Intermediary-based transcoding framework

by S. C. Ihde
P. P. Maglio
J. Meyer
R. Barrett

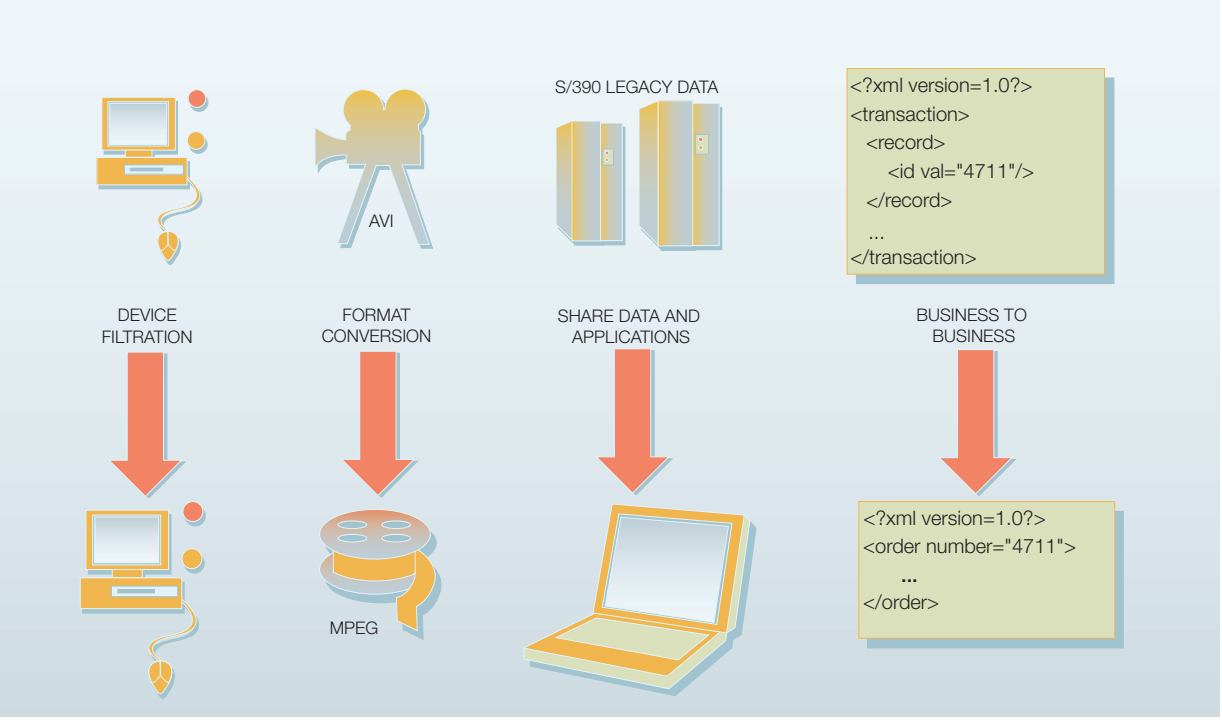
With the rapid increase in the amount of content on the World Wide Web, it is now becoming clear that information cannot always be stored in a form that anticipates all of its possible uses. One solution to this problem is to create transcoding intermediaries that convert data, on demand, from one form into another. Up to now, these transcoders have usually been stand-alone components, converting one particular data format to another particular data format. A more flexible approach is to create modular transcoding units that can be composed as needed. In this paper, we describe the benefits of an intermediary-based transcoding approach and present a formal framework for document transcoding that is meant to simplify the problem of composing transcoding operations.

Today, content providers on the World Wide Web (WWW) are under constant pressure to make information available in a variety of formats and for a variety of purposes. For example, the Yahoo!** catalog server provides information formatted in HyperText Markup Language (HTML) for standard Web browsers, and also provides some of this information formatted for handheld devices such as Palm Pilots** and wireless phones. In this case, content is formatted differently for displays that have different capabilities, and is also delivered differently for devices that have different connectivity. Concern for network bandwidth limitations in particular has spurred many projects aimed at minimizing the amount of data transmitted for Web transactions. For instance, Fox and colleagues[1–3] developed a proxy-based architecture for distilling or transforming content so that thin devices receive only the data they can handle (e.g., devices with monochrome displays do not receive color images), thus minimizing the network bandwidth needed to transmit information.

Moreover, as more and more companies explore the WWW as a place to do business, large amounts of information of various types and in a variety of formats will be made available on the Web. This leads to the problem of converting data to enable applications to handle data that might come from a variety of sources. In this context, bandwidth limitations or client resources (e.g., CPU power and disk space) are not a major concern. The main questions here are: what form is the information in; what form does it need to be in? The ability to convert content from one form to another lets systems that use different languages and conventions communicate and interoperate. The Extensible Markup Language[4] (XML) is particularly suited to the needs of businesses to convert data from one form to another, as it provides means for specifying semantic structure.

The process of converting, distilling, or transforming content is often referred to as *transcoding*[1,5] (see Figure 1). In particular, this term is also used when referring to algorithms for transforming certain data, such as images and movies, from one format into another. For example, video transcoding is the process of converting between different compression formats, or of further reducing the bit rate of a previously

Figure 1 Transcoding intermediaries connect a wide variety of devices, applications, and services.

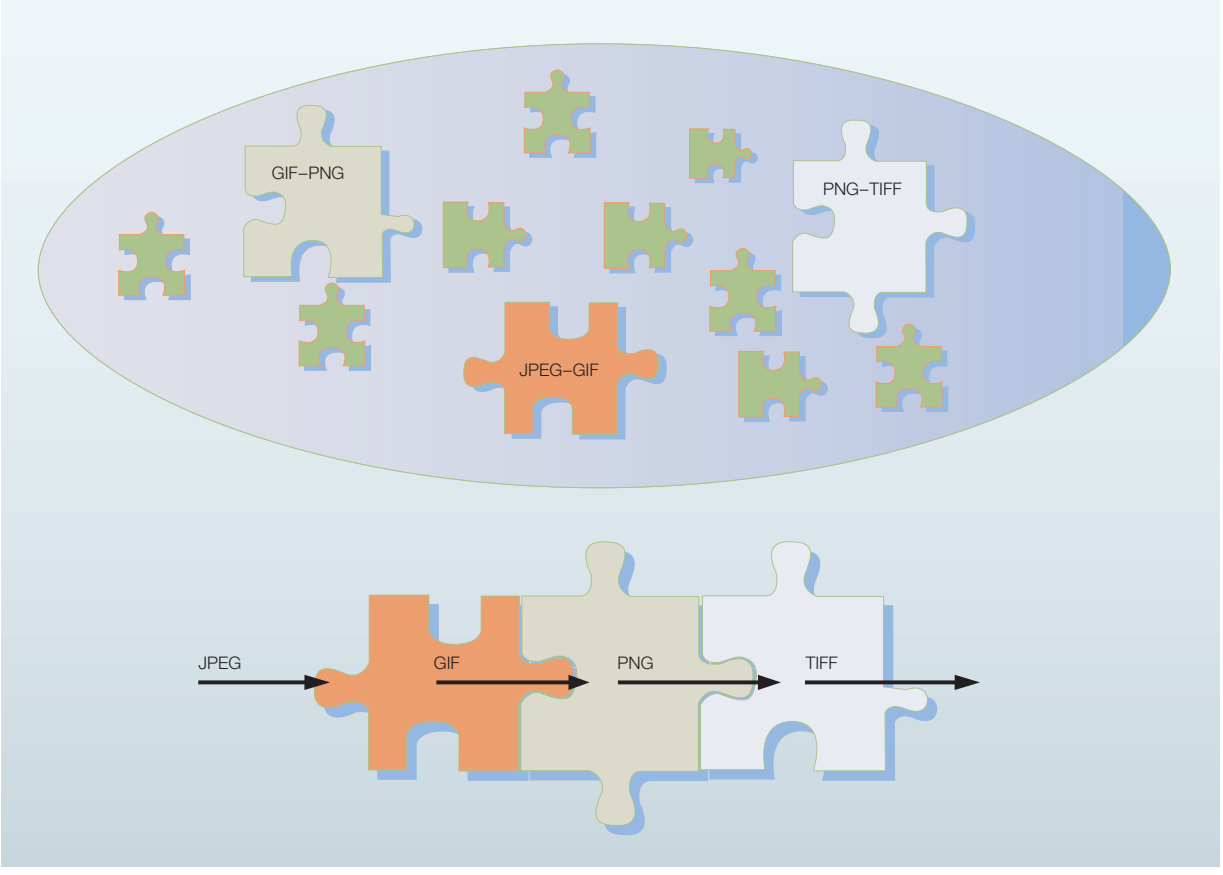


compressed signal.[6] Character transcoding is the process of translating characters from one encoding (e.g., EBCDIC, or extended binary-coded decimal interchange code) to another (e.g., to ASCII, or American National Standard Code for Information Interchange). Image transcoding is the process of converting an image from one format (e.g., JPEG, or Joint Photographic Experts Group) into another (e.g., to GIF, or graphic interchange format), and possibly modifying some of its properties, such as size, resolution, or color depth.[7] The main motivation behind these sorts of transcoding operations is to overcome network constraints, such as limited bandwidth, and to allow clients with limited resources (e.g., processing power, display size) access to Web content.[7,8] Many transcoding operations do not usually alter the semantics, or meaning, of the object being transcoded. However, certain kinds of transcoding operations may alter the semantics, at least to some extent. For example, lossy compression of an image, or summarization of a document, may result in an object that is very different, even unrecognizable when compared to the original.

In general, transcoding operations are applied on demand, rather than precomputed and stored. The simple reason is that it is difficult and expensive to anticipate all possible transformations. As the requirements for transcoding operations increase—new document types to be converted, parameters set for transcoding, and so on—building specialized transcoders will become more and more complex. It seems clear to us that a simpler solution is to develop a framework for combining or composing simple transcoders to perform more complex jobs. For example, given three image converters, GIF to JPEG, JPEG to PNG (Portable Network Graphics), PNG to TIFF (tagged image file format), our hope is to make it easy to convert GIF to TIFF by combining (or chaining) these three (see Figure 2). A system that merely applies specialized transcoding operations would require the implementation of many transcoders that anticipate all possible combinations of input and output formats (see Figure 3).

Because transcoding operations apply to data flows, it is natural to view them as a kind of intermediary-

Figure 2 A transcoding framework simplifies the tasks of combining and reusing transcoders.

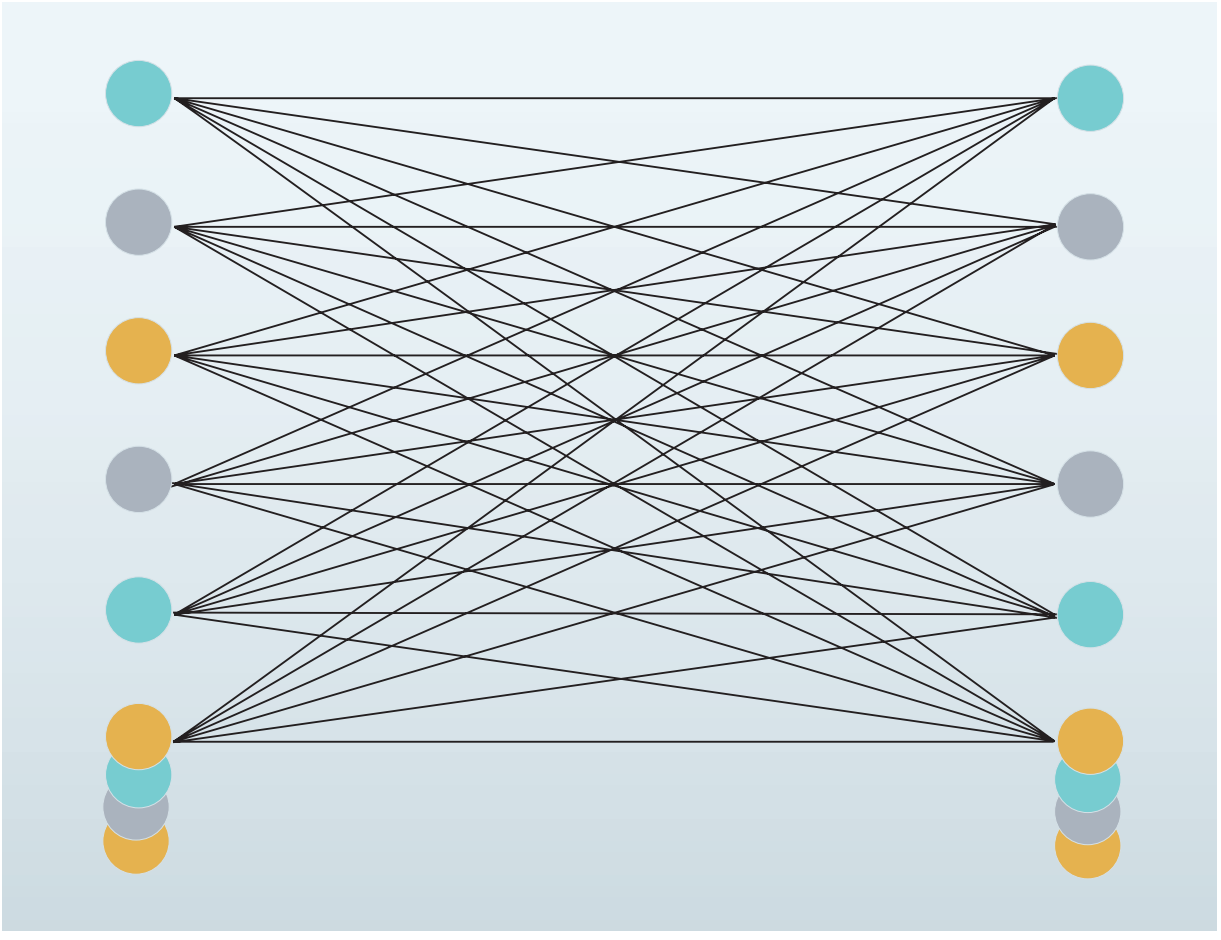


based computation.[9–11] Of course, final results and partial results of intermediary-based transcoding can be cached for quick access later.[2] Our point is that all possible transcodings are not generally precomputed and stored as if they are simply alternate formats for maintaining content. Transcoding is a fundamentally active process of recasting content when it is needed, and therefore lends itself to implementation as an intermediary process. This raises the issue of how to build transcoding applications as intermediaries. As mentioned, the approach of chaining transformations seems reasonable, but how can a system guarantee that any chain produces the desired result?

Gribble and Fox's DOLF[12] system is an intermediary-based scheme that automatically adapts content for particular clients. Specifically, DOLF converts a document's format—that is, its Multipurpose Internet Mail Extensions (MIME) type[13–15]—to one that is appropriate for the client. DOLF can create chains of MIME-type transcoders to convert documents of one type to documents of another type. To create the proper chain, DOLF must know which transformations its transcoders can perform, but DOLF's understanding of the process is limited to just MIME type transformations. Thus, DOLF itself cannot be used to summarize a document or translate from one language to another, because these transformations do not affect a document's MIME type.

The transcoding framework we have developed is similar to DOLF in several ways. Most notably, both systems perform dynamic transformations on requested objects by composing a pipeline of computational elements. However, our framework includes a language for describing in detail the abilities of the individual transcoders and mechanisms for packag-

Figure 3 Without the ability to compose operations, each transcoding operation must be preprogrammed, leading to a combinatorial explosion of transformations.



ing transcoders so they can be added or removed at run time. The language allows transcoders to specify semantic transformations on objects regardless of whether the transformation preserves or alters type. This allows a richer set of operations to be described formally. Within such a framework, different parties can supply feature-rich transcoders that can cooperate automatically.

In the next section of this paper, we describe our transcoding framework in some detail, including a formal specification of our language for defining and manipulating transcoding operations. Next, we present a specific intermediary-based implementation of our transcoding framework. Finally, we outline future directions for this work.

A transcoding framework

Before describing our framework, we first define some terms. Intuitively, *data objects* represent content, *type* indicates the form the data are in, *properties* represent attributes of particular data types, and *format* combines a type and a set of properties. More precisely,

- A data object consists of a sequence of bytes.
- A type is one of the MIME types.[14,15] It represents the semantic nature of data objects of that type and specifies the byte-level encoding used to represent the data. For example, data objects of the type "image/gif" are images in which the data are encoded according to the GIF format.

- A property is an attribute particular to a type or set of types that may take on values represented as strings. The special value "*" represents any value. For example, the type "text/xml" might have the property "DTD" (document type definition), which could take on values such as "http://www.w3.org/TR/1999/PR-xhtml1-19990824," "foo," or "*."
- A format consists of one type and zero or more property-value pairs. For example, ("text/xml," (("DTD," "foo"))) is a valid format, as is ("text/html," (("case," "upper"), ("length," "500"))).
- A transcoding operation takes an input data object d_{in} in a particular format f_{in} (we use the notation $d_{in}(f_{in})$) and converts it into an output data object d_{out} in the specified format f_{out} (using again the notation $d_{out}(f_{out})$). Thus, we denote a transcoding operation by

  $$(d_{in}(f_{in}), (f_{in}, f_{out})) \rightarrow d_{out}(f_{out})$$

  The type of the input format f_{in} and the type of the output format f_{out} may be identical. If so, and if the set of properties specified by f_{out} is empty, the transcoding operation is the identity transformation and d_{out} is identical to d_{in}.

Although several examples of properties are given in this paper, it is our intent that the ontology for these be extensible and *ad hoc*. Thus, for transcoders from different authors to be able to interoperate, authors must use the same name for the same property. Properties describe transformations of a more semantic nature than those the MIME type system covers. These transformations may or may not be reversible. For example, text summarization is not reversible, nor is increasing the compression factor in a JPEG image, because both transformations discard information. Other transformations may be reversible, such as inverting the color map of an image. In any event, all such transformations can be described by properties in our system, because they do not affect the type of the object.

We define *intermediaries*[9–11] as a general class of computational entities that act on data flowing along an information stream. In general, intermediaries are located in the path of a data flow, possibly affecting the data as they flow by. For HyperText Transfer Protocol (HTTP) streams, for instance, intermediary computation might be used to customize[11,16,17] content with respect to the history of interaction of individual users, to annotate[16,18] content, such as marking up URLs (uniform resource locators) to indicate network delay, to add awareness and interaction in order to enable collaboration[19,20] with other users, to transcode[7,21] data from one image type to another, or to cache and aggregate content.[2]

In general, we distinguish six broad applications of intermediary computation: customization, annotation, collaboration, transcoding, aggregation, and caching. More precisely, we distinguish these intermediary functions by the kinds of information they take into account when performing their actions. Customization takes account of information about the user or the user's environment when modifying data on the stream, such as adding links the user visits often. Annotation takes account of information about the world outside the user or user's environment, for instance, by determining link speed. Collaboration takes account of information about other users, such as what Web page they are currently visiting. Transcoding takes account of information about the data's input and desired output format, for instance, transforming a JPEG format to a GIF. Aggregation takes into account additional data streams, for instance, merging results from several search engines into a single page of search results. Caching takes account of when data were last stored and last changed.

Our transcoding framework is implemented as an intermediary for a well-defined protocol for document or object retrieval, such as HTTP or the Wireless Application Protocol (WAP). This intermediary can inspect or modify both requests for objects and the responses, that is, the objects themselves. The intermediary performs transcoding operations on these objects. Clients may request that the intermediary transcode responses on its behalf, or the intermediary may make that decision itself based on other factors. To perform this transcoding, the intermediary relies on a set of transcoders, each of which *advertises* its capabilities. The advertisements specify what sort of output object a transcoder is able to produce given certain constraints on its input. A transcoder that translates Japanese Web pages into English might specify

```
((text/html, (("language", "ja"))),
 (text/html, (("language", "en"))))
```

A "*" on the right side of an (attribute, value) pair indicates that the transcoder can produce any value requested for the corresponding attribute. A transcoder can advertise more than one capability.

Figure 4 Simple BNF grammar for the request and capability language. We distinguish between a transcoding request (TR) and capability advertisement (CA).

A transcoding request (TR) specifies a format pair (FP) describing the input format and the desired output format.

```
TR := FP
```

A capability advertisement (CA) contains one or more format pairs describing the supported transcoding operations by a particular transcoder.

```
CA := FP+
```

A format pair describes two formats (F), which are described by a mimetype and an optional key value list (KVL). The terminal <mime-type> denotes the standard MIME types such as "text/plain" or "application/pdf".

```
FP := "(" F "," F ")"
F  := "(" <mime-type> "0, (" [KVL] "))"
```

The key value list (KVL) contains one or more key value pairs (KVP). The terminals <key> and <value> denote characteristics of a particular MIME type, such as compression ratio for "image/jpeg" or language for "txt/*".

```
KVL := KVP | KVP "," KVL
KVP := "(" <key> "," <value> ")"
```

A simple BNF (Backus Naur form) grammar for the language that we use to describe transcoder capabilities and requests is given in Figure 4.

Once the desired transformation is determined, the capabilities of each transcoder are examined in order to identify a set of transcoders that can perform the requested operation. Once an appropriate set of transcoders has been selected, each one is invoked in turn with two inputs that specify the *transcoding request*: (1) the output of the previous transcoder (or the original input, in the case of the first transcoder); and (2) a *transcoder operation*, which specifies one or more of the operations advertised in a transcoder's capabilities statement. Each transcoder operation includes the input format of the object supplied and the output format of the object to be produced, both of which must respect the transcoder's advertised capabilities.

More precisely, a *transcoding request R is valid for a transcoder T* given that T advertised a set of capabilities $\{C_1, \ldots, C_n\}$ if:

1. There exists at least one capability C_i such that the types specified in the input and output formats of C_i are identical to the types specified in the input and output formats of R. Let the set $\{D_1, \ldots, D_m\}$ denote all members of $\{C_1, \ldots, C_n\}$ that meet this criterion.

2. There exists a subset E of D such that the union of all property-value pairs in the output formats of the members of E is identical to the set of property-value pairs of the output format of R, and the union of all property-value pairs in the input formats of the members of E is identical to the set of property-value pairs of the input format of R, subject to the following conditions:

   a. Any property-value pair in any input format or in the output format of R with a value of "*" is meaningless; it is as though the pair were not present.
   b. Any property-value pair in an output format of a member of E with a value of "*" will be considered identical to a property-value pair in R with an identical property and with any value.

The operations of different transcoders can be composed in a straightforward way. Generally, the idea is to break down an overall request into a series of subrequests to different transcoders, each of which accomplishes part of the overall goal or some other necessary subgoal. Specifically, a list of subrequests $(S_1, \ldots, S_n)$ can be considered equivalent to an overall request R if the following conditions are met:

- The type of the input format of the first request S_1 is identical to the type of the input format of R.
- The type of the output format of the last request S_n is identical to the type of the output format of R.
- Each property-value pair in the input format of R is present in the input format of some subrequest S_i.
- Each property-value pair (P, V) in the output format of R is present in the output format of some subrequest S_j such that there does not exist any

subrequest S_k, $k > j$, whose output format contains a property-value pair (P, V'), $V \neq V'$.

The net effect of these conditions is that every property specified in the output format of a request R may take on any value at various stages throughout the chain, as long as the final value that it takes is the one requested in R.

As noted previously, our framework is similar in many ways to DOLF.[12] However, several key differences distinguish it. In our framework, the use of properties in addition to MIME types allows transcoders to declare their ability to change attributes of an object other than its type. DOLF relies on other, "stacked" proxies[1,2,21] to perform such transformations. By contrast, our system allows type-preserving and type-changing transformations to be automatically combined in any number and order within the scope of a single intermediary. This would be difficult to achieve with a combination of DOLF and a second proxy that communicate only through HTTP. For instance, stacked or chained proxies operate in a predefined order. If one proxy handles type-altering transformations and another one handles type-preserving transformations, the order in which the proxies are stacked dictates the only order in which those transformations may be performed. Of course, it might be possible to give the proxies detailed knowledge of each other. With this knowledge, they could forward requests back and forth until all required transformations are performed. However, one of the advantages of a stacked proxy architecture is that one proxy generally does not know what function the other proxies perform, and may not even know of their existence. This lack of knowledge allows a clean architectural separation of function, but if the functions are heavily intertwined, it makes more sense and is more efficient to combine them in a single intermediary.

In addition, our use of a formally specified language to describe the abilities of transcoders allows transcoders to be packaged and interchanged in a simple and automated way, enabling the creation of transcoder repositories. Thus, an intermediary unable to satisfy a request might automatically search for, retrieve, install, and use an appropriate transcoder. For instance, if IBM creates transcoders that operate between the AFP (Advanced Function Presentation) and SVG (Scalable Vector Graphics) formats, and Adobe creates transcoders that operate between PostScript and SVG, a third-party transcoder repository service could publish these transcoders, enabling any system that knows about the repository to discover, download, and combine the transcoders seamlessly.

Transcoding framework in action. Consider the case of a worker away from the office. Suppose he or she is traveling by car, perhaps making a sales call. Suppose further that this worker's Internet-connected mobile phone can request data from the office via a transcoding intermediary, and that he or she wants to hear an English audio summary of a long document, such as a sales contract. The mobile phone browser requests the document from the transcoding intermediary. The phone-browser knows that the user wants an audio summary, either because it has been preconfigured or through some other means (e.g., because an earpiece is plugged into the phone). Suppose the original document is a PDF (Portable Document Format) document written in Spanish. In this case, the phone-browser might specify its preference for an audio summary by including with its request a header line such as,

```
Transcoding-Request:
((application/pdf, (("language","es"))),
 (audio/mp3, (("summarize", "10.0"),
              ("language", "en"))))
```

To satisfy the request, the intermediary first retrieves the original document from its origin. Because a transcoding specification was included in the original request for data, the intermediary must transcode the data before passing the data along to the client. To satisfy the transcoding request, the intermediary first looks for a single transcoder that can do the job of transforming Spanish PDF documents into summarized, English audio. Because there are no special transcoders for this, the intermediary next tries to find a chain of transcoders that, when applied in sequence, satisfies the request.

The chain of transcoders is determined by simple backward chaining, with the desired output type examined first. If there is no single transcoder that can produce "audio/mp3," then the request cannot be satisfied. If a transcoder is available, the input requirements of the transcoder are examined in order to identify another transcoder that can output a matching type. This process repeats until the input type of the last transcoder selected matches the input type of the original transcoding request. Furthermore, the output format of the original transcoding request must be satisfied by the chain of transcoders.

Let us consider this example more carefully:

1. The desired output type is "audio/mp3" and the only available transcoder that can output "audio" is the following:

```
((text/plain, (("language","en"))),
 (audio/mp3, ()))
```

   The transcoder's capability advertisement states that plain text written in English can be converted into audio. This transcoder will be selected as the last transcoder in the chain.

2. Because the transcoder selected in Step 1 only accepts English input in plain text, the framework must find a transcoder that outputs plain text in English. Suppose a language-translation transcoder is available:

```
((text/plain, (("language","es"))),
 (text/plain, (("language","en"))))
```

   At this point, two jobs remain. First, we must find a transcoder that can summarize text and, second, we must find a transcoder that can convert PDF into plain text.

3. Suppose there are two such transcoders available, a PDF-to-text converter,

```
((application/pdf, ()),
 (text/plain, ()))
```

   and a text summarizer,

```
((text/plain, ()),
 (text/plain, (("summarize","*"))))
```

   One final problem remains: ordering these last two transcoders. If the PDF converter is selected first, the next step would be to find a summarizer that outputs a PDF document. Because there is no such PDF summarizer, the chain of transcoders cannot be completed. Because our framework implements a search process that can backtrack, it can revoke the selection of the PDF converter and select the summarizer, which then leads to the selection of the PDF-to-text converter.

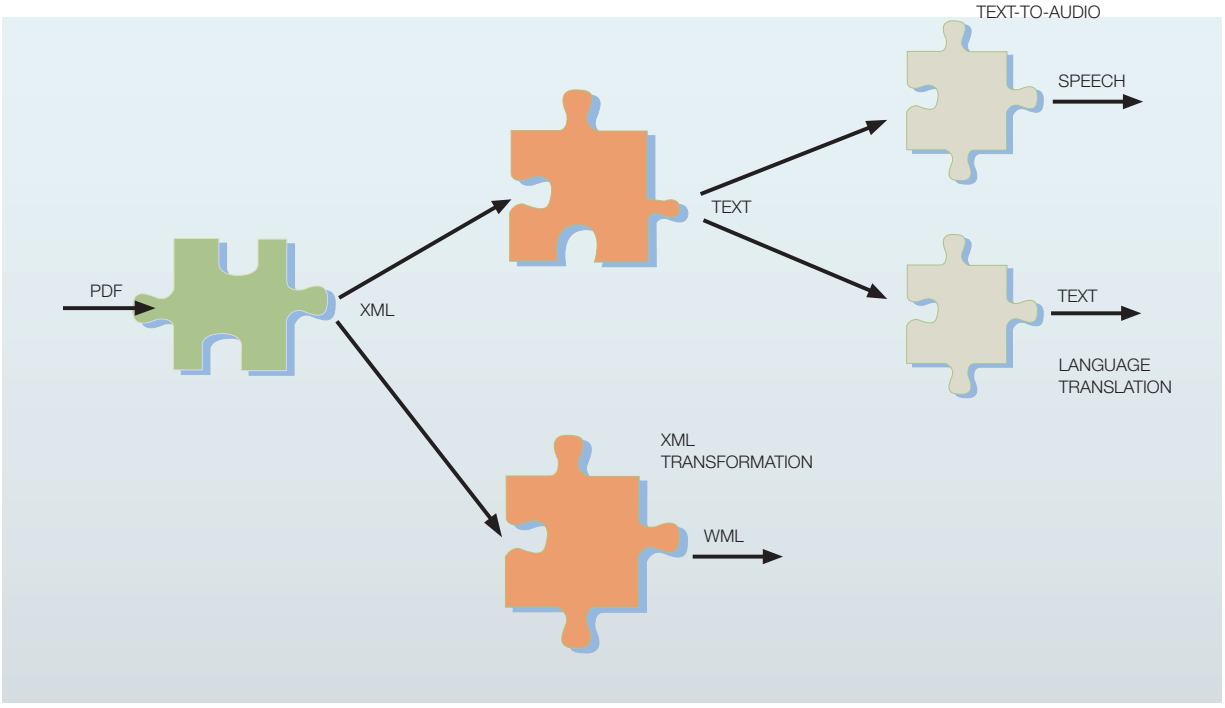
   The overall sequence of transcoding operations for our example request is

```
((application/pdf, ()),
 (text/plain, ()))

((text/plain, (())),
 (text/plain, (("summarize","10.0"))))

((text/plain, (("language","es"))),
 (text/plain, (("language","en"))))

((text/plain, (("language","en"))),
 (audio/mp3, ()))
```

Note that the individual transcoding units in this example are reusable and can be combined in many ways with other transcoders (see Figure 5). The text-summarization transcoder might be used together with a text-to-WML (Wireless Markup Language) transcoder to allow a WML phone to display a summarized document. The text-to-speech transcoder can be used alone to turn text input into audio output. A language translation transcoder can be combined with the text-to-speech transcoder to turn Spanish text into English audio. A PDF document can be transcoded to text in order to take advantage of a summarization or text-to-speech transcoder. Text-to-speech conversion might be used alongside a CAD-to-VRML (computer-aided design to virtual reality modeling language) transcoder that allows one to walk through an immersive, audio-guided tour of a building blueprint.

Limitations of the transcoding framework. Our transcoding framework has two main limitations: (1) the language used by transcoders to express their capabilities is somewhat simplified, and (2) the correct operation of the system depends on the cooperation among the transcoders in setting the properties of the input and output formats. As a result of (1), it is cumbersome for a transcoder to express that it cannot accept input with certain combinations of properties. When a transcoder lists several property-value pairs in a single advertisement, they represent a conjunction of properties. To express disjunction, the properties must be listed in separate advertisements. For example, a transcoder that can accept textual input only in English or German must list all its other restrictions on the input twice, once in conjunction with English, once with German. If a transcoder has several such restrictions on its input, the list of advertised capabilities will quickly become long and unwieldy.

Figure 5 A pluggable, reusable framework allows individual transcoders to be combined in different ways to achieve a variety of results.

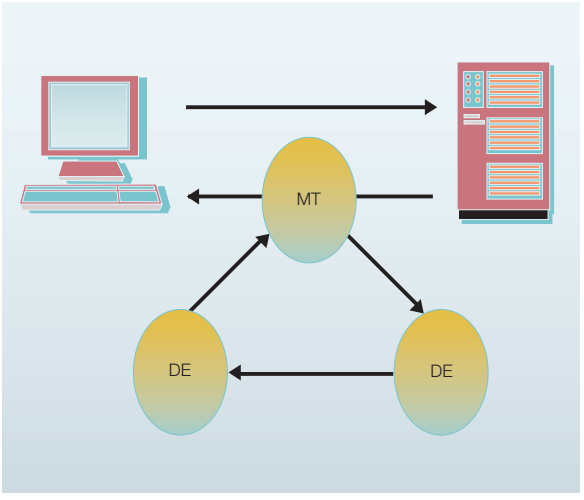


The result of the second limitation of our transcoding framework is that the usefulness of the system as a whole depends on the judicious and correct use by transcoder authors of properties in the input and output formats they advertise. If different transcoders use properties in different ways, or have different policies with respect to when properties should be specified in formats, the system will not function effectively. For example, consider the type "application/xscript," which has three different "levels," "1," "2," and "3." One transcoder might understand all three levels of xscript, and never make any mention of "level" in its capabilities advertisement. Another transcoder might only understand levels 1 and 2, and thus advertise that it can accept "application/xscript" input with the property ("Level," "1") or ("Level," "2"). These two transcoders could not work together effectively on xscript documents because the output produced by the first transcoder does not specify "Level" at all, and therefore cannot be used as input to the second transcoder.

An intermediary-based implementation

The *Web Intermediaries (WBI) Development Kit* is an implemented framework for adding intermediary functions to the WWW.[9–11,16] WBI is a programmable proxy that was designed for ease of development and deployment of intermediary applications. Using WBI, intermediary applications are constructed from four basic building blocks: request editors, generators, document editors, and monitors. We refer to these collectively as MEGs (monitors, editors, generators). Monitors observe transactions without affecting them. Editors modify outgoing requests or incoming documents. Generators produce documents in response to requests. WBI dynamically constructs a data path through the various MEGs for each transaction. To configure the data path for a particular request, WBI has a rule associated with each MEG that specifies a Boolean condition indicating whether the MEG should be involved in a transaction based on header information about the request or response. An application (WBI plug-in) is usually comprised

Figure 6 The master transcoder (MT) chains document editors (DEs). The DEs perform the actual transcoding.



of a number of MEGs that operate in concert to produce a new function.

Because transcoding is an intermediary application, we built our transcoding framework on top of WBI. In particular, the transcoding framework is implemented as a WBI plug-in that consists of several MEGs, specifically, the master transcoder, and various specific transcoders (such as GIF-to-JPEG transcoder, or an XML-to-XML converter based on XSL (Extensible Stylesheet Language) processing, etc.).

In WBI terms, the master transcoder is a document editor that receives the original object (e.g., a GIF image) as input and produces a modified object (e.g., a JPEG image) as output according to some transcoding requirements. The master transcoder intercepts the data flow between client and server. For each object in the data flow, WBI calls the master transcoder so that it may inspect the request and the original object in order to make an appropriate response. If transcoding is necessary, the master transcoder determines the appropriate transcoder or combination of transcoders. The master transcoder arranges for the appropriate transcoders to be subsequently called by WBI in the correct order (see Figure 6).

As mentioned, the means by which the desired output format is determined are external to the master transcoder and beyond the scope of this paper (but see Reference 22 for an approach to determining what output format is desired). To demonstrate a useful system and to keep the implementation simple, we describe some simple policies as though they were implemented in the master transcoder itself. Our current implementation separates this decision from the master transcoder, and the desired output format is communicated to the master transcoder through extra header data attached to the HTTP request stream.

WBI offers various protocol-specific keywords that allow the specification of rules. During transaction processing, the information available about the transaction (e.g., requested URL, host, content type, etc.) is matched against the rules of all registered MEGs. If the rule of a MEG is satisfied, the particular MEG will be the next one in the chain to handle the data stream. The master transcoder is registered with WBI in a way that allows it to inspect every request. In the Java code below:

```
...
MasterTranscoder mt = new
                        MasterTranscoder();
mt.setup( "MasterTranscoder", "%true%" );
...
```

the special rule "%true%" is satisfied in every case, and any MEG specifying this rule is invoked for every request that passes through WBI. Thus, the master transcoder can decide when transcoding is necessary, and if it is, it can then decide on the chain of specific transcoders.

The specific transcoders, which are also document editors, are registered with the master transcoder in order to advertise their capabilities. In addition, the transcoders are also registered with WBI by specifying a rule that determines the conditions under which they will be invoked during a transaction. In addition to the protocol-specific keywords mentioned earlier, an extension mechanism, known as *extra rule keys*,[23] is available. This mechanism is used by the master transcoder to tell WBI which transcoders to invoke and in what order. Extra rule keys consist of a key-value pair that may be set by any MEG. MEGs can register their rules with WBI so that they will be invoked when another MEG sets a certain value for an extra rule key.

A transcoder that converts GIFs to JPEGs might be registered this way:

```
...
GifToJpegTranscoder g2j = new
                    GifToJpegTranscoder();
g2j.setup( "GIF-To-JPEG",
          "$GIF-To-JPEG = %true%", 10 );
...
```

If the master transcoder determines that the GIF-to-JPEG transcoder should be called, it simply sets the extra rule key condition "$GIF-to-JPEG = %true%." WBI will then invoke this transcoder to perform the conversion. This obviously works for single-step transcoding, as all the master transcoder must do is set the special condition, and the rest is done by WBI itself.

Things become a little more complicated when the transcoding request can only be satisfied by a combination of transcoders. In this case, the master transcoder must first determine which transcoders to invoke to accomplish the transformation, and the individual transcoders must then be applied in the proper order. To determine which transcoders are needed, the master transcoder considers the input format, the requested output format, and the advertised capabilities of available transcoders. If a single transcoder is available to perform the operation (transforming the input format to the desired output format), it is simply used. If not, the master transcoder searches for a chain of individual transcoders such that

1. The type of the output format of each transcoder matches the type of the input format of the next transcoder in the chain, and
2. Each property contained in the input format of a transcoder appears with an identical value (or with the "*" wildcard) in the output format of a transcoder in the proper place in the chain (or in the input format of the object itself); that is, the most recent instance of the property in the chain must have the correct value.

It can be shown that the overall request is considered satisfied if a hypothetical transcoder with an input format identical to the requested output format can be added to the end of the chain. Thus, this process implements a simple backward-chaining, state-space search in which the goal state is the output format, the initial state is the input format, and the state-transition operators are individual transcoders.

Once an appropriate chain of transcoders is found, there remains the problem of invoking the transcoders in the correct order. We solved this problem using WBI's *transaction data*, which let MEGs associate arbitrary objects with a transaction (HTTP request/response), allowing MEGs later in the processing chain to use objects (information) generated by previous MEGs. If the transcoding request can only be served by chaining multiple transcoders, the master transcoder simply determines the order of participating operations and stores this information in an object that is then attached to the transcoding request. The master transcoder still sets the condition for the first transcoder in the chain so that WBI can invoke it. The first MEG and each of the following MEGs then set the condition for the next MEG (based on the object stored in the transaction data) until no more MEGs need to be called.

WBI provides various ways for the master transcoder to gather the information it uses to determine the desired output format of an object. One very simple mechanism is to automatically draw conclusions from the HTTP request, such as information about the client that is requesting the data. For example, the following HTTP request could have been issued by a handheld device:

```
GET http://www.ibm.com/image.jpg HTTP/1.0
Accept: */*
User-Agent: tiny-PDA
...
```

The master transcoder interprets this as an indication that the client is a device with limited resources for display and connectivity. Of course, there must be a lookup mechanism to identify transcoding operations with a particular device (see Reference 22) such that the master transcoder can match the device's capabilities, for example, by transcoding each JPEG into a smaller GIF with reduced color depth (monochrome). This saves bandwidth and allows the device to display the image. The User-Agent field is a convenient way to determine standard transcoding operations, such as type conversions or size reduction.

This method can be extended, such that the client specifies allowable or desired transcoding operations in additional HTTP header fields:

```
GET http://www.ibm.com/image.jpg HTTP/1.0
Accept: */*
...
Transcoding-Request:
```

```
        (image/jpg, ()), (image/gif, ())
Transcoding-Request:
                (text/xml, ("DTD","a")),
                (text/xml, ("DTD","b"))
...
```

In this example, the client specifies explicitly which transcoding operations should be performed through additional header fields "Transcoding-Request." In the above request, the client asks to transcode each JPEG into a GIF, and to translate XML documents that correspond to the DTD "a" into XML documents that correspond to the DTD "b." WBI provides the necessary infrastructure for each MEG to access protocol-specific header information.

The mechanisms we have described work for the HTTP protocol but may not work with every HTTP client, much less other protocols, such as FTP or SMTP (File Transfer Protocol or Simple Mail Transfer Protocol). If the protocol underlying a request does not offer such functionality, clients can simply register with the transcoding intermediary and maintain a client or device profile. These profiles are made accessible to the master transcoder, such that it only needs to determine the client during a transaction to perform a lookup and decide whether a transcoding operation is necessary. Of course, such a service requires the transcoding intermediary to provide a mechanism for clients to register their transcoding needs. Other implementations might make transcoding decisions in different ways. For example, rather than having clients register with the transcoder, the transcoder could present a simple form-based interface that would allow a user to request transcoded objects on a per-request basis, or a specific intermediary might always make certain kinds of transcoding decisions based on a mandate to preserve bandwidth.

As previously described, our WBI-based prototype system supports both type and semantic transformations. Although this prototype has demonstrated our framework in action, we have not yet deployed it on a large scale. A full evaluation of the system requires that we examine how transcoders from a variety of sources interact and whether the type and property ontologies we have described are adequate. Furthermore, in an operational environment performance is bound to be an issue. It is possible, for example, that a heuristic search would be needed to construct the chains of transcoders because a simple depth-first or breadth-first search might be too time consuming. Another performance issue concerns the individual transcoders themselves. Though we cannot improve the performance of transcoders supplied by third parties, it may be necessary to take into account the relative performance of individual transcoders while constructing the chains so that we can choose between two equivalent chains on the basis of performance.

Conclusion and future directions

In this paper, we have described a framework and an intermediary-based implementation for transcoding. Our approach is flexible because our framework can be used to seamlessly combine a set of transcoding operations in a way that guarantees conversion from arbitrary input formats to arbitrary output formats. Moreover, by locating transcoding at the intermediary rather than at the server or at the client, our approach enables content conversions that have not been anticipated by the owner or creator of the data. The result of formalizing our framework is that we can combine type transformations with semantic transformations to express a rich set of transcoding operations in a uniform way. Because many types of transformations can be expressed in the same language, it is simply a matter of performing a search and then combining the selected transcoding operations.

There are many opportunities for future work. Our architecture does not currently support certain computational optimizations. The framework could be extended to allow transcoders to advertise the quality of their output or their consumption of computing resources. This would enable the transcoding engine to do a better job of optimizing the data flow through a series of transcoders.

Another direction is to more carefully consider the details of how the master transcoder derives the path through the pool of available transcoders. One can imagine many situations in which more than one chain of transcoders might satisfy a request. How does the master transcoder decide which path to take? Dynamic measurements of the past performance of each transcoder could be considered, or information as to whether type or other conversions are known to be lossy.

In addition, many enhancements can be made to the language that describes transcoder capabilities and requests. A more expressive way of describing patterns might be useful, for instance, one that enables the system to match ranges of numbers. Currently,

each transcoder is free to choose any name it wishes for the properties specified in its input and output formats, leading to the possibility of name space collisions. A mechanism for avoiding such collisions is needed, such as creating a standard ontology for properties.

Finally, the very notion of transcoding raises many intellectual property issues. For example, is it legal to change the format of content owned or copyrighted by others? It is possible that the formal type-transforming and property-transforming distinctions made in our transcoding framework can be used in determining whether copyrighted content has actually been modified. Though such legal issues are clearly beyond the scope of this paper, we feel certain that these will be addressed by legislatures and courts in the near future, for the legal battle has already begun: a group of on-line publishers has sought to stop one transcoding service from modifying its copyrighted content.[24]

Acknowledgments

We thank Stephen Farrell and Ralph Case for helpful discussions on this topic, Jim Jennings for pointing out the need to formalize our transcoding framework, and three anonymous reviewers for many helpful and insightful comments on the initial version of this paper.

**Trademark or registered trademark of Yahoo! Inc., or Palm, Inc.

Cited references and notes

1. A. Fox, S. D. Gribble, E. A. Brewer, and E. Amir, "Adapting to Network and Client Variability via On-Demand Dynamic Distillation," *Proceedings of the 7th International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS-VII)*, Cambridge, MA, ACM, New York (1996).
2. A. Fox, S. D. Gribble, Y. Chawathe, and E. A. Brewer, "Adapting to Network and Client Variation Using Active Proxies: Lessons and Perspectives," *IEEE Personal Communications* **5**, 10–19 (1998).
3. A. Fox, S. D. Gribble, Y. Chawathe, E. A. Brewer, and P. Gauthier, "Cluster-Based Scalable Network Services," *Proceedings of the 1997 Symposium on Operating Systems Principles (SOSP-16)*, Saint-Malo, France, ACM, New York (1997).
4. R. J. Glushko, J. M. Tenenbaum, and B. Meltzer, "An XML Framework for Agent-Based E-Commerce," *Communications of the ACM* **42**, No. 3, 106–114 (1999).
5. M. Hori, R. Mohan, H. Maruyama, and S. Singhal, "Annotation of Web Content for Transcoding," W3C Note, World Wide Web Consortium (1999). Available at http://www.w3c.org/TR/annot.
6. P. N. Tudor and O. H. Werner, "Real-Time Transcoding of MPEG-2 Video Bit Streams," *IEEE Conference Publication of International Broadcasting Convention*, IEEE, New York (1997), pp. 286–301.
7. J. R. Smith, R. Mohan, and C. Li, "Transcoding Internet Content for Heterogeneous Client Devices," *Proceedings of IEEE Conference on Circuits and Systems (ISCAS)*, IEEE, New York (1998).
8. P. Noble, M. Price, and M. Satyanarayanan, "A Programming Interface for Application-Aware Adaptation in Mobile Computing," *Proceedings of the Second USENIX Symposium on Mobile and Location-Independent Computing*, Ann Arbor, MI, USENIX (1995).
9. R. Barrett and P. P. Maglio, "Intermediaries: An Approach to Manipulating Information Streams," *IBM Systems Journal* **38**, No. 4, 629–641 (1999).
10. R. Barrett and P. P. Maglio, "Intermediaries: New Places for Manipulating and Producing Web Content," *Computer Networks and ISDN Systems* **30**, No. 1–7, 509–518 (1998).
11. P. P. Maglio and R. Barrett, "Intermediaries Personalize Information Streams," *Communications of the ACM* **43**, No. 8, 96–101 (2000).
12. S. Gribble and A. Fox, "*Digital Objects with Lazy Fixations*," University of California, Berkeley (1996). Available at http://www.cs.berkeley.edu/~gribble/cs294-5_digdoc/project.html.
13. MIME types are listed at ftp://ftp.isi.edu/in-notes/iana/assignments/media-types/.
14. N. Freed and N. Borenstein, "Multipurpose Internet Mail Extensions (MIME) Part One: Format of Internet Message Bodies," RFC 2045 (1996). Available at http://www.rfc-editor.org/rfc/rfc2045.txt.
15. N. Freed and N. Borenstein, "Multipurpose Internet Mail Extensions (MIME) Part Two: Media Types," RFC 2046 (1996). Available at http://www.rfc-editor.org/rfc/rfc2046.txt.
16. R. Barrett and P. P. Maglio, "How to Personalize the Web," *Proceedings of the ACM Conference on Human Factors in Computing Systems (CHI '97)*, ACM Press, New York (1997).
17. 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).
18. C. S. Campbell and P. P. Maglio, "Facilitating Navigation in Information Spaces: Road Signs on the World Wide Web," *International Journal of Human-Computer Studies* **50**, 309–327 (1999).
19. P. P. Maglio and R. Barrett, "Adaptive Communities and Web Places," *Second Workshop on Adaptive Hypertext and Hypermedia*, Pittsburgh, PA, ACM, New York (1998).
20. P. P. Maglio and R. Barrett, "WebPlaces: Adding People to the Web," *Poster Proceedings of Eighth International World Wide Web Conference* (1999).
21. A. Fox and E. A. Brewer, "Reducing WWW Latency and Bandwidth Requirements by Real-Time Distillation," *Proceedings of the Fifth International World Wide Web Conference (WWW5)* (1996).
22. K. H. Britton, R. Case, A. Citron, R. Floyd, Y. Li, C. Seekamp, B. Topol, and K. Tracey, "Transcoding: Extending e-business to New Environments," *IBM Systems Journal* **40**, No. 1, 153–178 (2001, this issue).
23. WBI programming tutorial. Available at http://www.almaden.ibm.com/cs/wbi/doc/Programming.html.
24. *News firm sues over wireless access to content*, August 8, 2000, Associated Press (2000). Available at http://www.canada.cnet.com/news/0-1004-202-2468835.html.

Accepted for publication September 26, 2000.

Steven C. Ihde *IBM Research Division, Almaden Research Center, 650 Harry Road, San Jose, California 95120 (electronic mail: ihde@almaden.ibm.com).* Mr. Ihde is a software engineer at the IBM Almaden Research Center. He obtained a B.S. degree in computer science from Stanford University in 1995. He joined IBM Research, where he has worked on an intelligent help system, pointing devices, embedded systems, and Web intermediaries.

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

Joerg Meyer *IBM Research Division, Almaden Research Center, 650 Harry Road, San Jose, California 95120 (electronic mail: jmeyer@almaden.ibm.com).* Mr. Meyer is a software engineer at the IBM Almaden Research Center. He holds a diploma engineering degree in computer science from the University of Applied Sciences in Hamburg, Germany. He joined IBM Research in 1998, where he completed his diploma thesis about P3P. Since graduating in 1999, he has worked on WBI, XML transcoding, and search and indexing problems.

Robert Barrett *IBM Research Division, Almaden Research Center, 650 Harry Road, San Jose, California 95120 (electronic mail: barrett@almaden.ibm.com).* Dr. Barrett is a research staff member at the IBM Almaden Research Center. He holds B.S. degrees in physics and electrical engineering from Washington University in St. Louis, and a Ph.D. degree in applied physics from Stanford University. He joined IBM Research in 1991, where he has worked on magnetic data storage, pointing devices, and Web intermediaries.