Building the InfoBus: A Review of Technical Choices in the Stanford Digital Library Project

Andreas Paepcke †
Michelle Baldonado ††
Chen-Chuan K. Chang †
Steve Cousins ††
Hector Garcia-Molina †

† Stanford University (paepcke,kevin,hector@cs.stanford.edu)
†† Xerox PARC (baldonado,cousins@parc.xerox.com)

http://www-diglib.stanford.edu

Abstract

We review selected technical challenges addressed in our digital library project. Our InfoBus, a CORBA-based distributed object infrastructure, unifies access to heterogeneous document collections and information processing services. We organize search access using a protocol (DLIOP) that is tailored for use with distributed objects. A metadata architecture supports novel user interfaces and query translation facilities. We briefly explain these components and then describe how technology choices such as distributed objects, commercial cataloguing schemes and Java, helped and hindered our progress. We also describe the evolution of our design tradeoffs.

Keywords: InfoBus, digital library, CORBA, distributed objects, USMARC, query processing, interoperability, heterogeneous systems, metadata, state management.

This material is based upon work supported by the National Science Foundation under Cooperative Agreement IRI-9411306. Funding for this cooperative agreement is also provided by DARPA, NASA, and the industrial partners of the Stanford Digital Libraries Project. Any opinions, finding, and conclusions or recommendations expressed in this material are those of the author(s) and do not necessarily reflect the views of the National Science Foundation or the other sponsors.

1. Introduction

Four years ago we set out to create a technical infrastructure that would support the construction of digital libraries. These were to be comprised of widely distributed resources that could be autonomously maintained by different organizations and did not require adherence to uniform interfaces. We wanted to cover a vertical slice spanning the space from low-level transport and search facilities to user interfaces. The resulting testbed includes unified access to over 20 diverse sources, some of them with hundreds of associated collections. Online processing services include document summarizers, bibliography converters, resource discovery tools, a copyright violation detector, online payment, rights management, selective dissemination of information, a result set analysis tool, query translation, and document annotation (e.g. [1, 2]). Our user interfaces include a graphical, animated, drag/drop digital library desktop (e.g. [3]) and a browser for blind users.

An 'information bus' (InfoBus) pulls all these components together. It provides plug-in integration for repositories, information processing services, and user interfaces alike (Figure 1). It is implemented as a distributed, CORBA-based object system [4, 5], using Xerox PARC's ILU implementa-

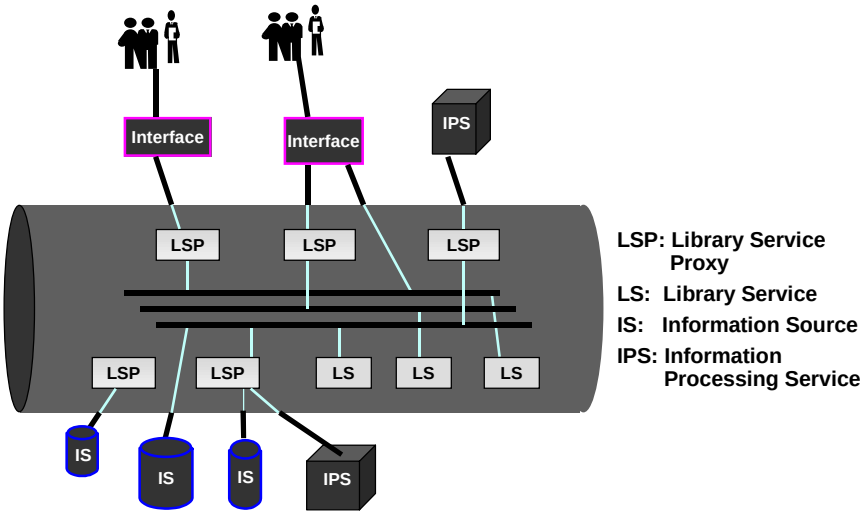

Figure 1: InfoBus Vision of the Stanford Digital Library Project

tion. Heterogeneous repositories are wrapped with *Library Service Proxy* (LSP) objects that shield client programs from as much heterogeneity as technically feasible and appropriate from an end-user perspective. Library services (LS) built into the InfoBus provide the necessary support functions, such as query translation, metadata facilities, and rights management. Documents are modeled as objects. Their instance variables contain document fields, such as author or title. They are materialized from the underlying collections, which may or may not be object-oriented.

For this article we have selected several technical components of the system. We describe these components briefly; details can be found on our Web site and in the more than 55 articles and five dissertations project members published during the course of the project. The purpose here is to discuss some of the tradeoffs we made around the selected components, and how they worked out for us—or did not. Our discussion moves bottom-up and touches on several major issues that must be addressed in a comprehensive interoperability effort for digital libraries. We begin with the choice of using distributed object technology for our underlying infrastructure. We then discuss the protocol we developed for the interaction with objects that provide search services. We particularly focus on the question of where, and how long document result sets are maintained. The subsequent discussion of how documents are represented on the InfoBus highlights our experience with using library cataloguing schemes in a digital environment. We close with a discussion of deployment issues and the influence of user traditions on our technical choices.

2. The Ups and Downs of Distributed Objects

Our use of CORBA has worked well for us, although the experience has not been painless, especially since our system was a very early, extensive use of this technology. The four main advantages of CORBA are the modularity inherent in the distributed object approach, the ability to use different programming languages for interoperating programs, platform independence, and the coordinated specification of object services such as naming. The first was fully realized right from the start. We had previously written large Web services based on CGI-bin scripts and began immediately to appreciate the much cleaner designs a distributed object approach

enables. CORBA includes the concept of an ‘interface’ that formally specifies each class and its methods. Each interface is captured in an interface file using a specification language that is similar in syntax to a C++ program, except that no procedural code is included. Multiple different implementations may realize the same interface. This is of fundamental importance for our InfoBus: Every one of our search proxy objects is an implementation of a single interface we designed—whether that proxy serves out a small email folder, the multiple terabytes of data in the Dialog Corporation’s holdings, or the services of a Web search engine. Consequently, we were able to construct several radically different clients for our services, because the interface clearly describes which methods clients can use to interact with proxy objects.

Language independence refers to the ability to write different implementations of an interface in different programming languages. This capability has been somewhat slower in coming to full fruition, as each CORBA vendor focuses on some small number of languages they either like, or believe to be good business choices. The evolution of both the CORBA specification over the last few years, and the emergence and unceasing transformations of Java somewhat impeded language independence. Nevertheless, very large portions of our digital library infrastructure are written in the Python language, but communicate with an equally large stock of Java programs, and a small number of C++ facilities. The ability to experiment with Java very early without having to discard our large stock of Python code has been a major boon for us.

Platform independence allows programs to run on different processors and operating systems, while interoperating with each other. Like language independence, this capability has been designed into CORBA from the start, and is more or less well achieved by different vendors. After the initial growing pains, this feature is moving closer to reality. At this point, platform incompatibilities introduced by Java’s (maybe overly) rapid evolution is more of an obstacle to cross-platform interoperability than faulty CORBA implementations. Our testbed interoperates among several Unix versions, as well as Windows 95 and NT, and it is being expanded to include some personal digital assistants (PDAs).

CORBA object services, finally, will *eventually* be a great plus for CORBA systems, but their implementations have lagged behind. The idea is for the industry to agree on specifications for services needed in distributed systems. CORBA vendors would then provide corresponding implementations. Many service specifications have been worked out. These include object naming and migration, global event delivery, and transactions. Our project has benefited from the study of these specifications, but we needed to implement for ourselves whatever we required. Today, most CORBA implementations come at least with a naming service, which is essential. Of all the others we most missed an implementation of the event handling service. It could, for example, have been used elegantly for notifying clients of newly arrived documents.

In conclusion, the basic facilities of CORBA have helped us keep our testbed implementation organized, they allowed us to mix and match the programming languages that were appropriate for the tasks at hand, and they bridged the gap between workstations, laptops, and even smaller platforms. But once the system began to grow, three issues required increased attention: control over when to move documents among collections and clients, the complexity of combining locally threaded operation with wide-area object distribution, and overall reliability.

2.1 Controlling Object Movement

We were able to hide the distribution of collections and services from application programmers. For example, our user interface programs execute searches by calling methods on local ‘collection objects’, which contain document objects. The fact that the actual documents and the search proxies may be located elsewhere is successfully hidden. On the other hand, the implementations of the collection objects needed to be programmed with a higher level of complexity, most of which would have been avoidable. The problem was a missing feature of CORBA: pass-by-value object parameters.

When methods implementing CORBA interfaces are invoked and receive or return objects, those objects are passed by reference. This means that every access to those objects involves a remote call across the network. This is exactly the right behavior if most of the access invokes large com-

putations, and this was the case in many of our information processing objects. But in digital libraries, many of the objects are documents, and access to them usually just retrieves the contents of instance variables, that is portions of the document text. In the context of users browsing results, this is too slow across a wide-area network. The document objects need to reside where the access occurs. Users are accustomed to waiting for results after a search. But once they perceive the search to be completed and they begin their exploration of the results, delays are very distracting.

The technical answer is simple: we needed the ability to pass objects *both* by reference and by value. Unfortunately, the CORBA standards efforts have only very recently addressed pass-by-value parameters. Consequently, we needed to simulate pass-by-value by packaging all object state into arrays of differently typed values. Result sets returned from searches, which should simply have been sequences of document objects, instead needed to be two dimensional arrays of such packed document state. This was one of the two major sources of complexity in our search proxy interface, and this still haunts us today, because everyone implementing a client or search proxy must study the interface. Notice incidentally that HTTP has the opposite problem: it does not provide pass-by-reference.

2.2 Threading Across WANs

We found that our users very quickly wanted to perform multiple tasks at the same time: submitting further searches while analyzing previous results, or having remote services create document summaries while they browsed relevant materials. We responded to this demand by introducing multi-threaded operations. The CORBA implementations we used all supported threads, even though our programming languages had differing thread models. However, the complexity of threading was problematic in practice. It seemed as if everyone, in spite of taking operating system classes, needed to experience a deadlock first hand. The combination of interacting with remote objects as if they were local, and the well-known complexity even of locally multi-threaded systems required careful training and debugging patience. Tools that would not just help monitor and analyze the operations of local threads, but that would also allow remote debugging of distant objects that are served through CORBA and are also

threaded would have helped us tremendously.

2.3 Reliability

The distribution of services is technically sensible because it allows the distribution of load, and the avoidance of single-point failures. This is particularly true for our digital library where auxiliary services such as query translators and payment managers are required in addition to the primary services like collections or document summarizers.

Service distribution also mirrors digital library realities, because collections are maintained by different organizations, and autonomous control over access to the collections is required in many cases. However, high overall average availability of the library is more difficult to maintain than with a centralized solution. We needed to check multiple machines for any failures, and when things went wrong, the source of the trouble was not always easy to find. We therefore needed to introduce ‘object nannies’, which would call special status methods on the service proxies, and would periodically restart them. Another effective remedy was to combine multiple related services in one process. For example, we combined a copy each of all library service proxies for Web search engines into one (Unix) process. Monitoring that process thereby monitors basic availability of these multiple CORBA objects all at once. Similarly, restarting that one process restarts all the CORBA objects. From the clients’ point of view, this changes nothing, but system supervision is easier. Again, we would benefit greatly from tools that could monitor and restart processes, analogous to tools used for monitoring low-level network traffic.

3. Let There Be State?

One important factor influencing reliability and response time is whether documents are always accessible, where documents of a search result set are stored, and how long they are available. For digital libraries with long distances between clients and collections this brings up the question of whether stateful or stateless operation is most appropriate.

This issue manifests itself most in how the process consisting of query submission, the construction of a result set, and the client’s subsequent exam-

ination of those results is organized. Our InfoBus and any other such system must manage this process. Related protocols for controlling the process differ in where state is maintained and for how long. For example, the Z39.50 access protocol [6] requires services to retain result sets for the duration of a session with a client. Clients then only retrieve the pieces they need from that set. The World-Wide Web's HTTP protocol, on the other hand, is stateless. Clients request a service, wait until the server is finished processing, and then a result is returned. There is no notion of a session spanning multiple interactions between client and server. There are other important differences among client/server protocols as exemplified in Table 1.

System Behavior	HTTP	Z39.50
State maintenance	client	server
Sessions	no	yes
Connection mode	synch	both
Fixed/negotiated protocol	fixed	negotiated
Fixed/negotiated document format	negotiated	negotiated
Policies sensitive to link speed/load	no	no

Table 1: Some dimensions of client/server interaction protocols

Interaction between clients and servers can be synchronous or asynchronous; some protocols allow negotiation of protocol parameters such as speeds, or document formats. Neither of the protocols in Table 1 allow dynamic modification of client/server interaction based on observed link speed or changing loads.

When designing our protocol for search on the InfoBus, we were in a unique position, because all clients and services on our InfoBus are objects. Objects by nature have the ability, but no requirement to maintain state; their methods may be invoked either synchronously or asynchronously, and they can implement multiple interfaces at once. This technology base allowed us to create a protocol that was very flexible and lent itself well to experimentation with different protocol parameters, and with

dynamic load balancing. In particular, it allowed us to operate both in stateful and stateless modes, and to switch between the two dynamically as operating conditions changed. For example, a server can retain result sets while its load is low (stateful operation). When load increases, it can discard the state and reclaim the associated resources (stateless operation). We call the protocol DLIOP, for Digital Library Interoperability Protocol. We summarize its operation and discuss selected tradeoffs and experiences (refer to Figure 2). For a full description, see [7].

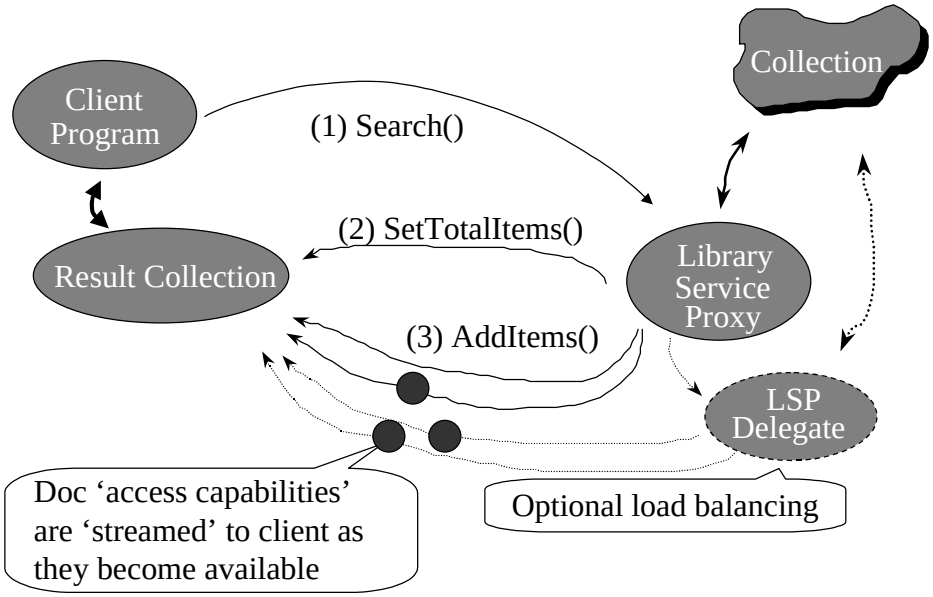

Figure 2: Summary of DLIOP search interaction on the InfoBus

Figure 2 shows a client on the left, and a service on the right. The service initially consists of the collection, and the Library Service Proxy object (LSP). The client creates a local 'result collection' object, which will eventually hold the results. The client then issues a query to the LSP, passing the query, a pointer to the result collection, the number of results requested with the initial return batch, and the document attributes to be included for each document (step 1 in Figure 2). For example, the client could ask for the titles and authors of the first ten documents that contain the words 'digital library'. The LSP processes the query (by interacting with the collection), and calls **`SetTotalItems()`** on the result collection to indicate how many results it found. Then the LSP adds ten new document objects to the client's result collection by calling the **`AddItems()`** method on the result

collection. Each object will contain two properties: 'author' and 'title'. Notice that the LSP may call **`AddItems()`** multiple times with partial results. This allows results to ‘migrate’ to the client as they become available on the server side. The client can then communicate locally with its result collection to retrieve these objects.

Notice in Figure 2 that the LSP may optionally create a delegate object, which is then responsible for pushing the requested results to the client-side result collection as they become available. The LSP might choose this strategy if it gets overloaded. Each time the LSP (or its delegate) invokes the **`AddItems()`** method on the result collection, it includes a service object identifier (OID) to use for any future interactions about this search result. Usually, this service OID will not change; it will be the LSP’s OID. But if the LSP needs to create a delegate in between two calls to **`AddItems()`**, the new OID lets the result collection know. The delegate could even repeat this process and create a second delegate, possibly on another machine.

If the result collection (prompted by the client program) requires more documents from the result set, or if it needs to retrieve additional properties for the documents it already has (e.g. ‘date’), it uses the most recent service OID to issue the appropriate request (not shown in Figure 2).

The description so far is stateful: result sets are maintained at the server side in anticipation of possible follow-up requests. However, the LSP or delegate have the option at any time to discard all state. In order for the result collection to retrieve more items, or additional document properties after server-side state is discarded, the result collection uses ‘access capabilities’, which are part of the documents delivered in the **`AddItems()`** calls. An access capability is a set of server-side cookies, which enable the LSP to re-create the state. Of course, this is a more costly operation for the LSP to accomplish than servicing such follow-on requests if the state is still available. But this facility allows implementors of LSPs to have LSPs adjust their behavior dynamically to run-time conditions. This realizes the benefits of both stateful and stateless operation.

There were two main issues with which we had trouble. The first was the

lack of a pass-by-value facility (described above), which made the interface that describes the DLIOP much more complicated than it should be. The second was our adoption of a Z39.50 convention for expressing queries in the call to the **`Search()`** method (step 1 of **`Figure 2`**). In order to avoid parsing query strings repeatedly during query delivery and search, Z39.50 specifies a syntax tree representation for parsed queries. Following the Z39.50 lead, we decided to parse queries at the client side, and to use the specification as the format for transporting queries in the call to **`Search()`**. Since this method is part of the LSP interface, we needed to create a CORBA interface description of the syntax tree. While we did accomplish this, it again made the interface specification much more complex than it needed to be, and we are still experiencing sporadic difficulties with some CORBA implementations coping with the complexity of the (correct) specification. We would have been much better off paying the price of potentially re-parsing query strings a couple of times but making the DLIOP's CORBA interface simple.

4. About Documents

Digital libraries, like their traditional cousins, need representations for both documents and metadata about documents. We quickly decided that documents would be modeled as objects, partly because we wanted the ability to give them behavior. Document attributes would be stored in the instance variables. This left two decisions to be made: how specifically we would prescribe the representation of the attribute values, and which scheme we would use to name the attributes.

Statically typing attribute values would provide more predictable document interaction capabilities, and it would have been a pleasing computer science solution, particularly if we had used inheritance to derive related document classes. For example, we could have insisted that all 'author' attributes would be sequences of strings in the CORBA interface. However, in the context of CORBA, this solution has drawbacks for the kind of dynamically extensible digital library we wanted to build with our InfoBus. We would have been required to define a different interface for each service proxy or document type that used a new attribute. A basic assumption for CORBA systems is that both service and client have access to the

shared interfaces, and that when new interfaces are developed, they are compiled ('stubbed') at both ends. This tight coupling was quite appropriate for our DLIOP, but it was too constraining for tracking attribute values when new resources were to be developed all the time, especially since the development of CORBA interfaces is not terribly pleasant for the casual user. All our attributes are therefore typed as ANY. For quite a while we communicated data types informally among creators of LSPs and clients; later, our metadata architecture (see below) provided a natural place to record them. We periodically re-visited this decision, considering self-describing representations for attribute values, such as extensions of MIME [8] or an in-house structured representation. Recently, the Resource Description Framework (RDF) has evolved, and we may re-visit our decision again in light of that new facility.

As for attribute naming conventions, we conferred with our librarians who explained that of the traditional cataloguing schemes, MARC, and specifically for us the US version, USMARC, was the most extensive developed over the years [9]. USMARC defines both a wire transport format, and field names for bibliographic entries. The wire transport was irrelevant for our object system, which took care of transport. But we initially decided to use the naming scheme. USMARC field names are numbers (e.g. '100' denotes one kind of author). There are hundreds of fields, and blocks of numbers are reserved for user extensions. The scheme was a good start for us.

The first problem arose from the fact that USMARC was ill equipped to describe journal articles, as opposed to books. The historical reason is that libraries have traditionally lacked the resources to catalogue down to the article level. We added appropriate fields.

Ironically, USMARC itself prodded us down a path that would eventually force us to expand beyond the scheme: in addition to covering directly search-relevant fields like title, author, etc., USMARC also enables coding of administrative information such as the physical size and condition of a book, or a document's issuing organization. The uniform treatment of search attributes and administrative metadata works very well in the digital realm as well. It is a small step from 'physical condition' to 'scan density',

and from ‘physical size’ to ‘number of bytes’. So for a while, the addition of administrative information to our document objects went smoothly. But we soon began to expand beyond bounds appropriate for USMARC: We used InfoBus facilities to search over collections of unusual ‘documents’. For example, we created collections of user profiles and access rights management contracts, which we could search to run our digital library. This required new attributes such as ‘contract duration’, and ‘identity verification method’. Our technology clearly made the uniform treatment of administrative and content materials the right choice, but we were stretching USMARC too far. The emergence of specialized attribute sets, such as GILS for geographic information [10], indicated that the rest of the community also felt the need for expansion outside of USMARC.

We consequently redesigned our metadata facilities from the ground up [11]. At the heart of the redesign is the notion of an *attribute model* that represents a naming scheme, such as USMARC, GILS, Z39.50’s BIB1, or Dublin Core [12]. Attribute models do not need to be flat. For example, one could construct a model that included hierarchical arrangements, such as the attribute ‘creator’ being a parent of attributes ‘corporate author’ and ‘individual author’. Computationally, attribute models are implemented as library service proxies whose ‘document’ objects represent metadata attributes containing information such as attribute name, and the data type its values may take on. For example one such attribute object might represent a USMARC ‘author’ attribute, containing the facts that the attribute’s name is USMARC.100, the type is sequence of string, and legitimate user-level nicknames include ‘100’ and ‘author’. This information is used by several components in the InfoBus, such as our query translation and user interface facilities.

Figure 3 shows the elements of our metadata architecture which supports the auxiliary services on our InfoBus. Attribute model proxies are shown on the left. Attribute model translators (top) are services that provide translation among the models.

The metadata facilities attached to all LSPs (right side of Figure 3) allow LSP clients to find out at runtime which attribute models the LSP supports. Most important for our query translator, they also contain information on

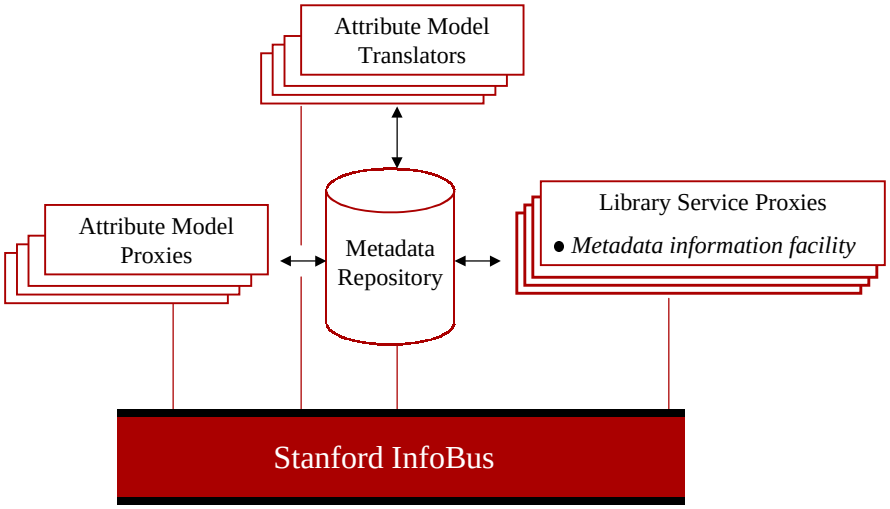

Figure 3: Stanford Metadata Architecture

how to translate each attribute into the proper form to use with the respective source. For example, consider our LSP that wraps the Dialog Corporation's collections, supporting the USMARC attribute model. It informs the query translator that in Dialog the USMARC.100 attribute needs to be referenced by the string 'au=', so that a query searching for a USMARC.100 value of 'Smith' can be translated to a query containing the string 'au=Smith'.

Metadata repositories (center of Figure 3) are caches of the information contained in the other components. When a metadata repository is started, it 'looks around' the CORBA environment name space to find running attribute models, translators, and LSPs. It then extracts some of their information and stores it. This provides us with one-stop access to the information and enables more complicated searches across different kinds of metadata. For example, we can find all LSPs (and therefore InfoBus-accessible collections) that can be accessed using Dublin Core attributes.

The ability to represent coherent attribute sets as searchable collections available on the InfoBus has freed us to introduce new document types much more easily. Again, the downside is increased interdependence of distributed components. Once services are distributed, it is very easy to set off a chain of failures if one of the services is unavailable. In the case of our metadata architecture, for example, we ensured that the query translator

can access some basic models locally, in case attribute models are not up and running when needed. In general, we found such graceful degradation facilities necessary and useful, albeit not as ‘pure’ as we would like our world to be.

5. Deployment

A major issue for us was the distribution of client code to libraries and other test sites. CORBA implementations are not trivial to install, so in the beginning, its use was a hindrance in this respect. We had alleviated the problem by building a special proxy that accepted DLIOP commands coded as strings through a TCP/IP socket and then made the corresponding real CORBA method calls on behalf of the original sender. This allowed non CORBA-capable clients to use the InfoBus initially at low investment cost. The solution worked, but it was roundabout and limiting.

About halfway into the project, Java became a feasible alternative for code distribution. This was extremely attractive for our user interface prototypes. We wanted to distribute them as applets that would display themselves, and would make the necessary CORBA calls as users manipulated our widgets on their screen. The original interface prototypes were written in Python, with a TK interface toolkit attachment. So a switch to Java with its own interface toolkit required a port. Worse, even though clients could be expected to provide a resident Java Virtual Machine, they could at that point not be expected to have an installed CORBA implementation as well. We therefore implemented a simple CORBA-ILU subset in Java, and soon after began distributing our user interfaces as applets that had rudimentary CORBA capabilities built in. Clients no longer needed to install CORBA as a prerequisite to using the InfoBus. They downloaded an applet over the Web which then began operating through CORBA.

The Java security facilities were a constant source of trouble in this context. We were using the prototypes internally, so we had no security concerns. Yet Java security managers and their interactions with browsers were not flexible enough for us to turn off, for example, the limitation on applets to make TCP/IP connections only to the site they came from. We had to create special relay facilities to defeat this limitation. And again,

Java growing pains as manifested in constant revisions and consequent incompatibilities often made development frustrating indeed. Remembering which version combinations of Java, the browser and the computing platform our demonstrations were able to run on at any given time was a challenge. But these are temporary issues that should be resolved over time. Recently, CORBA implementations have started to make their way into commercial Web browsers, which significantly decreases the size of the applets we need to distribute, and which returns us to full CORBA compliance.

6. Reconciling Diverse User Traditions

Once users had access to the InfoBus, we made many changes to our testbed system based on their feedback. One such issue is typical and illustrates how digital libraries are in a specially difficult situation at this time. This difficulty stems from the fact that comprehensive digital libraries are comprised of sizeable collections contributed by traditional library holdings and their online catalogs on the one hand, and the World-Wide Web on the other. These very different collections and associated search facilities each come with diverse audiences. Web users are often untrained and used to extremely simple content similarity search facilities (vector-space). Library users, even if they are not professional librarians, tend to come from a tradition of fielded, usually Boolean search. Digital libraries like ours need to combine and reconcile these different collections and traditions. This has challenging technical implications. Here is an example taken from our query translation work. The query translator allows queries formulated in a single query language to be submitted to collections supporting different native languages, both Boolean and vector-based.

At first, we strived to be very conservative in our query translation facility. If a front-end query was not exactly translatable for the desired target collection, we would indicate failure to the user. Our users' reactions to this approach indicate that many people working online are now so used to the obviously heuristic behavior of Web search engines that they are willing to trade assured correctness, which is typical for traditional library systems, for uniformity by accepting approximations in query translation.

For example, some of the collections we access allow keyword search over author fields (let us call these type A collections), while others (type B collections) support only phrase searches and right truncation (wildcards at the end of a word). Thus, in type A collections, a search for author ‘Mill’ would return documents with author fields ‘Mill’, ‘Mill, Tom’, ‘Frank Mill’, etc. Type B collections would only return documents whose author fields exclusively contained the string ‘Mill’, i.e., only the first item in the type A list above. In order to help users understand this difference, our translator and user interface insisted that users entered queries directed to type B collections as phrases by enclosing query terms in quotes. Keyword-only queries would generate an instructive error message if submitted to type B collections because the translator could not in good conscience automatically promote a sequence of keywords to be a phrase. In contrast, keyword queries would succeed for type A collections.

Users were unhappy with this because the difference in behavior was confusing. One way out of the dilemma is to allow the query translator to approximate translations for particular fields, rather than insisting on strict correctness. For example, the translator could automatically convert all keyword queries to phrases with right truncation when translating for type B collections. This would work well in that it would retrieve ‘Mill, Frank’ for the keyword query ‘Mill’, which is illegal under the strict strategy. But the approach is misleading in that it will fail to retrieve this record in response to the keyword query ‘Frank’. Left truncation would solve this problem, but most library systems do not support that feature. In general, the strategy would probably be a win for author fields because most users tend to work with last names much of the time. The situation is less clear with other fields. For example, many of our collections provide a phrase searchable field ‘Product Type’, with values such as ‘home electronics’. For this field the proper heuristic is less clear. Sophisticated, field-specific user interface support, such as drop-down lists of the controlled vocabulary, might be more appropriate in that case.

As more non-expert users approach our system, the pressure for intelligent translation approximation increases. The challenge is to keep users informed about just how much of an approximation they are seeing, and to keep the complexity of the translation facility manageable for large num-

bers of heterogeneous collections, as specialized heuristics are introduced.

The underlying issue for digital libraries that combine diverse collections is to find the right integration of traditional library facilities and current Web-based approaches. The former provide effective access to carefully indexed fields and offer rich tools for precision search. The latter excel in offering intuitive, albeit crude search over flat textual content. The answer is not just to throw away all the value-added facilities of library collections by restricting them to pure keyword search interfaces. Nor does it seem productive to attempt hoisting intricate power search tools onto uninterested lay searchers. One of the exciting challenges and opportunities for digital library technology is to build a bridge between these worlds. Some of this is already happening outside the research community: Specialized Web sites are re-introducing fielded search and selected Boolean notions for searches over subcollections like automobile data and email addresses. Integration projects like ours will continue to explore other techniques for reconciling all participating traditions.

In conclusion, we believe that our work on the Stanford InfoBus has yielded some useful insights into the operation of distributed object frameworks, as well as fundamental techniques for interoperation in digital libraries. We expect the InfoBus to continue to evolve, as we extend it to cover new services and as we provide facilities for gracefully recovering from failed services.

7. References

[1] Michelle Q Wang Baldonado and Terry Winograd. SenseMaker: An Information-Exploration Interface Supporting the Contextual Evolution of a User's Interests. In *Proceedings of the Conference on Human Factors in Computing Systems*, pp. 11–18. ACM Press, New York, Atlanta, Ga. March, 1997.

[2] Chen-Chuan K. Chang, Héctor García-Molina, and Andreas Paepcke. Boolean Query Mapping Across Heterogeneous Information Sources. *IEEE Transactions on Knowledge and Data Engineering*, 8(4):515-521, Aug, 1996.

[3] Steve B. Cousins, Andreas Paepcke, Terry Winograd, Eric A. Bier, and Ken Pier. The Digital Library Integrated Task Environment (DLITE). In

Proceedings of the Second ACM International Conference on Digital Libraries, 1997. Accessible at http://www-diglib.stanford.edu/cgi-bin/WP/get/SIDL-WP-1996-0049.

[4] Object Management Group. The Common Object Request Broker: Architecture and Specification. Dec, 1993. Accessible at ftp://omg.org/pub/CORBA.

[5] Andreas Paepcke, Steve B. Cousins, Héctor García-Molina, Scott W. Hassan, Steven K. Ketchpel, Martin Röscheisen, and Terry Winograd. Using Distributed Objects for Digital Library Interoperability. *IEEE Computer Magazine*, 29(5):61–68, May, 1996.

[6] *Information Retrieval: Application Service Definition and Protocol Specification*. ANSI/NISO, April, 1995. Preliminary Final Text.

[7] Scott W. Hassan and Andreas Paepcke. *Stanford Digital Library Interoperability Protocol*. Number SIDL-WP-1997-0054. Stanford University, 1997. Accessible at http://www-diglib.stanford.edu/cgi-bin/WP/get/SIDL-WP-1997-0054.

[8] N. Borenstein and N. Freed. MIME (Multipurpose Internet Mail Extensions) Part One: Mechanisms for Specifying and Describing the Format of Internet Message Bodies. September, 1993. Internet RFC 1521.

[9] USMARC Format for Bibliographic Data: Including Guidelines for Content Designation. Cataloging Distribution Service, Library of Congress, Washington, D.C., 1994.

[10] Government Information Locator Service (GILS). 1996. Accessible at http://info.er.usgs.gov:80/gils/.

[11] Michelle Baldonado, Chen-Chuan K. Chang, Luis Gravano, and Andreas Paepcke. Metadata for Digital Libraries: Architecture and Design Rationale. In *Proceedings of the Second ACM International Conference on Digital Libraries*, 1997. At http://www-diglib.stanford.edu/cgi-bin/WP/get/SIDL-WP-1997-0055.

[12] Stuart Weibel, Jean Godby, Eric Miller, and Ron Daniel. OCLC/NCSA Metadata Workshop Report. March, 1995.