service in the network without making extensive modifications to its source code. Furthermore, the same service could not be used differently in different contexts (e.g., charging for service in one case, and not charging in another).

Our architecture also lacked a standardized process status mechanism through which applications and services could monitor on-going processes. Each application implemented some sort of a polling mechanism for each service whose state it needed to monitor, resulting in extra code and interface inconsistencies.

For these reasons, we decided to re-implement our system architecture to use a BPEL engine for service orchestration. We chose the ActiveBPEL BPEL4WS engine to handle run-time orchestration and used the ActiveWebflow Professional Eclipse Plugin tool to build BPEL scripts. We built one script for each application function (e.g., printing, scanning, faxing, etc.). Each script creates a new process with a process ID, and handles service orchestration (when multiple services are involved), billing, and error handling. Figure 7 illustrates a sample BPEL script structure, this one for orchestrating a notification of an incoming Confirmed Digital Fax. The orchestration involves locating the AAA service, authenticating the user, locating the billing service, adding a pending charge, locating the notification service, performing the notification, and finalizing the transactions. It also includes fault handling for the constituent operations.

We also implemented PBELInterfaceService, a helper web service, that makes the status of ongoing transactions available to user interfaces and to other services, and implements some data transformation and UDDI functions used by the BPEL scripts. This service also implements the IDocumentSource web service interface that handles document access for the scripts that require it.

By orchestrating services through BPEL scripts, we reduced the code and complexity of application web services and made it easier to add new ones without extensive additional coding. Furthermore, each service could be used in multiple billing situations without additional changes.

Unfortunately, not all of our architectural patterns could be implemented directly in BPEL scripts. BPEL4WS is an emerging standard that will shortly reach its second revision. Version 1 left much scope for interpretation by early implementers and adopters. We encountered two main types of limitations when we ported our system to BPEL4WS: fault handling, and flow of control.

ActiveBPEL implements cascading fault behavior: a fault experienced in a parent scope will fire the fault handlers in all its child scopes. This led to some less than desirable workarounds in practice.

The ASP.NET WSDL generator does not implement the Fault type message to transmit exceptions back to the caller. This message is normally exposed in the BPEL4WS fault handlers. Its absence in the WSDLs generated by .NET prevented us from reporting detailed exception information to the end user. We expect the next version of the IDE to have better support for fault handling.

Finally, BPEL4WS contains limited constructs for controlling execution flow. Since there are no procedures or functions, code re-use can only be achieved by one BPEL script calling another via SOAP messages within the script engine or from engine to engine. It is anticipated that this would have

negative implications for performance in a real world implementation. Of course a sufficiently powerful editing tool could compensate for this limitation by operating at a different level of abstraction in which code constructs can be reused.

Figure 7. Sample BPEL script visualization showing a script to notify a user about an incoming Confirmed Digital Fax.

X. Conclusions and Future Work

We set out to build a web services framework for bringing documents and services to mobile users. In addition to the framework, we constructed a suite of services for printing and sharing documents, and created a unifying structure for integrating arbitrary 3rd party services. Billing and transactions were incorporated from the beginning in a way that encourages 3rd party service integration and organic network growth. The variety of application interfaces integrated into DoKumobility provide users with opportunistic access to documents, using the device at hand, rather than restricting them to specific platforms. The ability to route documents from one service to another, coupled with secure single-sign-on capability, creates a seamless user experience that focuses on documents rather than on discontinuities induced by unrelated services.

But this is just the beginning, and much remains to be done. Our infrastructure makes it easy to integrate services and to pass documents among them. It is not clear, however, how best to approach dynamic service integration in the user interface. The system as it stands can allow new services to be integrated without recompiling the code, but what of the user interface?

At present, to integrate a new service into DoKumobility applications requires us to change application source code to create appropriate screens. Although this is not difficult, it does not scale as an approach, particularly as the number of application services (e.g., Translation, Summarization, etc.) increases. We would like to automate interface composition to reflect the set of services a user chooses to apply to selected documents. Thus the system should determine automatically which services may be applied at any point during a user's interactions, allow the user to select some subset of these services (e.g., generate a document summary and then print it), and prompt for the parameters that are necessary for the selected services.

Another open issue related to interaction is how integrate user interaction with BPEL processes. Let's say that as the BPEL engine is executing a process, a service decides it needs input from the user, such as re-authentication because the session expired, or some other piece of information that was not anticipated when the job started. Once possibility is to roll back the transaction, get the required information, and repeat it. But this is wasteful and may require awkward state representation in the user interface. We would like to explore an interrupt-driven model through which services can express their information need in some abstract way that applications understand, pause themselves while waiting for the application to complete the missing information, and then resume computation. This ability to reach out the application and to initiate interaction with the user moves the system further from a client-server and closer to a peer-to-peer architecture.

Acknowledgments

The authors thank Jonathan Trevor, Jim Vaughan, Paul Murphy, and Sunil Sharaf for their contributions to DoKumobility code, Sara Bly and Pernilla Qvarfordt for the help with the field work, Kate Withey for the graphics, and Jim Baker for encouraging and sponsoring this work.

References

[1] OASIS Web Services Business Process Execution Language Technical Committee, "WSBPEL Version 2.0 Working Draft", http://www.oasis-open.org/

[2] R. Bentley, T. Horstmann, K. Sikkel, and J. Trevor. "Supporting collaborative information sharing with the World-Wide Web: The BSCW Shared Workspace system", In *Proceedings of 4th International WWW Conference*, Boston, MA, USA. Dec. 11-14, 1995.

[3] E.F. Churchill, A. Munro. " Work/place: mobile technologies and arenas of activity" *ACM SIGGROUP Bulletin*, XXIII (4), pp. 3-9, 2001.

[4] M. Lamming, M. Eldridge, M. Flynn, C. Jones, D. Pendlebury, "Satchel: providing access to any document, any time, anywhere" *ACM TOCHI* VII (3), 2000.

[5] M. Perry, K. O'Hara, A. Sellen, B. Brown, R. Harper. "Dealing with mobility: understanding access anytime, anywhere" *ACM Transactions on Computer-Human Interaction* VII (4), pp. 323-347, 2001.

[6] http://www.puregrain.com/mobiledoc/index.html

[7] http://www.xerox.com/ds

[8] https://www.gotomypc.com/

[9] http:// www.webdav.org/

[10] http://www.printme.com/

[11] http://www.interfax.net/

[12] https://www.printing.ne.jp/

[13] http://www.efax.com/

[14] http://www.jini.org/

Gene Golovchinsky received a BS ('87) in electrical engineering from UCLA (Los Angeles, CA, USA), and an M.Sci. ('93) and Ph.D. ('96) from the Department of Mechanical and Industrial Engineering (Human Factors) at the University of Toronto (Toronto, Ontario, Canada).

He has worked at Kaiser Electronics, IBM, GMD-IPSI in Darmstadt, Germany, and is currently employed by FX Palo Alto Laboratory, Inc. in Palo Alto, California. He has published numerous articles in the area of hypertext, information retrieval, and human-computer interaction, and holds 15 patents.

Dr. Golovchinsky is a member of the ACM.

Gerry Filby has worked as a software engineer for LGI Photo Agency (now Corbis/Microsoft), GTE Services Corp., Ascend Communications (now Lucent Technologies), and Click Commerce/Allegis Corp., and as a Chief Software Engineer for Validigm Corp. He is currently a contract software engineer at FX Palo Alto Laboratory, Inc.