Metadata for Digital Libraries: Architecture and Design Rationale *

Michelle Baldonado Chen-Chuan K. Chang Luis Gravano
Andreas Paepcke

Computer Science Department
Stanford University
Stanford, CA 94305-9040, USA
{michelle,kevin,gravano,paepcke}@db.stanford.edu

Abstract

In a distributed, heterogeneous, proxy-based digital library, autonomous services and collections are accessed indirectly via proxies. To facilitate metadata compatibility and interoperability in such a digital library, we have designed a metadata architecture that includes four basic component classes: attribute model proxies, attribute model translators, metadata facilities for search proxies, and metadata repositories. Attribute model proxies elevate both attribute sets and the attributes they define to first-class objects. They also allow relationships among attributes to be captured. Attribute model translators map attributes and attribute values from one attribute model to another (where possible). Metadata facilities for search proxies provide structured descriptions both of the collections to which the search proxies provide access and of the search capabilities of the proxies. Finally, metadata repositories accumulate selected metadata from local instances of the other three component classes in order to facilitate global metadata queries and local metadata caching. In this paper, we outline further the roles of these component classes, discuss our design rationale, and analyze related work.

Keywords: Metadata architecture, interoperability, attribute model, attribute model translation, metadata repository, InfoBus, proxy architecture, heterogeneity, digital libraries, CORBA.

1 Introduction

Travelers rely upon various types of information to point them to where they might want to go and what rules they need to follow. Maps, guidebooks, signs, and phrase books are just a few examples. Not surprisingly, users of digital libraries also require information about what networked resources and services to consult, as well as information about how to use them. Generically, we call this information *metadata*. In an earlier paper [1], we analyzed the needs of our digital library *InfoBus* architecture [2] in order to determine what types of metadata are important in this environment.

Our InfoBus architecture provides the infrastructure for a distributed, heterogeneous, proxy-based digital library. At its core are *proxies*—or wrappers—that provide uniform access to existing, autonomous digital library repositories and services. We implement these proxies as CORBA distributed objects [3]. These objects can be placed on any machine on the network and can be accessed remotely from anywhere. Clients use remote method calls to access these proxies conveniently. The proxies communicate with the services they represent via the native service access protocols, such as telnet, Z39.50, or HTTP. Examples of the diverse services we have wrapped with proxy objects are Knight-Ridder's Dialog Information Service, World-Wide Web search engines, automatic document summarizers, bibliography maintenance tools, OCR services, and others.

Within the InfoBus, we have designed a variety of services, including services for finding resources likely to satisfy a given query [4], for formulating queries that are appropriate for multiple sources [5], for translating queries [6], and for making sense of query results [7]. Analyzing these services leads to a list of metadata requirements (see [1]). However, addressing these metadata requirements on a service-by-service basis is problematic. We have a need for metadata compatibility and interoperability among these services. Accordingly, we have chosen to develop a metadata architecture for our digital library that addresses these requirements in a general fashion.

Our resulting metadata architecture sits on top of our InfoBus. It makes extensive use of its objects and services, especially those that pertain to documents and collections. Documents on the InfoBus are represented as objects. We use the CORBA property service [3] to store document attributes. For example, the `Title` attribute of a document is attached to the document object as a property of name `Title`. In addition, various methods allow us to extract metadata from document objects, such as a list of all the document's attributes. The contents of a document are represented as the value of a content attribute. For example, the full text of

*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.

a textual document would be included as the value of its `Full Text` attribute. The InfoBus *collection service* allows users to build their own repositories from documents they have created, located, or received. A collection is a CORBA object that allows users to store other objects, including documents or, recursively, other collections. Collections support querying through a special protocol that we have defined [8].

In our earlier paper [1], we analyzed our metadata needs and presented our initial metadata architecture for addressing those needs. In this paper, we outline the architecture, discuss its design rationale, and analyze related work.

2 Metadata Architecture Design

The foundation for our metadata architecture design is currently based upon four component classes and the communication pathways among them. The architecture is extensible, allowing for additional component classes to be introduced as the need arises. Each individual component instance is implemented by one or more CORBA objects. Figure 1 shows the current component classes, communication pathways, and InfoBus connections. The component classes include attribute model proxies, attribute model translators, metadata facilities for search proxies, and metadata repositories. We describe the specification of each component class and its associated communication pathways in turn.

2.1 Attribute Model Proxies

In the digital library environment, we build structured descriptions of resources and services out of *attributes* and *attribute values*. One level of aggregation beyond the individual attribute is the *attribute model*—a self-contained collection of attributes. Well-known attribute models include the USMARC set of bibliographic attributes (referred to as "fields" in the USMARC community) [9], the Dublin Core set of attributes [10], and so on.

In our metadata architecture, we reify both attributes and their encompassing attribute models as first-class objects. Attributes are instances of class `AttributeItem`. *Attribute model proxies* are implemented as InfoBus collections. Attribute model proxies represent real world attribute models, just as search proxies represent real world search services.

An `AttributeItem`'s properties include the following: model name, attribute name, aliases usable for queries, value type, documentation, various other information used by query translators, and so on.

The model name and attribute name are both strings that serve to identify the `AttributeItem` uniquely. As an example, an attribute might have the model name "Dublin Core" and the attribute name "Title." The model name is repeated in all items to make them self-contained. This is important when the items are passed around the system to components other than the "home" attribute model proxy. When examining an `AttributeItem`, a client can always determine to which attribute model it belongs.

The attribute value type information in an `AttributeItem` dictates the data type that can be contained in fields described by the `AttributeItem`. We use the interface specification language that is part of our CORBA implementation to specify these types. It is up to each search service proxy to ensure that the values it returns conform to these type specifications. If the external service that the proxy represents natively returns a different type, then the proxy is expected to transform the value into the specified type before returning it.

Attribute model proxies make attribute models first-class objects in our computational environment. They allow us to store and search over attribute-specific information that is independent of the capabilities possessed by any particular search proxy. Since an attribute model proxy is a collection, it is accessible via the same interface as all other search service proxies. In other words, the attribute model proxy has a search method that responds to a query by returning the appropriate subset of the included `AttributeItems`. Furthermore, attribute model proxies record what relationships hold among the included attributes. This is important for some services, such as sophisticated user interface services, that require *structured* attribute models. Figure 2 depicts a structured attribute model that encodes "`is-a`" relationships among its attributes. In our architecture, the attribute model proxy for this attribute model would be a collection containing `AttributeItems` subclassed to include `is-a()` methods. Other relationships between the attributes in an attribute model, like "`has-a`," can be treated analogously.

A search service proxy that supports this attribute model could use its relationship information when processing queries. For example, a search service proxy might determine that items match the query `Creator contains Ullman` if they contain "Ullman" in the value of the `Creator` attribute or if they contain "Ullman" in the value of descendant attributes (i.e., in the value of `Reporter` or `Author`).

2.2 Attribute Model Translators

In heterogeneous environments, many different attribute models co-exist. This inevitably leads to mismatches when InfoBus components that support different attribute models attempt to communicate with each other. For example, consider a bibliographic database proxy and a client of that proxy. The bibliographic database proxy might support only the Dublin Core attribute model, while the client might support only the USMARC bibliographic data attribute model. In order for this client and this proxy to communicate with each other, they must be able to translate from USMARC attributes to Dublin Core attributes and vice versa. In other words, they require intermediate *attribute model translators*. Attribute model translators serve to mediate among the different metadata conventions that are represented by the attribute model proxies. These translation services, available via remote method calls, translate attributes and their values from one attribute model into attributes from a second attribute model.

Of course, in some cases, translation may not be possible: consider translating from an attribute model designed for chemistry databases into an attribute model designed for ancient Greek texts. Similarly, Dublin Core is a much smaller set than USMARC, so some information that can be tagged in USMARC finds no equivalence in Dublin Core.

Even when translation is appropriate, the translation from one attribute model to another is often difficult and lossy. For example, the Dublin Core describes authorship using the single attribute `Author`. However,

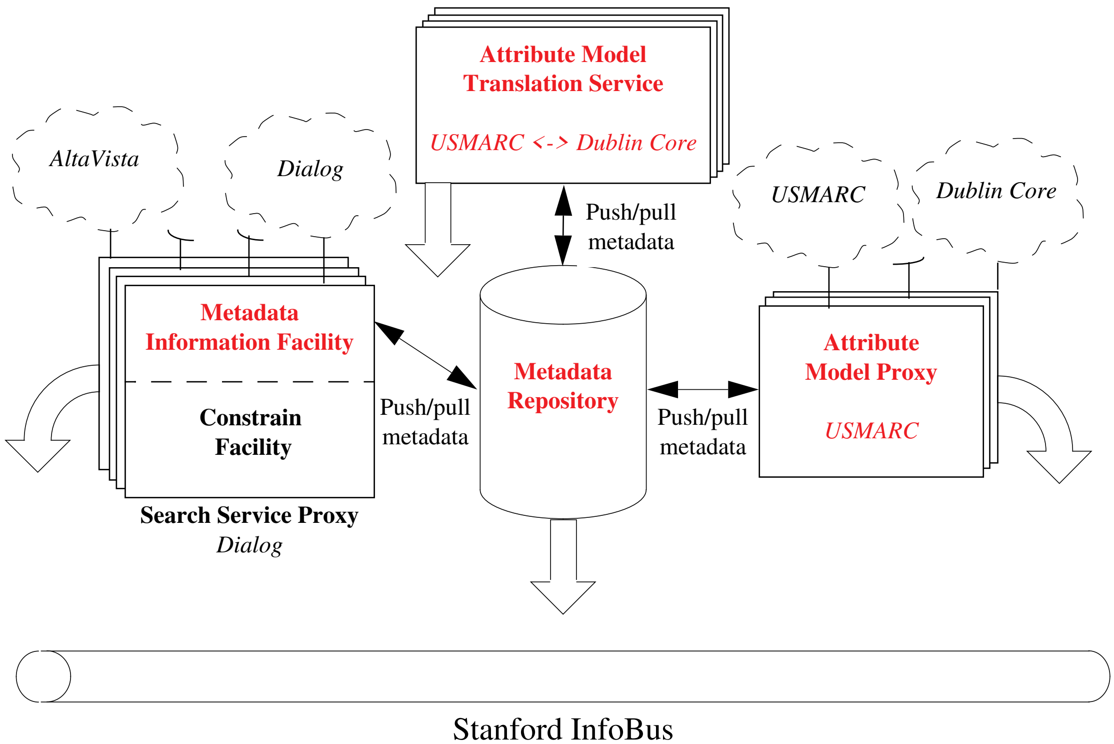

Figure 1: The Metadata Architecture Design

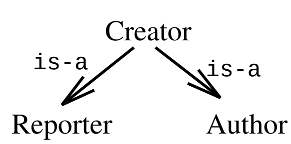

Figure 2: A complex attribute model with the "is-a" relationship among its attributes.

USMARC distinguishes among several different types of authors, including `Corporate Author` (recorded in the `100` attribute) vs. `Individual Author` (recorded in the `110` attribute). When translating an `AttributeItem` from Dublin Core to USMARC, a decision must be made whether to translate the Dublin Core `Author` attribute value into a USMARC `100` attribute value or a USMARC `110` attribute value. This may be hard-coded, or the translation may be performed heuristically and may take into account the other attribute values present in the item being translated.

Translation services do more than map source attributes onto target attributes. They must also convert each attribute value from the data type specified for the source attribute into the data type specified for the target attribute. This conversion can be quite complex. For example, one attribute model might call for authors to be represented as lists of records, where each record contains fields for first name, last name, and author address. Another model might call for just a comma-separated string of authors in last-name plus initials format. When translating among these values, some information may again be lost if, for example, the address is simply discarded.

Our attribute model translation services may be accessed by a variety of other InfoBus components, including search service proxies. For example, a search service proxy might choose to use attribute model translators to be attractive to more clients, because it can then advertise that it deals in multiple attribute models. On the other hand, clients might use attribute model translators in order to ensure their ability to communicate with a wide variety of search services.

2.3 Metadata Facilities for Search Proxies

The *metadata facility* that we attach to each search service proxy is responsible for exporting metadata about the proxy as a whole, as well as for exporting metadata about the collections to which it provides access. Collection metadata includes descriptions of the collection, declarations as to what attribute models are supported, information about the collection's query facilities, and the statistical information necessary for resource discovery services like *GlOSS* [4] to predict the collection's relevance for a particular query. Clients can use the information to determine how best to access the collection maintained by the search service (i.e., what capabilities the search service supports).

We have decided to make the interface for accessing the metadata facility of search service proxies very simple in order to encourage proxy writers to provide this information. Search service proxy metadata is accessed via the `getMetadata()` method, which returns two metadata objects. Alternatively, each proxy may opt to "push" these metadata objects to its clients. The first metadata object (Table 1) contains the general service information, and it is based heavily on the *source metadata* objects defined by STARTS [11]. The general service information includes human-readable information about the collection, as well as information that is used by our query translation facility. Examples for the latter are the type of truncation that is applied to query terms, and the list of stopwords. Our current query translation engine is driven by local tables containing this kind of information about target sources.

An interesting attribute of our first metadata object is `contentSummaryLinkage`. The value for this attribute is a URL that points to a *content summary* of the collection. Content summaries are potentially large, hence our decision to make them retrievable using ftp, for example, instead of using our protocol. The content summary follows the STARTS [11] content summaries, and consists of the information that a resource-discovery service like *GlOSS* needs. Content summaries are formatted as Harvest SOIFs [12].

Example 2.1 *Consider the following content summary for a collection:*

```
@SContentSummary{
Version{10}: STARTS 1.0
Stemming{1}: F
StopWords{1}: F
CaseSensitive{1}: F
Fields{1}: T
NumDocs{3}: 892

Field{5}: Title
DocFreq{11023}: "algorithm" 53
                "analysis" 23
...

Field{6}: Author
DocFreq{1211}: "ullman" 11
               "knuth" 15
...
}
```

This summary indicates that the word "algorithm" appears in the `Title` *of 53 items in the collection, and "ullman" as an* `Author` *of 11 items in the collection, for example. The numbers in braces indicate the length of the field values, to facilitate parsing.*

The second metadata object returned by the `getMetadata()` method (Table 2) contains attribute access characteristics. This is attribute-specific information for each attribute that the proxy supports. Recall that attribute model proxies contain only information that is independent from any particular search services. Attribute access characteristics complement this information in that they add the service-specific details for each attribute.

For example, some search services allow only phrase searching over their `Author` attributes, while others allow keyword searching. Similarly, some search services may index their `Publication` attributes, while others may not. The attribute access characteristics describe this information for each attribute supported by the collection specified in the `getMetadata()` call. The query translation services need this information to submit the right queries to the collections.

Notice that this design does not allow clients to query search service proxies directly for their metadata. Search service proxies only export all their metadata in one structured "blob." The reason for this is that the InfoBus includes many proxies, and more are being constructed as our testbed evolves. We therefore want proxies to be as lightweight as possible. Querying over collection-related metadata as exported by search service proxies is instead available through a special component, the metadata repository.

Metadata Attribute	*Description*
`version`	Version of the metadata object
`collectionName`	Name of the collection being described
`attrModelNames`	Attribute models supported
`attrNames`	Attributes supported
`booleanOps`	Boolean operators supported
`proximity`	Type of word proximity supported
`truncation`	Truncation patterns supported
`implicitModifiers`	Modifiers implicitly applied (e.g., stemming)
`stopWordList`	Stopwords used
`languages`	Languages present (e.g., `en-US`, for American English)
`contentSummaryLinkage`	URL of the content summary of the collection
`dateChanged`	Date the metadata object last changed
`dateExpires`	Date the metadata object will be reviewed
`abstract`	Abstract of the collection
`accessConstraints`	Constraints for accessing the collection
`contact`	Contact information of collection administrator

Table 1: The attributes present in the first metadata object for a collection, describing general service characteristics.

Access Characteristic	*Description*
`collectionName`	Name of the collection being described
`attrModelName`	Model of the attribute
`attrName`	Name of the attribute
`searchRetrieve`	Whether the attribute is searchable, retrievable, or both
`modifierCombinations`	Legal combinations of modifiers (e.g., `stemming`, `>`) for the attribute

Table 2: The attribute access characteristics for one search attribute, as described in the second metadata object for a collection.

2.4 Metadata Repositories

Metadata repositories are local, possibly replicated databases that cache information from selected attribute model proxies, attribute model translators, metadata facilities for search proxies, and other InfoBus services in order to produce one-stop-shopping locations for locally valuable metadata. We allow for metadata repositories to pull metadata from the various facilities, as well as for the facilities to push their metadata to one or more repositories directly. The intent is for these repositories to be a local resource for finding answers to metadata-related questions and for finding specialized metadata resources. A metadata repository has a search-service-proxy interface, and includes:

- *The `AttributeItems` from locally relevant attribute model proxies.* This data is useful, for example, to search for attribute models that contain concepts of latitude and longitude.
- *Translator information for locally relevant attribute models.* This data is useful, for example, to search for translators to or from particular models. Searches return pointers to the translator components.
- *General service information for locally relevant search service proxies.* This data is obtained through `getMetadata()` calls on the proxies, as discussed in the previous section, and is useful, for example, to search for collections whose abstracts match a user's information need. It is also useful for more technical inquiries, such as for finding search proxies that support a given attribute model, or proxies that support proximity search.
- *The attribute access characteristics of the locally relevant search proxies.* This data is primarily useful to the query translators. Translators can, for example, find out which proxies support keyword-searchable Dublin-Core `Author` attributes.

2.5 Implementation Status

We have implemented prototype instances of all four component classes of our metadata architecture: several attribute model proxies, two attribute model translators, a metadata repository, and a metadata facility for a search proxy.

Our attribute model proxies include implementations of proxies for Z39.50's Bib-1, Dublin Core, Refer, BibTeX, GILS, and a subset of USMARC. We can, for example, search over our USMARC proxy for all attributes containing the word `Title` in their description. This returns five entries, including attributes for `Title Statement`, `Varying Form of Title`, and `Main Entry -- Uniform Title`. This information will be used in our user interface, for example, to help users select proper attributes for search.

Our first attribute model translators provide attribute translations between Refer and BibTeX, and between our subset of USMARC and BibTeX. We have not implemented the value type translations yet.

Our implementation of a metadata facility for search proxies works over our NCSTRL proxy, which communicates with the NCSTRL collection of Computer Science technical reports.

Our initial metadata repository implements an information pull model. It currently maintains metadata information from the attribute model proxies and attribute model translators. We have not yet integrated information from the metadata facilities for search proxies. When the repository starts up, it finds all running instances of attribute model proxies and attribute model translators and extracts relevant information from them.

3 Design Rationale

As we have designed and implemented the metadata architecture outlined in this paper, we have broadened our understanding of the complex tradeoff space in which digital library systems are embedded. In the ensuing sections, we discuss and situate some of our fundamental design decisions. We also articulate some of the difficult challenges that must still be addressed by metadata architectures.

3.1 Attribute Model Proxy Issues

We decided to implement attribute model proxies as simple, searchable collections of `AttributeItem` objects. This implementation is a good reflection of the purpose behind attribute model proxies: to represent external attribute set standards, which consist of sets of attribute elements. Representing these elements as objects gives us the modeling flexibility we need. Earlier, we mentioned our ability to subclass `AttributeItem` to add inter-attribute relationships. There are other reasons why we need such extensibility. For example, while Bib-1 elements are flat, Dublin Core and USMARC elements may each contain subordinate additional information. When this information is to be modeled as well, subclassing is an easy mechanism allowing us to do that.

Our implementation of attribute models as searchable collections allows us to interact with the models just as we interact with our search proxies (e.g., AltaVista). In fact, our use of the same protocols for communicating with both attribute models and search proxies has contributed to the simplicity of our architecture.

Another design decision around attribute model proxies concerns the form in which the admissible values for each attribute in an attribute model are specified. Attribute value types are frequently not specified in attribute set standards. Sometimes, some of the types are specified, while others are not. For example, some attribute sets specify that dates are to be maintained in some standard form, but they may not specify the exact form in which publisher addresses are to be entered. We wanted our attribute model proxies to isolate client programs from the variations in attribute values. Specifying admissible value types in the model, and making sure the corresponding value translations occur when values are extracted from the underlying sources is one way of accomplishing this isolation. But how should the value types be specified?

A spectrum of type specification formality is possible. At one extreme, we could introduce an entire extensible type system to describe how values for each attribute are constructed. We could then build value parsers that transform values to convenient formats. This would provide for maximum stability in system maintenance because all code changes could be verified against the type specifications. It would also enable completely separate development of code implementing information sources and code implementing information clients. Unfortunately, this solution is expensive in terms of supporting code, and in building metadata models.

At the other extreme, we could have a typeless system. A hybrid system might allow attribute type specifications as hints to help make implementations less verbose. For example, a `Date` field value could just be described as `String`. This solution is easier to implement and maintain, and it would allow implementations at least to guard against basic type faults and related crashes; but it would limit the usefulness of the type information.

In our initial prototype, we are experimenting with specifying attribute value types in the same CORBA interface language we are using to specify the types of parameters passed to methods in our larger distributed object system. We are also considering an adaptation of the MIME standard [13] to let us describe complex attribute values, such as values consisting of multiple parts. Neither of these is satisfactory in describing the details of how basic types are used. For example, while we can express concisely that a value is to be of type `Integer`, we cannot say formally in our current prototype that the integer is to represent a date in "seconds since 1970." Similarly, we cannot describe whether the `String` holding a document's authors is organized as last-name first or vice versa. This information can, of course, be included in the human readable description slot of each `AttributeItem`. While we could borrow from knowledge representation technologies to accomplish more refined value typing, we need to weigh the usefulness of such descriptions against the complexity of the code that deals with them throughout the system.

The final design decision we discuss in the context of attribute model proxies concerns their relationship to each other. In our initial prototype we decided not to model such relationships at all. For example, the emerging Geo1 attribute set [14] is a superset of the GILS attribute set, which in turn subsumes Bib-1. If we allowed attribute model proxies to reference each other, we could model these subsumption relationships. Instead, we construct each attribute model proxy from scratch, replicating all common `AttributeItems`.

This approach has the standard drawbacks of replicated data: if an evolving standard changes, or if we decide to change the description of a Bib-1 `AttributeItem`, we must propagate this change manually. Similarly, we cannot currently express that the value type of attribute A in attribute model X is to be the same as the value type of attribute B in attribute model Y.

We decided to experiment with our simpler design in the interest of controlling overall complexity. It remains to be seen whether we will need to revise this decision as we gain more experience with the system.

3.2 Attribute Model Translator Issues

Attribute model translation has some intrinsic limitations. We have described how translation can lose information when multiple attributes of one model map into a single attribute of another (the many-to-one problem). The one-to-many problem is the dual: if an attribute in one model has multiple, finer granularity equivalents in another, automated translation is difficult because it requires analysis of the attribute value, or of other attributes in the same document. For example, one could imagine a heuristic that checks whether an author value is an individual or a company. We indeed do similar heuristic analyses when we parse Web pages returned from search engines to extract attribute values. We also use such heuristics when converting untyped Refer records to BibTeX records, which are typed. However, the current implementation of our attribute translators does not employ such techniques for dealing with the one-to-many problem.

Instead, when asked to map an `AttributeItem` from a source model to its equivalent in a target model, our translators return lists of target attributes when multiple attributes in the target model may be correct translations. The translator client then needs to decide what to do. For example, our BibTeX to USMARC translator returns several USMARC `AttributeItem`s as possible translations for a BibTeX `Author`. Varying approaches for making this decision are appropriate under different circumstances. For example, when the client's goal is to create a document tagged according to the target model, the client might set all the related target attributes to the same value—the value of the single source attribute. If instead the goal is to create a document representation using the simpler source model, the value of the single relevant attribute could simply be set to a concatenation of relevant values in the target. For example, if a Dublin Core `Author` field in one document representation is to be initialized from another document representation organized according to the USMARC model, all USMARC shades of `Author`-like fields might be combined and separated by semi-colons. While our translators do not make this decision, they at least inform the client which target attributes are relevant.

Apart from these intrinsic difficulties of attribute model translation, there is the cost of writing translators to translate from n attribute models to m other models. Initially, this does not seem to be a big problem, because there are only a rather limited number of official attribute set standards in common use. But once our metadata components began to come online, it became clear that we wanted to build specialized attribute models for use locally within our own applications. For example, we quickly discovered that our metadata architecture would be handy also for attribute models concerned with online financial transactions.

In order to keep the cost of translator generation in check as the number of attribute models increases, we initially constructed one large, generic, intermediate, attribute model. When translating from model X to model Y, we first translated from X to the rich intermediate model, and from there to model Y. This approach served us well for quite a while. It does make translation easier. But, of course, the intermediate model becomes too unwieldy as the number of attribute models grows. We are now planning to experiment with another variant of two-stage translation. We will first pick one representative in each group of similar models. For example, we will select one rich model that handles geo-spatial material. We pick another for bibliographic data. Then, instead of always translating to a single rich model, we will translate to one of a handful of models, and from there to the final target.

A particularly sophisticated approach to translation is possible for the hierarchical attribute models we described earlier (Figure 2). If the translation of a leaf attribute fails, we will first attempt to generalize to a level higher in the hierarchy, and then repeat the translation attempt.

3.3 Service Proxy Metadata Facility Issues

Our specification of how metadata can be obtained from search service proxies, and which format it will be delivered in is an important first step towards dealing with source heterogeneity. We need collection content summaries to predict which of a potentially large number of sources is most likely to contain results relevant to some given query. Services such as our query translator also need information about which operations are supported by each service.

Naturally, data source content summaries provided by proxies are only as good as the information that can be obtained from the actual sources they represent. The STARTS specifications [11] are intended to allow search engine providers and their customers to deliver the content summaries. Carl Lagoze at Cornell University has built a reference implementation of STARTS in Java that provides the STARTS information for freeWAIS services. This will make it easier for information providers to deliver the meta-level information described in Section 2.3.

The metadata called for from proxies in Section 2.3 is still rather text oriented. Our content summaries, for example, contain information about occurrence statistics of keywords. Similarly, the descriptions of service functionality are geared towards text search related operations, like stemming, or stop word elimination. More and more innovative metadata is coming on line that goes beyond text. For example, Carnegie-Mellon's Digital Library project provides small, automatically generated sequences of low-resolution images to summarize the contents of a video (`http://www.informedia.cs.cmu.edu/`). Over time, we will need to add such advanced notions to the repertoire of metadata that can be requested from our service proxies. For the initial step reported on here, we decided to focus on the more traditional text medium, because there is a longer tradition, and broader common understanding of standard search related operations.

Limiting functionality related service metadata to information about operations related to text is actually not enough to maintain a manageable degree of complexity. There are many advanced capabilities that search engines might provide, but that our proxy metadata format does not include. One example that came up in discussion is the ability to search for all documents that are written in one language, but that contain some particular word in another language. Our STARTS metadata format does not make it easy to advertise this capability, although the format could easily be extended. Again, the decision to address only commonly available operations initially is grounded in our goal of getting a first version of all four metadata component classes working, so that we can study the effect they have on our overall system. Since the ultimate success of STARTS depends on the participation of information providers, simplicity in this part of our metadata facilities was especially important.

Our self-imposed guideline of keeping metadata components simple is also particularly relevant for all aspects of search service proxies. Our overall InfoBus system benefits greatly from keeping service proxies simple. We want it to be easy for third parties to construct proxies to their services, so that InfoBus clients can take advantage of them. Therefore, any metadata facilities added to proxies had to be easy to implement. This is why we decided against making service metadata on proxies searchable. We did not want to require a search engine to be included in all the proxies. Each proxy just needs to support the `getMetadata()` method. To make up for this low functionality interface, our metadata repositories provide for searching over source content summaries. They accumulate metadata from the proxies and prepare it for searching as necessary. This presents no additional implementation expense, since metadata repositories need search engines anyway, and because we do not expect others to construct different kinds of metadata repositories on a regular basis.

We will, of course, always need to accommodate sources that are not equipped to deliver the search service related metadata we like to have. We are constructing our clients and the metadata repository such that they are tolerant of these deficiencies.

3.4 Metadata Repository Issues

The notion of our metadata repositories is rather straightforward. Their purpose is to provide for local caching of metadata and to enable querying across metadata from multiple sources. In spite of this simplicity, there are design decisions we needed to make, and further investigations to undertake.

One question concerns the system-wide usage of metadata repositories. We have not yet gathered enough experience to know how many local repositories will be desirable, and how much information they will each cache. If metadata repositories cache most of the system's metadata, and if the total amount of metadata is large, we may need to organize metadata repositories hierarchically. It is possible that the Internet's domain name service (DNS) could serve as a model. So far, we have not seen a need for such a step.

If there are many metadata repositories and sources of metadata, the problem of cache consistency may become a problem. We do not expect this to be an immediately pressing issue, because metadata generally does not tend to change as frequently as other data. A related problem that is more likely to demand attention arises when services that provide metadata come online after a metadata repository wishing to store their metadata has already been started. In this case, our pull model implementation will not acquire the new metadata. Our design does support metadata pushing, as well as pulling. We decided to start with a pull model, because without further facilities, it is not clear which metadata repositories a newly started metadata source should push its data to. We would in this case need to add a registration facility where metadata repositories could advertise their interest. Alternatively, we can retain the pull model, and have metadata repositories scan for new services periodically. We can do this through the use of our object name service.

Our current implementation supports very simple queries over repositories, in the sense that only one class of metadata can be addressed by a single query. For example, while we can respond to queries asking for attribute models containing `AttributeItems` related to authorship, we cannot ask for all search services that support attribute models containing `AttributeItems` related to authorship. This question would need to be answered in two steps. First, we would find the attribute models, then we would find all services that support these models. If we moved our metadata implementation to a database that supports joins, we could make the query facility more powerful. Again, for now we opted for simplicity.

4 Related Work

Much recent work has focused on the specification of attribute sets intended for the metadata description of information objects. The Bib-1 attribute set [15, 16] registers a large set of bibliographic attributes, while the Dublin Core [10] is intended to be a simple yet usable set of attributes. These standard metadata sets are represented as attribute models in our architecture. Our attribute models go beyond these metadata sets because they can optionally include structures and because they are reified as searchable collections.

Given the existence of various metadata sets for information objects, and the fact that no single set covers all the possible aspects, there have also been significant efforts in developing architectures that support the integration and interoperation of various metadata sets. The Warwick Framework, for example, is essentially an encoding scheme to integrate various metadata packages. Another framework, proposed by the Alexandria Digital Library [17], models document meta-information with attribute-value pairs, in which multiple "languages" can be used. Common to these architectures and ours is the support of multiple metadata sets in modeling documents. However, because we represent documents as a flat set of attribute-value pairs, the encoding scheme of the Warwick Framework is more sophisticated because its structure supports recursion and indirection. On the other hand, our work complements the Warwick Framework in that we do provide attribute models as searchable registries for metadata attributes.

In addition to the work on metadata for information objects, there are also proposals that support metadata for search services. For instance, the GILS profile for Z39.50 [18], while including all of the Bib-1 attributes, defines many attributes for describing search services. The University of Michigan Digital Library Conspectus describes the contents and capabilities of the available search services using a special language [19]. The efforts that are probably closest to ours are the Explain facility of Z39.50-1995 [15] and Stanford's STARTS [11], both of which require services to export their "source metadata."

Z39.50 servers present metadata about their services via the Explain facility so that clients can dynamically configure themselves to match individual servers. The metadata is structured as another database that can be queried by the clients via the Z39.50 protocol. The Explain facility provided by Z39.50 servers corresponds to the metadata information facility supported by our service proxies. The major distinction is that our proxies support simpler interfaces for accessing the service metadata. We have decided to shift the more complex functionalities (e.g., searchable metadata) to the metadata repositories, to make proxies as lightweight as possible and therefore easy to implement. On the other hand, this simplicity may not be an issue for Z39.50 because both their clients and servers will necessarily have most of the required capabilities [20]. However, our architecture can benefit from the Explain facility; it should be relatively easy to build proxies to Explain-compliant services that will support our proposed metadata facility.

Another relevant effort on top of which we built our architecture is STARTS [11]. STARTS, coordinated by Stanford, is an informal "standards" effort whose main goal is to facilitate the interoperability of search engines for text. STARTS specifies what metadata should be exported by each collection of text documents. This information is essentially what the search proxies export through their metadata facility (Section 2.3). STARTS does not describe, however, a more sophisticated metadata architecture. It just specifies the information that the collections should provide. As with the Explain facility, the architecture that we present in this paper benefits from STARTS-compliant services: for such services, it is easy to build proxies that will satisfy our metadata requirements.

Finally, our proposed metadata repository has roots in the WAIS "directory of servers." However, our metadata repository has a more sophisticated structure, more complex update mechanisms, and a broader scope.

5 Conclusion

Our experience in designing and implementing a metadata architecture has illustrated once again the importance and complexity of metadata issues. As many digital library builders have observed, there are many classes and uses for metadata. In this paper, we have outlined a metadata architecture that addresses a broad class of the needs found in a distributed, heterogeneous, proxy-based digital library. The component classes around which the architecture is built—attribute model proxies, attribute model translators, metadata facilities for search proxies, and metadata repositories—facilitate the sharing of both document and service metadata. On the document side, attribute model proxies, attribute model translators, and metadata facilities for search services combine to enable clients to understand and make use of structured document descriptions that are encoded according to a wide array of standards. On the service side, the content summaries and attribute access characteristics available from the metadata facilities for search services help clients to make "smart" choices of search services and to tailor their service requests according to the services' capabilities. Our metadata repositories organize both document and service metadata so that it can be easily found.

In analyzing our design and implementation of this metadata architecture, we made explicit in this paper our stances on a variety of metadata issues. On the architectural level, we have striven for simplicity and interoperability wherever possible. In this way, we hope to facilitate the provision and sharing of metadata in our digital library. In our design of attribute model proxies and translators, we have adopted an approach that allows us to accommodate a large number of existing metadata sets. We have also paved the way for the rich, structured attribute models that will become important in digital libraries. In our design of metadata facilities for search services, we have balanced the need for detailed service information with the requirement that the metadata facilities be both lightweight and easy to write. Finally, our conceptualization of metadata repositories addresses what we see as the important problem of where to look first for metadata in the intricate web of services that makes up a large-scale, distributed digital library.

Acknowledgments

Many useful comments on our work were provided by Terry Winograd, Héctor García-Molina, and other members of the Stanford Digital Library Project. Carl Lagoze constructed our STARTS reference implementation, which is the basis for our NCSTRL collection metadata experiments.

References

[1] Michelle Baldonado, Chen-Chuan K. Chang, Luis Gravano, and Andreas Paepcke. The Stanford Digital Library metadata architecture. *International Journal of Digital Libraries*, 1(2), February 1997.

[2] Andreas Paepcke, Steve B. Cousins, Héctor García-Molina, Scott W. Hassan, Steven K. Ketchpel, Martin Röscheisen, and Terry Winograd. Towards interoperability in digital libraries: Overview and selected highlights of the Stanford Digital Library Project. *IEEE Computer Magazine*, May 1996.

[3] Object Management Group. The Common Object Request Broker: Architecture and specification. Accessible at `ftp://omg.org/pub/CORBA`, December 1993.

[4] Luis Gravano, Héctor García-Molina, and Anthony Tomasic. The effectiveness of *GlOSS* for the text-database discovery problem. In *Proceedings of the 1994 ACM SIGMOD Conference*, May 1994.

[5] Steve B. Cousins. A task-oriented interface to a digital library. In *CHI 96 Conference Companion*, pages 103–104, 1996.

[6] 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, August 1996.

[7] Michelle Q Wang Baldonado and Terry Winograd. SenseMaker: An information-exploration interface supporting the contextual evolution of a user's interests. In *Proceedings of CHI 97*, 1997.

[8] Scott W. Hassan and Andreas Paepcke. Stanford digital library interoperability protocol. Technical Report SIDL-WP-1997-0054, Stanford University, 1997. Accessible at `http://www-diglib.-stanford.edu/cgi-bin/WP/get/-SIDL-WP-1997-0054`.

[9] USMARC format for bibliographic data: Including guidelines for content designation, 1994.

[10] Stuart Weibel, Jean Godby, Eric Miller, and Ron Daniel, Jr. OCLC/NCSA metadata workshop report. Accessible at `http://www.oclc.org:5047/-oclc/research/publications/weibel/-metadata/dublin_core_report.html`, March 1995.

[11] Luis Gravano, Chen-Chuan K. Chang, Héctor García-Molina, and Andreas Paepcke. STARTS: Stanford proposal for Internet meta-searching. In *Proceedings of the 1997 ACM SIGMOD Conference*, 1997.

[12] Darren R. Hardy, Michael F. Schwartz, and Duane Wessels. Harvest user's manual, January 1996. Accessible at `http://harvest.transarc.com/afs/-transarc.com/public/trg/Harvest/-user-manual`.

[13] 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.

[14] Z39.50 application profile for the content specification for digital geospatial metadata or GEO, October 1995. Accessible at `ftp://h2o.usgs.gov/-wais/docs/AppProfile_GEO1.2.ps`.

[15] National Information Standards Organization. *Information Retrieval (Z39.50): Application Service Definition and Protocol Specification (ANSI/NISO Z39.50-1995)*. NISO Press, Bethesda, MD, 1995. Accessible at `http://lcweb.loc.gov/-z3950/agency/`.

[16] Z39.50 Maintenance Agency. Attribute set Bib-1 (Z39.50-1995): Semantics. Accessible at `ftp://-ftp.loc.gov/pub/z3950/defs/bib1.txt`, September 1995.

[17] Terence R. Smith, Steven Geffner, and Jonathan Gottsegen. A general framework for the meta-information and catalogs in digital libraries. In *Proceedings of the First IEEE Metadata Conference*, Silver Spring, Maryland, April 1996. IEEE. Accessible at `http://www.nml.org/resources/misc/-metadata/proceedings/smith/ieee.html`.

[18] Eliot Christian. Application profile for the government information locator service (GILS), Version 2, October 1996. Accessible at `http://www.usgs.-gov/gils/prof_v2.html`.

[19] William P. Birmingham. An agent-based architecture for digital libraries. *D-Lib Magazine*, July 1995.

[20] Denis Lynch. Implementing Explain. Accessible at `ftp://ftp.loc.gov/pub/z3950/articles/-denis.ps`.