Modeling Decisions for Digital Content

Naomi Dushay and Carl Lagoze

Cornell Digital Library Research Group
Department of Computer Science
Cornell University
Ithaca, NY 14853-7501
{naomi, lagoze}@cs.cornell.edu

Abstract

The organization of digital information poses unique challenges not only due to rapidly evolving data formats and software, but also because digital content may be dynamic, distributed, or executable. Managers of digital information need flexible, extensible architectures that facilitate maintainability as well as accessibility and long-term utility of content. However, the flexibility of these architectures presents difficult organizational, or modeling, decisions. To explore these modeling decisions and their consequences, we introduce four dimensions of digital content modeling: aggregation, interfaces, transformations and indirection. We discuss these dimensions individually and also examine their interaction; in essence, this paper examines general design patterns for digital content modeling. As new and powerful architecture evolve, it is vital that information mangers understand and consider these design decisions.

1 Introduction

Organizing large quantities of information to promote accessibility and maintainability has always been challenging. With digital content, additional levels of complexity are present, not only because of rapidly evolving software, hardware and data formats, but also due to new management and organization challenges that result from the new technologies. For example, digital information can be remotely located, dynamic (today's weather prediction for Ithaca) or executable (an interactive program, e.g. a backgammon game).

While the Internet can deliver dynamic, distributed, and executable content, two difficult tasks for managers of digital information are 1) making complex digital content comprehensible to users and 2) managing the digital content and its complex interrelationships. Digital content stewards need a flexible, extensible architecture that enables maintainability as well as accessibility and utility of content. But with such an architecture comes myriad organizational, or modeling decisions. How should digital content be modeled to promote ease of maintenance? To ensure continued utility of the information? To promote easy access? What are the tradeoffs of choosing one organization scheme over another?

To discuss such modeling decisions, we introduce four dimensions of digital content modeling:

1. ***Aggregation***: What information will be stored in a digital object? What bit streams will be stored, and which bit streams will be aggregated into a single digital object?

2. ***Interfaces***: How will information be made available and presented to users? Using object-oriented terms, what methods will be published for use by agents and users? What disseminations of bit streams will be delivered to users as a result of those methods? What methods should be grouped into a single interface, and which interfaces should be associated with a digital object?

3. ***Transformations***: How does the data stored in a digital object correspond to the disseminations available through its interfaces? What mechanisms will be used to transform what is stored to

what is delivered to users? What aspects of transformations will be static, what aspects parameterized?

4. ***Indirection***: What information will be stored within the digital object, and what will be accessed at a remote location? How will remote information be accessed?

This paper will explore these dimensions and some of the ways they interact when considering different modeling choices. Our experience has shown that the nature of and motivations for these choices can impact digital library design and management. For this very reason, we cannot prescribe particular modeling decisions. However, as new and powerful architectures evolve it is vital that information managers understand and carefully consider the implications of these design decisions.

The remainder of this paper is organized as follows. In section 2 we describe the features of an enabling architecture for digital content that sets the context for modeling decisions. In section 3 we explore two examples to begin understanding the dimensions of digital content modeling. We take a closer look at each of the modeling dimensions and how they interact in section 4 and we state our conclusions in section 5.

2 A Flexible, Extensible Architecture for Digital Content

Our analysis presumes digital content architectures of considerably greater power than those currently available through the World Wide Web. In particular, we assume architectures that:

- accommodate new types of digital content as they emerge, as well as mechanisms for transforming and manipulating that content,
- support aggregation of heterogeneous, possibly distributed digital content into single entities,
- support multiple, extensible disseminations of information from these aggregated entities,
- have an open, well-defined protocol facilitating repository interoperability and confederation [1].

Within our digital library research at Cornell we have been experimenting with one such architecture, Fedora (Flexible and Extensible Digital Object and Repository Architecture)[1], described in [2]. The Cornell Digital Library Research Group[2] and the Corporation for National Research Initiatives (CNRI)[3] have each implemented an interoperable version of the Fedora architecture in Java, using CORBA as a transport layer [3]. The University of Virginia Library has implemented an alternative version of Fedora using MYSQL and various scripts, which uses HTTP as a transport layer [4]. There are similar architecture efforts in the digital library community, such as Smart Object, Dumb Archive (SODA)[5], Dienst[4] [6], Multivalent Documents [7], and that proposed by the Making of America II project [8].

Please note that the example digital content models in this paper are conceived within the Fedora architecture, but the modeling concepts and issues are applicable to other architectures. In some sense, we are describing general design patterns for digital content modeling.

Fedora, like SODA, builds on the concepts of the Kahn/Wilensky Framework [9] and the model presented in the Warwick Framework [10]; Fedora uses a container abstraction to aggregate distinct resources into a single abstract container, called a *digital object*. These resources may either be local to the digital object – i.e., physically contained within the object – or derived from external requests to other digital objects or other networked resources. Fedora builds on the Distributed Active Relationship (DAR) abstraction [11] [12] to provide extensible disseminations from that digital object.

---

[1] http://www.cs.cornell.edu/cdlrg/FEDORA.html
[2] http://www.cs.cornell.edu/cdlrg/
[3] http://www.cnri.reston.va.us/
[4] http://www.cs.cornell.edu/cdlrg/dienst/architecture/architecture.htm

Another key facet of the Fedora architecture is the segregation of digital object structure, interfaces and mechanisms. This segregation allows digital content storage, disseminations and transformations to be specified independently of each other. It is this independence that allows us to view the dimensions of digital information modeling as orthogonal.

Throughout this paper, we use visual metaphors to represent Fedora digital objects. We introduce these metaphors in Figure 1, and also present some fundamentals of the Fedora architecture.

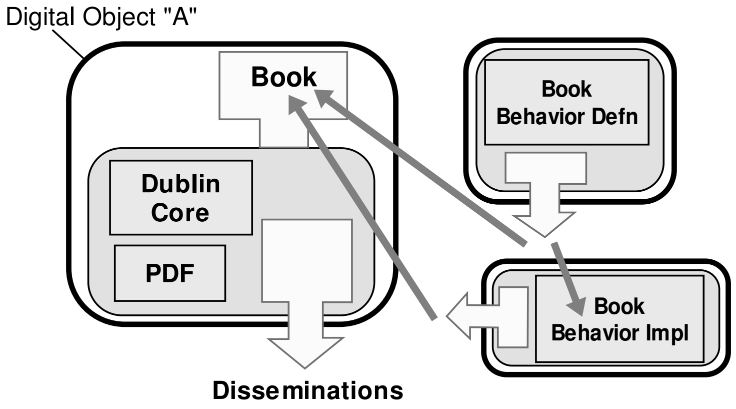

Figure 1 - three Fedora digital objects

Digital Object "A" in Figure 1 physically contains two bit streams: a PDF file and a Dublin Core[5] record. It has one interface, or set of behaviors, associated with it: a BOOK interface. The BOOK interface is defined by a set of behaviors called the Book Behavior Definition; the individual behaviors might include getPage(pageNo), getTitle(), getTableOfContents(), etc. Figure 1 shows only one interface attached to digital object "A", but Fedora digital objects can have any number of attached interfaces.

A digital object disseminating a Book Behavior Definition is shown in the upper right of the figure. Note that there is no interface associated with this digital object: Fedora provides a bootstrap mechanism for disseminations of Behavior Definitions. Similarly, the digital object in the lower right disseminates a Book Behavior Implementation via a bootstrap mechanism.

The Book Behavior Implementation is executable code that transforms stored content into disseminations based on behavior requests. This transformation occurs when a user requests a dissemination from digital object "A". In our example, a getTitle() request on digital object "A" would cause the Book Behavior Implementation to parse the stored Dublin Core data and deliver the title to the user. Note that the separation of the interface (BOOK behavior definition in our example) from the behavior implementation allows the transformations to change while the interface remains stable. This will be illustrated in the next section.

3 Getting Started – A Comparison of Two Models

We begin exploring the digital content modeling dimensions by simplifying the problem. In the two examples below, we hold the interface dimension constant: in both cases, the user has access to the same set of behaviors (or methods) to request disseminations. However, we vary the three other dimensions: aggregation, transformations and indirection. What is stored in the digital objects is different, which in turn impacts the transformations from physical storage to available disseminations, and we introduce a form of indirection in the second example.

[5] http://purl.org/dc/

The DOCUMENT interface is attached to digital objects “B” and “C” in the figures below. Because the interface is the same, it refers to the same Document Behavior Definition. Let’s assume the behavior definition has two methods: getBibliographicData() and getPostScriptRendition().

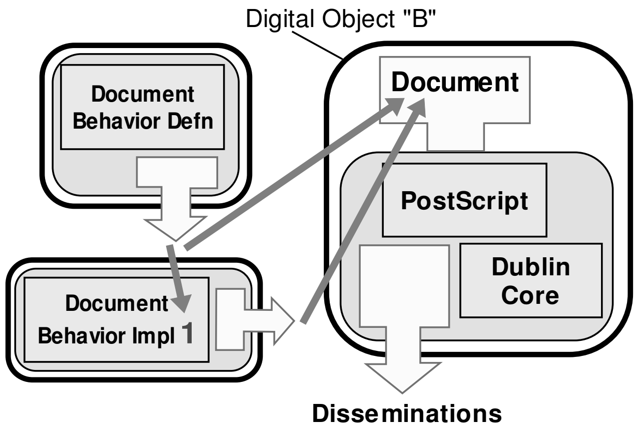

Figure 2 - digital object “B”

In Figure 2, the content stored in digital object “B” is 1) a Dublin Core bit stream containing bibliographic metadata for the document and 2) a PostScript representation of the document. The transformations to get from the stored content to the disseminations for digital object “B” are quite simple: getBibliographicData() will return the Dublin Core bit stream and getPostScriptRendition() the PostScript bit stream. Document Behavior Implementation 1, shown in the lower left, carries out these simple transformations.

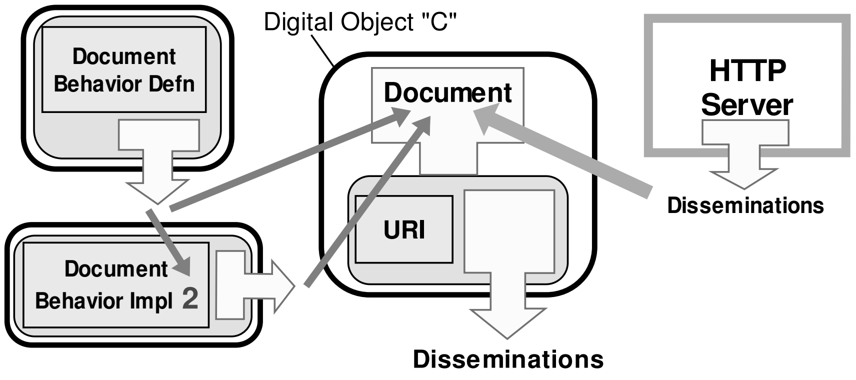

Figure 3 - digital object "C"

The content stored in digital object “C” in Figure 3 and the transformations performed on that content are completely different from that in digital object “B”, even though the interface is the same: getBibliographicData() and getPostScriptRendition() are still the dissemination request methods. Digital object “C” contains only a URI, which is used by Document Behavior Implementation 2 to make HTTP requests to an external server. When digital object “C” receives a getBibliographicData() request, Document Behavior Implementation 2 uses the stored URI to make an HTTP request to an external server, and then parses the dissemination from the HTTP server in order to pull out Dublin Core data to deliver to the user. Similarly, a getPostScriptRendition() request causes Document Behavior Implementation 2 to make an HTTP request and then parse the results for PostScript data to deliver to the user. These transformations are more complex than those carried out by Document Behavior Implementation 1 in Figure 2.

Because the stored URI in digital object "C" is used to get information external to the digital object, a form of indirection is present. We could even say that digital object "C" is a Fedora wrapper for a particular document (indicated by the URI) in the HTTP server.

3.1 Why choose one model over another?

In the previous two examples, we held the interface dimension constant while altering the aggregation, transformation and indirection dimensions: the user can get exactly the same information from these two digital objects, but what is stored and how it is transformed is different. So why choose one model over another?

To examine the issues, let's assume that the HTTP server in Figure 3 is storing the same Dublin Core and PostScript bit streams shown in Figure 2. This means digital object "B" in Figure 2 repeats the storage of files: Dublin Core and PostScript files are stored in the HTTP server as well as in digital object "B". Figure 3 avoids repetitious storage, but adds vulnerability due to indirection: it is dependent on URI reliability, the robustness of an external service as well as network stability. There may also be issues of maintainability: if changes are made to the external files, then the digital object in Figure 2 will need to be updated, while the digital object in Figure 3 requires no changes. But if we don't believe the external repository will be maintained for the life of the digital object, then Figure 2's model may be preferable to Figure 3's. Aside from these preservation concerns, there may also be security concerns, such as access rights to disseminations from the HTTP server vs. disseminations from the digital objects in the repository.

Most digital object modeling decisions will require similar considerations -- anticipating not only future needs of users, but future maintenance of services outside the scope of the repository or even the owning institution, e.g. a remote service or connectivity to that service. We will further explore modeling decisions and their ramifications in the rest of this paper.

4 A Closer Look at the Modeling Dimensions

In section 3, the interface for the digital objects was already known – we wanted our digital object to "speak" DOCUMENT. But the starting point for modeling decisions may vary. There may be situations where aggregation is the first dimension to be considered: digital content stewards could have lots of digital content which first needs to be organized for maintainability and then have interfaces provided. Or there might be initial questions about indirection: what should be stored in the local repository, and what should be requested from external sources?

In this section, we will analyze choices for each dimension and their ramifications. Since the dimensions aren't truly orthogonal, we will also explore how decisions for one dimension affect the others. We begin with the interface dimension, and then we examine the aggregation dimension, including storage and maintenance of mechanisms. We next discuss transformations and finish with a discussion of indirection.

4.1 Interfaces

In many cases, defining interfaces for a digital object is one of the most challenging modeling issues. It's not only a matter of anticipating what information a user will want to get out of a digital object, but also of understanding the appropriate granularity and grouping of behaviors. For example, Dublin Core behavior might be defined as getTitle(), getCreators(), getDescription(), etc. Or it might be defined as getDCElement(elementName), or even getDCRecord(), which would return the entire Dublin Core record to be parsed by a client application. This last option requires further decisions: in what format should the Dublin Core record be disseminated? XML? Plain text? RDF? Or should that be specified by the user via parameter, e.g. getDCRecord(format)?

Payette suggests that access control and other policies will be enforced during the execution of behaviors [13]; in effect, the granularity of behavior definitions is closely coupled with policy enforceability. In our Dublin Core example, we must decide if we want to have policies pertaining to the entire Dublin Core record (getDCRecord()) or to individual elements (getDCElement(elementName) or getTitle(), getCreator(), ...). There may be security issues in providing data to users or agents: it may be okay to disseminate the title, but a security violation to send the whole Dublin Core record.

Another facet of interface choices lies in the grouping of requests into behavior definitions. For example, should the LECTURE behavior definition include a getHomeworkAssignment() request or should there be a separate HOMEWORK behavior definition with getAssignment() and getSolution() requests?

Stepping back further, we must determine which interfaces will be associated with any given digital object. In addition to DUBLIN CORE, we might want a given object to speak BOOK and PHOTO ALBUM or any number of other interfaces. We might create new interfaces for a digital object, or use existing ones. The Fedora architecture allows new interfaces to be associated with a digital object at any time, so all of these decisions need not be made at digital object creation time.

Another question to consider is who defines interfaces? Who maintains them and makes them available? We envision digital content providers, either singly or in association, taking on these tasks.

4.2 Aggregation

In some cases, digital content stewards need to decide what to store locally and what to access at a remote location, while in other cases, digital content stewards already have lots of digital content and need to decide how to organize it. We address the former situation in section 4.4 when we discuss indirection; the latter situation is the main focus of the aggregation dimension – which bit streams should be grouped together into single digital objects?

Sensible aggregation choices may depend on anticipating the uses for and interfaces to be attached to a digital object: should we package images of different resolutions and a thumbnail together into one object, or should each image be in a separate object? If we expect the group of images to be used together, then we want to package them together, but if we anticipate individual images being used in particular contexts, then we may want to package them separately.

Preservation considerations such as maintainability and long-term utility of information may also impact aggregation decisions. Or we might want to apply policies such as access permissions at the package level, which would also impact aggregation choices.

There's a new wrinkle in the aggregation question when we consider the bit streams for mechanisms – the executable code containing behavior definitions or behavior implementations.

4.2.1 Where should mechanisms be stored?

Figures 1, 2 and 3 show each mechanism stored in a separate digital object. But the architecture may not require these to be stand-alone digital objects. In fact, Fedora digital objects can contain any number of behavior definitions and any number of behavior implementations, raising some interesting modeling questions.

For example, a digital object can contain all behavior definitions and implementations it uses – the mechanisms can be stored in same container as the "content." An example of such a "self-contained" digital object is shown in Figure 4. BOOK and DUBLIN CORE interfaces are attached to this digital object, and the bit streams for the mechanisms required for the interfaces' behavior definitions and behavior implementations are all stored in this digital object, along with "content" bit streams of type XML, PostScript and GIF.

Self-contained digital objects reduce dependency on external software and files: they do not need to locate or retrieve the mechanisms for their interfaces. One tradeoff for that reliability gain is the potentially repetitious storage of mechanism executables: the BOOK interface's mechanisms will be duplicated in every self-contained digital object using that interface. There are also implications if the BOOK interface is updated – would containing digital objects be updated, or would they continue using an older version?

How would this versioning of be tracked? Moreover, would the enforcement of policies on this digital object allow the mechanisms to be used by other digital objects or by external users? How shareable are these mechanisms? The Fedora architecture allows mechanisms to be globally shareable from anywhere, depending on policy decisions.

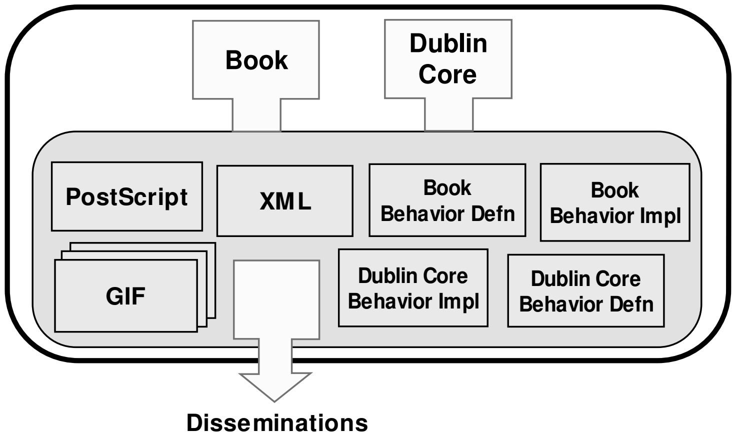

Figure 4 - self-contained digital object

If we choose not to store the mechanisms with the content in self-contained digital objects, then we are still faced with aggregation choices for the mechanisms. Should mechanisms be in stand-alone digital objects, as shown in section 3, or should they be grouped? We have returned to the aggregation question for mechanisms.

Mechanisms could be grouped by interface. For example, the behavior definition and the behavior implementation for the BOOK interface could both be packaged in a single digital object. If additional behavior implementations were created for the BOOK behavior definition, they might be added to this digital object. This avoids repetitious storage of mechanisms, and may make version handling of mechanisms easier. But what if someone who doesn't have write access to this digital object creates a new BOOK behavior implementation?

Another possible organization of mechanisms would be to create a "BehaviorDefinition" digital object and a "BehaviorImplemenation" digital object. So, in our example, the BOOK and DUBLIN CORE behavior definitions would be in the former digital object, while the behavior implementations would be in the latter digital object. This might make maintenance easier if a repository contained lots of interface mechanisms.

Any combination of the above aggregation schemes is possible in Fedora. For example, if an implementation of BOOK behaviors was likely to be used only by a particular digital object, it might make sense to store that BOOK behavior implementation in that digital object, even though the BOOK behavior definition would be accessed in another location. In another scenario, if Cornell's BOOK behavior implementation was proprietary, it might be easiest to enforce rights management if it was in a stand-alone digital object.

4.3 Transformations

Transformations by their very nature are constrained by what is stored in a digital object and the behaviors they are implementing. For instance, if we're storing a Dublin Core record in XML, and we have an interface with behaviors getTitle() and getAuthor(), then it's clear the transformation will involve parsing the XML, finding the relevant Dublin Core tags, and disseminating the correct information to the user. But there are many cases where transformation choices must be made, even given storage and behavior constraints.

As an example, let's assume we're storing a large, unsorted list of names, and we have a behavior request getSortedNames(). There are multiple sorting algorithms that will all produce the correct results, but the different algorithms require different amounts of resources such as memory or processing time. These types of decisions have been in the bailiwick of programmers in the past and we expect them to remain there, not with digital content stewards. On the other hand, given multiple behavior implementations that are functionally equivalent, digital content stewards will have to choose which one to use.

There are further interdependencies between aggregation, interfaces and transformations, especially concerning parameters in behavior requests. Consider the interface and aggregation scenarios for a digital object providing a dissemination of a document, as represented in Table 1. The interface could define the getDocument() behavior request with or without a format parameter, and the aggregation could include the document as a PostScript bit stream or as a LaTeX bit stream.

Table 1 – behavior requests vs. aggregated renditions

	contains PostScript rendition	**contains LaTeX rendition**
getDocument(format)	transformation A	transformation B
getDocument()	transformation C	transformation D

When we set the format parameter to "PostScript" for the first line of the table, transformation A is trivial: it merely delivers the stored PostScript bit stream to the user. On the other hand, transformation B must take the LaTeX bit stream and convert it to PostScript before delivering it to the user. For this row of the table, the aggregation and interface dimensions are constraining the transformation dimension: if the transformation cannot produce the format specified by the parameter, the dissemination request produces an error.

When no format parameter is specified, as in the second line of the table, the interface could be specified such that transformation C will return a PostScript bit stream, while transformation D will return a LaTex bit stream. In essence, the storage provides the value of the absent format parameter. In this case, the aggregation and the transformation dimensions are affecting the interface dimension.

We can also see that the complexity of the transformation is affected by the interface definition if we view the first column of the table. Transformation A may need to produce multiple disseminations corresponding to different values of the format parameter, but transformation C may be trivial, merely delivering the stored PostScript bit stream to the user.

Another interaction between parameters and aggregation can be illustrated by recalling Figure 3. In this figure, a URI is stored in the digital object. But what if the URI was specified as a parameter in a behavior request? We will explore this in the next section.

4.4 Indirection

We want to take full advantage of the networked environment of digital information and allow access to digital content that is stored remotely. Figure 3 showed one type of this indirection – stored URIs are used to access remote content. Architectures may enable other forms of indirection, providing ways to aggregate distributed content into digital objects.

The Fedora architecture supports two forms of indirection for the aggregation dimension: it provides two ways to specify formal requests for remote content within its content container. Since requests for external bit streams are specifically supported by the architecture, problems such as invalid requests can be caught when these remote bit stream requests are stored. In Figure 3, problems accessing the data via the URIs will not be caught until a dissemination is requested.

Figure 5 shows an indirect PostScript bit stream that is a request for a specified PostScript dissemination from a specified Fedora digital object. Such indirection occurs entirely within the Fedora architecture, enabling multiple digital objects to use the same content while minimizing repetitious storage.

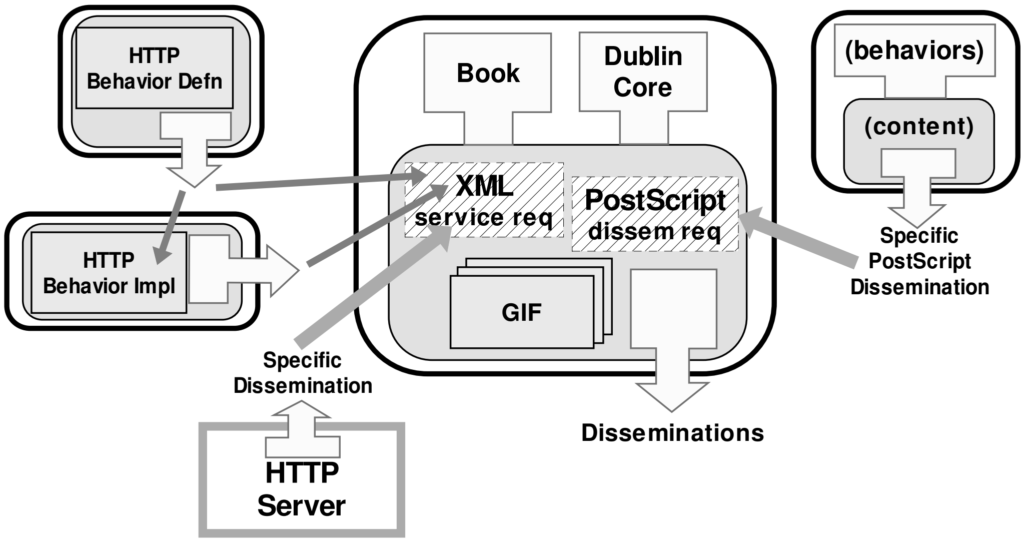

Figure 5 - remote content

Figure 5 also shows an indirect XML bit stream that is built from a particular dissemination from a particular HTTP server. The stored "service request" specifies the dissemination from the HTTP server via a behavior request from the HTTP Behavior Definition. The HTTP Behavior Implementation is the mechanism that extracts information from the stored service request, gets the requested dissemination from the specified HTTP server, and then, in this case, transforms the dissemination results into an XML bit stream, as shown in the figure. This form of indirection involves a service outside the Fedora architecture, and as with the URIs in Figure 3, the out-of-band communication limits policy enforcement capabilities and adds dependencies on external services and network reliability. However, since the service request itself is specified within the Fedora architecture, we have some degree of policy enforcement for out-of-band activity.

Note that accessing an external service via Fedora, such as HTTP in our example, requires service request mechanisms to support the indirection. These mechanisms must be stored and maintained just like any other mechanisms; the modeling choices described in section 4.2.1 apply here also.

Both forms of indirection illustrated in Figure 5 may reduce duplication of storage, but add vulnerability due to dependence on content outside the digital object. Clearly, remote content within the same Fedora repository exposes fewer vulnerabilities than remote content in an out-of-band service in an external repository located far away. Deciding what to store locally and what to access remotely will require weighing repetitious storage and versioning issues against expected vulnerabilities.

Indirect bit streams within the Fedora architecture raise interesting aggregation questions, as we now have a means to combine locally controlled content in unlimited ways. For example, what if a JPEG image of Cornell's McGraw tower is used in a multimedia presentation on the chimes housed there, in a video presentation of the Cornell campus and in a treatise on architectural photography? The Fedora architecture allows us to store the JPEG image in one digital object and refer to it from other digital objects. Figure 6 shows a digital object comprised solely of references to other Fedora digital objects.

The University of Virginia Library takes advantage of this indirection within the architecture in their modeling decisions. They create "building block" digital objects that contain a single entity such as an image (possibly multiple resolutions packaged together) or an EAD rendition of a manuscript, and then they create aggregate digital objects such as the one shown in Figure 6, which refer to the "building block" objects. This maximizes the utility, maintainability, and flexibility of their content for their purposes – they can easily aggregate bit streams in multiple ways without duplicating storage. Moreover, interfaces can be associated with any of the aggregate or "building block" digital objects at any time: the interface dimension is orthogonal to both the aggregation and the indirection dimensions.

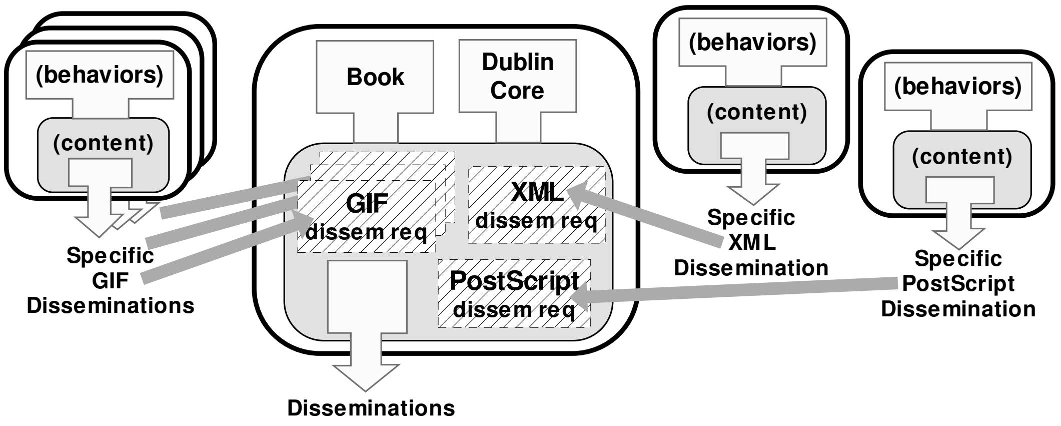

Figure 6 - no local content

We can use the "building block" principle for indirect bit streams outside the Fedora architecture. Sensible building blocks could be digital objects containing either a sole indirect bit stream request to an external service or multiple requests to a single external service. Other aggregate digital objects could then refer to these "building block" digital objects. Given these modeling decisions, it might make sense to store relevant service request mechanisms in the building block digital objects along with the indirect bit streams. Organizing digital content in this way may make maintaining references to external services easier.

4.4.1 Surrogate Digital Objects

Another form of indirection is when a digital object acts as a surrogate for content stored elsewhere. Figure 3 shows a digital object that acts as a wrapper for a particular document in an external repository – the Fedora digital object is a surrogate for the externally stored document. Fedora digital objects can also be surrogates for entire external services, not just documents contained there.

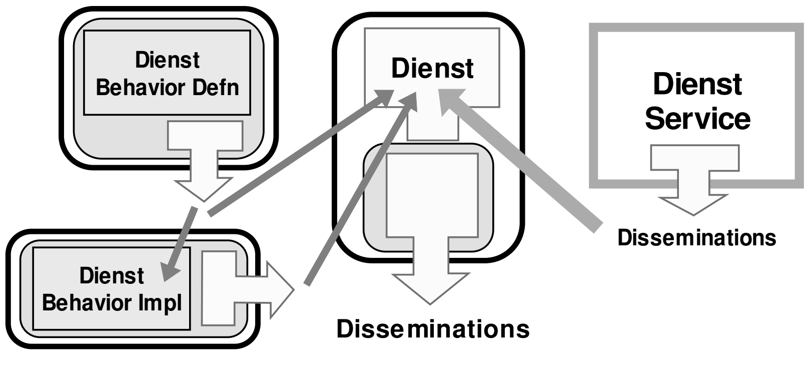

Figure 7 - surrogate digital object with no content

The digital object shown in Figure 7 is a surrogate for the Dienst service. This digital object has no content, just an attached interface: behavior implementations are not dependent on aggregation but instead on the interface definition (parameters, granularity of the methods) and/or the transformation definition.

The user of this Fedora digital object requests a specific dissemination at runtime – one of the many possible Dienst disseminations. This contrasts with the service request stored in Figure 5, which identifies a specific external dissemination when the indirect bit stream is created.

If the interface to an external service is stable, then a surrogate digital object like that in Figure 7 may be a good way to provide access via the Fedora architecture. Of course, this is another layer of indirection: besides needing the network and the external service to be reliable, we now require the external service's interface to be stable. On the other hand, we can represent an external service as a logical entity, facilitating interoperability as well as allowing some degree of policy enforcement on an out-of-band service. We can also control the amount of potential access to the external service via the interface and transformation dimensions: the interface might only define a subset of the possible requests, for example.

So given certain aggregation, interface, and transformation choices, digital objects can act as surrogates for external services or subsets of them. It's worth noting that the indirection supported by the Fedora architecture allows digital objects to be surrogates, or wrappers; no API is needed at the repository or architecture level to communicate with external services.

Mechanisms required by surrogate digital objects can be stored in the surrogate digital object – this might be a good choice if the mechanisms will only be used by dissemination requests on the surrogate digital object.

5 Conclusions

Digital content stewards face difficult organizational decisions as they try to anticipate future user needs and future maintenance issues for content within their control, as well as anticipate the level of maintenance for services beyond their control. In order to explore these decisions, we introduced four dimensions of digital content modeling: 1) aggregation 2) interfaces 3) transformations and 4) indirection. We discussed the complexities of choices for each dimension and the inter-relationships among the dimensions.

Information managers must consider the benefits and costs of various choices, often weighing vulnerabilities for one choice against another. For example, indirection can make digital content dependent on forces outside of the content provider's control, while repetitious storage may adversely affect maintainability of the same digital content. There may be security or preservation issues as well.

We provided no answers to the modeling questions, because there is no single solution.

Acknowledgments

This work was supported by National Science Foundation Grant No. IIS-9817416. This paper does not necessarily represent the views of the NSF. The authors are grateful to Thorny Staples and Ross Wayland of the University of Virginia Library, and especially Sandy Payette of Cornell for their contributions.

References

1. Leiner, B.M., *The NCSTRL Approach to Open Architecture for the Confederated Digital Library*, in *D-Lib Magazine*. 1998.

2. Payette, S. and C. Lagoze. *Flexible and Extensible Digital Object and Repository Architecture (FEDORA).* in *Second European Conference on Research and Advanced Technology for Digital Libraries*. 1998. Heraklion, Crete.

3. Payette, S., *et al.*, *Interoperability for Digital Objects and Repositories: The Cornell/CNRI Experiments.* D-Lib Magazine, 1999. **May**.

4. Staples, T. and R. Wayland, *Virginia Dons FEDORA: A Prototype for a Digital Object Repository.* D-Lib Magazine, 2000. **July**.

5. Maly, K., M.L. Nelson, and M. Zubair, *Smart Objects, Dumb Archives: A User-Centric, Layered Digital Library Framework.* D-Lib Magazine, 1999. **March**.

6. Lagoze, C., *et al.*, *Dienst Implementation Reference Manual*, . 1995, Cornell University Computer Science.

7. Wilensky, R. and T.A. Phelps, *Multivalent Documents: A New Model for Digital Documents*, . 1998, University of California, Berkeley.

8. (DLF), D.L.F., *The Making of America II Testbed Project White Paper*, . 1998, Digital Library Federation.

9. Kahn, R. and R. Wilensky, *A Framework for Distributed Digital Object Services*, . 1995, Coporation for National Research Initiatives: Reston.

10. Lagoze, C., C.A. Lynch, and R.D. Jr., *The Warwick Framework: A Container Architecture for Aggregating Sets of Metadata*, . 1996, Cornell University Computer Science.

11. Daniel Jr., R. and C. Lagoze. *Distributed Active Relationships in the Warwick Framework*. in *IEEE Metadata Conference*. 1997. Bethesda.

12. Daniel Jr., R., C. Lagoze, and S.D. Payette. *A Metadata Architecture for Digital Libraries*. in *Advances in Digital Libraries*. 1998. Santa Barbara.

13. Payette, S. and C. Lagoze. *Policy-Enforcing, Policy-Carrying Digital Objects*. in *Fourth European Conference on Research and Advanced Technology for Digital Libraries*. 2000. Lisbon.