

# Smart Objects, Dumb Archives: Insuring the Long-Term Integrity of Digital Information

Michael L. Nelson  
NASA Langley Research Center  
m.l.nelson@larc.nasa.gov

Paper presented at the workshop on  
Web-Based Language Documentation and Description  
12-15 December 2000, Philadelphia, USA.

**Abstract.** *Within the context of DLs, we are making information objects first-class citizens. We decouple information objects from the systems used for their storage and retrieval, allowing the technology for both DLs and information content to progress independently. We feel that dismantling the current stovepipe of DL-archive-content is the first step in both building richer DL experiences for users and insuring the long-term availability and integrity of digital information. To demonstrate this partitioning between DLs, archives and information content, we introduce Buckets, aggregative, intelligent, object-oriented constructs for publishing in digital libraries. Buckets exist within the Smart Objects, Dumb Archives (SODA) DL model, which dictates that functionalities and responsibilities traditionally associated with archives are pushed down into the buckets, making the buckets smarter and the archives dumber.*

## 1. Introduction

Discussion of digital libraries (DLs) is often dominated by the merits of various archives, repositories, search engines, search interfaces and database systems. While these technologies are necessary for information management, information content and information retrieval systems should progress on independent paths and each should make limited assumptions about the status or capabilities of the other. Information content is more important than the systems used for its storage and retrieval. Digital information should have the same long-term survivability prospects as traditional hardcopy information and should not be impacted by evolving search engine technologies or vendor vagaries in database management systems.

Digital information can achieve independence from archives and DL systems through the use of buckets. Buckets are an aggregative, intelligent construct for publishing in DLs and allow the decoupling of information content from information storage and retrieval. Buckets exist within the Smart Objects, Dumb Archives (SODA) model for DLs in that many of the functionalities and responsibilities traditionally associated with archives are pushed down (making the archives dumber) into the buckets (making them smarter). Some of the responsibilities imbued to buckets are the enforcement of their terms and conditions, and maintenance and display of their contents. These additional responsibilities come at the cost of storage overhead and increased complexity for the archived objects. However, tools have been developed to manage the complexity, and storage is cheap and getting cheaper; the potential benefits buckets offer DL applications appear to outweigh their costs.

## 2. SODA: Smart Objects, Dumb Archives

The observation that motivates the SODA model for DLs is that digital objects are more important than the archives that hold them. Many DL systems and protocols are reaching a point where DL interoperability and object mobility are hindered by the complexity of the archives that hold the objects. The goal of the current work is to increase the responsibilities of objects, and decrease the responsibilities of archives. If digital objects themselves handle presentation, terms and conditions and their own data management, it will be easier to achieve interoperability between heterogeneous DLs as well as increase object mobility and longevity. As a consequence, it should be easier to build specialized DLs for targeted user communities. SODA defines a DL model composed of three strata (Fig. 1):

- *digital library services* - the user functionality and interface: searching, browsing, usage analysis, citation analysis, selective dissemination of information (SDI), etc.
- *archive* - managed sets of digital objects. DLs can poll archives to learn of newly published digital objects, for example.
- *digital object* - the stored and trafficked digital content. These can be simple files (e.g., PDF or PS files), or more sophisticated objects such as buckets.

In most DLs, the digital library services (DLS) and the archive functionality are tightly coupled. A digital object is placed in an archive, and this placement uniquely determines in which DL it appears. We believe that if there is not a 1-1 mapping between archives and DLs, but rather a N-M mapping, the capacity for interoperability is greatly advanced. A DL can draw from many archives, and likewise, an archive can contribute its contents to many DLs.

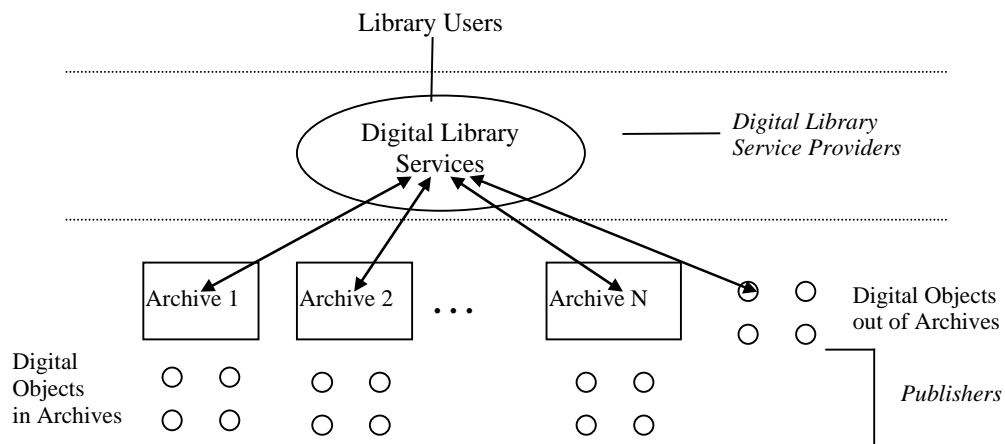


FIG. 1. The three strata of DLs

However, since we can no longer be sure which DL will be used for the discovery and presentation of an object, it is necessary to evolve the notion of the object and to imbue it with greater functionality and responsibility. DL objects should be self-sufficient, intelligent, and aggregative and capable of enforcing their own terms and conditions, negotiating access, and displaying their contents.

## 2.1 Archive Design Space

In SODA, archives exist primarily to assist DLs in locating objects -- they are generally not for direct user access. Archived objects, not archives, should be responsible for the enforcement of terms and conditions, negotiation and presentation of content, etc. In contrast, the Dienst protocol (Lagoze, Shaw, Davis, & Krafft, 1995) contains a built-in document object model, and this limits its applicability in different domains and makes it more difficult to transition to evolving document object models. Although it is expected that some archive implementations will retain portions of the above functionality — indeed, SOSA (Smart Objects, Smart Archives) may become the most desirable DL model — a Dumb archive model is used here to illustrate the full application of smart objects (buckets). When archives become Smart again, it will with other functionalities, not duplication of bucket functionality. Using this terminology, Table 1 illustrates how the archive design space partitions.

TABLE 1. The archive design space.

	Smart Archives	Dumb Archives
Smart Objects	SOSA: Smart Objects, Smart Archives DL Example: none known	SODA: Smart Objects, Dumb Archives DL Example: NCSTR+L
Dumb Objects	DOSA: Dumb Objects, Smart Archives DL Example: NCSTR	DODA: Dumb Objects, Dumb Archives DL Example: any anonymous FTP server with .ps.Z files

## 2.2 Publishing in the SODA Model

Separating the functionality of the archive from that of the DLS allows for greater interoperability and federation of DLs. The archive's purpose is to provide DLs the location of buckets (the DLs can poll the buckets themselves for their metadata), and the DLs build their own indexes. And if a bucket does not want to share its metadata (or contents) with certain DLs or users, its terms and conditions will prevent this from occurring. For example, it is expected that the NASA digital publishing model will begin with technical publications, after passing through their respective internal approval processes, to be placed in a NASA archive. The NASA DL (which is the set of the NASA buckets, the NASA archive(s), the NASA DLS, and the user communities at each level) would poll this archive to learn the location of buckets published within the last week. The NASA DL could then contact those buckets, requesting their metadata. Other DLs could index NASA holdings in a similar way: polling the NASA archive and contacting the appropriate buckets. The buckets would still be stored at NASA, but they could be indexed by any number of DLs, each with the possibility for novel and unique methods for searching or browsing. Or perhaps the DL collects all the metadata, then performs additional filtering to determine applicability for inclusion into their DL. In addition to an archive's holdings being represented in many DLs, a DL could contain the holdings of many archives. If all digitally available publications are viewed as a universal corpus, then this corpus could be represented in N archives and M DLs, with each DL customized in function and holdings to the needs of its user base. Figure 2 illustrates the SODA publishing model.

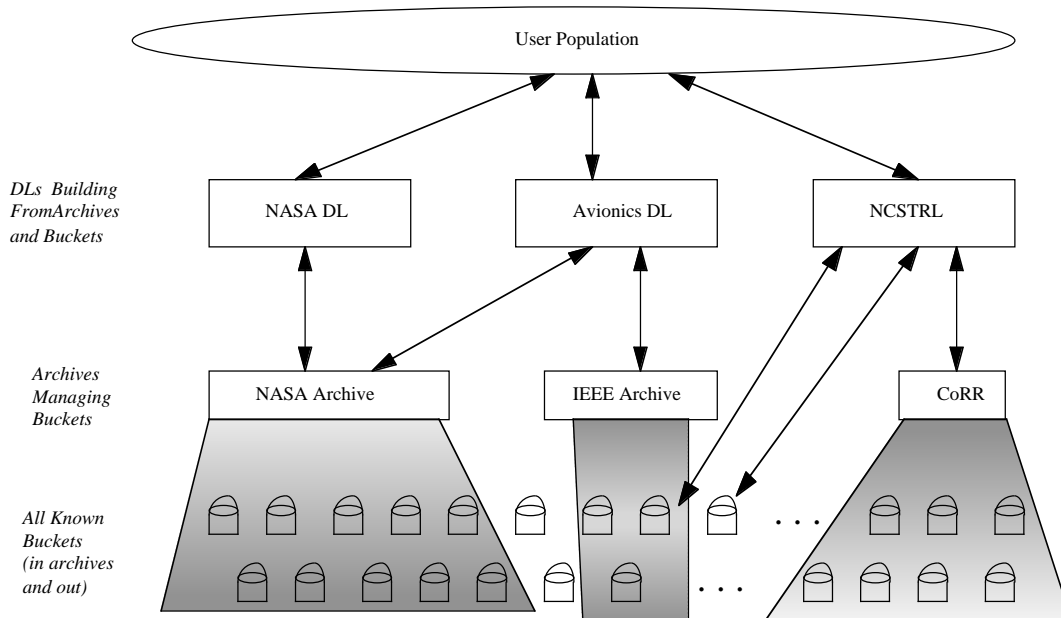


FIG. 2. The SODA publishing model.

### 3. Buckets: Smart Objects

Buckets are our implementation of smart objects. The motivation for buckets came from previous experience in the design, implementation and maintenance of NASA scientific and technical information DLs, including the Langley Technical Report Server (LTRS) (Nelson, Gottlich & Bianco, 1994), the NASA Technical Report Server (NTRS) (Nelson, Gottlich, Bianco, et al., 1995), and the NACA Technical Report Server (NACATRS) (Nelson, 1999). In early user evaluation studies of these DLs, one reoccurring theme was detected. While access to the technical report (or re/pre-print) was desirable, users particularly wanted the raw data collected during the experiments, software used to reduce the data, and ancillary information used in the production of the published report. In response, we wanted to permit DLs to accommodate requests for substantially more information types than just reports. However, rather than setup separate DLs for each information type or stretch the definition of a traditional report to include various multi-media formats, we defined an arbitrary digital object that can capture and preserve the potentially intricate relationship between multiple information types.

Additionally, our experiences with updating the NASA DLs and making the content accessible through other DLs and web-crawlers led to the decision to make the information objects intelligent. We wanted the objects to receive maximum exposure, and did not want them trapped inside our DLs, with the only method for their discovery coming from our DL interface. However, the DL should have more than just an exportable description of how to access the objects in the DL. The information object should be independent of the DL, with the capability to exist outside of the DL and move in and out of different DLs in the future. However, to not assume which DL was used to discover and access the buckets means that the buckets must be self-sufficient and perform whatever tasks are required of them, potentially without the benefit of being arrived at through a specific DL.

A bucket is a storage unit that contains data and metadata, as well as the methods for accessing both. It is difficult to overstate the importance of aggregation as a design goal. In our experience with

other NASA DLs, data was often partitioned by its semantic or syntactic type: metadata in one location, PostScript files in another location, PDF files in still another location, etc. Over time, different forms of metadata were introduced for different purposes, the number of available file formats increased, the services defined on the data increased, new information types (software, multimedia) were introduced, the logging of actions performed on the objects became more difficult. The result of a report being on the DL eventually represented so much DL jetsam - bits and pieces physically and logically strewn across the system. We responded to this situation with extreme aggregation. As suggested by the SODA model, buckets have unique requirements due to their emphasis on minimizing dependence on specific DL implementations. The design goals are: aggregation, intelligence, self-sufficiency, mobility, heterogeneity and archive independence.

### 3.1 Bucket Model

The first focus of the aggregation was for the various data types. Based on experience gained while designing, implementing and maintaining LTRS and NTRS, we initially decided on a two-level structure within buckets:

- buckets contain 0 or more *packages*
- packages contain 0 or more *elements*

Actual data objects are stored as elements, and elements are grouped together in packages within a bucket. In LTRS and NTRS, a two-level architecture was sufficient for most applications, so this two-level architecture was retained as a simplifying assumption during bucket implementation. Future work will implement the semantics for describing arbitrarily complex, multi-level data objects. An element can be a pointer to another object: another bucket, or any other arbitrary network object. By having an element point to other buckets, buckets can logically contain other buckets. Although buckets provide the mechanism for both internal and external storage, buckets have less control over elements that lie physically outside the bucket. However, it is left as a policy decision to the user as to the appropriateness of including pointers in an archival unit such as a bucket. Buckets have no predefined size limitation, either in terms of storage capacity, or in terms of number of packages or elements. Buckets are accessed through 1 or more URLs. For an example of how a single bucket can be accessed through multiple URLs, consider two hosts that share a file system:

```
http://host1.foo.edu/bar/bucket-27/  
http://host2.foo.edu/bar/bucket-27/
```

Both of these URLs point to the same bucket, even though they are accessed through different hosts. Also, consider a host that runs multiple http servers:

```
http://host1.foo.edu/bar/bucket-27/  
http://host1.foo.edu:8080/bucket-27/
```

If the http server running on port 8080 defines its document root to be the directory `bar` then the two URLs point to the same bucket. Elements and packages have no predefined semantics associated with them. Authors can model whatever application domain they desire using the basic structures of packages and elements. One possible model for bucket, package, and element definition is based on NASA DL experiences. In NASA DL buckets, packages represent semantic types (manuscript, software, test data, etc.) and elements represent syntactic representations of the packages (a .ps version, .pdf version, .dvi version, etc.). Other bucket models using elements and packages are possible. For example, we have used buckets for entire research projects (Fig. 3) and university classes (Fig. 4) as well as for STI

publications. Though the display of the two buckets is different, the two-level architecture of packages and elements is evident.

Buckets have the capability of implementing different policies as well: one site might allow authors to modify the buckets after publishing, and another site might have buckets be frozen upon publication. Still another site might define a portion of the bucket to receive annotations, review, or contributions from the users, while keeping another portion of the bucket frozen, or only changeable by authors or administrators. Buckets provide mechanism, not policy.

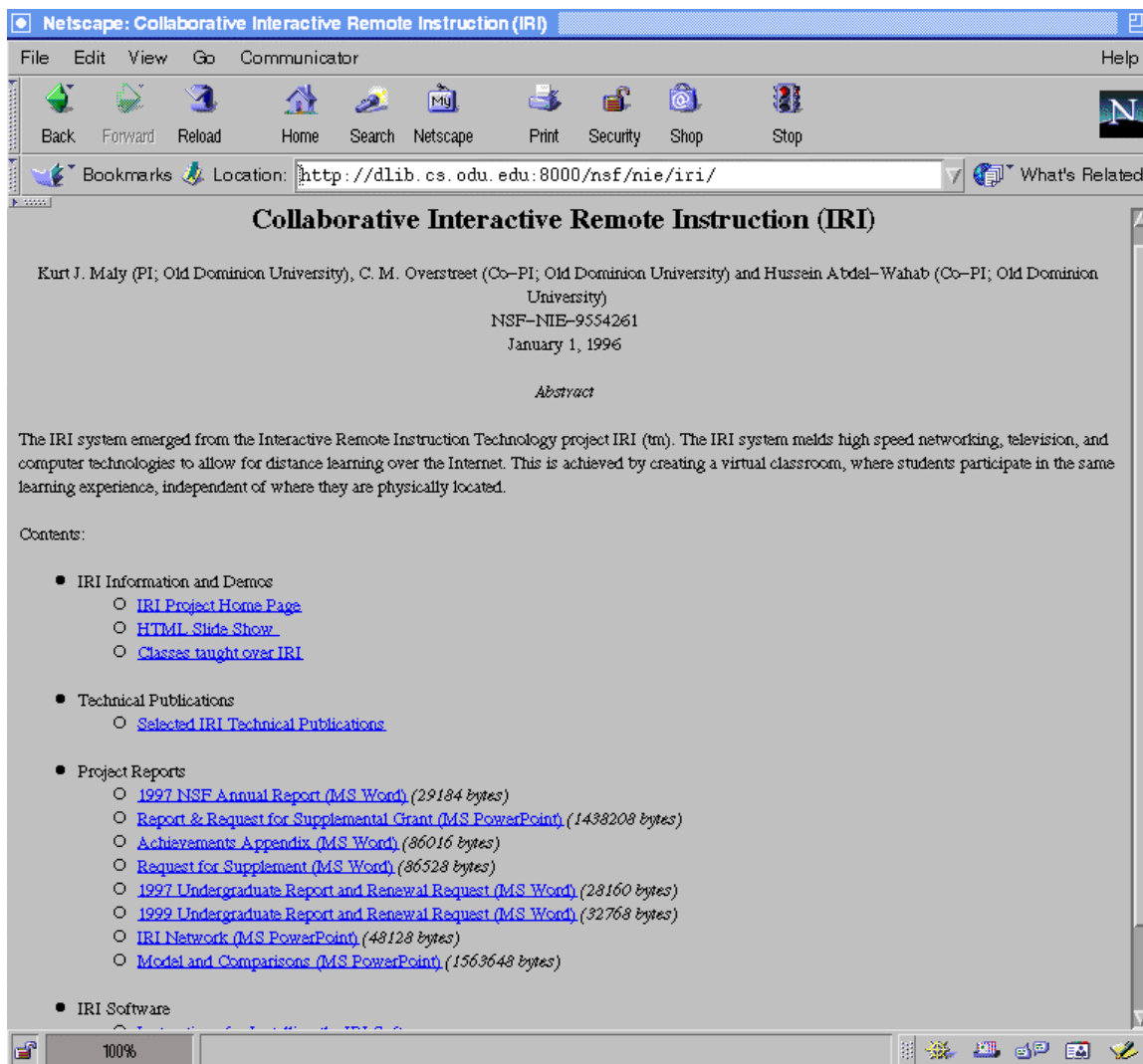


FIG. 3. Sample project bucket.

Another focus of aggregation was including the metadata with data. In previous experiences, we found that metadata tended to drift over time, becoming decoupled from the data it described or locked in specific DL systems and hard to extract or share with other systems. For some information types such as reports, regenerating lost metadata is possible either automatically or by inspection. For other information types such as experimental data, the metadata cannot be recovered from the data. Once the metadata is lost, the data itself becomes useless. Also, we did not want to take a proscriptive stance on metadata. Although the bucket itself has to ultimately choose one metadata format as canonical for

structural purposes, buckets needed to be able to accommodate multiple metadata formats. Buckets do this by storing metadata in a reserved package and using methods for reading and uploading new metadata formats as elements in the metadata package. As a result, buckets can accommodate any number of past, present or future metadata formats.

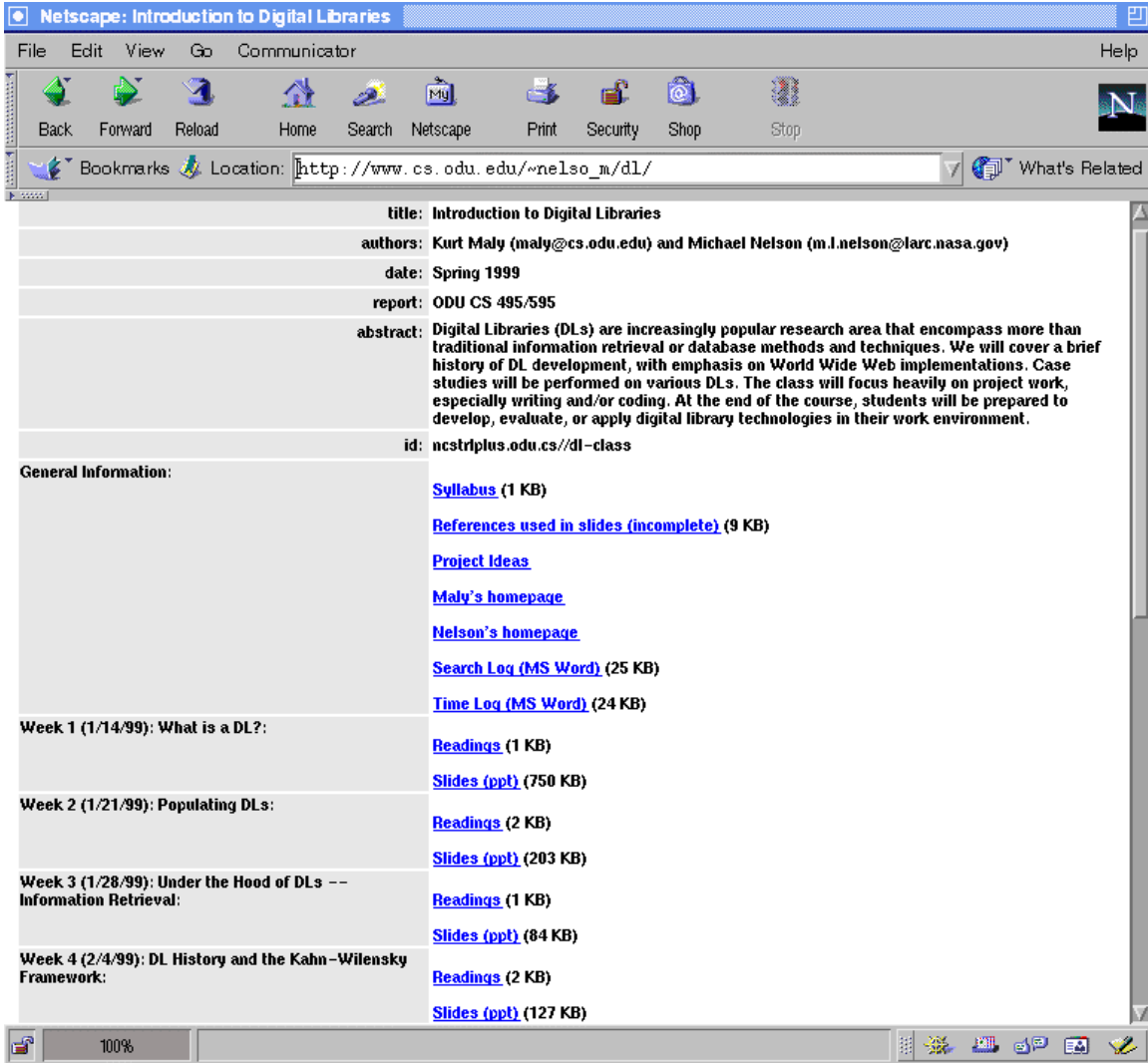


FIG. 4. Sample course bucket.

The final aggregation focus was on the services defined on buckets and the results of those services. The default state is for everything the bucket needs to display, disseminate, and manage its contents is contained within the buckets. This includes the source code for all the methods defined on the bucket, the user ids and passwords, the access control lists, the logs of actions taken on the bucket, Multipurpose Internet Mail Extensions (MIME) (Borenstein & Freed, 1993) definitions and all other supporting technologies necessary for the bucket to function. The self-sufficiency and mobility design goals dictate that a bucket cannot make many assumptions about the environment that it will reside in and should require no server modifications to function.

## 3.2 Bucket Implementation

The buckets described here are version 1.6.2. Buckets are currently written in Perl 5 and use http as the transport protocol. However, buckets can be written in any language as long as the bucket API is preserved. Buckets were originally deployed in the NCSTRL+ project (Nelson, Maly, Shen, & Zubair, 1998), which demonstrated a modified version of the Dienst protocol. Owing to their Dienst-related heritage, bucket metadata is stored in RFC-1807 format (Lasher & Cohen, 1995), with package and element information stored in NCSTRL+ defined optional and repeatable fields. Although buckets use RFC-1807 as their native format, they can contain and serve any metadata type. Dienst has all of a document's files gathered into a single Unix directory. A bucket follows the same model and has all relevant files collected together using directories from file system semantics. The bucket is accessible through a CGI script that enforces terms and conditions, and negotiates presentation to the WWW client.

Aside from Perl 5, http, and CGI, buckets make no assumptions about the environment in which they will run. Mobility is one of the design goals of buckets, and a corollary of that is that buckets should not require changes in a reasonable http server setup; where reasonable is defined to be allowance of the `index.cgi` convention. Once these assumptions have been met, buckets by default take care of everything themselves with no server intervention, including MIME typing, terms and conditions, and support libraries. Although bucket development was conducted under Solaris (Unix), buckets have been tested on a variety of system configurations (Table 2).

TABLE 2. System configurations used for bucket testing.

Architecture	Operating System	Perl	http server
Sparc	Solaris 2.7	5.005_03	Apache 1.3.9
Sparc	Solaris 2.7	5.005_03	NCSA httpd 1.5.2
Sparc	Red Hat 6.0 (Linux 2.2.5-15)	5.005_03	Apache 1.3.6
Intel x86	Windows NT 4.0 (1381 / SP 5)	Active Perl 5.005_03	Apache 1.3.12
Intel x86	Mandrake Linux 6.2	5.005_03	Apache 1.3.6
MIPS R10000	IRIX 6.5	5.004_04	Apache 1.3.4
RS/6000	AIX 4.2	5.002	Apache 1.3.12
PowerPC 604	Linux 2.0.33 (MkLinux)	5.004_01	Apache 1.2.6

## 3.3 Bucket Operation

Our reference implementation implements the bucket API using http encoding of messages. Buckets appear as ordinary URLs and casual users should not realize they are not interacting with a typical web site. Although possible, users are not expected to directly invoke methods — the applicable methods for accessing its contents are automatically built into the bucket's HTML output. The other creation and management-oriented methods are expected to be accessed by various bucket tools. If no method is invoked via URL arguments, the `display` method is assumed by default. This generates a human-readable display of the bucket's contents. For example, a bucket version of a NACA Technical Note:

```
http://www.cs.odu.edu/~nelso_m/naca-tn-2509/
```

which is the same as:

```
http://www.cs.odu.edu/~nelso_m/naca-tn-2509/?method=display
```

Both URLs produce the output in figure 5. These URLs could be reached through either a searching or browsing function within a DL, or typed in directly — buckets make no assumptions about their discovery. However, a DL can pass in preferences to alter the appearance of the bucket. For example, a view (figure 6) of the bucket suitable for library staff can be specified with:

```
http://www.cs.odu.edu/~nelso_m/naca-tn-2509/?method=display&view=staff
```

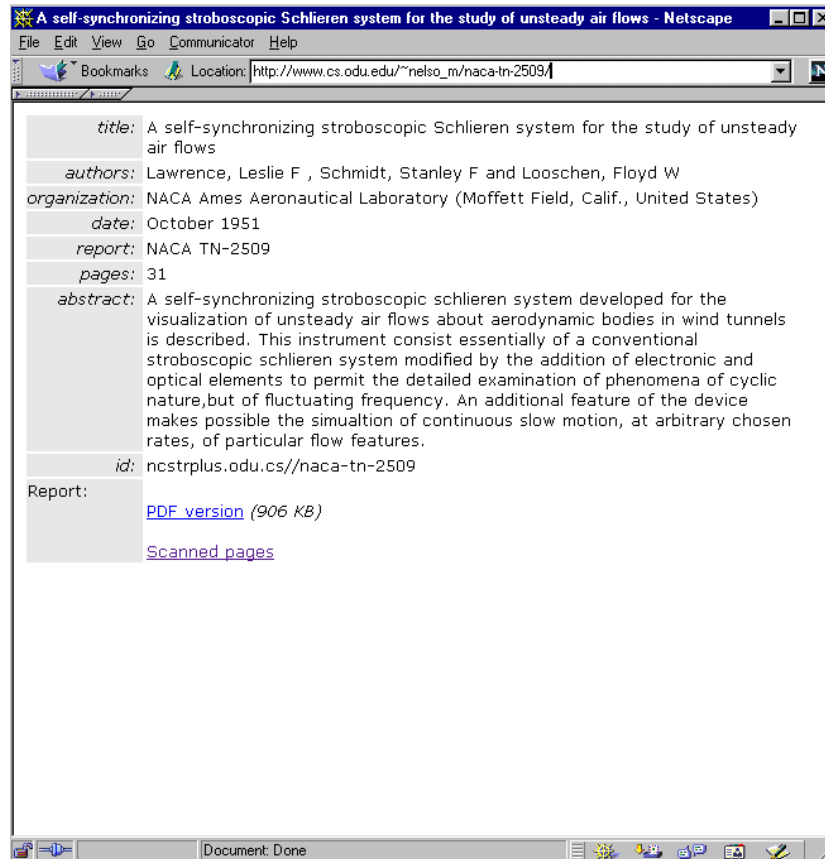


FIG. 5. The default display method reveals the bucket contents in a human readable format.

From the human readable interface the `display` method generates, if users wish to retrieve the PDF file, they click on the PDF link that was automatically generated:

```
http://www.cs.odu.edu/~nelso_m/naca-tn-2509/?method=display&pkg_name=report.pkg&element_name=naca-tn-2509.pdf
```

which would cause the WWW browser to launch the PDF reader. Similarly, if users wished to display the scanned pages, selecting the automatically created link would send the following arguments to the `display` method:

```
http://www.cs.odu.edu/~nelso_m/naca-tn-2509/?method=display&pkg_name=report.pkg&element_name=report.scan
```

which would produce the output in figure 7. To the casual observer, the bucket API is transparent. However, if individual users or harvesting robots know a particular URL is actually a bucket, they can exploit this knowledge. For example, to extract the metadata in default (RFC-1807) format, the URL is:

`http://www.cs.odu.edu/~nelso_m/naca-tn-2509/?method=metadata`

which would return the metadata in a structured format, suitable for inclusion in an index being automatically built by a DL. If a user or agent wishes to determine that nature of a bucket, a number of methods are available. For example, to determine the bucket's version:

`http://www.cs.odu.edu/~nelso_m/naca-tn-2509/?method=version`

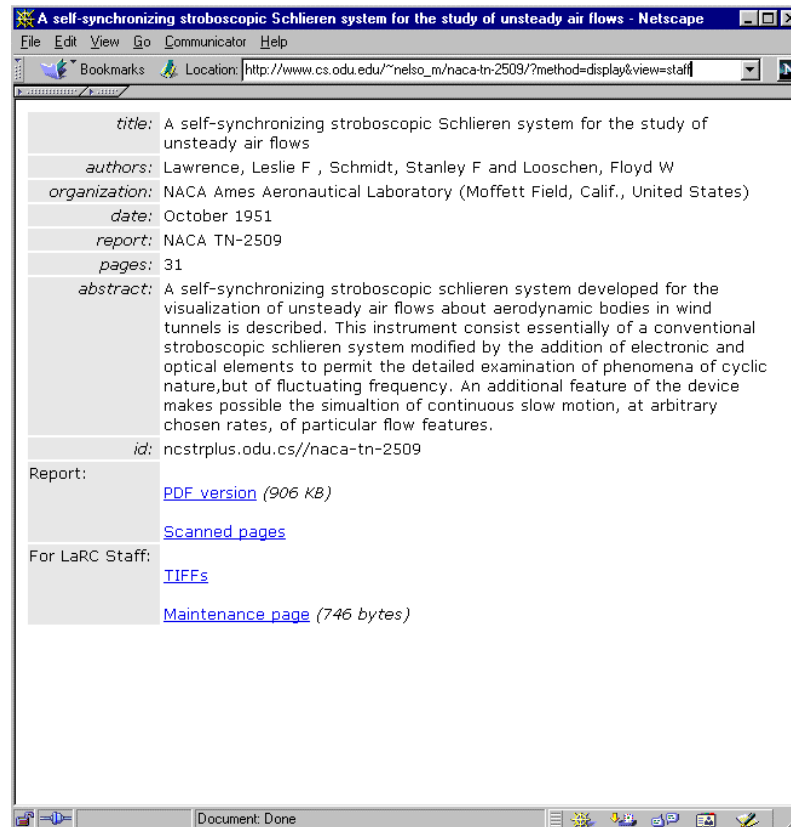


FIG. 6. Extra options are available for library staff.

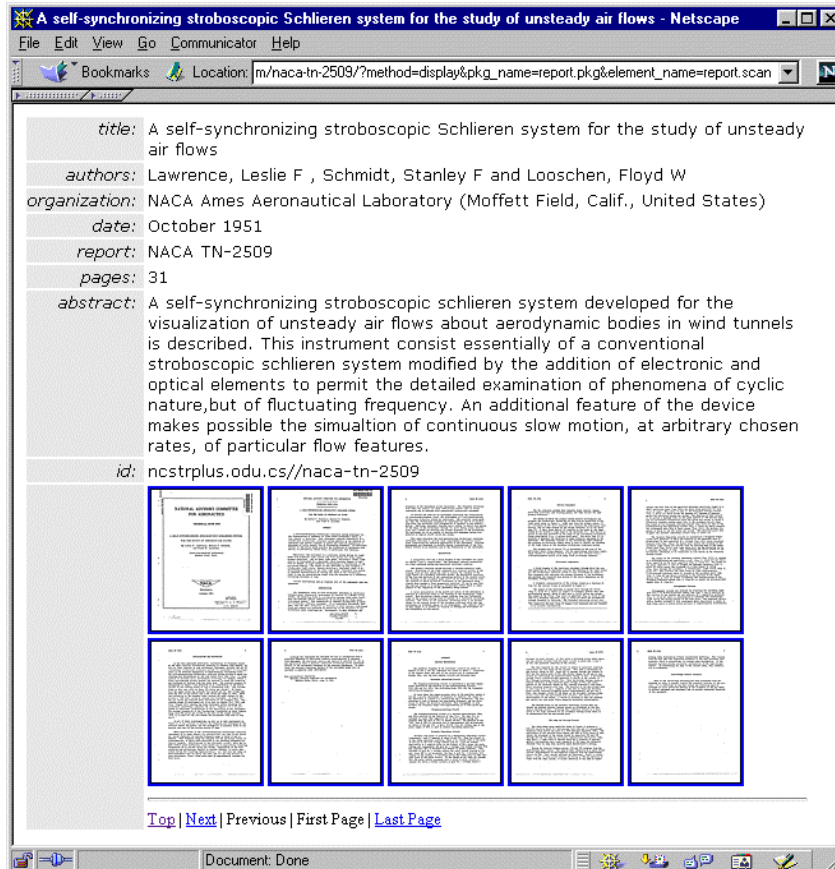


FIG. 7. The first 10 scanned thumbnails within the bucket are displayed, along with pagination control.

To see what methods are defined on a bucket:

`http://www.cs.odu.edu/~nelso_m/naca-tn-2509/?method=list_methods`

However, if a harvester is not bucket-aware, it can still crawl or spider the bucket URLs as normal URLs, extracting information from the HTML human-readable interface generated by the `display` method (assuming the `display` method is not restricted by T&C). Buckets offer many expressive options to the users or services that are bucket-aware, but are transparent to those who are not bucket-aware. The full bucket API is defined and discussed in (Nelson, 2000).

### 3.4 Bucket Communication Space

The Bucket Communication Space (BCS) is partially motivated by Linda, the parallel communication library (Carriero & Gelernter, 1989). In Linda, processes effectively pass messages by creating tuples that exist in tuple space. These data objects are created with the `eval` primitive, and filled with data by processes using the `out` primitive. Processes use `in` and `in` for reading and reading-removing operations, respectively. These primitives allow processes to communicate through tuple space, without having to know the details (e.g. hostnames, port numbers) of where the processes are. The messages written to tuple space can have regular expressions and control logic to specify who should read them. When a `in` tuple sees an `out` tuple and the conditions of the former match that of the latter, the message is communicated to the receiving process and the tuple is removed from tuple space. Though it imposes a performance overhead, the Linda environment provides a useful layer of abstraction for inter-process communication.

We wished to provide something similar for buckets: buckets communicating with other buckets without having to know the details of bucket location. This is especially important if the buckets are mobile, and a bucket location is not guaranteed to be static. The BCS also provides a method for centralizing functionality that cannot be replicated in individual buckets. This could be either because of efficiency concerns (the resulting bucket would be too bloated) or implementation limitations (a service is not available on the architecture that is serving the bucket). Buckets need only know how to communicate to a BCS server (maintained through a bucket preference), which can handle their requests for them.

The BCS model opens up many possible service areas. A subtle element of the BCS is that buckets, not people, are responsible for the provision and coordination of these services. We provide proof-of-concept implementations for four significant services: file format conversion, metadata conversion, bucket messaging, and bucket matching. The goal of providing these services is to allow buckets to individually migrate the formats of their metadata and data according to their preferences and BCS capabilities. It is possible for buckets, over time, to become the union of all desirable formats and encodings for their information content. The BCS is further discussed in (Nelson, 2000).

## 4. OAI: Dumb Archives

Just as buckets are a possible implementation of smart objects, there are also different possible implementations of dumb archives. Originally, a separate protocol for archives was defined and implemented in (Nelson, 2000), but it is no longer being developed in favor of the evolving Open Archive Initiative (OAI) and its metadata harvesting protocol. Similar to the DA, OAI archives for NASA DLs are implemented as modified buckets. These OAI buckets have all the functionality of regular buckets, plus the source code to implement the six verbs of OAI metadata harvesting protocol.

The OAI metadata harvesting protocol defines six verbs (Table 3) that allow the creators of DLs (known as service providers in OAI parlance) to query archives (data providers) to determine the nature of the archive and produce full or partial dumps of an archive's metadata (Open Archives Initiative, 2000). Most of the six verbs take various arguments such as timestamps or archive-defined sets to allow for partial harvesting. Although any metadata format can be provided by a data provider, in the interest of easing the task of creating service providers, the Dublin Core (Weibel, Kunze, Lagoze & Wolf, 1998) is defined as the default metadata format required for OAI compliance.

TABLE 3. OAI verbs.

Verb	Function
Identify	machine readable description of archive
ListMetadataFormats	metadata formats supported by archive
ListSets	sets defined by archive
ListIdentifiers	OAI unique ids contained in archive
ListRecords	listing of all records
GetRecord	listing of a single record

It should be noted that the OAI protocol is not defined as a stand-alone system; only a layer for retrieving items is available. An OAI interface is always a front-end to some other archival system — for example: a relational database management system, directory service, filesystem, or Dienst server.

The OAI grew out of the meeting surrounding the presentation of the Universal Preprint Service (UPS) demonstration DL. The UPS is a large DL testbed introduced in October 1999 and is based on NCSTRL+ software. The UPS prototype was a feasibility study for the creation of cross-archive end-user services. With the premise that users would prefer to have access to a federation of digital libraries, the main aim of the project was the identification of the key issues in actually creating an experimental end-user service for data originating from important existing, production archives. This included a total of

almost 200,000 buckets harvested from six existing production DLs. A full discussion of the results from the UPS project and the role of buckets in the DL can be found in (Van de Sompel, Krichel, Nelson, et al., 2000).

## 5. Related and Future Work

There are projects with similar aggregation goals as buckets from the DL community, such as Multivalent Documents (Phelps & Wilensky, 2000) and the Kahn-Wilensky Framework (Kahn & Wilensky, 1995) and its derivatives (Warwick Framework (Lagoze, Lynch & Daniel, 1996) and FEDORA (Payette & Lagoze, 2000)). But these projects do not feature mobility, self-sufficiency or the SODA-inspired motivation of freeing the information object from archival control. Most DL intelligent agent projects focus on aids to the DL user or creator; the intelligence is machine-to-human based. Buckets are unique because the information objects themselves are intelligent, providing machine-to-machine (or, bucket-to-bucket) intelligence.

An additional advantage is buckets require no server modification or additional software installation. Buckets are designed to work in a minimalist computer environment and not depend on extensive infrastructure. A result of this self-sufficiency is that buckets are mobile and portable — able to move about various servers and archives. Similarly, buckets are not dependent on the existence (or absence) of any particular search engine, database or archival software. The aggregative nature of buckets has allowed for easy object-level addition of value-added services such as the SFX reference linking service (Van de Sompel & Hochstenbach, 1999) without the trouble of modifying DL system source code.

Buckets are having a significant impact in how NASA, Los Alamos National Laboratory and the Air Force Research Laboratory are designing their next generation DLs. A technical report interchange project between these institutions is currently underway that allows OAI cross-harvesting and indexing of buckets without modification of these institutions' respective DLs. Future bucket development plans include the continued development of high quality tools for bucket creation, management and maintenance in a variety of application scenarios. Other short-term goals include alternate implementations of buckets, discipline-specific buckets, and more sophisticated T&C support. Long-range plans include significant utilization of bucket mobility and bucket intelligence, including buckets actively involved with their long-term survivability and buckets interacting with DL services to report their observed usage patterns.

## 6. Conclusions

Buckets were born of our experience in creating, populating and maintaining several production DLs for NASA. The users of NASA DLs repeatedly wanted access to data types beyond that of the technical publication, and the traditional publication systems and the digital systems that automated them were unable to adequately address their needs. Instead of creating a raft of competing, separate-but-equal DLs to contain the various information types, a container object was created capable of capturing and preserving the relationship between any number of arbitrary data types. Buckets have been successfully deployed in several prototype DLs. In addition, they have been modified to provide functionality and services outside of their original scope.

There are a number of projects that have similar aggregation goals as buckets. Some are from the DL community, and others are from e-commerce and computational science. Most do not have the SODA-inspired motivation of freeing the information object from the control of a single server. The mobility and independence of buckets are not seen in other DL projects. Most DL projects that focus on intelligence or agency are focused on aids to the DL user or creator; the intelligence is machine-to-human

based. Buckets are unique because the information objects themselves are intelligent, providing machine-to-machine (or, bucket-to-bucket) intelligence.

## References

- Borenstein, N. & Freed, F. (1993). MIME (multipurpose Internet mail extensions) part one: mechanisms for specifying and describing the format of Internet message bodies. Internet RFC 1521. Available at <ftp://ftp.isi.edu/in-notes/rfc1521.txt>.
- Carriero, N. & Gelernter, D. (1989). Linda in context. *Communications of the ACM*, 32(4), 444-458.
- Kahn, R. & Wilensky, R. (1995) A framework for distributed digital object services. cnri.dlib/tn95-01. Available at <http://www.cnri.reston.va.us/home/cstr/arch/k-w.html>.
- Lagoze, C., Shaw, E., Davis, J. R., & Krafft, D. B. (1995). Dienst: implementation reference manual. Cornell University Technical Report TR95-1514. Available at <http://ncstrl.cs.cornell.edu/Dienst/UI/1.0/Display/ncstrl.cornell/TR95-1514>.
- Lagoze, C., Lynch C. A., & Daniel, R. (1996). The Warwick framework: a container architecture for aggregating sets of metadata. Cornell University Computer Science Technical Report TR-96-1593. Available at <http://ncstrl.cs.cornell.edu/Dienst/UI/1.0/Display/ncstrl.cornell/TR96-1593>.
- Lasher, R. & Cohen, D. (1995). A format for bibliographic records. Internet RFC-1807. Available at <ftp://ftp.isi.edu/in-notes/rfc1807.txt>.
- Nelson, M. L., Gottlich, G. L., & Bianco, D. J. (1994). World Wide Web implementation of the Langley technical report server. NASA TM-109162. Available at <http://techreports.larc.nasa.gov/ltrs/PDF/tm109162.pdf>.
- Nelson, M. L., Gottlich, G. L., Bianco, D. J., Paulson, S. S., Binkley, R. L., Kellogg, Y. D., Beaumont, C. J., Schmunk, R. B., Kurtz, M. J., Accomazzi, A., & Syed, O. (1995). The NASA technical report server. *Internet Research: Electronic Network Applications and Policy*, 5(2), 25-36. Available at <http://techreports.larc.nasa.gov/ltrs/papers/NASA-95-ir-p25/NASA-95-ir-p25.html>.
- Nelson, M. L., Maly, K., Shen, S. N. T., & Zubair, M. (1998). NCSTRl+: adding multi-discipline and multi-genre support to the Dienst protocol using clusters and buckets. *Proceedings of the IEEE forum on research and technology advances in digital libraries* (pp. 128-136), Santa Barbara, CA. Available at <http://techreports.larc.nasa.gov/ltrs/PDF/1998/mtg/NASA-98-ieeedl-mln.pdf>.
- Nelson, M. L. (1999). A digital library for the National Advisory Committee for Aeronautics. NASA/TM-1999-209127. Available at <http://techreports.larc.nasa.gov/ltrs/PDF/1999/tm/NASA-99-tm209127.pdf>.
- Nelson, M. L. (2000). Buckets: Smart Objects for Digital Libraries. Ph.D. Dissertation, Department of Computer Science, Old Dominion University, August 2000. Available at <http://home.larc.nasa.gov/~mln/phd/>.
- Open Archive Initiative. (2000). Available at <http://www.openarchives.org/>.

- Payette, S. & Lagoze, C. (2000). Policy-Carrying, Policy-Enforcing Digital Objects. In J. Borbinha & T. Baker (eds.), *Research and advanced technology for digital libraries*, fourth European conference, ECDL 2000 (pp. 144-157), Berlin: Springer.
- Phelps, T. A. and Wilensky, R. (2000). Multivalent documents. *Communications of the ACM*, 43(6), 82-90.
- Van de Sompel, H. & Hochstenbach, P. (1999). Reference linking in a hybrid library environment: part 2: SFX, a generic linking service. *D-Lib Magazine* 5(4). Available at [http://www.dlib.org/dlib/april99/van\\_de\\_sompel/04/van\\_de\\_sompel-pt2.html](http://www.dlib.org/dlib/april99/van_de_sompel/04/van_de_sompel-pt2.html).
- Van de Sompel, H., Krichel, T., Nelson, M. L., Hochstenbach, P., Lyapunov, V. M., Maly, K., Zubair, M., Kholief, M., Liu, X. & O'Connell, H. (2000a). The UPS prototype: an experimental end-user service across e-print archives. *D-Lib Magazine*, 6(2). Available at <http://www.dlib.org/dlib/february00/vandesompel-ups/02vandesompel-ups.html>.
- Weibel, S., Kunze, J., Lagoze, C. & Wolf, M. (1998). Dublin Core metadata for resource discovery. Internet RFC 2413. Available at <ftp://ftp.isi.edu/in-notes/rfc2413.txt>.