2001

Smart Objects and Open Archives

Michael L. Nelson
Old Dominion University

Kurt Maly
Old Dominion University

Follow this and additional works at: https://digitalcommons.odu.edu/computerscience_fac_pubs
Part of the Computer Sciences Commons, and the Digital Communications and Networking Commons

Repository Citation
Nelson, Michael L. and Maly, Kurt, "Smart Objects and Open Archives" (2001). Computer Science Faculty Publications. 6.
https://digitalcommons.odu.edu/computerscience_fac_pubs/6

Original Publication Citation

This Article is brought to you for free and open access by the Computer Science at ODU Digital Commons. It has been accepted for inclusion in Computer Science Faculty Publications by an authorized administrator of ODU Digital Commons. For more information, please contact digitalcommons@odu.edu.
Smart Objects and Open Archives

Michael L. Nelson  
NASA Langley Research Center  
MS 158  
Hampton, VA 23681  
m.l.nelson@larc.nasa.gov

Kurt Maly  
Old Dominion University  
Computer Science Department  
Norfolk, VA 23592  
maly@cs.odu.edu

Abstract

Within the context of digital libraries (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 believe dismantling the stovepipe of "DL-archive-content" is the first step in building richer DL experiences for users and insuring the long-term survivability 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 Object, Dumb Archive" (SODA) DL model, which promotes the importance and responsibility of individual information objects and reduces the role of traditional archives and database systems. The goal is to have smart objects be independent of and more resilient to the transient nature of information systems. The SODA model fits well with the emerging Open Archives Initiative (OAI), which promotes DL interoperability through the use of simple archives. This paper examines the motivation for buckets, SODA and the OAI, and initial experiences using them in various DL testbeds.

1.0 Introduction

Digital library (DL) discussions are often dominated by the merits of various archives, repositories, search engines, search interfaces and database systems. Information content is more important than the systems used for its storage and retrieval, and information content and information retrieval systems should progress independently and make limited assumptions about the status or capabilities of the other. Digital information should have the same long-term survivability prospects as traditional hardcopy information and should not be impacted by churning technologies or vendor vagaries. DL technologies have allowed commercial publishers to become more involved with library functions, serving on the World Wide Web
Smart Objects and Open Archives

(WWW) the byproducts of their publishing processes (PostScript, PDF, etc.). However, ultimately the goals of publishers and the goals of libraries are not the same, and the long-term commitment of publishers to provide library-quality archival and dissemination services is in doubt (Arms, 1999). While not a panacea, an institution's application of DL technologies will be an integral part of their knowledge usage and preservation effort, in either supplanting or supplementing traditional libraries.

All of this has tremendous impact on a U.S. Government agency like NASA, whose ultimate product is information. The deliverables of NASA's aeronautical and space projects are information for either a targeted set of customers (often industrial partners) or the scientific community at large. The information can have many forms: publications in the open literature; a self-published technical report series; and non-traditional STI media types such as data and software. NASA contributions to the open literature are subject to the same widening gap in conservation and output identified in (Henderson, 1999). For some, the NASA report series is either unknown or difficult to obtain (Roper, et al., 1994). For science data, NASA has been criticized for poor preservation of this data (United States General Accounting Office, 1990).

However, NASA has identified and is addressing these problems with ambitious goals. From the NASA STI Program Plan (NASA, 1998):

"By the year 2000, NASA will capture and disseminate all NASA STI and provide access to more worldwide mission-related information for its customers. When possible and economical, this information will be provided directly to the desktop in full-text format and will include printed material, electronic documentation, video, audio, multimedia products, photography, work-in-progress, lessons-learned data, research laboratory files, wind tunnel data, metadata, and other information from the scientific and technical communities that will help ensure the competitiveness of U.S. aerospace companies and educational institutions."

Although tempered with the phrase "possible and economical", it is clear that the expectations are higher than simply automating traditional library practices. Much of the STI identified above has historically not been included in traditional library efforts, primarily because of the mismatch in hard- and soft-copy media formats. However, the ability to now document the entire research process, and not just the final results, presents new challenges about how to acquire and manage this increased volume of information. To effectively implement the above mandate, additional DL technology is required.

2.0 Information Survivability

The Task Force on Archiving of Digital Information (1996) distinguishes between: refreshing, periodically copying the digital information to a new physical media; and migrating, updating the information to be compatible with a new hardware/software combination. The nature of refreshing necessitates a hardware-oriented approach (perhaps with secondary software assistance). Software objects cannot directly address issues such as the lifespan of digital media or availability of hardware systems to interpret and access digital media, but they can implement a migration strategy in the struggle against changing file formats. An aggregative software object could allow long-term accumulation of converted file formats. Rather than successive (and possibly lossy) conversion of: Format 1 -> Format 2 -> ..-> Format n, we should have the option of storing conversion of a format to any arbitrary format: Format 1 -> Format 2, Format 1 -> Format 3, Format 2 -> Format 4 with each intermediate format stored in the same location. This permits the "throw away nothing" philosophy, without burdening the DL directly with increasing numbers of formats.

For example, a typical research project at NASA Langley Research Center produces...
information tuples: raw data, reduced data, manuscripts, notes, software, images, video, etc. Normally, only the report is officially published and tracked. The report might reference online resources, or even include a CD-ROM, but these items are likely to be lost, degrade, or become obsolete over time. Some portions, such as software, can go into separate archives (i.e., COSMIC -- the official NASA software repository) but this leaves the researcher to locate the various archives, then re-integrate the information tuple by selecting pieces from different and possibly incompatible archives. Most often, the software, datasets, etc., are simply discarded or effectively lost in informal, short-lived personal archives. After 10 years, the manuscript is the only likely surviving artifact of the information tuple. As an illustration, COSMIC ceased operation in July 1998; its operations were turned over to NASA's technology transfer centers. However, at the time of this writing there appears to be no operational successor to COSMIC. Unlike their report counterparts in traditional libraries or even DLs, the software contents of COSMIC have been unavailable for several years, if not completely lost.

Additional steps can be taken to insure the survivability of information objects. Data files could be bundled with the application software used to process them, or if common enough, different versions of the application software, with detailed instructions about the hardware system required to run them, could be part of the DL. Furthermore, they could include enough information to guide the future user in selecting (or developing) the correct hardware emulator.

3.0 Buckets

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, et al., 1995), and the NACA Technical Report Server (NACATRS) (Nelson, 1998). In early user evaluation studies of these DLs, one recurring 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, rather than creating separate DLs for each information type or stretching the definition of traditional reports to include various multi-media formats, we defined an arbitrary digital object to 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 requires the buckets to be self-sufficient and perform whatever tasks are required of them, potentially without the benefit of being arrived at through a specific DL.

Buckets are our implementation of smart objects. A bucket is a storage unit that contains data and metadata, as well as the methods for accessing both. Buckets are similar in design to Kahn-Wilensky Digital Objects (Kahn & Wilensky, 1995), but with a few architectural changes and optimizations specific to DLs. The bucket design goals are: aggregation, intelligence, self-sufficiency, mobility, heterogeneity and archive independence. It is difficult to over-stress the importance of aggregation as a design goal. In our DL experience, 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, and new information types (software, multimedia) were
introduced. The result of a report being "in 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.

3.1 How Buckets Work

The first focus of the aggregation was on the various data types. Based on experience gained
with LTRS and NTRS, we 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, and this was a simplifying assumption during bucket implementation. Future
work will implement arbitrarily complex, multi-level data objects. An element can be a
"pointer" to another bucket or any 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 to the user to consider 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 number of packages and elements.

The buckets described here are version 1.6.2. Buckets are currently written in Perl 5 and use
http as the transport protocol for sending messages defined in the bucket application
programming interface (API). 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,
et al., 1998), which demonstrated a modified version of the Dienst protocol (Davis & Lagoze,
2000). Owing to their Dienst-related heritage, bucket metadata is stored in RFC-1807 format
(Lasher & Cohen, 1995). Although buckets use RFC-1807 as their native format, they can
contain and serve any metadata type. The bucket is accessible through a common gateway
interface (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. A corollary of the mobility design goal is that buckets should not require changes
in a "reasonable" http server setup; where "reasonable" is defined as allowing 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,
buckets have been tested on a variety of systems (Table 1).

<table>
<thead>
<tr>
<th>Architecture</th>
<th>Operating System</th>
<th>Perl</th>
<th>http Server</th>
</tr>
</thead>
<tbody>
<tr>
<td>Sparc</td>
<td>Solaris 2.7</td>
<td>5.005_03</td>
<td>Apache 1.3.9</td>
</tr>
<tr>
<td>Sparc</td>
<td>Solaris 2.7</td>
<td>5.005_03</td>
<td>NCSA httpd 1.5.2</td>
</tr>
<tr>
<td>Sparc</td>
<td>Red Hat 6.0 (Linux 2.2.5-15)</td>
<td>5.005_03</td>
<td>Apache 1.3.6</td>
</tr>
<tr>
<td>Intel x86</td>
<td>Windows NT 4.0 (1381 /SP 5)</td>
<td>Active Perl 5.005_03</td>
<td>Apache 1.3.12</td>
</tr>
<tr>
<td>Intel x86</td>
<td>Mandrake Linux 6.2</td>
<td>5.005_03</td>
<td>Apache 1.3.6</td>
</tr>
</tbody>
</table>
Buckets are accessed through one or more URLs. For an example of a single bucket 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 URLs point to the same bucket, even though they are access through different hosts. Also consider a host with multiple http servers:

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

If the http server on port 8080 defines its document root to be the directory "bar", then the two URLs point to the same bucket. That a bucket is potentially accessible from different hosts and servers illustrates the importance of buckets maintaining their own logs.

Elements and packages have no predefined semantics. Authors can model any application domain using the basic structures of packages and elements. In NASA DL buckets, packages represent semantic types (manuscript, software, test data, etc.) and elements represent syntactic representations of the packages (.ps version, .pdf version, .dvi version, etc.). Other bucket models are possible, and we have used buckets for entire research projects and university classes as well as technical publications.

Buckets provide mechanism, not policy. Buckets can implement different policies: one site might allow authors to modify buckets after publishing, and another site might "freeze" buckets 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.

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 manually. 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 prescriptive stance on metadata. Although the bucket must ultimately choose one metadata format for structural purposes, buckets can 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.

The final aggregation focus was on the services defined on buckets and the results of those services. In object-oriented fashion, we wanted method source code to be resident in the bucket. Although they can be "factored out" to shared locations, by default everything the bucket needs to display, disseminate, and manage its contents is contained within the buckets. This includes the method source code, 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 its environment and should require no server modifications to function. Such self-sufficiency results in a storage overhead of approximately 100Kb per bucket, although space savings can be achieved by decreasing the mobility and self-sufficiency.

3.2 Bucket Messages

Even though the bucket API encodes its messages using http, buckets appear as ordinary URLs and casual users should not realize they are not interacting with a regular 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 specified, the default "display" method is assumed. 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/

is the same as:

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

Figure 1. The default display method.
Both URLs produce the output in Figure 1. These URLs could be reached through either searching or browsing in 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 2) 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

![Netscape A self-synchronizing stroboscopic Schlieren system for the study of unsteady air flows](http://www.cs.odu.edu/~nelso_m/naca-tn-2509/)

Figure 2. A display altered with preference.

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 in the HTML output:


which would cause the WWW browser to launch the PDF reader. Similarly, if users wish 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 3. To the casual observer, the bucket API is transparent. However, if individual users or harvesting robots know a particular URL is a bucket, additional actions are possible. For example, to extract the metadata in default (RFC-1807) format, the URL is:

http://www.cs.odu.edu/~nelson_m/naca-tn-2509/?method=metadata
Figure 4. The metadata method.

which would return the metadata in a structured format (Figure 4), suitable for indexing by a DL. If a specific metadata format was desired, it could be requested:

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

which will result in a MARC record being returned if the bucket has the metadata in MARC format, or can have it converted into MARC format (see the next section). If a user or agent wishes to determine the 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

To see what methods are defined on a bucket (Figure 5):

http://www.cs.odu.edu/~nelso_m/naca-tn-2509/?method=list_methods
Figure 5. The list_methods method.

And to determine the bucket's T&C before attempting potentially restricted methods (Figure 6):

http://www.cs.odu.edu/~nelso_m/naca-tn-2509/?method=list_tc
However, if a harvester is not bucket-aware, it can still "crawl" or "spider" the buckets as normal URLs, scraping information from the HTML interface generated by the "display" method. Buckets offer many expressive options to users or services that are bucket-aware, but are transparent to those who are not bucket-aware. All bucket methods are listed in Table 2, and a full discussion of the methods and their arguments is in (Nelson, 2000). Appendix 1 provides a tour of bucket methods.

### TABLE 2. Bucket methods.

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>add_element</td>
<td>Adds an element to a package</td>
</tr>
<tr>
<td>add_method</td>
<td>Adds a method to the bucket</td>
</tr>
<tr>
<td>add_package</td>
<td>Adds a package to the bucket</td>
</tr>
<tr>
<td>add_principle</td>
<td>Adds a user id to the bucket</td>
</tr>
<tr>
<td>add_tc</td>
<td>Adds a T&amp;C file to the bucket</td>
</tr>
<tr>
<td>delete_bucket</td>
<td>Deletes the entire bucket</td>
</tr>
<tr>
<td>delete_element</td>
<td>Deletes an element from a package</td>
</tr>
<tr>
<td>delete_log</td>
<td>Deletes a log file from the bucket</td>
</tr>
<tr>
<td>delete_method</td>
<td>Deletes a method from the bucket</td>
</tr>
</tbody>
</table>
4.0 Bucket Communication Space

The Bucket Communication Space (BCS) is inspired by Linda, the parallel communication library (Carriero & Gelernter, 1989). In Linda, processes 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 "rd" 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 the processes. Though it imposes a performance overhead, Linda provides a useful layer of abstraction for inter-process communication.

We desired something similar for buckets: buckets communicating with other buckets without knowing the details of bucket location. This is especially important if the buckets are mobile, and a bucket's location is not guaranteed to be static. The BCS also provides a method for centralizing functionality that cannot be replicated in individual buckets due to efficiency concerns (the resulting bucket would be too bloated) or implementation limitations (a service is available only on select architectures). Buckets need only know how to communicate to a BCS server, which can handle their requests for them. The location of a BCS server is stored as a preference within the bucket. The BCS defined methods are listed in Table 3. We provide proof-of-concept implementations for four services: file format conversion, metadata conversion, bucket messaging, and bucket matching. Appendix 2 provides a tour of the BCS methods.

Table 3. BCS methods.

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>delete_package</td>
<td>Deletes a package from the bucket</td>
</tr>
<tr>
<td>delete_principal</td>
<td>Deletes a user id from the bucket</td>
</tr>
<tr>
<td>delete_tc</td>
<td>Deletes a T&amp;C from the bucket</td>
</tr>
<tr>
<td>display</td>
<td>Displays and disseminates bucket contents</td>
</tr>
<tr>
<td>get_log</td>
<td>Retrieves a log file from the bucket</td>
</tr>
<tr>
<td>get_preference</td>
<td>Retrieves a preference(s) from the bucket</td>
</tr>
<tr>
<td>get_state</td>
<td>Retrieves a state(s) from the bucket</td>
</tr>
<tr>
<td>id</td>
<td>Displays the bucket's unique id</td>
</tr>
<tr>
<td>lint</td>
<td>Checks the bucket's internal consistency</td>
</tr>
<tr>
<td>list_logs</td>
<td>Lists all the log files in the bucket</td>
</tr>
<tr>
<td>list_methods</td>
<td>Lists all the methods in the bucket</td>
</tr>
<tr>
<td>list_principals</td>
<td>Lists all the user ids in the bucket</td>
</tr>
<tr>
<td>list_source</td>
<td>Lists the method source</td>
</tr>
<tr>
<td>list_tc</td>
<td>Lists all the T&amp;C files in the bucket</td>
</tr>
<tr>
<td>metadata</td>
<td>Displays the metadata for the bucket</td>
</tr>
<tr>
<td>pack</td>
<td>Returns a &quot;bucket-stream&quot;</td>
</tr>
<tr>
<td>set_metadata</td>
<td>Uploads a metadata file to the bucket</td>
</tr>
<tr>
<td>set_preference</td>
<td>Changes a bucket preference</td>
</tr>
<tr>
<td>set_state</td>
<td>Changes a bucket state variable</td>
</tr>
<tr>
<td>set_version</td>
<td>Changes the version of the bucket</td>
</tr>
<tr>
<td>unpack</td>
<td>Overlays a &quot;bucket-stream&quot; into the bucket</td>
</tr>
<tr>
<td>version</td>
<td>Displays the version of the bucket</td>
</tr>
</tbody>
</table>
4.1 File Format Conversion

File format conversion provides bi-directional conversion of image (e.g., GIF, JPEG) formats and page description formats (e.g., PostScript, PDF). Format conversion is an obvious application -- additional formats will become available after a bucket's publication, and the ability to either place them in the bucket or dynamically create them will be useful in information migration. This service is implemented as a wrapper around the popular ImageMagick and Image Alchemy programs.

4.2 Metadata Conversion

Metadata conversion is similar to file format conversion, providing conversion between some of the more popular metadata formats. Metadata conversion is extremely important because, although buckets ultimately have to choose a single format on which to operate, it is unreasonable to assume all applications needing metadata from the bucket must choose the same format. Being able to specify the desired format to receive from a bucket also leaves the bucket free to change its canonical format in the future. Similar to the file format conversion, this service is implemented as a wrapper to our "mdt" metadata conversion script (Nelson, et al., 1999).

4.3 Bucket Messaging

Messaging allows multiple buckets to receive a message if they match specific criteria. While point-to-point communication between buckets is always possible, bucket messaging provides a method for discovering and then sending messages to buckets. Messaging provides functionality close to the original inspiration of Linda, and can be used as the core of a "bucket-multicasting" service that sends pre-defined messages to a subset of registered buckets. This could be used in turn to implement a metadata normalization and correction service, such as that described by French, et al. (1997) or Lawrence, Bollacker & Giles (1999).

4.4 Bucket Matching

A compelling demonstration of the BCS is bucket matching. Matching provides the capability to create linkages between "similar" buckets. Consider a technical report published by the Old Dominion University computer science department that is also submitted to a conference. The report exists on the DL maintained by the department and the publishing authority is ncstrl.odu_cs. If the conference paper is accepted, it will eventually be published by the conference sponsor. Say the conference sponsor is the Association for Computing Machinery, and publishing authority is ncstrl.acm. Although the conference paper will appear in a modified format (edited and perhaps abbreviated), the technical report and the conference paper are clearly related, despite being separated by publishing authority, date of publication, and...
editorial revisions. Two separate but related objects now exist, and are likely to continue to exist.

How best to create the desired linkage between the two objects? It is easy to assume ncsrl.acm has neither the resources nor the interest to spend the time searching for previous versions of a manuscript. Similarly, ncsrl.udcs cannot link to the conference bucket at the creation time of the technical report bucket, since the conference bucket did not exist then. It is unrealistic to suggest the relevant parties will go back to the ncsrl.udcs archive and create linkages to the ncsrl.acm bucket after six months to a year have passed. However, if both buckets are registered in the same bucket communication space (by way of sending their metadata or fulltext), they can "find each other" without human intervention. When a match, or near match (the threshold for "match" being a configurable parameter) is found, the buckets can either automatically link to each other, or inform a human reviewer that a potential match has been found and request approval for the linkage.

This technique could also be used to find related work from different authors and even duplications (accidental or plagiarism). In tests using approximately 3000 National Advisory Committee for Aeronautics (NACA) buckets, multi-part reports were found (e.g., Part 1, Part 2) and matched, as were Technical Notes (archival equivalent of a computer science technical report) that were eventually published as Reports (archival equivalent of a journal article), and a handful of errors where duplicate metadata was erroneously associated with separate reports. See Appendix 2 for examples of bucket matching with NACA data.

5.0 Smart Objects, Dumb Archives

Buckets are part of the larger "Smart Object, Dumb Archive" DL Model (Maly, Nelson, & Zubair, 1999). SODA is a reaction to the vertically integrated (and non-interoperable) DLs that tended to grow from the ad-hoc origins of many popular DLs (Esler & Nelson, 1998). Separating the functionality of the archive from that of the DL 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. 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 would poll the NASA 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 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.

6.0 Open Archives Initiative

Just as buckets break the dependency of the information objects on archives, the Open Archives Initiative (OAI) breaks the dependency of archives on DLs. A separate protocol, "DA", for archives was originally defined and implemented in (Nelson, 2000) but the protocol is no longer being developed. Instead, the DA functionality is now provided by the evolving Open Archives Initiative (OAI) and its metadata harvesting protocol. The OAI does not address the issue of smart objects, but the archives in the OAI are very similar to the archives described in
the SODA model in that they have minimal functionality. OAI archives aim for greater interoperability through performing less sophisticated functions (e.g., no keyword search functions defined, T&C is not handled at the protocol level) -- a sort of Reduced Instruction Set Computer (RISC) philosophy for archives.

The OAI metadata harvesting protocol addresses the problem of metadata being "locked" within a DL and not being easily exportable. The OAI metadata harvesting protocol defines six "verbs" (Table 4) 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 (Van de Sompel & Lagoze, 2000). Most of the six verbs take various arguments such as datestamps or archive-defined sets to allow for partial harvesting, and there are optional flow-control provisions to throttle harvesting on busy or large archives. Although any metadata format can be provided by a data provider, in the interest of easing the task of creating service providers, unqualified Dublin Core (Weibel, et al., 1998) is defined as the minimally required metadata format.

**TABLE 4. OAI verbs.**

<table>
<thead>
<tr>
<th>Verb</th>
<th>Function</th>
</tr>
</thead>
<tbody>
<tr>
<td>ListMetadataFormats</td>
<td>metadata formats supported by archive</td>
</tr>
<tr>
<td>ListSets</td>
<td>sets defined by archive</td>
</tr>
<tr>
<td>ListIdentifiers</td>
<td>OAI unique ids contained in archive</td>
</tr>
<tr>
<td>ListRecords</td>
<td>listing of all records</td>
</tr>
<tr>
<td>GetRecord</td>
<td>listing of a single record</td>
</tr>
</tbody>
</table>

It should be noted the OAI archives are not intended for user-interaction, and the OAI protocol is not defined as a stand-alone system: an OAI interface is always a front-end to some other archival system (e.g., a relational database management system, directory service, filesystem, or Dienst server). The goal of the OAI is to provide a standard mechanism for a DL to expose its metadata to external harvesters and to encourage the creation of value-added DLs that provide resource discovery to content from multiple archives for targeted user communities. To increase the chances of wide-spread adoption, the OAI purposely defines a minimal harvesting protocol, with the provision of hooks for community-specific specializations. On the theory that existence of many data providers will encourage the creation of service providers, in cases where complexity can reside either with the service provider or the data provider, the OAI shifts the burden to the service provider.

The OAI grew out of the meeting surrounding the presentation of the Universal Preprint Service (UPS) demonstration DL. The UPS was introduced 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 access to a federation of DLs, the project aimed at identifying the key issues in actually creating an experimental end-user service for data originating from existing production archives. This included almost 200,000 buckets harvested from six popular 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, et al., 2000).

Two OAI compliant archives have been built for LTRS and NACATRS. They are, respectively:

http://techreports.larc.nasa.gov/ltrs/oai/
http://naca.larc.nasa.gov/oai/

Examples of valid OAI messages to these archives are:
It is worth noting that both of these archives are implemented as modified buckets. In addition to the standard buckets defined in Table 2, the above buckets also have the 6 additional methods defined in Table 4 included (bucket "methods" and OAI "verbs" are synonymous). They also carry as a data file a support library that maps the general implementation of the OAI verbs to match the structure of the specific DLs they service, which can be easily modified using the standard bucket methods for uploading new data elements. The BCS server is similarly implemented as a modified bucket (implementing the methods defined in Table 3), as was the now defunct DA archive. Although originally designed to transport information content, buckets have proven to be useful, extensible computational WWW entities.

7.0 Future Work

The lessons learned from implementing buckets and supporting technology in NCSTRL+ and UPS point to many areas of future work. Currently, there is a project planned for bucket usage between NASA, the Air Force Research Laboratory and Los Alamos National Laboratory to use buckets and the OAI for DL interoperability. Buckets are especially suited for the multiplicity of files and formats resulting from the older technical reports which have to be scanned, and OAI archives will be used to allow each site's DLs to harvest and ingest the metadata from each other. A number of bucket features are being improved currently (such as replacing the two-level architecture with an arbitrary level architecture), but there are also a number of longer range areas of research.

7.1 Alternate Implementations

Although Perl and CGI are good development platforms, other bucket implementations should be explored. This includes making the bucket API available through non-http environments, such as CORBA (Vinoski, 1997), and implementing buckets using other languages and relational database management systems.

7.2 Pre-defined Packages and Elements

Some functionality improvements could be made, not through new or modified methods but through conventions established on the current infrastructure. One convention already adopted was the use of a BCS_Similarity.pkg package to hold the resulting links of the BCS similarity indexing. Other possible uses include standard packages (or elements) for bibliographic citation information, possibly in multiple encodings and standard package or element names for previous revisions of bucket material. The ability to store derived works and intermediate results in the object itself is likely to prove useful.

7.3 Increased Intelligence

There are a number of functionality hooks in place that have not yet been fully automated. For example, the "lint" method can detect internal errors and misconfigurations in the bucket, but it does not yet attempt to repair a damaged bucket. Similarly, a bucket preference could control the automatic updating of buckets when new releases are available, while still maintaining the bucket's own configuration and local modifications. The updated bucket could then be tested for correct functionality, and rolled back to a previous version if testing fails. The option of removing people from the bucket update cycle would ease a traditional administration burden.

7.4 Security, Authentication and Terms & Conditions
Currently, buckets have no line of defense if the http server or the system software itself is attacked. Having buckets employ some sort of encryption on their files that is decoded dynamically would offer a second level of security, making the buckets truly opaque data objects that could withstand at least some level of attack if the system software was compromised. Authentication is currently done through standard http procedures. However, more sophisticated authentication technologies will be necessary for general large-scale deployment.

7.5 Usage Analysis

There are several DL projects that focus on determining the usage patterns of their holdings and dynamically arranging the relationships within the DL holdings based on these patterns (Bollen & Heylighen, 1997; Rocha, 1999). These projects are similar in that they extract usage patterns of passive documents, either examining the log files of the DL, or instrumenting the interface to the DL to monitor user activity, or some hybrid of these approaches. An approach that has not been tried is for objects themselves to participate in determining the usage patterns, perhaps working in conjunction with monitors and log files. Since the buckets are executable code, it is possible to instrument not just the resource discovery mechanisms, but also the archived objects. We have experience instrumenting buckets to extract additional usage characteristics, but have not combined this strategy with the other projects.

7.6 Software Reuse

Buckets could impact software reuse as well. If a bucket stores code, such as a solver routine, it would be limited to a model where users extract the code and link it into their application, but rather the bucket could provide the service, and be accessible through remote procedure call (RPC)-like semantics. Interfaces between distributed computing managers such as Netsolve (Casanova & Dongarra, 1998) or NEOS (Czyzyk, Mesnier, & More, 1998) and "solver buckets" could be built. Data, and the routines to derive and manipulate it, could reside in the same bucket in a DL. This would likely be tied to a discipline specific application, such as a bucket having a large satellite image and a method for dynamically partitioning and disseminating portions of the data. The traditional model of "data resides in the library; analysis and manipulation occurs outside the library" can be circumvented by making the archived objects also be computational objects.

8.0 Related Work

There are projects from the DL community with similar aggregation goals as buckets, such as Multivalent Documents (Phelps & Wilensky, 2000) and FEDORA (Payette & Lagoze, 2000). Some projects, such as the VERS Encapsulated Objects (VEOs) of the Victorian Electronic Record Strategy (VERS) (Waugh, et al., 2000), focus primarily on digital preservation goals. But none of these feature mobility, self-sufficiency or the SODA-inspired motivation of freeing the information object from archival control and dependency. 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 intelligence is embedded in the object and not just something applied to the object.

9.0 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 the
technical publication, and the traditional publication systems and the digital systems that automated them were unable to adequately address users' 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 are aggregative, intelligent, WWW-accessible digital objects that are optimized for publishing in DLs. Buckets implement the philosophy that information itself is more important than the DL systems used to store and access information. Their aggregative capabilities allow retention of past formats and derived works, with the hopes of increasing the future usability of the object. Buckets are designed to imbue the information objects with certain responsibilities, such as the display, dissemination, protection and maintenance of its contents. As such, buckets should be able to work with many DL systems simultaneously, and minimize or eliminate the necessary modification of DL systems to work with buckets. Ideally, buckets should work with everything and break nothing. This philosophy is formalized in the SODA DL model: objects become "smarter" at the expense of the archives (that become "dumber"), as functionalities generally associated with archives are moved into the data objects themselves. This shift in responsibilities from the archive into the buckets results in a greater storage and administration overhead, but these overheads are small in comparison to the great flexibility that buckets bring to DLs. Freeing the information objects from the dependency of specific archive software, databases or search engines should increase their chances at long-term survivability. The SODA model can be implemented using buckets as "smart objects" and using OAI compliant archives as the "dumb archives".

Buckets are already having a significant impact in how NASA and other organizations such as Los Alamos National Laboratory, Air Force Research Laboratory are designing their next generation DLs. Buckets, through aggregation, intelligence, mobility, self-sufficiency, and heterogeneity, provide the infrastructure for information object independence. The truly significant applications of this new breed of information objects remain undiscovered.

Appendices

**Appendix 1:** A Bucket Demo

**Appendix 2:** A Bucket Communication Space Demo

References


