Magi: An Architecture for Mobile and Disconnected Workflow

GREGORY ALAN BOLCER
Endeavors Technology Inc.

Automated workflow systems do not yet adequately address information mobility or tasks disconnected from the rest of a business process. While the current Web infrastructure lends itself to disseminating information, it also inhibits the decentralization of process definitions and their execution required by workflow systems. The Micro-Apache Generic Interface—based on the popular Apache HTTP server—is an architecture that can overcome some of these limitations.

Although much of today's office work is cooperative, the automation of workflows governing the handoff and routing of business documents and data are often hard-scripted into supporting software. This tight coupling of processes and supporting applications limits the modeling, deployment, and execution of work. These shortcomings are compounded by changes in organizational structures to accommodate dynamic, geographically dispersed workgroups as well as telecommuting, outsourcing, and an increasingly mobile workforce.

Although some Web-based systems have increased the accessibility of guidance and automation workflows, little support exists for cross-platform interoperability—and even less for nontraditional computing platforms such as smart phones, personal digital assistants (PDAs), and other Internet-enabled devices.[1] The result is limited e-business information access and activity coordination, particularly for mobile workflow participants or even those who are just away from their desks.

The Micro-Apache Generic Interface (Magi, http://magi.endeavors.org/) is an open-architecture framework that explicitly addresses e-business messaging and deployment issues across a broad range of computing platforms. Magi is a superset of the Apache Project's protocol modules and a subset of its Web-server technologies (see the sidebar "The Apache Project"). The Magi architecture has two key components: the micro-Apache HTTP server and the extensible generic interface.

This article begins by looking at some issues motivating the development of this architecture, followed by a description of its components and an example scenario. It concludes by comparing Magi with other architectural frameworks and summarizing its current status.

LIMITATIONS OF HTTP

HTTP has served as the cornerstone protocol for the World Wide Web since 1990. Because it is easy to parse and extend, HTTP can be easily integrated with distribution mechanisms of many other tools. For these reasons, HTTP has become the de facto messaging and interoperability format for numerous Web-enabled devices.

 1089-7801/00/$10.00 ©2000 IEEE

However, HTTP does not provide universal communication. In fact, as currently implemented, HTTP actually inhibits the free flow of information between clients and servers. There are three specific problems that limit its use for mobile and disconnected workflow.

No Event Backchannel

HTTP calls are initiated by a client to a server. Information that frequently changes is typically stored on a server and is dynamically generated upon a client request. Updates require client polling of the server for new information. This can be accomplished using a "refresh" HTML tag embedded in the Web content or an HTML page wrapper around some non-HTML content.

This model severely limits return events to clients. Information is updated only when the client can determine that the update is needed. Even if the server sends e-mail notifications of an event, the client must initiate an intermediate SMTP-compliant server call supporting POP or IMAP.

Routing and Naming Conventions

One reason backchannel event notification is so difficult is that clients typically do not follow Web server naming conventions. Web servers often use names that users can easily identify, such as apache.org or yahoo.com. The default behavior in most Web clients is to resolve these names to the Web server that responds to HTTP on port 80 on the host machine named www in, for example, the apache.org top-level domain. On the other hand, there is no mechanism by which the host machine can initiate a connection or transfer information to a named client without a request. Client naming requires a generative scheme and succeeds only if the client identifies itself.

Most Web servers can log a client's environment variables and IP address; however, typical clients are not configured to respond to server-initiated connections. Even if a client were configured to respond, there is no guarantee beyond a short half-life that a server-initiated connection will succeed. The call may fail because the client disconnects, or even worse, because another client is assigned its dynamic IP address, causing the message to go to another client altogether.

Nonubiquity

Despite claims to the contrary, another problem with the Web as a communication infrastructure is that HTTP is not ubiquitous. HTTP clients typically reside on desktop computers, and access to information is difficult when users are out of the office or even just down the hall. Although many mobile clients such as smart phones and PDAs now support wireless access to the Web from any location, their ability to access and display information quickly is limited.

The Apache Project

The Apache Project is a collaborative software development effort, founded in 1995, to create a robust, commercial-grade, featureful, and freely available source-code implementation of an HTTP server. Eight programmers interested in improving the original NCSA Web server built by Rob McCool found each other on the Net and started exchanging ideas, changes, and fixes.

The project is now jointly managed by the Apache Group—volunteers located around the world who use the Internet and the Web to communicate, plan, and develop the server and its related documentation. In addition, hundreds of users have contributed ideas, code, and documentation to the project.

The Apache server is used in over 57 percent of publicly available Web sites—more than twice that of its nearest competitor. Apache is extremely well known in Web development circles, but its success is sometimes overshadowed by other high-profile open-source projects, such as Linux. However, the project's success speaks volumes—particularly last year when IBM dropped its own Web server in favor of using Apache in its products.

To learn more about the Apache Project, see http://apache.org/.

WIRELESS WORKFLOW AND THE WEB

Unlike other wireless approaches, HTTP has the ability to scale from devices to desktops. HTTP, and thus Magi, is content neutral. This means that Wireless Application Protocol's (WAP) wireless markup language as well as other formats can be used with Magi while maintaining interoperability with the rest of the Web. Because a Magi server can run on the device, the same protocol can be used independent of where the message is initiated, and the devices themselves become named destinations.

Internet-scale describes a software system that can grow beyond a predictable and enumerable set of people, devices, or agents and still maintain interoperability. The Web is a perfect example of an Internet-scale software infrastructure,[2] but it trades scalability for "full-featuredness" and lacks some of the primitives needed to provide rich collaboration and coordination support. To circumvent this limitation, traditional workflow vendors that want to deploy

their wares on the Web typically wrap an HTTP server around a database-centric workflow engine and allow thin-client access to data and tasks. Efforts to scale beyond a centralized server thus require changes to installed clients such as specialized plug-ins or complex configuration settings, which limit their interoperability and ability to communicate across organizational boundaries. E-business processes that can be deployed and executed elegantly in a controlled and customized environment do not map well to the simple but scalable Web infrastructure.[3]

In a perfect world, switching contexts would be as easy as sweeping papers into a folder.

Even more vexing is the problem of how to scale down these systems to support mobile and disconnected activities better. Spotty connections, lack of bandwidth, and low processing power in handheld devices make support of even whittled-down HTML interfaces to these workflow engines untenable for most mobile workers.

Three other complications deserve special attention.

No Store-and-Forward Event Queuing

An advantage of e-mail as a messaging platform is the fundamental assumption that messages share store-and-forward behavior. Standard procedures exist for handling forwarding failures. The process includes delaying, rerouting, or even bouncing the message. Some e-mail systems even provide advisory notices when certain failures or retries occur.

To date, messaging efforts on the Web using HTTP have focused on server-side issues such as shortening the distance between the client and server through caching and incremental updating of content. Unsuccessful client calls typically elicit "Page 404" error messages with no other information.

In a workflow system, where the caller may be an automated agent or where the callee may be temporarily disconnected (as in a mobile workflow situation), the system can generate an exception. In an Internet-scale environment, correcting state and redeploying processes are very expensive operations to ensure, particularly without server-to-client notification.

Access Control Complexities

Publishing an arbitrary document on an intranet for internal viewing is beyond the skills of most users. Even worse, if the requested document requires dissemination to someone outside the company's intranet, users will resort to e-mail rather than brave the technical details required for access control and security.

The problem with e-mail is that multiple copies of documents are created with no way to track them. Changes to information may not reach users who depend on it. If a user sends the latest copy of a document to a coworker, there is no way to determine whether that document is being edited or is simply sitting idle as an unread e-mail attachment.

Many mobile clients such as smart phones and PDAs now support wireless access to the Web, but the chain of tasks needed to complete the handoff from these devices suffers many limitations:

- Access speed is typically slower than that of a standard Internet office connection.
- Most handheld and wireless Internet clients offer only limited browser support of filtered content.
- Once they find the relevant information, users switch to e-mail to send the URL—which is difficult to type on these devices or to send as an e-mail attachment—potentially incurring extra time and costs.

Work Context versus Data Access

Standing up and walking down the hallway is a context switch. In a perfect world, switching contexts would be as easy as sweeping a desktop full of papers into a folder or briefcase.

On the Web, browsers allow users to recreate a context by storing their bookmarks and preferences on a server so that any number of clients can use the same profile. This lets users bookmark resources for access away from the office. Users need the same functionality for arbitrary resources. For example, sometimes they need the actual information rather than a pointer to it.

MAGI

Magi is an architectural framework that explicitly addresses the coordination of e-business messaging and deployment across a range of computing platforms. It is an open-source interoperability specification consisting of complementary protocol standards, formats, and implementations.

Magi leverages several protocol standardization efforts, offering a simple, robust container archi-

tecture that trades features for scalability. By piecing together open-protocol standards, Magi supports a more natural mapping of the Workflow Management Coalition's model to Web primitives.[4]

In particular, Magi makes the configuration and construction of XML and Java distributed applications easier by providing an event destination for any desktop, laptop, or palmtop running a Magi server. Magi's dynamic DNS lookup and URL mapping maintains the XML definitions to provide dynamic access controls using buddy lists to Web directories and files.

Key Components

The Magi architecture has two key parts:

- *A micro-Apache HTTP server.* A scaled-down, low-memory-footprint version of an Apache HTTP stack provides interoperability across a wide range of intelligent devices, including Internet-enabled wireless PDAs that support human-in-the-loop remote activities.
- *An extensible generic interface.* Based on its deployment platform, Magi supports additional incremental utility. Apache's modular architecture and HTTP's extensibility mechanism allow easy customization of services.

Magi, then, is essentially an Apache HTTP server using HTTP as its core communication protocol. It implements other protocol definitions as Apache modules or HTTP extensions that can be loaded on demand. With appropriate permissions, users can remotely configure Magi servers to provide just-in-time service matching. For any Web protocol extension, Magi can fill the role of both client and server in peer-to-peer relationships with other Magi servers.

Magi currently supports the following Web protocols:

- *HTTP/1.1.* HTTP's first version, often called HTTP/0.9, supported the Get method across a variety of platforms. HTTP has since evolved, adding other method calls. HTTP/1.1 incorporates the Put method, which allows direct writing of a specific resource.[5] Because of its extensibility and support by many nontraditional computing devices, HTTP/1.1 serves as the communication protocol source for all Magi services.
- *WebDAV.* The Web Distributed Authoring and Versioning protocol was developed by adding several new methods and headers to HTTP/1.1, creating a standard mechanism for collaborative authoring of Web resources.[6] WebDAV defines HTTP methods such as (1) Lock and Unlock, for preventing two or more people from overwriting each other's changes; (2) Propfind and Proppatch, for allowing discovery and assignment of name-value pairs such as title, author, publisher, and other metadata about the resource; and (3) several other methods that support remote authoring of arbitrary Web resources. Any DAV-compliant tool can easily build policies for synchronization and dissemination of arbitrary data, such as extensible markup language (XML), executable process descriptions, or even binary data across Magi servers.[7]
- *SWAP/Wf-XML.* As with WebDAV, the Simple Workflow Access Protocol was developed because many workflow vendors were building proprietary extensions in order to transition their engines to the Web.[8,9] SWAP reuses many WebDAV method extensions to HTTP/1.1 and defines several of its own. SWAP supports the creating and listing of process instances running on a server. Also, agents or observers can register to be notified of state changes using Subscribe and Unsubscribe and can view a process-specific execution history using Gethistory. Long-running processes on a server can send Notify events to subscribers who have registered their callback URL. Likewise, a client can stop a running process in the face of changing information using the Terminate method. HTTP, SWAP, and DAV allow users to easily chain and nest workflows across a series of Magi servers.
- *CPAM.* CPAM is loosely related to SWAP in that it is an HTTP/1.1 extension intended to start, stop, and monitor long-running processes on a server.[10] In its method definitions, CPAM includes certain prediscovery of process values before executing a request that may incur expenses. The Estimate method allows a client to get a cost estimate for a process or a service before committing to its execution. Furthermore, CPAM uses Examine to test the status of or keep a running track of the progress of an invoked method. Using CPAM methods, clients or agents can evaluate services based on cost factors and terminate electronic services that suffer cost overruns.

Integration Architecture

Magi servers maintain interoperability with current HTTP Web servers. The protocol technologies are grouped into service modules. Other clients and servers can query the Magi server using the HTTP

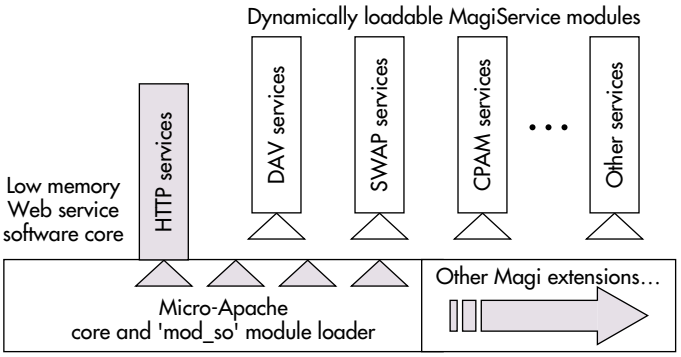

Figure 1. Magi configuration and component architecture. Magi services can be added on the fly, thus allowing other Magi servers or clients to discover if a certain Magi server can help answer the particular request.

Options method to see which protocol extensions are available. As shown in Figure 1, Magi servers load services in response to requests.

The Web service software core can be as simple as a low-memory-footprint HTTP stack serving up a collection of resources. For simple retrieval and placement of resources, this HTTP stack can be as small as 38 to 64 Kbytes plus data. The media types that a Magi server can transfer may need to be similarly dynamically configurable and extensible.

Magi devices are configured with either a statically or dynamically allocated IP address. Static IPs are simply handled as a named host and use traditional DNS routing and naming. Dynamically allocated IP addresses require a dynamic DNS intermediary that keeps online aliasing information. Each Magi server contains the identity of the user, a unique public and private key, and an XML "buddy list." Buddy lists keep track of named Magi servers and the electronic services they support. Buddies, which may be either human or automated agents, represent trusted entities.

When launched explicitly or on establishment of a live Internet connection, Magi servers register with the dynamic DNS server and then notify each client on the buddy list. The local Magi server may allow or restrict browsing, searching, publishing, and other functions on a buddy, directory, or file-by-file basis. This combines the flexibility of a fully configurable Web server with the familiarity users have with manipulating buddy lists.

Magi users can

- hand off or edit documents in place using any WebDAV-compatible Web publishing client;
- create files and folders on any Magi server where they have permission; and
- submit information or paperwork to a buddy's Magi server by dragging and dropping the required document to any desktop-mounted Web folder.

Magi servers can capture and influence several types of events. To create an event, they can register actions of another Magi server going on- or offline, publishing a document, or even simply downloading a file. Further, users may register a set of workflow actions with an event. Events can trigger one-time actions or can perform recurring workflows to synchronize data or collect and assemble the latest information from a variety of Web destinations.

From the client's perspective, the URL on the Magi server that initiates, monitors, or terminates the server-executed workflow process represents a trackable electronic service. From the server's perspective, the Magi server represents a collection point for initiating an electronic business process.

Levels of Support

Magi consists of seven compliance levels that come into play depending on use and platform support to build up more complex transactions. Magi's levels are generally additive; however, some Magi servers may skip functionality. They scale from a simple resource service that employs HTTP methods Get and Head to secure peer-to-peer discovery and leasing of complex services. HTTP's Options method allows users to discern each level's functionality.

Users can tailor the implementation and deployment of a Magi-compliant server to platform and application-specific requirements while maintaining interoperability with other Web services. Table 1 (next page) details the levels, the particular methods that each level supports, and the feature descriptions and rationales.

EXAMPLE SCENARIO

Because Magi was designed to specifically address the advantages of paper-based workflow, we'll examine how it applies to a common paper-based process. The manager of a company needs to send a memo on changes in health insurance and to include a survey about the appropriateness and effects of the changes. Because the manager needs all recipient signatures on one form, a routing slip is stapled to the memo and survey for each person to sign their initials and check off relevant information. Further, the manager puts a yellow sticky note on the top that

Table 1. Magi protocol levels 1 to 7.

Level	Methods	Description/Rationale
Magi-1	Options, Get, Put, Head	Minimum, low-memory infrastructure to support putting information "on the wire" compatible with the HTTP specification
Magi-2	Post, Delete, Trace	A more powerful host server for back-end execution of server-side commands and information processing
Magi-3	Lock, Unlock, Copy, Move, Proppatch, Propfind, Publish	Collaborative manipulation of resources on a server; prevention of overwrites; manipulation of namespace; assignment and querying of properties and metadata; publication of arbitrary data
Magi-4	Subscribe, Unsubscribe, Notify, Logon, Logoff	Internet-wide notification, event management, and presence information; subscription and unsubscription to event channels
Magi-5	Createprocess(instance), List(process)instances, Gethistory, Run, Complete, Terminate	Remote tracking and management of long-duration processes; initiation/cessation of server-to-server interactions; display of history of process over time, including status and events; tracking of progress across a set of cooperating servers
Magi-6	Examine, Estimate	Estimation of a service before committing; maintenance of running totals of an electronic service; examination of cost factors
Magi-7	Register, Unregister, Proxy, Search, Filter, Commit, Rollback, and other application-specific methods	Service-specific queries to determine applicability of an electronic service; intelligent resource matching, alliance formation primitives; "I know who can help" e-service proxying and advertising; renting and leasing of human- and agent-based electronic services

states the workflow rules: (1) After you have signed this, give it to the next person on the list. (2) After everyone has signed this, return it to me. After a week or two of bouncing around in people's in boxes, the forms get signed and returned.

The paper-based workflow usually works, except that there is no way to determine exactly how many people have signed the form at any given time. Even e-mail would not allow the manager to determine whether a particular form has been received or read. A Magi server on each desktop would allow documents—and the process definition—to migrate from desktop to desktop. A paper-based system also lacks a way to route the information around people on vacation. You cannot (1) determine whose inbox the forms are sitting in; (2) change course or review the intermediate results ("I need it back now!"); or (3) change participants or steps of the workflow in response to exceptions ("Peter is in London this week"). Magi servers can track the status and handoff of materials well beyond the current organizational boundaries of the workgroup or company.

OTHER APPROACHES

Magi's architecture focuses on Internet-scale issues—which are different from those of other highly scalable enterprise-wide systems—by supporting inconsistent data. One factor that has made the Web so successful is that it allows broken links, which arise when one site changes a target resource that other sites link to. When the number of links to a locally owned resource is unknown, it is impossible to notify all sites pointing to the resource of the change. This results in inconsistent data.

Closed systems limit the actions that a user can perform on a resource (such as moving or changing it) in order to maintain consistency. The Web takes an open approach, allowing inconsistent data and thereby accommodating large numbers of users and agents.

Several researchers have explored the requirements for Internet-scale event notification and Internet-scale namespaces. Magi attempts to consolidate the lessons learned into an Internet-scale workflow architecture.

Internet Workflow Approaches

Workflow architectures have historically been either message-based or database-based. More recently, workflow researchers and vendors have begun using the Web as the underlying execution and deployment transport. Because of the limitations discussed earlier, designers have made efforts to extend the infrastructures. These efforts fit into three schools.

E-Mail/Store and Forward. Messaging workflow systems typically use e-mail to send assignments. This allows anyone with e-mail to participate in a

workflow process, regardless of location. The store-and-forward mechanisms of e-mail allow asynchronous distribution of assignments.

Because a wide range of devices support e-mail, messaging workflow systems can scale to large workgroups. These systems have tried to scale to the Internet by extending SMTP and IMAP to better support store-and-forward application architectures. One such effort is the application core protocol (Applcore),[11] whose set of primitives handles the challenges of messaging-based extensions in store-and-forward protocols. One feature of Applcore is its ability to bring a "state" back to a "stateless" protocol. Stateless protocols such as HTTP require synchronous behaviors because of the Web's server-to-client limitations.

Workflow software has evolved mainly because of the acceptance of the WebDAV and XML protocols.

Using a Magi server on the client side with appropriate routing and naming brings the benefits of asynchronous messaging into a peer-to-peer context. SMTP servers communicate peer to peer between servers but resort to client polling for final delivery. The ability to map store-and-forward channels back to a client using a Magi server creates a more flexible model with increased visibility and tracking for messaging-based workflow.

Database/Distributed Objects. In a database-centered, distributed-objects workflow architecture, work items reside in a centrally located storage area on a server. The centralized control of objects makes it easier to tightly manage and track changes, but participants must have access to the appropriate server. Accessibility, location, presentation, and performance can all be difficult issues in creating and maintaining workflows.

The typical architecture for Web-based workflow limits the deployment of processes and data. A Web server with a database back end permits access to work items and accomplishes workflow execution by pulling dynamically generated HTML pages from the server and submitting changes or updates to it. To overcome notification, routing, and naming limitations, developers hope the next generation of HTTP will be more like a programmable distributed object protocol. HTTP/1.1 would be replaced with an HTTP next-generation or HTTP-2.0 specification more like an efficient, multiplexing messaging transport that can interoperate with Java's remote method invocation (RMI), CORBA's Internet Inter-ORB protocol (IIOP), and remote procedure call protocols.

Web Extensions. Time and again, the HTTP suite of protocols has been "just useful enough" for a variety of Web application communities. HTTP's extensibility mechanisms have given birth to a whole set of new primitives, protocol-specific headers, message formats, and Internet media types.

Workflow software has evolved mainly because of the widening acceptance of the WebDAV and XML protocols. WebDAV has opened the door to more complex human- and agent-based content publishing and sharing. It represents a calculated compromise between defining a near-infinite number of HTTP methods recognizable by only a small percentage of Web servers and overloading existing HTTP methods such as Post.

Enterprise and Internet-Scale Approaches

Two major enterprise-scale offerings are Hewlett-Packard's E-Speak/E-Services and Sun Microsystems' combination of Jini and Enterprise Java Beans. Magi fits well with both approaches, whose electronic service discovery and execution models map well to the Magi HTTP method primitives.

E-Speak. E-speak was introduced to create, compose, deploy, and manage electronic services, which are defined as anything that can be digitally transmitted, including access to the communication channel itself. E-Speak contains an infrastructure that resides on a logical computing device, and its core works within that framework to provide coordination of low-level, multitiered electronic services across a federation of devices.

Every collocated service registers its name and metadata with the E-Speak core on the platform where it "lives." Each task, typically a client, can then query the core for name-value attributes matching a desired service. Every task has at least one outbox connected to the core and may also have inboxes. Messages contain both an envelope, which contains core-related data, and a payload, which contains application data. When services are not available

within that particular E-Speak core, the client passes the request to an intermediary, which may perform proxy functions to match the task with other advertised services it knows about.

E-Speak constructs services with three visible abstractions: a *service* is simply some action that a client invokes; a *contract* defines the service-level interface; and a *vocabulary* describes the range of services available for discovery. E-Speak is built on top of a network object model. Electronic services can map onto various platforms and implementation languages as long as they adhere to the E-Speak Internet protocol and the specific interface contracts that have been registered, and can receive messages delivered using the E-Speak service bus.

Jini. Sun Microsystems' Jini architecture also allows a federation of devices to work together without extensive planning, installation, or human intervention. Each device can advertise services that other devices on the network may need. When a Jini device plugs into a network, it polls the network to locate a Jini technology lookup service, which is similar to an E-Speak service repository. But, instead of storing name-attribute metadata about electronic services, Jini can actually store the objects that implement the service.

Users can invoke services in Jini by using Java's RMI or by downloading the RMI-serialized object into the device's address space and executing it on the local virtual machine. Furthermore, the Jini device can upload a service object to the Jini server for each service it provides.

E-Speak, Jini, and Magi. Both E-Speak and Jini map well to Magi's architecture. Configuration, control, and remote authoring of services blend well with Magi's WebDAV implementation. Magi's use of the Apache HTTP server allows it to leverage the entire security infrastructure that has been integrated with it. Although Apache and plug-in security measures lack the fine-grained security implementations of both E-Speak and Jini/Java's security manager, the functionality that has been deployed is adequate for an abundance of electronic commerce, collaboration, and other Web-based applications.

Jini's initial focus has been on the Internet device level. Magi complements this to leverage users' familiarity with common desktop metaphors and the Web to abstract away many of the details of security and collaboration. Access control is handled through buddy lists similar to AOL's Instant Messenger. Other security mechanisms, such as approaches to certification, authentication, filtering, and encryption, are borrowed from the software and Web-standards communities.

CURRENT STATUS

Magi is an open-source project and is freely downloadable at http://magi.endeavors.org. The current release includes Apache 1.3.12, the Xerces XML tool set, the Jserv Java servlet engine, and Java Runtime Environment 1.2.2.

Specific Magi components include

- Magi_DAV, the dynamic WebDAV implementation that supports XML-based buddy lists and access controls;
- Magi_GUI, a graphical client for determining presence information, exchanging files, and managing access controls; and
- Magi_DNS, the dynamic IP mapping service.

Magi_DNS can be downloaded and installed in a local workgroup and can double as a metadata repository of all files located on any mobile or disconnected Magi installation. This allows the searching of information that is not immediately connected to the network and the discovery of Web resources or documents that are offline.

Magi_DAV can be downloaded and integrated into any client or server workflow system or other desktop tool independent of the other Magi components. Because Magi_DAV adheres to the WebDAV standard, desktop tools such as Microsoft Office 2000 (PowerPoint, Word, and Excel), Internet Explorer, Netscape Communicator, and others are already compatible with the Magi server. Magi_DAV also supports additional HTTP-derived protocol extensions such as SWAP/Wf-XML.

The first release of Magi_SWAP—a SWAP implementation supporting dynamic buddy-list permissions and remote service handoff and tracking—will be implemented in Java and should be available this summer. Because the Xerces XML parser is supported in both Java and C, both languages will support subsequent implementations of both Magi_DAV and Magi_SWAP.

In conjunction with the generic interface portions of Magi, the current Magi server runs on most desktop operating systems, including Solaris, Win32, Linux, and Mac OS X. Other Unix and desktop ports are possible but have not been tested.

The notification infrastructure, which includes the nongraphical back-end primitives, is implemented using Java servlets. Adding runtime components as

Java servlets for presence information, user configuration, and Endeavor's workflow execution classes adds anywhere from 3 to 8 Mbytes.[3] Although 8 Mbytes is large for some systems, it represents the unoptimized runtime versions of all the software and associated toolkits. The trade-off is that not all target systems will need full support of all Magi levels.

Combining the trends of optimization and miniaturization of computing with greater availability of application memory on handheld and wireless devices, full Magi support of smart phones or embedded devices seems inevitable. In addition, the experimental PalmOS Magi implementation has only static support for HTTP/1.1 methods and placeholders for various WebDAV methods. The implementation fits into about 50 Kbytes and runs on a Palm III.

CONCLUSION

Management can deploy a federation of Magi servers across an organization to seed communication, collaboration, coordination, and costing of electronic services beyond current organizational structures. These server-primitives enhance the management, synchronization, and archiving of online information.

As workflow technology evolves from centralized systems to a ubiquitous, embeddable service across a wide variety of information technology and e-commerce Web applications, users will be able to move beyond simple document exchange. Organizations will be able to manage and track information that was previously out of reach—namely, that of mobile and disconnected devices and users. Remote monitoring and tracking of the status of everything from embedded manufacturing components to off-site and in-transit workers or automated agents will extend workflow management beyond traditional workplace boundaries. Magi can empower the next generation of e-business applications to scale beyond current Web limitations to allow advertising and discovery of Internet-scale electronic services across a broad range of mobile and disconnected workers and devices. ■

ACKNOWLEDGMENTS

I would like to thank the following individuals for their exchange of ideas: Tim Byars, Clay Cover, Steve Dossick, Roy Fielding, Michael Gorlick, Art Hitomi, Peter Kammer, Rohit Khare, Peyman Oreizy, Dick Taylor, and Jim Whitehead. In addition, I would like to recognize the generous support of Bernard Hulme, Graham Brown, and Una Richens of Tadpole Technology, PLC.

REFERENCES

1. G. Abowo and B. Schilit, "Ubiquitous Computing: The Impact on Future Interaction Paradigms and HCI Research," *Proc. CHI '97,* ACM Press, New York, 1997.
2. D. Rosenblum and A. Wolf, "A Design Framework for Internet-Scale Event Observation and Notification," *Proc. Sixth European Software Eng. Conf./ACM SIGSoft-Fifth Symp. on the Foundations of Software Eng.,* ACM Press, New York, 1997, pp. 344-360.
3. G. Bolcer and R. Taylor, "Advanced Workflow Management Technologies," *J. Software Process: Improvement and Practice,* vol. 4, no. 3, Oct. 1998, pp. 173-182.
4. D. Hollingsworth, "The Workflow Management Coalition Reference Model," Tech. Report WFMC-TC-1003, version 1.1, Workflow Management Coalition, Lighthouse Point, Fla., Jan. 1995.
5. R. Fielding et al., "Hypertext Transfer Protocol—HTTP/1.1," Internet Proposed Standard RFC-2068, Univ. of California, Irvine, DEC, MIT/LCS, Jan. 1997.
6. J. Whitehead, "Collaborative Authoring on the Web: Introducing WebDAV," *Bulletin of the American Soc. for Information Science,* Vol. 25, No. 1, Oct./Nov. 1998, pp. 25-29.
7. T. Bray, J. Paoli, and C.M. Sperberg-McQueen, "Extensible Markup Language (XML) Technical Report," W3C*;* available online at http://www.w3.org/TR/PR-xml-971208 (current 24 Apr. 2000).
8. G. Bolcer and G. Kaiser, "SWAP: Leveraging the Web to Manage Workflow," *IEEE Internet Computing,* vol. 3, no. 1, Jan.-Feb. 1999, pp. 85-88.
9. K. Swenson, "Simple Workflow Access Protocol (SWAP)," work in progress*;* http://www.ics.uci.edu/pub/ietf/swap/swap-prot.txt (current 24 Apr. 2000).
10. D. Beringer and G. Wiederhold, "Cost Estimation in CPAM: An Access Protocol for Remote and Autonomous Services," *Proc. Workshop on Cross-Organizational Workflow Management and Coordination, WACC '99,* http://www.Zurich.ibm.com/~hlu/WACCworkshop/papers/Beringer.htm (current 24 Apr. 2000).
11. R. Khare, "Building the Perfect Beast: Dreams of a Grand Unified Protocol," *IEEE Internet Computing,* vol. 3, no. 2, Mar.-Apr. 1999, pp. 89-93.

Gregory Alan Bolcer is founder and interim CEO of Endeavors Technology Inc., a wholly owned subsidiary of Tadpole UK. His current work includes technologies for managing Internet-scale event notification and namespaces. Bolcer received an MS in computer science at the University of Southern California in 1993 and a PhD in information and computer science at the University of California, Irvine, in 1998.

Readers may contact the author at gbolcer@endtech.com or http://www.endtech.com/.