An Architecture for Digital Object Typing

Christophe Blanchi, Jason Petrone
Corporation for National Research Initiatives
Reston, Virginia 20191
{cblanchi, jpetrone}@cnri.reston.va.us

Abstract

Though the Internet provides a powerful platform for distributed access to data, current standards fail to encompass management of complex information. The ability to mediate between heterogeneous information formats is a key foundation on which more powerful information systems can be built. This paper outlines our efforts in creating a digital object architecture that facilitates the management of distributed heterogeneous information. The key features of the architecture are: (i) the ability to aggregate multiple data sources and relevant processing services into a single logical entity; (ii) a flexible, distributed, digital object typing mechanism; and (iii) support for dynamic aggregation of information processing services.

1 Introduction

The Web's rapid growth has made the task of finding and interacting with content increasingly difficult. Although simple access of web content is generally straightforward, programmatic interaction with web applications requires extensive coordination between the content provider and recipient. This results largely from the scarcity of standards for describing complex digital content. Distributed object technologies, such as CORBA, RMI, and WSDL/SOAP provide a convenient platform for building transport transparent APIs, but do not address the greater problem of information-level interoperability. To achieve this degree of interoperability, system developers must create new protocols for each type of information exchange. While current efforts such as ebXML[Org01] and Microsoft's BizTalk[Mic00] attempt to address this need for exchange protocol definition, they primarily deal with creation of static protocols that can not proliferate in an automated fashion. In this paper we introduce an architecture that facilitates management of heterogeneous information through the use of a dynamic and distributed digital object typing mechanism.

We at CNRI have been researching information-centric architectures for managing content and services in the context of distributed networked environments. Our research originated in the digital library community where we developed experimental digital object systems with the Library of Congress[ABL97] and Defense Technical Information Center[DVL]. Although our work was first geared to solving specific digital library problems such as the need for Universal Resource Names (URN)[ADD$^+$96] and mechanisms for aggregating data with metadata, the resulting architecture addresses issues present in general distributed information management[xiw97, KL01].

Our implementation provides a general framework and methodologies for interaction with content and services within a distributed network environment. At the center of this architecture is the notion of the digital object— a uniquely identified data abstraction that encapsulates content and access policies while providing a high level, self-describing, type definition. Digital objects are described through the use of an abstract typing mechanism that we refer to as a *content type*. Content types describe the intents of use that a digital object creator has for the object by defining the finite set of views that can be acquired from it. These views are referred to as *disseminations* of digital objects.

The following sections describe the various components of the digital object architecture, further define the notions of content types, and explain the concept of digital object chaining.

This research is supported by the Defense Advanced Research Project Agency (DARPA) on behalf of the Digital Libraries Initiative under Grant No. N66001-98-1-8908.

2 Digital Object Architecture

The digital object architecture defines a core set of services for identification, access, and management of digital objects. These services represent the minimum functionality required for interoperability.

In our current implementation, digital objects and other elements within the architecture are uniquely identified using the Handle System^{TM}[SL03]. The Handle System^{TM} is a comprehensive persistent naming system. Identifiers within the Handle System^{TM} are called "handles". We use handles within the architecture for their ability to provide globally unique persistent identification and location independent references.

2.1 Digital Object Repositories

A digital object repository exposes an external interface for creation, modification, and access of digital objects, as well as assuming storage responsibilities. Repositories enforce the access policies that pertain to each digital object and provide a safe environment for generation of content type disseminations of digital objects. Repositories also provide the necessary functionality for creating new digital object content types. There is no limit to the number of repositories that may coexist within the architecture.

Repositories themselves are digital objects. This enables them to provide additional services through the use of disseminations. For example, a repository could allow disseminations of indices of its metadata or of administrative information, such as server logs.

In our interoperability experiments with Cornell[PL98, PBLO99], we defined the minimum required interface and protocol specifications needed to guarantee interoperability between independent repository implementations. The specifications allow for a great degree of flexibility in repository implementation and storage facilities. Digital object repositories have been implemented using object oriented databases, relational databases[SR00], and simple file system storage with no perceptible change in behavior.

2.2 Repository Access Protocol

The Repository Access Protocol(RAP) is used to access repositories and their respective digital objects. RAP includes operations for creation, deletion, modification, metadata access, and dissemination of digital objects. RAP has been defined using CORBA IDL[Obj99] and as a binary protocol on top of TCP/IP connections.

The location of a repository is specified in RAP in the form of a handle. A repository's handle resolves to the information required by a client to establish a connection. Repositories using the CORBA interface have a CORBA object reference called an IOR stored in their handle. Repositories accepting connections via TCP/IP have an IP address or host name stored in their handle.

2.3 Digital Objects

Digital objects provide the primary form of information representation within the architecture. The original notions of digital objects were first described in "A Framework for Distributed Digital Object Services"[KW95] and were further developed in subsequent research[PL98, ABL97, BP01].

At an abstract level, digital objects are uniquely identified network entities that encapsulate, describe, and provide value-added access to data. Digital objects are managed using the operations defined in the RAP protocol. Digital objects can be used to aggregate multiple elements of relevant data, so that complex, multi-part information may be managed as a single entity. This is a useful feature for implementation of security policies, mirroring, and archiving. Clients accessing these complex digital objects typically do not retrieve the entire object all at once. Instead, clients can request specific disseminations of a digital object based on individual needs and access permissions.

Digital objects are identified using handles. A digital object's handle resolves to the handle of the repository that contains its respective digital object. Creators of digital objects may also elect not to assign a handle to a digital object and instead use an arbitrary identifier. Accessing a digital object without a handle requires prior knowledge of the repository in which it resides.

All digital object RAP read operations are referred to as digital object disseminations. Within this general notion of digital object dissemination, we identify two separate classes, primitive disseminations and content type disseminations.

Primitive disseminations provide direct access to the contents and attributes of the digital object as they were deposited into the object. Unlike content type disseminations, primitive disseminations are not extensible. The full set of primitive digital object disseminations is shown in Table 1.

Content type disseminations provide intent-of-use based access to a digital object. They provide a view of a digital object's content that is independent of its

Operation	Description
CreateDataStream	Attaches a typed byte stream to the digital object.
GetDataStream	Retrieves an attached data stream and its associated metadata from the digital object.
DeleteDataStream	Deletes an attached data stream from the digital object.
ListDataStreams	Returns a list of data streams attached to the digital object.
GetDissemination	Invokes a content type dissemination on the digital object and returns the resulting data.
CreateDisseminator	Creates a content type disseminator for the digital object by associating a set of data streams with the handle for a specific content type.
DeleteDisseminator	Removes a content type disseminator from the digital object.
ListDisseminators	Returns a list of content type disseminators for the digital object.

Table 1: Primitive Digital Object Disseminations

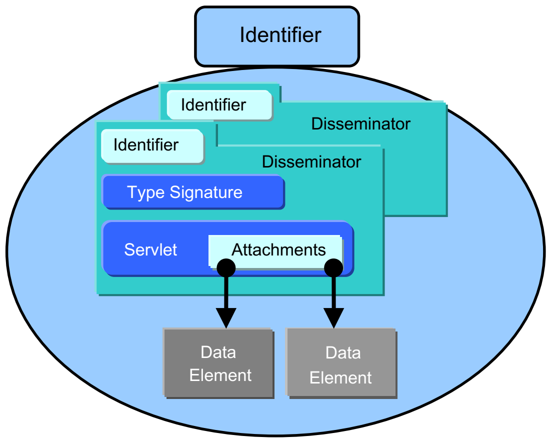


Figure 1: Digital Object Structure

content-specific encoding or underlying complexity. A content type can be bound to a digital object by its creator. When a content type dissemination of the digital object is requested, the content type executable is executed within the context of the object, where it can generate a content type dissemination from the object's data.

Digital objects achieve all their functionality through the use of two internal data structures: *data elements* and *disseminators*. An illustration of the structures of a digital object can be found in Figure 1.

Data elements store sets of sequences of bytes within a digital object. Each digital object can have any number of uniquely identified data elements. Each data element maintains its set of byte streams, as well as a few minimal metadata fields: data type, stream length, date created and date last modified. Client interaction with data elements occurs over RAP through the use of primitive disseminations.

A disseminator is a structure that binds a content type to a set of data elements within a digital object. A separate structure is needed so that different content types can be associated with arbitrary permutations of the same data elements. A digital object can have any number of uniquely identified disseminators. Each disseminator can be bound only to a single content type, though multiple disseminators within a digital object may be bound to the same content type. The data elements associated with a disseminator are referred to as *disseminator attachments*.

3 Content Types

As described in the previous sections, digital objects are typed with what we refer to as content types. Content types represent a flexible mechanism for loose association of data with relevant distributed services. A content type is defined as a set of operations used to act on a particular class of content. Each operation has a semantically relevant name, as well as a human readable description of its purpose.

Content type operations receive input from two sources: input parameters and digital object data elements. An operation receives the data for its input parameters from incoming dissemination requests. The entity requesting the dissemination provides these parameters. Each parameter has a semantically relevant name and a human readable description of its relation to the functionality of the operation.

An content type operation has access to data elements as the streams of raw content contained within the digital object to which the content type is applied. An operation refers to data elements using an identifier called a role. This identifier is defined by the content type creator and is understood by creators of digital objects that use the content type. The creator of a digital object associates attached data elements with a particular role. The content type uses a data element's role to distinguish it from other elements within the digital object.

Operation parameters, data elements, and dissemination results are all typed using traditional data types. These data types include primitive types such as integer, byte arrays, and character strings, as well as higher level representations such as MIME types[FB96].

This technique of data typing bears some resemblance to MIME, a standard designed to facilitate interoperability in Internet email attachments. However, there are a number of differences between MIME and digital object content types. MIME types are concerned with expressing the particular structure within a set of bytes, while content types denote the manner in which the data is to be used, irrespective of representation format. For example, an SGML file containing the script for the play Hamlet would have a MIME type of `text/sgml`. Since SGML is a generic file format and requires a separate document type definition (DTD) file for interpretation, the MIME type in of itself does not provide any context for interpreting the contents of the file. A content type for this same script could be "`Script`" or even as specific as "`ShakespereanDrama`". These content types could define operations for accessing the content by retrieving a single act or providing a version of the play typeset in a manner consistent with the plays genre. The MIME type does not provide any aid for understanding how to make advanced usage of the play, and it would require a knowledgeable person to acquire the appropriate tools to do so.

While both MIME types and content types are registered with unique identifiers, the process of registration for each differs greatly. MIME registration requires submitting a proposal to the Internet Engineering Steering Group (IESG) for peer review[FKP96]. Registration of a MIME type is contingent upon IESG approval. To prevent the MIME type registry from becoming overburdened, few MIME types are adopted as standards.

Unlike MIME type registration, content type registration is dynamic. Content types are registered using the Handle System so that the registry may be distributed. This allows for registration of many content types without the scalability problems of a centralized index. It also allows for content type

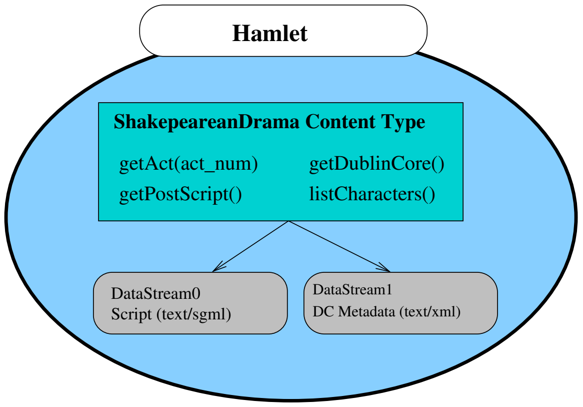


Figure 2: Digital Object for Hamlet

providers to independently administer their own registries. This means individual organizations can globally register content types with autonomy.

Content types are represented in two parts, a *type signature* and a *servlet*. A type signature defines the exact set of views that can be disseminated from a digital object. More specifically, a type signature provides an abstract definition of the operations that a client can request using content type dissemination. It is a static structure containing only names and descriptions of the methods and parameters. Type signatures are stored within digital objects and are uniquely identified with handles. The handle for a type signature resolves to the handle of the digital object in which it resides.

A servlet implements the operations defined in one and only one type signature. The servlet provides the executable program that a digital object uses to generate content type disseminations. As with the type signature, the servlet has its own unique handle. This handle resolves to the location within the digital object architecture where the servlet resides.

Separating the interface from the implementation makes it possible to provide a consistent interface for functionally similar digital objects with different underlying structures. Even though the internal representation of the information could be dramatically different, the access interface remains consistent.

In the `ShakespereanDrama` content type we discussed earlier, the internal format of the data was SGML. Another digital object could use the `ShakespereanDrama` type, but with a servlet that operates on LaTeX formatted text using a special stylesheet instead. Even though the internal representation of the information is dramatically different, the access layer remains consistent. An illustration of this example can be found in Figure 2.

After a new content type's signature and associated servlets are written, they are assigned handles and deposited into their own digital object. The handle values for the type signature and servlet contain the information necessary to locate them from within the architecture. From that moment onward, the rest of the system has the ability to acquire and use the new content type. A digital object creator can associate the new content type with a disseminator by referencing the handles of the content type's type signature and servlet.

In experimenting with the digital object content types in the context of our architecture, we have demonstrated that they can facilitate homogeneous interaction with complex heterogeneous data types in a flexible and standardized manner.

4 Digital Object Chaining

The functionality of a digital object can be extended by sequentially chaining digital objects with each other. This works much like command piping in Unix-like operating systems[Joy86]. The output from a digital object dissemination is used as the input for the dissemination of another digital object. Digital object chains can be predefined in order to provide a composite service built from multiple special purpose components. Clients wishing to obtain functionality not explicitly provided by the existing content types can also create chains on the fly. Furthermore, it is possible for a digital object to dynamically discover new chain combinations so that it may offer new operations as the system expands.

As described in section 2, digital objects can aggregate content in two forms. Content can be stored in its raw form, inside the digital object. The data element for the content contains the bytes that make up the content, just as it would be stored on disk in a file. Digital objects can also store references to other digital objects as data elements. For example, an object that analyzes weather data could have a data element that is a reference to a digital object that acquires weather data. When the weather analysis object is accessed, the repository performs a content type dissemination on the weather acquisition object and attaches the result of that dissemination to the servlet. Using this technique, different stages in a data transformation process can be compartmentalized to promote reuse and individual management.

Reference data elements are somewhat limited in that they are defined at the time the object is created. It is often desirable to specify a sequence of disseminations independent of the structure of the digital objects involved. Less static chaining can be accomplished by writing servlets that can retrieve digital object dissemination on their own. Rather than relying on the repository to dynamically attach the data, servlets implemented in this fashion can retrieve different disseminations depending on the given context.

Consider the digital object containing the play Hamlet written in English. This object could be chained with an object that generates play summarizations. The output from the summarization object could then be then fed into a German translation digital object. In this manner, a summary of Hamlet in the German language could be generated, even though the original digital object was not explicitly designed to provide this functionality.

To facilitate this aggregation, the use of a special chaining object is required. When a dissemination of the chaining object is invoked, the methods from different digital objects are disseminated in succession, passing along the data down the chain. The technique is illustrated in Figure 3.

In this example, Hamlet is stored in a chaining digital object of type `EnglishText`, which provides the operation `SummarizeInGerman()`. The operation first invokes a dissemination of the summarization object using the text of Hamlet as a parameter. The method then invokes a dissemination of the German translation object using the results from the summarization dissemination as an argument. This effectively aggregates the functionality provided by the translation and summarization objects in a manner transparent to the user.

A more difficult means of aggregating digital objects is by manually chaining at time of use. Continuing with the Hamlet example, consider the case where the digital object's creator does not include desired functionality. The object containing Hamlet may provide operations `SummarizeInEnglish()`, and `SummarizeInFrench()`, but not `SummarizeInGerman()`. Since the client wishing to obtain a German summary may not have sufficient system access to modify the object to include this operation, chaining must be done using a different technique. In this situation, a special utility chaining object can be used to manually connect digital objects. This run-time chaining would ideally be accomplished using a GUI application that could provide visual feedback to the user as digital objects are connected.

In addition to the methods of chaining described above, the possibility exists for digital objects to discover and dynamically build chains based on infor-

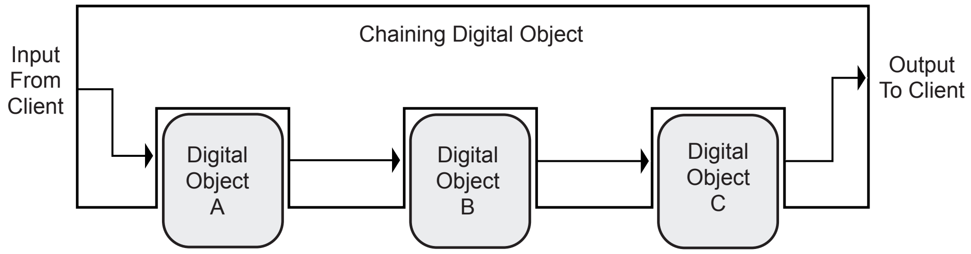


Figure 3: Using the chaining mechanism

mation found within the operation descriptions for digital object types. Each operation in a digital object type specifies its input requirements as MIME typed data streams and basic primitive values such as integers and booleans. A human readable description is also included for each input. For output specifications, the digital object provides a human readable description as well as a list of all possible MIME types that it may return as output. Chains are built dynamically by linking digital objects where the output from any given object within the chain satisfies the input restraints of the subsequent object.

Because the information describing content types is limited, there is no guarantee that a hypothetical chain will execute correctly or produce valid results. Chains should be viewed as possibilities and must be subject to manual review before execution. The manual review could be accomplished within an interactive tool, by presenting human readable descriptions of the state of the data as it moves through the chain. This should provide enough evidence for a user to determine if a chain is likely to produce valid results. Figure 4 shows an example of such a utility.

5 Future Work

In addition to refining the foundation of the architecture laid out in the previous sections, we are also working on extending the system to encompass other functionality useful in distributed information management.

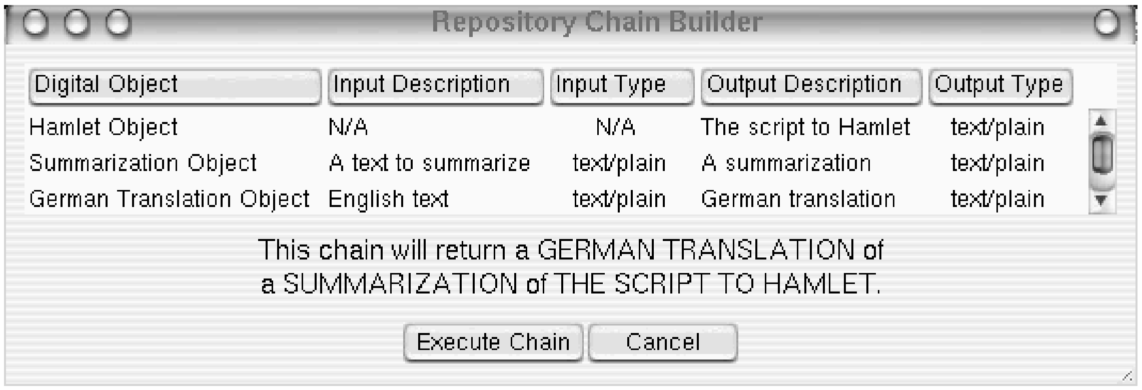
Though we do not directly address security in this paper, we're working on implementing an extensible security layer for the architecture. There are several facets to security that we must address. The architecture should provide for authentication, which involves assigning client identities and supplying the means for determining those identities at a later time. Security also involves authorization. Once an identity for a client has been determined, the system must have a way to specify what system operations the client is allowed to perform. It is important for this functionality to operate at the digital object level so that the information the object holds is protected and not just the channel used to access it[KL97], as is the case with end-to-end security systems. To maintain the level of extensibility the system provides, each digital object should be able to provide its own security mechanisms, independent of the repository in which it resides[PL00]. Since a digital object may not fully trust its host repository, the object must also be capable of having its contents encrypted while carrying references to the mobile code needed for decryption.

It is our goal to design an extensible security framework that allows for the coexistence of diverse cryptographic and rights management mechanisms. This framework will provide for the protection of both digital objects and the repositories that contain them.

Digital object replication and mirroring is another area we are beginning to explore. Since digital objects can be easily moved between repositories, we must develop strategies for optimal replication and migration. At the present time, repositories have the ability to perform migration and replication tasks, but only when explicitly instructed to do so. The system design permits much richer, dynamic forms of digital object movement. Digital objects can move from repository to repository in order to optimize access and to provide load balancing. However, this additional movement introduces concurrency issues. We are now investigating possible schemes for effective caching and replication of digital objects.

6 Conclusion

This document describes an architecture that allows for interoperability of high level information. By typing information based on intent of use, and not just encoding, it is possible to access and



Digital Object	Input Description	Input Type	Output Description	Output Type
Hamlet Object	N/A	N/A	The script to Hamlet	text/plain
Summarization Object	A text to summarize	text/plain	A summarization	text/plain
German Translation Object	English text	text/plain	German translation	text/plain

This chain will return a GERMAN TRANSLATION of a SUMMARIZATION of THE SCRIPT TO HAMLET.

Figure 4: Tool for manipulating chains

manage heterogeneous content in a consistent fashion. We have applied a prototype of this system to the Library of Congress' National Digital Library project[ABL97]. An implementation of the architecture is also currently in use for the Defense Technical Information Center's Defense Virtual Library[DVL]. The interoperability and extensibility of the system has been demonstrated in a series of experiments conducted in conjunction with the Digital Library Research Group at Cornell University[PL98, PBLO99]. These projects have clearly demonstrated the potential of the architecture to simplify access and exchange of heterogeneous information.

References

[ABL97] W.Y. Arms, C. Blanchi, and C. Lagoze. An architecture for information in digital libraries. *D-Lib Magazine*, February 1997.

[ADD+96] W. Arms, L. Daigle, R. Daniel, D. LaLiberte, Michael Mealling, Keith Moore, and Stuart Weibel. Uniform resource names: A progress report. *D-Lib Magazine*, February 1996.

[BP01] C. Blanchi and J. Petrone. Distributed interoperable metadata registry. *D-Lib Magazine*, December 2001.

[DVL] Defense Technical Information Center. Defense Virtual Library. http://dvl.dtic.mil.

[FB96] N. Freed and N. Borenstein. Multipurpose internet mail extensions (MIME) part two: Media types. Request for Comments 2046, Internet Engineering Task Force, November 1996.

[FKP96] N. Freed, J. Klensin, and J. Postel. Multipurpose internet mail extensions (MIME) part four: Registration procedures. Request for Comments 2048, Internet Engineering Task Force, November 1996.

[Joy86] W. Joy. An introduction to the c shell. In *UNIX User's Supplementary Documents*, volume 1. Computer Systems Research Group, Department of Electrical Engineering and Computer Science, April 1986.

[KL97] U. Kohl and J. Lotspiech. Safeguarding digital library contents and users: Protecting documents rather than channels. *D-Lib Magazine*, September 1997.

[KL01] R. Kahn and P. Lyons. Representing value as digital objects. a discussion of transferability and anonymity. *D-Lib Magazine*, May 2001.

[KW95] R. Kahn and R. Wilensky. A framework for distributed digital object services. Unpublished manuscript, Corporation for National Research Initiatives, Reston, Va., May 1995.

[Mic00] Microsoft Corporation. *BizTalk Framework 2.0: Document and Message Specification*, December 2000.

[Obj99] Object Management Group. *The Common Object Request Broker: Architecture and Specification*, October 99.

[Org01] Organization for the Advancement of Structured Information Standards. *ebXML Technical Architecture Specification*, February 2001.

[PBLO99] S. Payette, C. Blanchi, C. Lagoze, and E. Overly. Interoperability for digital objects and repositories - The Cornell/CNRI experiments. *D-Lib Magazine*, May 1999.

[PL98] S. Payette and C. Lagoze. Flexible and extensible digital object and repository architecture. 1998.

[PL00] S. Payette and C. Lagoze. Policy carrying, policy-enforcing digital objects. In *Proceedings of the Fouth European Conference on Research and Advanced Technology for Digital Libraries*, 2000.

[SL03] S. Sun and L. Lannom. Handle system^{TM} overview. Internet draft, Internet Engineering Task Force, January 2003.

[SR00] T. Staples and Wayland R. Virginia dons FEDORA: A prototype for a digital object repository. *D-Lib Magazine*, July 2000.

[xiw97] Managing access to digital infomation: An approach based on digital objects and stated operations. Technical report, May 1997.