Intermediaries: An approach to manipulating information streams

by R. Barrett
P. P. Maglio

Information flows all around us all the time. Whether on computer networks, on telephone lines, or within the wiring of everyday devices such as coffeemakers or gas pumps, data are constantly being transmitted from one place to another. Much of the time, such data flow directly between information producers and information consumers. Sometimes, however, intermediary processes stand in the way of a simple data flow, for instance, to monitor traffic, to bridge between incompatible communication streams, or to customize or extend the functions that are natively available on a stream. Intermediaries can turn ordinary information streams into* smart *streams that enhance the quality of communication. Because information flows are now everywhere, there is a new opportunity for taking advantage of intermediary computation, but general principles and approaches have not yet been developed. This paper provides an introduction to the intermediary approach, describes an implemented Web intermediary framework and applications, and proposes extending intermediaries to other information streams.

As information becomes ever more pervasive and important, people increasingly rely on a variety of information streams to meet their information needs. Rather than one stream replacing another in this economy of information, each stream has developed its own niche. Thus, newspapers did not disappear when radio was developed, radio remains after the advent of television, and the telephone did not obviate the need for postal mail. More recently, e-mail, news groups, chat rooms, push technologies, pagers, cellular phones, personal digital assistants, and the World Wide Web (WWW, or Web) have greatly expanded the set of information streams to which we all have access. Importantly, being connected to many streams is very nearly a necessity of the modern world.

An information stream conveys data from an information provider to an information consumer. For instance, on the WWW, servers generally provide information, and browsers generally consume information. Of course, streams can be bidirectional, so that the same endpoint might be both a provider and a consumer at different times. A telephone is just this kind of stream, enabling two (or more) parties to freely exchange information in any direction. Often, the stream simply conveys the information without additional processing, as the telephone does, but sometimes information can be usefully injected or modified along the stream. For instance, some telephone companies provide real-time language translation, or some Web communication passes from one network to another through a firewall.

We define *intermediaries* as computational entities that operate on information as it flows along a stream

Figure 1 An intermediary is a computational element that lies between an information producer and an information consumer on an information stream.

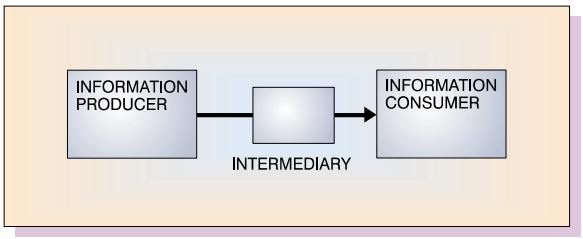
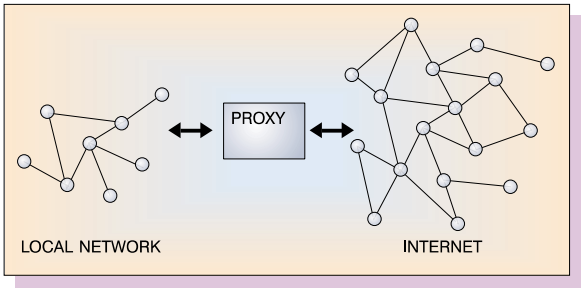


Figure 2 A Web proxy is an intermediary that tunnels requests and responses between two disjoint networks.



(see Figure 1). Because of the tremendous number and kind of information streams that are now available, there is a new opportunity to take advantage of intermediary computation. We believe intermediaries can add value in several different ways. Namely, an intermediary can (1) produce new information by injecting it into the stream, (2) enhance the information that is flowing along a stream, and (3) connect different streams, possibly translating communication protocols in the process. Note that intermediaries do not create new information devices (such as telephones or Web browsers) but increase the value of existing devices by improving the streams upon which the devices operate. Likewise, intermediaries do not create new information streams but enhance existing streams.

Many information streams operate properly without intermediaries. An information device connects to a wire that then connects to another information device. A Web browser connects through the Internet to a Web server. A telephone connects through a telephone line to another telephone. For these, intermediaries are not strictly necessary but can be added to improve the existing system in some way. For example, in the case of the Web, it is often desirable to protect the internal network of a corporation from the Internet at large. A firewall can block unauthorized external access to the internal network, but this changes the network topology so that the browser can no longer reach external Web servers. To solve this problem, firewalls often contain a Web proxy[1] that conveys Web traffic from internal Web browsers to external Web servers (see Figure 2). This Web proxy is an intermediary that selectively connects two networks, thus adding value to the Web stream. Web proxies often add other features too. For instance, a proxy may cache Web pages that have been viewed by users within the firewall (as an example, see Yu and MacNair[2]). Subsequent requests for those Web pages can then be satisfied by the proxy, rather than requiring additional Internet traffic. This caching function is another case of an intermediary adding value to an information stream.

Intermediaries can do more than simple network translation and caching. For example, a Web intermediary can compress large images before sending them across a slow network link such as a telephone line. Or it can lay out a Web page for the limited display of a PalmPilot** before sending the page along to the device.[3] A telephone intermediary can take speech input and automatically translate spoken names into dialing tones—effectively dialing the phone when the person picks up the receiver and says, "Call Aunt Marion." This action suggests one key advantage of using intermediaries over simply computerizing existing devices. Systems already exist for connecting a telephone to a computer and then using voice recognition on the computer to dial the telephone.[4] But an intermediary *hides* the computer inside the telephone line so that the user need not learn how to manipulate a new device. This intermediary gives the conventional telephone the ability to understand and act on a human voice.

The utility and power of an information stream depends on many factors, including its (1) quality of information, (2) availability when information is needed, (3) breadth of information, (4) ease-of-use, (5) reliability, and (6) relevance of information. The telephone is an important stream because it connects one person directly to another, which implies high-quality information. The breadth of available information is vast because a large fraction of the world's population is reachable by telephone, and ease-of-use is high because dialing about 15 digits can con-

nect one person to another anywhere in the world. Availability is also increasing as people use mobile phones more, enabling others to contact them anytime and anywhere. Of course, with increased phone usage (e.g., unsolicited sales calls), the amount of unwanted or irrelevant information increases and causes people to limit distribution of phone numbers and to not respond to all calls. In any event, improving performance in these six areas can increase the value of an information stream.

In this paper, we take the first steps toward outlining general principles for intermediary computation and discuss ways to improve a variety of information streams through the use of intermediaries. The paper is organized in six parts. First, we describe what an intermediary is in detail through a series of examples. Second, we unpack the notion of intermediary in an attempt to more precisely formalize it. Third, we describe a general architecture for adding intermediary computation to the Web. Fourth, we sketch several scenarios for using intermediaries to coordinate functions among several different streams. Fifth, we discuss the role of intermediary computation in pervasive computing devices. And finally, we outline future directions of this work.

Intermediaries are everywhere

The concept of an intermediary is not a new one. In fact, intermediaries are so commonplace that it is sometimes difficult even to notice them. For instance, human intermediaries abound. Travel agents translate customer requests into data entered into airline reservation computers. In this way, a travel agent acts as a protocol-translating intermediary, effectively connecting a customer on the telephone to a mainframe computer running the airline reservation system. Of course, human travel agents provide all manner of additional functions, such as suggesting ways to lower costs, explaining the results of computer queries, faxing itineraries, and so on. But the main work of a travel agent is to intelligently connect the information streams of telephone, fax, mainframe computer, printer, mail, and e-mail. Because many of these tasks can be handled without human intelligence, several automated Web-based travel agents now translate Web-based forms into airline reservation requests and then translate the responses back into Web pages for the customer.[5] Similarly, just a few years ago, human gas station attendants functioned as intermediaries who controlled gas pumps for customers in exchange for cash. Now, computerized intermediaries within gas pumps handle financial transactions. Nevertheless, many other roles of human intermediaries are not so easily replaced, such as librarians, secretaries, or any other role requiring intelligent interaction.[6]

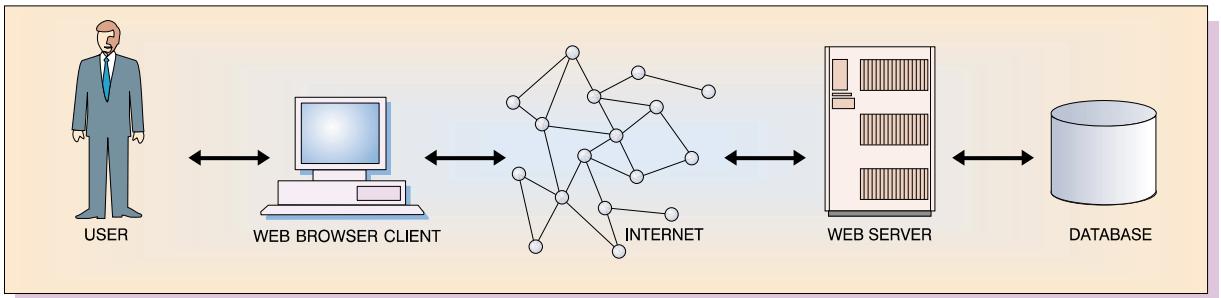
Scientific journal editors comprise another intermediary-based system. Journal editors receive submissions from authors, send manuscripts to reviewers, forward reviews back to authors, receive corrections from authors, and deliver final copy to printers. All these activities are designed to add value to the information stream that results from authors writing down their findings. The journal editor enhances the value of this information stream by providing competent reviewing, ensuring anonymity of reviews, organizing and indexing articles, and editing completed manuscripts. With so much publishing now moving to the Internet, it is important that journals identify these (and other) contributions to the publishing process to maintain their business and their valuable service to the scientific community.

Not only are journal editors intermediaries, but so are the paper copies of the journals themselves. Journals are tangible representations of the information contributed by authors and therefore enhance the information stream coming from these authors by providing archival forms of their findings and targeting interested readers. Of course, one could conceivably do without journals; scientists might telephone one another with their findings, but this is obviously inefficient. Journals are intermediaries that enhance an information stream that connects scientific communities.

Intermediaries are common in many other kinds of information streams as well. In fact, e-mail depends on intermediaries to hold messages after they have been sent and before they have been received. The POP3 protocol[7] explicitly defines how such an e-mail intermediary works. Many other e-mail intermediary functions can be envisioned. For instance, e-mail intermediaries might provide: (1) local replicas of remote e-mail repositories so that e-mail can be handled off line, (2) automatic summarization of long e-mail messages, (3) intelligent e-mail routing to correct e-mail addressing errors, and (4) services that log and index e-mail for later retrieval.

The World Wide Web is another, and immensely popular, information stream that is managed by computational processes.[8] Because the stream between browser and server is completely automated, it is an obvious place to find computational intermediaries.

Figure 3 The basic elements of Web use



For example, MetaCrawler**[9] is a search service that provides a single user interface for queries to multiple Web search services, combining results to produce a single list of documents. As an intermediary, MetaCrawler enhances the query-result information stream. The collection of Web directories in Yahoo!**[10] is another example of a Web intermediary. These directories do not themselves contain topical information but provide an intermediary service for connecting Web users to information. In a similar way, all Web search engines provide a sort of intermediary service with the same purpose. One search engine, AltaVista**,[11] has recently added a language translation intermediary service that reads its input from the original Web page and performs machine translation to produce a version of the page in another language. This service adds value to the information stream for readers who cannot understand the original language of the page.

Analyzing such commonplace and complex systems of information flow in terms of information origin, destination, and intermediaries illuminates design principles for computational intermediary systems.

Anatomy of an intermediary

Though intermediaries on information streams are ubiquitous in both human and computational systems, a thorough and systematic study of their properties has not been undertaken. In this section, we begin such a study by considering carefully what parts are needed to make up an intermediary process, and we then provide a classification of types of information streams and the sorts of intermediaries appropriate for each.

Endpoints and middle points. Information streams consist of origin and destination endpoints, the stream itself, and various intermediaries that are located at middle points and that operate upon the stream. A serious complication in analyzing such systems is that they may be decomposed into these constituent elements in many ways and at many different levels. Consider the case of a person browsing a database using the Web. The basic elements (depicted in Figure 3) include the user, a Web browser, the Internet, the Web server, and a database. In one decomposition of the system, the database is the origin endpoint, the Web browser is the destination endpoint, and the Internet and Web server are intermediaries. However, Web servers are often regarded as the information origin, even if they actually use a database to find the data that they serve. In that case, the Internet is the only intermediary. But a networking engineer may choose to look more deeply at the Internet element, breaking it down into a collection of intermediaries that includes Ethernets, token rings, routers, gateways, name servers, hubs, and so on. An electrical engineer may choose to go even further, considering line drivers, laser diodes, and optical fibers as essential intermediaries in the information stream. In the other direction, a sociologist may abstract away all of the details of the computer systems and focus on the origin endpoint as the person who produced the information that resides in the database. In this case, the destination endpoint would be the Web-browsing user—or possibly the person that will receive the report that user writes. Thus, the Web browser, Internet, Web server, and database are all lumped together as a single seamless intermediary that connects user with information producer.

Partitioning an information stream into origin endpoint, destination endpoint, and intermediaries involves selecting several division points: everything beyond the chosen origin point is the origin; every-

thing beyond the chosen destination point is the destination; and various points in between are chosen as breaks between intermediaries. Many decompositions are possible, but the most suitable one matches the analyst's needs. Designing an intermediary information stream is analogous to building a complex UNIX** command by piping data from an information origin (e.g., stdin) through any number of intermediaries (e.g., grep, more) and to an information destination (e.g., stdout).

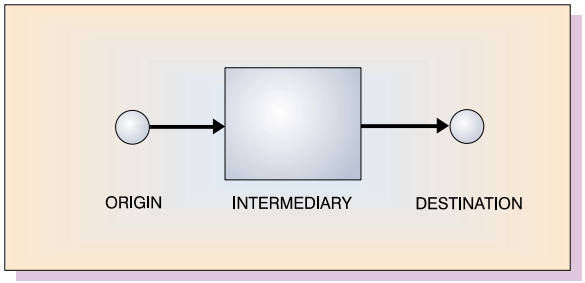
Now consider the functions of the three basic entities: origin endpoint, destination endpoint, and intermediary (see Figure 4). The origin endpoint has one connection point and transmits information to it. It may also receive requests for information, or it may transmit proactively. The destination endpoint also has one connection point and receives information from it. It may also transmit requests for information, but it is not required to do so. An intermediary is most easily conceptualized by considering everything on one side of it to be an origin endpoint and everything on the other side to be a destination endpoint. The intermediary has two connection points: one connecting to an origin and the other to a destination. The side connecting to the origin behaves like a destination: receiving information from the stream (and possibly transmitting requests for information). The other end of the intermediary connects to the destination and behaves like an origin: transmitting information to the stream (and possibly receiving requests for information).

From this decomposition, an endpoint is simply another kind of intermediary. Whether origin or destination, an endpoint can be realized as an intermediary that ignores one of its connections. An origin endpoint is an intermediary that ignores its destination-like connection; a destination endpoint is an intermediary that ignores its origin-like connection. Put differently, endpoints do not have any additional capabilities beyond those of intermediaries: they are simply intermediaries that speak to a vacuum on one side.

We now turn to our classification of information streams to explore the implications of various structures on the endpoints and intermediaries in the stream.

Message streams. Information streams vary widely in complexity, which affects the roles of the processes involved in information transfer. The simplest information stream consists of a unidirectional flow from origin to destination. An example of such a flow is a telemetry system where the origin endpoint transmits status information to a destination endpoint. Intermediaries on such streams have one connection that acts as a destination and one that acts as an origin. Because the data on the stream consist of discrete messages in most such systems, we refer to this system as a *unidirectional message stream*.

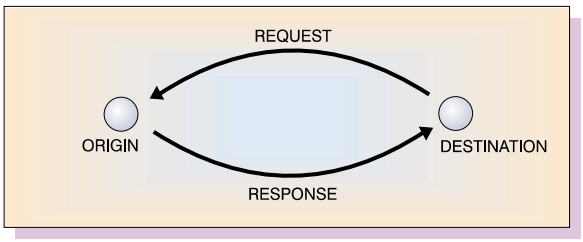
Figure 4 An intermediary lies between an origin endpoint and a destination endpoint



The next level of complexity occurs when the origin and destination endpoints are allowed to play both roles; that is, the destination can transmit messages back to the origin. If the endpoints can act in either role arbitrarily, the system is a *bidirectional message stream*. One example of such a system is a simple two-party chat because either party can chat at any time. Another example is the conventional telephone because either party can talk at any time.

Transaction streams. A more structured bidirectional system results if the endpoints reverse roles in a regular way. The most common example is when the destination sends a request message to the origin, and then the origin sends a response message to the destination (see Figure 5). The terms *origin* and *destination* are chosen in this way because the request normally includes a description of some desired information and the response contains that information. The desired information flows from origin to destination; the request is simply a mechanism for accessing the desired information. An example of this system is the HyperText Transfer Protocol (HTTP) that is used on the World Wide Web. A browser sends a request message to a server with a uniform resource locator (URL) that describes the desired information. The server then sends a response message that contains the information referred to by the URL. We refer to this system as a *unidirectional transaction stream*. A transaction is defined as a single request-response pair.

Figure 5 In a request-response type of information stream, the destination actually requests specific information from the origin, which in turn sends it along the stream.



The final structure we consider is the *bidirectional transaction stream*, an extension of the previous case in which origin and destination can reverse roles arbitrarily. It is a special case of the bidirectional message stream because each request requires a response, rather than simple message transmission. Note that many real-world implementations of transaction streams allow for a transaction to be aborted before completion. Such an abortive action seems to change a transaction stream into a message stream, but the key difference is that aborts are not part of the usual operation of the stream.

To see this classification scheme in action, consider the standard telephone system, which consists of three parts: an origin telephone, a destination telephone, and a telephone central office intermediary. When the system is in its quiescent state, both telephones are "on-hook" and idle. One party lifts the receiver, which sends an "off-hook" request down the stream. This request is intercepted by the telephone central office intermediary, which sends a "dial tone" response back to the telephone, completing the first transaction. The originating party then dials a destination phone number (which could be broken down into a series of requests, but we consider it to be a single request in this example), such as "555-1212." The central office intermediary intercepts this request, sends a "ring" request to the destination telephone, and sends a "ringing tone" response back to the origin telephone. The second transaction is now complete, and the third transaction has begun. When the destination phone is answered, it sends an "off-hook" response to the central office intermediary, which completes the third transaction. The central office now connects the two telephones together and begins acting as a transparent intermediary, simply passing audio messages back and forth between the two telephones. The system has switched modes: from a unidirectional transaction system to a bidirectional message system.

In summary, we have defined four types of information streams along two dimensions: (1) whether information flow is unidirectional or bidirectional, and (2) whether communications are message- or transaction-based. In unidirectional streams, information flows from origin to destination, whereas in bidirectional streams, information can flow in either direction. Arbitrary messages can flow along message streams, whereas only well-defined request-response message pairs can flow along transaction streams.

Building intermediaries

Many intermediary applications, sometimes termed "agents," have been built, such as Letizia[12] and WebWatcher[13] in the Web domain, Oval[14] and Remembrance Agent[15] in the e-mail domain, and NewsWeeder[16] in the Usenet news domain. These intermediary applications and many others can be much more easily constructed with appropriate frameworks and toolkits. To simplify and systematize the development of Web intermediaries, we have implemented WeB Intermediaries (WBI),[17,18] an infrastructure for designing and building Web-based intermediaries. As noted, the Web is a unidirectional transaction stream, and therefore we believe the WBI approach is applicable to other systems with the same sort of data flow.

WBI architecture. The Web is an attractive system for applying intermediary technology because it is both popular and simple. The Web is unidirectional (only the browser initiates transactions) and transactional (every browser request produces exactly one server response). It is also stateless, which means that each transaction is handled independently without reference to any history of transactions. Of course, browser, server, and intermediaries can all maintain their own internal state, but there is no inherent notion of state on the stream itself. These three characteristics make the framework for Web intermediaries particularly straightforward.

As an intermediary, WBI connects to both browser and server during a Web transaction. But because the browser initiates all transactions, WBI need only listen to the browser connection to start. Connections to the server are initiated by WBI as needed. Thus, WBI need not wait for server connections or initiate connections to the browser. These simplifi-

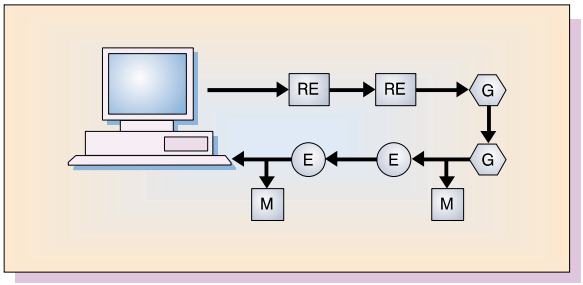
cations result from the unidirectional quality of the Web. Because the Web is transaction-based and stateless, each transaction can be treated independently, since it involves exactly one request and its associated response.

From the perspective of a Web browser, WBI acts as an HTTP request processor, receiving a request and sending a response. The processing that happens in between is completely controlled by the functions WBI is set up to provide. WBI might operate like a conventional Web server—using the request to access an internal document and sending it back to the browser. It might operate like a programmable Web server—using the request to execute a program that produces the resulting page. It might operate like a transparent Web proxy—forwarding the request to the appropriate Web server and then forwarding the result back to the client. Or WBI could perform any of a range of intermediary functions, such as personalizing contents,[17,19] transcoding one representation to another,[3] or caching results to be sent back directly the next time a page is accessed, and so on. The WBI architecture provides a simple and powerful programming framework for developing any such Web applications.

A WBI transaction flows through three basic stages (see Figure 6): request editors, generators, and editors (which might be more appropriately called document editors). *Request editors* receive a request and may modify it before passing it along. *Generators* receive a request and produce a corresponding response (i.e., a document). *Editors* receive a response and may modify it before passing it along. When all steps are completed, the response is sent to the originating client. A fourth type of processing element, the *monitor*, can be designated to receive a copy of the request and response but cannot otherwise modify the data flow. Collectively, we refer to monitors, editors, and generators as MEGs (see Table 1).

WBI dynamically constructs a data path through various MEGs for each transaction. To configure the route for a particular request, WBI has a rule with a priority number associated with each MEG. The rule specifies a Boolean condition that indicates whether the MEG should be involved in a transaction. This condition may test any aspect of the request header or response header, including the URL, content type, client address, server name, and so on. Priority numbers are used to order the MEGs whose rules are satisfied by a given request or response. More precisely, when WBI receives a request, it follows these steps:

Figure 6 A transaction (request/response) flows through a series of WBI MEGs.



1. The original request is compared with the rules for all request editors. All request editors whose rule conditions are satisfied by the request are allowed to edit the request in priority order.
2. The request that results from this request editor chain is compared with the rules for all generators. The request is sent to the highest-priority generator whose rule is satisfied. If that generator rejects the request, subsequent valid generators are called in priority order until one produces a document.
3. Both request and response are used to determine which editors and monitors should see the document on its way back to the client. Each editor whose rule condition is satisfied modifies the document in priority order. Monitors are also configured to monitor the document either (a) as it is produced from the generator, (b) as it is delivered back to the client, or (c) after a particular editor.
4. Finally, the response is delivered to the client.

A WBI application is usually composed of a number of MEGs that operate in concert to produce a new function. Such a group of MEGs forms a *plug-in*, which is the basic unit of granularity of WBI for installation and configuration. Each MEG is associated with a particular plug-in. When loaded, a plug-in creates and registers its MEGs with WBI, which maintains a pool of available MEGs. This arrangement allows several Web applications to work together simultaneously. Figure 7 illustrates three plug-ins that provide MEGs to WBI. WBI instantiates registered plug-ins at startup. These plug-ins then register MEGs with WBI, along with their conditions for firing. When a request comes into WBI, it routes the request through the various MEGs, according to the conditions. The final

Table 1 The basic MEG building blocks used to build intermediary applications

MEG Type	Input	Output	Action	Examples
Request editor	Request	Request	Modify request	Redirect request to new URL, modify header, insert form information, add or remove cookies
Generator	Request	Response	Produce response or reject request	Read response from local file, forward request, dynamically compute response (as CGI), compose response from database, produce control page, such as "document moved"
Response editor	Request and response	Response	Modify response	Add annotations, highlight links, add toolbars, translate response, change form data, add scripts
Monitor	Request and response	None	Receive request and response, perform computation	Gather usage statistics, record user trail, store documents in a cache, record filled-out forms

result is returned to the client. When WBI handles a transaction, MEGs are selected according to their conditions. WBI only considers the pool of MEGs and is unconcerned with which plug-ins created them.

The WBI architecture can be used to build many different applications. Here, we sketch three: a Web server, a caching proxy, and document transcoder.

Web server. A Web server is an intermediary that takes a request from a client and produces a document in response. More precisely, an HTTP request is made of a server by connecting to the HTTP port (typically port 80) of the server and issuing a command to GET a specific file. From the point of view of WBI, this amounts to a request whose URL does not specify a host from which to obtain the file (as in an ordinary proxy request). Thus, a WBI plug-in can implement a simple server by attaching a generator to a rule that checks the URL of the request for a host. If no host is specified, the generator can use the path name of the URL as the name of a file in the local file system, returning the file if it is available or returning an error code if it is not. If a host is specified, WBI can invoke the default generator to pass the request along to that host.

Caching proxy. A caching proxy is also easy to implement as a WBI plug-in. In this case, several MEGs must work together to monitor transactions to store retrieved documents and to generate documents from a local store if available. More precisely, the monitor maintains a database of document contents (the cache) that is indexed by URL. To create this cache, the monitor triggers on an essentially empty (i.e., always true) rule, storing data whenever a document is retrieved. Of course, if the content of the URL is already stored in the cache (with the same meta-information, such as date and expiration), the monitor simply does nothing. The generator triggers on an empty rule as well, checking the requested URL against the URLs contained in the cache. If none is found, the generator rejects the request, and WBI chooses the next highest-priority generator from the list of matching generators (possibly invoking the default generator to retrieve the document from the Web). If the requested URL is found in the cache, the generator can simply send the stored data back to the client.

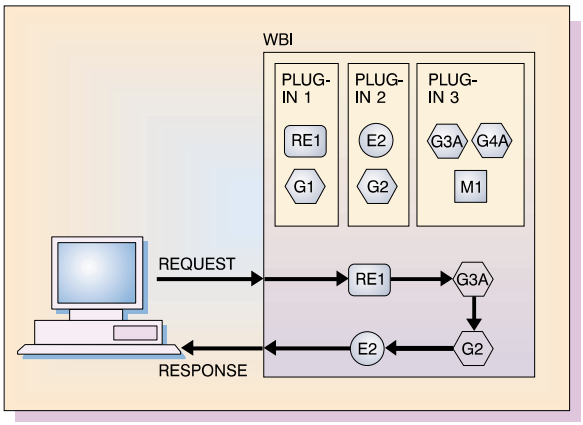
Transcoder. Transcoding might seem more complicated but is nonetheless straightforward to implement in WBI. Recall that transcoding is the process of converting one type of document into another type of document. For instance, it might be convenient to convert Microsoft Word** documents to HyperText Markup Language (HTML) documents so that Web browsers can display them, or to convert images containing millions of colors to images containing four bits of gray scale to save network bandwidth. The trick is to use a WBI document editor to read from the original information stream and to write

to the new converted information stream. To convert a Microsoft Word document to HTML, the editor would trigger on a rule that matches either the content-type of the response (e.g., "x-application/msword") or the file extension of the URL (e.g., "*.doc"). This editor can then use whatever software is necessary to effect the conversion, such as invoking the proper transformations using OLE (object linking and embedding) objects built into Microsoft Word. The WBI infrastructure handles all the plumbing, allowing the application writer to concentrate on application details rather than on HTTP.

Generalized intermediary architecture. The WBI framework can be used to build intermediaries for any unidirectional transaction stream. To demonstrate this, we are currently developing a generalized version of WBI for streams other than HTTP. To handle other protocols, this version of WBI has a modular connection component and a modular protocol component. The connection component accepts connections from the client, whether they are on a Transmission Control Protocol/Internet Protocol (TCP/IP) socket, an RS-232 port, or even a telephone line. The protocol component interprets the signals that come across the connection, parsing them into requests. The request is then parameterized into a set of attribute-value pairs that describe the request. These attribute-value pairs are fed into the rule engine that determines how the MEGs are interconnected to process the request and produce a response. For example, an HTTP request has attributes such as Method (e.g., GET, POST), URL, User-Agent, Client IPAddress, and so on. A telephone request might have attributes such as Action (e.g., Off-Hook, KeyPress) and Value (e.g., "#").

Although the WBI architecture seems a reasonable model for handling unidirectional transaction streams, it is not obvious how to extend it to handle all other sorts of streams. Bidirectional transaction streams might be a straightforward extension of WBI. In this case, because both endpoints can initiate transactions, an intermediary process must listen for requests on both sides. A request handler can then process each type of request. One difficulty is concurrency control between transactions on each side—the intermediary framework might need to provide mechanisms for maintaining a consistent state so that transactions do not interfere. This is a problem for the unidirectional case, but it is amplified in the bidirectional case.

Figure 7 Three plug-ins provide MEGs to WBI



Message-based information streams are more difficult to handle within a general framework because there are fewer constraints on the possible sequence of messages. The transaction case provides a constrained type of message in which requests and responses can be easily identified. But many information streams are more open-ended than simple request-response pairs. One unsatisfying workaround is to consider nontransactional messages to be requests that have a null response. In fact, the WBI framework does not require that a response be generated for each request; the transaction can simply be terminated with no response sent back to the client.

Message streams. Although considering messages to be transactions with null responses is in theory complete, there are better ways to think about message-based streams. An intermediary connected to such a stream can: (1) pass a message through unchanged, (2) block a message from being passed through, (3) send one or more new messages back to the originator of the message, or (4) send one or more new messages along to the destination. These basic functions can be combined to achieve any result. The central problem is to design the state machine to control the selection of operations.

We envision a framework built out of message handlers (MHs), which correspond to MEGs in WBI. Like MEGs, each MH has a rule that determines when it is to be involved in the information stream. This rule can depend on characteristics of the message as well as the state of the intermediary. Such state variables might have different scopes, such as global and

stream, so that the state of the entire intermediary and the state of the current information stream can be handled independently. They will also need to be scoped by plug-in to avoid interference between multiple applications that are sharing the same intermediary. Finally, messages typically have different levels of granularity. Some MHs will wish to operate on messages at a fine granularity and others at a coarse granularity. For example, a log-in may correspond to three messages: (1) client gives user identification to server, (2) client gives password to server, and (3) server confirms or denies log-in. Some MHs may wish to operate on these messages individually, others may wish to aggregate the user identification and password messages together, and still others may wish to aggregate all three messages together into one log-in transaction. It would be helpful for the intermediary framework to offer mechanisms for MHs to identify useful aggregations and mechanisms for MHs to trigger on and operate on aggregated messages.

In summary, the WBI architecture provides a basis for modeling intermediary computation more generally. Though WBI was originally created to handle HTTP streams, it is a straightforward extension to handle other transaction streams. Message-based streams, however, require many more modifications to the WBI model.

Intermediary-rich information streams

Intermediaries can play a substantial role in coordinating multiple information streams. By integrating a variety of functions into intermediary processes, systems become more modular, more extendible, and more interoperable. For instance, suppose a user named Susan has a telephone intermediary that communicates with her PalmPilot synchronization intermediary. The PalmPilot intermediary monitors modifications made to Susan's address book, which contains the names and phone numbers of her main contacts. These names and phone numbers are communicated to the telephone intermediary, which stores them. When Susan picks up her telephone, rather than being connected directly to the telephone line, her telephone intermediary sends a distinctive dial tone. Rather than looking up a number or starting up a computerized dialer, Susan simply says, "Dial Bob Smith." The telephone intermediary performs a speech recognition transformation, effectively translating Susan's audio information stream into a text stream. The text stream is taken as a request, which a request editor in turn transforms into "Dial 555-1212." A generator then forwards that request to the telephone line, transparently connecting Susan's audio stream to the audio stream of the telephone line. If the number is found to be out-of-date, Susan presses the "#" key to gain the attention of her phone intermediary, and says, "Bob Smith's new number is 555-2121." This request is translated to text and then routed to a generator that communicates the change to Susan's PalmPilot intermediary, which in turn sends back a text response that is translated into audio by a speech synthesizer: "I will remember that Bob Smith's new number is 555-2121." The next time Susan synchronizes her PalmPilot, her intermediary will inject the correction into the information stream so that her system is kept up-to-date.

When Susan is away from her telephone, she can simply call her telephone to gain access to this functionality. Her intermediary answers (it also acts as her answering machine), allows Susan to authenticate herself, and then provides her access to all of her familiar features through a second phone line.

Suppose further that Susan's telephone intermediary is connected to her Web browser intermediary. When Susan calls various people, her telephone intermediary identifies the phone numbers dialed by observing her dialing and by looking up the numbers in her company's on-line directory. It also records spectral analyses of the peoples' voices so that it can identify them by their speech patterns. Now when Susan is on a conference call, she can view a "current phone call" Web page that is produced by her Web intermediary. This dynamically generated page receives its information from her telephone intermediary, showing the directory information for each person on the call and highlighting each person when they speak so she can keep up with the conversation.

Finally, suppose Susan's telephone intermediary can also communicate with her e-mail intermediary. Now when she picks up her telephone (or calls her telephone from another location), she is greeted by a "You've got mail" message rather than her normal dial tone. Susan's e-mail intermediary keeps her telephone intermediary up-to-date on her e-mail inbox. It can now retrieve her messages and read them to her over the phone.

These scenarios rely on straightforward combinations of intermediaries and information streams. As must be apparent, much value can be gained by in-

terconnecting different information streams through intermediaries.

We have been developing message-oriented blackboard systems[20] that can serve as the clearinghouse for information to be shared among intermediaries on different streams (see also Wyckoff et al.[21]). Such a blackboard system enables intermediaries to post messages to be routed to interested parties without requiring all interested parties to be available at the same time.

Pervasive computing streams

As computers become more pervasive—embedded in gas pumps, coffee machines, automobiles, televisions, or other common everyday devices—intermediary computation will play an even larger role in the design of future systems. For example, gas pumps formerly simply delivered gasoline, but now they also accept cash payments, perform electronic credit card transactions, and print receipts. It is not practical to replace a modern gas pump to add a new capability, such as performing ATM (automatic teller machine) transactions, automatically selecting the user's preferred octane level, or providing selected gasoline additives. Unless all possible uses have been anticipated, an embedded device must be extended to take advantage of new situations and new opportunities. We believe that intermediaries provide the right hooks to easily extend pervasive computers.

Note that the user interface of an embedded system communicates with the physical device through messages that indicate the user's commands, the state of the device, and available options. To personalize a gas pump by automatically selecting the user's preferred octane rating, an intermediary must understand how to modify the information stream between user interface and physical pump to set the octane rating. In a computerized coffeemaker, for instance, data flows from the interface that the user interacts with to control brewing temperature and starting time to the machinery that actually brews the coffee. For the coffeemaker, intermediaries along the path from user interface to brewing machinery can add functions not envisioned by the original designers, such as the ability to brew half-decaffeinated and half-regular coffee. All embedded computational devices must be separable into at least user interface and mechanical components, which correspond to origin and destination endpoints. The key is to break the single pervasive computing device into two endpoints and a data flow.

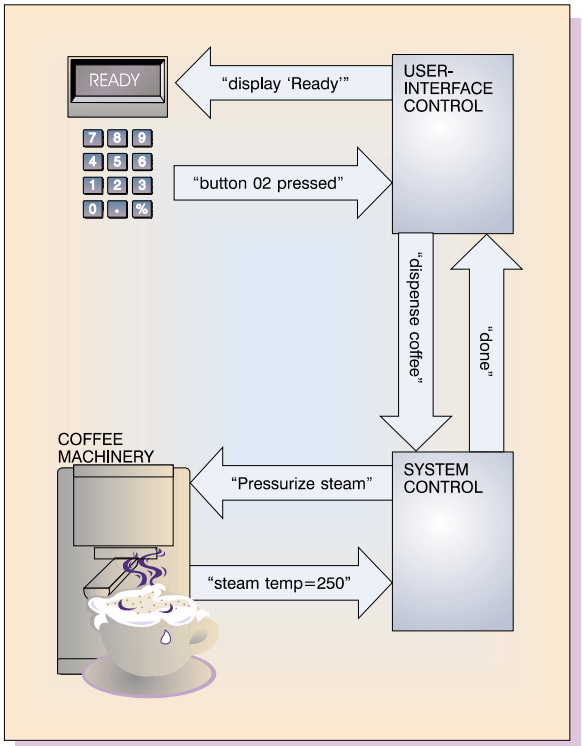
To examine the role of intermediaries in pervasive computing devices, consider an automatic cappuccino maker that is designed to offer the single-shot or double-shot options, the more-or-less steamed milk option, and the regular or decaffeinated option. It cannot offer triple-shots or half-regular and half-decaffeinated coffee, though the mechanics of the

We believe that intermediaries provide the right hooks to easily extend pervasive computers.

machine are capable of such features. For the cappuccino machine, there is at least a data flow between the user interface controls and displays and the control electronics that govern the functioning of the machine. This flow consists of three layers: physical, protocol, and data. The physical layer includes the wires and associated electronics that convey digital data between user interface and control electronics. The protocol layer defines a mechanism by which messages can be transferred between user interface and control electronics. The data layer defines the semantics of these messages, such as "grind coffee beans," "dispense an eight-ounce cup," or "display 'Please Wait'." The data flow provides a place within the device for modifying the operation of the device.

Given an information stream, intermediaries can be introduced to add new functions to the system. Note that even simple devices might have several different data flows for a given transaction. For example, pressing buttons on the cappuccino machine could produce a series of messages that say, "button 01 pressed," "button 23 pressed," and "button 12 pressed." When such a message is received, other messages might be sent along another data flow that says, "brew eight-ounce cappuccino with extra steamed milk." When this message is received, another series of messages might be sent along yet another data flow that say, "grind one shot of regular coffee," "tamp grounds," "pressurize steam," "turn on steam feed," and so on. Figure 8 shows information flows inside a hypothetical cappucino machine. The endpoints are the user controls and the coffee

Figure 8 Flows of information inside a hypothetical cappuccino machine



machinery. The user-interface controls and system controls act as intermediaries.

Adding automatic payment to the cappuccino machine would be straightforward if the machine were capable of having an intermediary added to it. When the "brew eight-ounce cappuccino with extra steamed milk" message came along, the intermediary would note the message but not forward it to the control electronics. Instead, it would send a message back to the display that said, "Display 'Swipe card'." The card reader could simply be a peripheral device connected to the intermediary. When the card had been read, the intermediary could deduct the payment from the user's account and then forward the "brew" message along to the control electronics.

There are many other ways to extend the machine through the use of intermediaries. For example, many people always order the same drink from the cappuccino machine. Instead of going through several layers of menus to define the desired product, the customer could simply swipe a card through the card reader of the machine and press a button confirming that the regular order is desired. Intermediaries can bypass the normal user interface using this shortcut by issuing the appropriate commands to the control electronics.

Intermediaries effectively open up pervasive computing devices, establishing new ways to manipulate data within the data flows of the device. Thus, intermediaries can adapt such devices to particular users and their needs and can enable functions that were unanticipated when the device was built.

Future work and summary

Our work with intermediaries has just begun. The WBI architecture provides only a single model for intermediary computation on a single sort of information stream. As mentioned, we are extending this model to other streams, such as those that are message-based, and to handle protocols other than HTTP. As we continue to explore other information streams, we will undoubtedly uncover additional assumptions built into the preliminary analysis provided here. For instance, the HTTP stream is both stateless and sessionless, enabling WBI to use a simplified transaction model. To create intermediaries for an information stream such as net news (i.e., using NNTP[22]) that requires session identifiers means building far more machinery into the intermediary to manage this state. The case of information streams that have more than two endpoints, such as Internet Relay Chat (IRC[23]), also complicates the issue.

In summary, intermediaries can turn ordinary information streams into *smart* streams that enhance the quality of available information and provide additional services. Because information flows everywhere today, there is a new opportunity for taking advantage of intermediary computation. Until now, however, general principles of intermediaries had not been developed. In this paper, we have taken only the first steps toward creating a general method for injecting intermediary computation into information streams. Much work remains to be done.

**Trademark or registered trademark of 3Com Corporation, Go2Net, Inc., Yahoo! Inc., Compaq Computer Corporation, X/Open Company, Ltd., or Microsoft Corporation.

Cited references

1. A. Luotonen, *Web Proxy Servers*, Prentice Hall, Englewood Cliffs, NJ (1997).

2. P. S. Yu and E. A. MacNair, "Performance Study of a Collaborative Method for Hierarchical Caching in Proxy Servers," *Computer Networks and ISDN Systems* **30**, 215–224 (1998).
3. A. Fox and E. Brewer, "Reducing WWW Latency and Bandwidth Requirements by Real-Time Distillation," *Proceedings of the 5th International World Wide Web Conference* (1996).
4. *Portico: It's magicTalk*, General Magic, Inc., available as http://www.generalmagic.com/portico/portico.html.
5. *Travelocity*, Travelocity, Inc., available as http://www.travelocity.com/.
6. B. A. Nardi and V. O'Day, "Intelligent Agents: What We Learned at the Library," *Libri* **46**, 59–88 (1996).
7. J. Meyers and M. Rose, *Post Office Protocol—Version 3*, RFC-1939, IETF Network Working Group (1996).
8. T. Berners-Lee, R. Calliau, A. Luotonen, H. Frystyk-Neilsen, and H. Secret, "The World Wide Web," *Communications of the ACM* **37**, No. 8, 76–82 (1994).
9. *MetaCrawler*, Go2Net, Inc., available as http://www.metacrawler.com/.
10. *Yahoo!*, Yahoo! Inc., available as http://www.yahoo.com/.
11. *AltaVista*, Compaq Computer Corporation, available as http://www.altavista.com/.
12. H. Lieberman, "Letizia: An Agent That Assists Web Browsing," *International Joint Conference on Artificial Intelligence* (1995), pp. 924–929.
13. T. Joachims, D. Freitag, and T. Mitchell, "WebWatcher: A Tour Guide for the World Wide Web," *Proceedings of the International Joint Conference on Artificial Intelligence* (IJCAI'97) (1997).
14. T. W. Malone, K. Y. Lai, and C. Fry, "Experiments with Oval: A Radically Tailorable Tool for Cooperative Work," *ACM Transactions on Information Systems* **13**, 177–205 (1995).
15. B. Rhodes and T. Starner, "Remembrance Agent: A Continuously Running Automated Information Retrieval System," *Proceedings of the First International Conference on the Practical Application of Intelligent Agents and Multi-Agent Technology* (PAAM '96) (1996), pp. 487–495.
16. K. Lang, "NewsWeeder: Learning to Filter Netnews," *Proceedings of Machine Learning* (1995).
17. R. Barrett, P. P. Maglio, and D. C. Kellem, "How to Personalize the Web," *Proceedings of the Conference on Human Factors in Computing Systems (CHI '97)*, ACM Press, New York (1997).
18. R. Barrett and P. P. Maglio, "Intermediaries: New Places for Producing and Manipulating Web Content," *Computer Networks and ISDN Systems* **30**, 509–518 (1998).
19. P. P. Maglio and R. Barrett, "How to Build Modeling Agents to Support Web Searchers," *Proceedings of the Sixth International Conference on User Modeling*, Springer-Verlag, New York (1997).
20. G. M. Underwood, P. P. Maglio, and R. Barrett, "User Centered Push for Timely Information Delivery," *Computer Networks and ISDN Systems* **30**, 33–41 (1998).
21. P. Wyckoff, S. W. McLaughry, T. J. Lehman, and D. A. Ford, "T Spaces," *IBM Systems Journal* **37,** No. 3, 454–474 (1998).
22. M. Horton and R. Adams, *Standard for Interchange of USENET Messages*, RFC-1036, IETF Network Working Group (1987).
23. J. Oikarinen and D. Reed, *Internet Relay Chat Protocol*, RFC-1459, IETF Network Working Group (1993).

Accepted for publication April 21, 1999.

Rob Barrett *IBM Research Division, Almaden Research Center, 650 Harry Road, San Jose, California 95120 (electronic mail: barrett@almaden.ibm.com).* Dr. Barrett holds B.S. degrees in physics and electrical engineering from Washington University in St. Louis, and a Ph.D. in applied physics from Stanford University. He joined IBM Research in 1991, where he has worked on magnetic data storage, pointing devices, and human-computer interactions in large information systems.

Paul P. Maglio *IBM Research Division, Almaden Research Center, 650 Harry Road, San Jose, California 95120 (electronic mail: pmaglio@almaden.ibm.com).* Dr. Maglio holds an S.B. in computer science and engineering from the Massachusetts Institute of Technology and a Ph.D. in cognitive science from the University of California, San Diego. He joined IBM Research in 1995, where he studies how people use information spaces, such as the World Wide Web.

Reprint Order No. G321-5709.