1998

Buckets: Aggregative, Intelligent Agents for Publishing

Michael L. Nelson
Old Dominion University

Kurt Maly
Old Dominion University

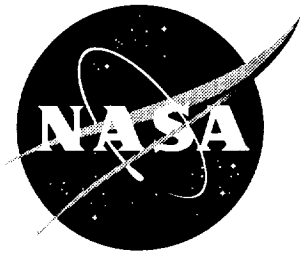
Stewart N. T. Shen
Old Dominion University

Mohammad Zubair
Old Dominion University

Original Publication Citation

Nelson, M.L., Maly, K., Shen, S.N., & Zubair, M. (1998). Buckets: Aggregative, intelligent agents for publishing. *NASA Technical Memorandum: 208419*. Hampton, VA: NASA Langley Research Center.

NASA/TM-1998-208419



Buckets: Aggregative, Intelligent Agents for Publishing

Michael L. Nelson
Langley Research Center, Hampton, Virginia

Kurt Maly, Stewart N. T. Shen, and Mohammad Zubair
Old Dominion University, Norfolk, Virginia

National Aeronautics and
Space Administration

Langley Research Center
Hampton, Virginia 23681-2199

May 1998

Available from the following:

NASA Center for AeroSpace Information (CASI)
7121 Standard Drive
Hanover, MD 21076-1320
(301) 621-0390

National Technical Information Service (NTIS)
5285 Port Royal Road
Springfield, VA 22161-2171
(703) 487-4650

Buckets: Aggregative, Intelligent Agents for Publishing

Michael L. Nelson
NASA Langley Research Center
MS 158
Hampton, VA 23681-0001, USA
E-mail: m.l.nelson@larc.nasa.gov

Kurt Maly, Stewart N. T. Shen, Mohammad Zubair
Old Dominion University
Computer Science Department
Norfolk, VA 23529, USA
E-mail: {maly, shen, zubair}@cs.odu.edu

ABSTRACT
Buckets are an aggregative, intelligent construct for publishing in digital libraries. The goal of research projects is to produce information. This information is often instantiated in several forms, differentiated by semantic types (report, software, video, datasets, etc.). A given semantic type can be further differentiated by syntactic representations as well (PostScript version, PDF version, Word version, etc.). Although the information was created together and subtle relationships can exist between them, different semantic instantiations are generally segregated along currently obsolete media boundaries. Reports are placed in report archives, software might go into a software archive, but most of the data and supporting materials are likely to be kept in informal personal archives or discarded altogether. Buckets provide an archive-independent container construct in which all related semantic and syntactic data types and objects can be logically grouped together, archived, and manipulated as a single object. Furthermore, buckets are active archival objects and can communicate with each other, people, or arbitrary network services.

KEYWORDS: Digital library architectures, agents, archiving, multi-format, bucket, container, package.

INTRODUCTION
Digital libraries (DLs) are an important research topic in many scientific communities and have already become an integral part of the research process. However, access to these DLs is not as easy as users would like. Digital libraries are partitioned both by the discipline they serve (computer science, aeronautics, physics, etc.) and by the format of their holdings (technical reports, video, software, etc.). There are two significant problems with current DLs. First, interdisciplinary research is difficult because the collective knowledge of each discipline is stored in incompatible DLs that are known only to the specialists in the subject. The second significant problem is that although scientific and technical information (STI) consists of manuscripts, software, datasets, etc., the manuscript receives the majority of attention, and the other components are often discarded (Figure 1) [20]. Although non-manuscript digital libraries such as the software archive *Netlib* [2] have been in use for some time, they still place the burden of STI reintegration on the customer. A NASA study found that customers desire to have the entire set of manuscripts, software, data, etc. available in one place [19]. With the increasing availability of all-digital storage and transmission, maintaining the tight integration of the original STI collection is now possible.

Old Dominion University and NASA Langley Research Center are developing NCSTRL+ to address the multi-discipline and multi-genre problems. NCSTRL+ is based on the Networked Computer Science Technical Report Library (NCSTRL) [5], which is a highly successful digital library offering access to over 100 university departments and laboratories since 1994, and is implemented using the Dienst protocol [9]. During the development stage, NCSTRL+ includes selected holdings from the NASA Technical Report Server (NTRS) [14] and NCSTRL, providing clusters of collections along the dimension of disciplines such as aeronautics, space science, mathematics, computer science, and physics, as well as clusters along the dimension of publishing organization and genre, such as project reports, journal articles, theses, etc. The DL aspects of NCSTRL+ are discussed in [15, 16]. Although developed for NCSTRL+ and with our modified version of the Dienst protocol in mind, buckets are protocol and archive independent, needing only standard World Wide Web (WWW) capability to function. This paper gives an overview of bucket functionality, examines similar work, and discusses current implementation and future plans.

OVERVIEW
Buckets are object-oriented container constructs in which logically grouped items can be collected, stored, and transported as a single unit. 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 part of this information tuple is officially published and tracked. The report might reference on-line resources, or even include a CD-ROM, but these items are likely to be lost or degrade over time. Some portions such as software, can go into separate archives (i.e., COSMIC or the Langley Software Server) but this leaves the researcher to re-integrate the information tuple by selecting pieces from multiple archives. Most often, the software and other items, such as datasets are simply discarded. After 10 years, the manuscript is almost surely the only surviving artifact of the information tuple.

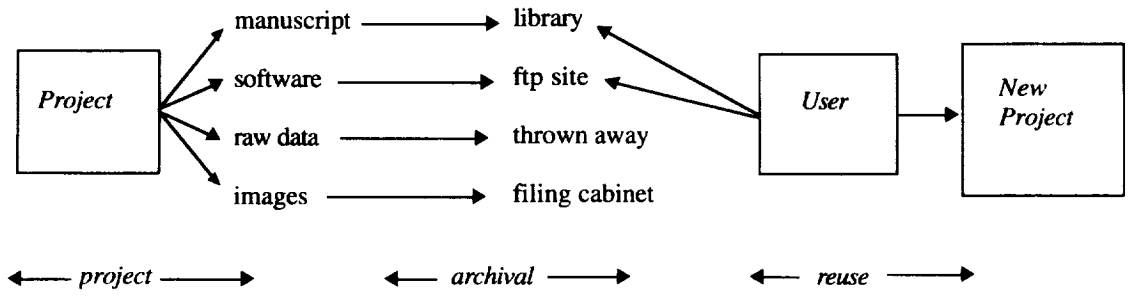


Figure 1: STI Lost in Project / Archival / Reuse Process

Large archives could have buckets with many different functionalities. Not all bucket types or applications are known at this time. However, we can describe a generalized bucket as containing many formats of the same data item (PS, Word, Framemaker, etc.) but more importantly, it can also contain collections of related non-traditional STI materials (manuscripts, software, datasets, etc.) Thus, buckets allow the digital library to address the long standing problem of ignoring software and other supportive material in favor of archiving only the manuscript [20] by providing a common mechanism to keep related STI products together. A single bucket can have multiple *packages*. Packages can correspond to the semantics of the information (manuscript, software, etc.), or can be more abstract entities such as the metadata for the entire bucket, bucket terms and conditions, pointers to other buckets or packages, etc. A single package can have several *elements*, which are typically different file formats of the same information, such as the manuscript package having both PostScript and PDF elements. Elements correspond to the syntax of a package. Packages and elements are illustrated in Figure 2.

Bucket Requirements

All buckets have unique ids, handles [7], associated with them. Buckets are intended to be either standalone objects or to be placed in digital libraries. A standalone bucket can be accessible through normal WWW means without the aid of a repository. Buckets are intended to be useful even in repositories that are not knowledgeable about buckets in general, or possibly just not about the specific form of buckets. Buckets should not lose functionality when removed from their repository. The envisioned scenario is that NCSTRL+ will eventually have moderate numbers of (10s - 100s of thousands) of intelligent, custom buckets instead of large numbers (millions) of homogenous buckets. Figure 3 contrasts a traditional architecture of having the repository interface contain all the intelligence and functionality with that of a bucket architecture where the repository intelligence and functionality can be split between the repository and individual buckets. This could be most useful when individual buckets require custom terms and conditions for access (security, payment, etc.). Figure 3 also illustrates a bucket gaining some repository intelligence as it is extracted from the archive en route to becoming a standalone bucket. A high level list of bucket requirements include:

- a bucket is of arbitrary size
- a bucket has a globally unique identifier
- a bucket contains 0 or more components, called packages (no defined limit)
- a package contains 1 or more components, called elements (no defined limit)
- an element can be a file, or a "pointer" to another
- both packages and elements can be other buckets (i.e., buckets can be nested)
- a package can be a "pointer" to a remote bucket, package, or element (remote package or element access requires "going through" the remote hosting bucket)
- packages and elements can be "pointers" to arbitrary network services, foreign keys to databases, etc .
- buckets can keep internal logs of actions performed on them
- interactions with packages or elements are made only through defined methods on a bucket
- buckets can initiate actions; they do not have to wait to be acted on
- buckets can exist inside or out of a repository

Table 1 lists the required bucket methods; other methods can be custom defined. Note that Table 1 differs from protocols such as the Repository Access Protocol (RAP) [10]. RAP defines what actions the Repository understands, while we define the actions that buckets understand. Although the two are not mutually exclusive, the current plan is to not implement RAP for NCSTRL+. Table 2 lists the default private methods for the bucket. We expect this list to grow as the public methods are refined, especially as the current terms and conditions model moves past its current hostname and username/password capability.

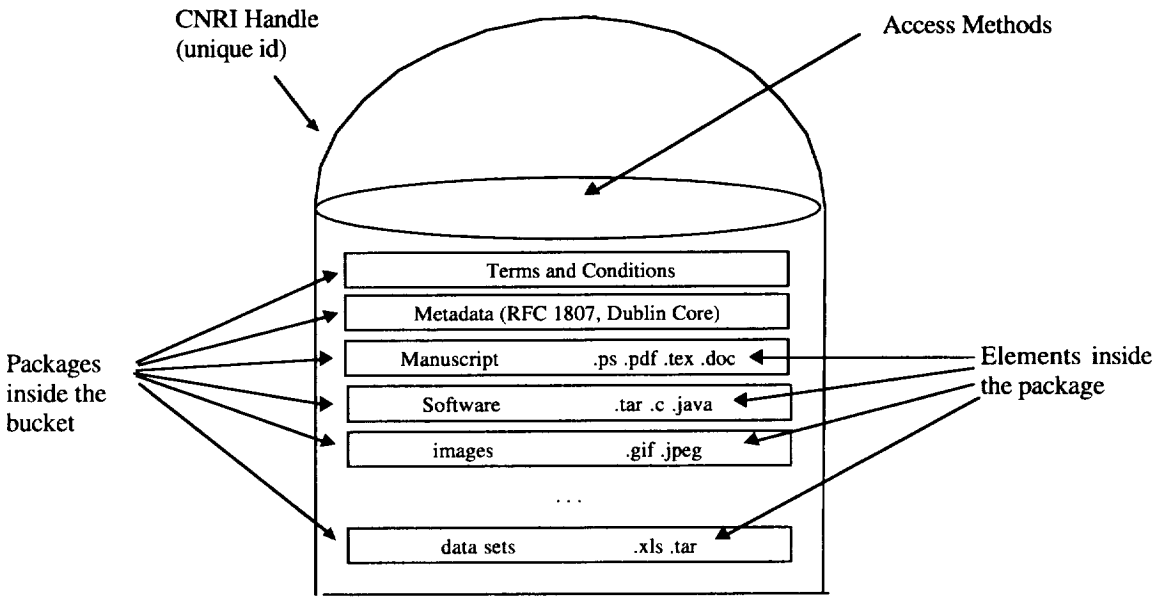


Figure 2: Bucket Architecture

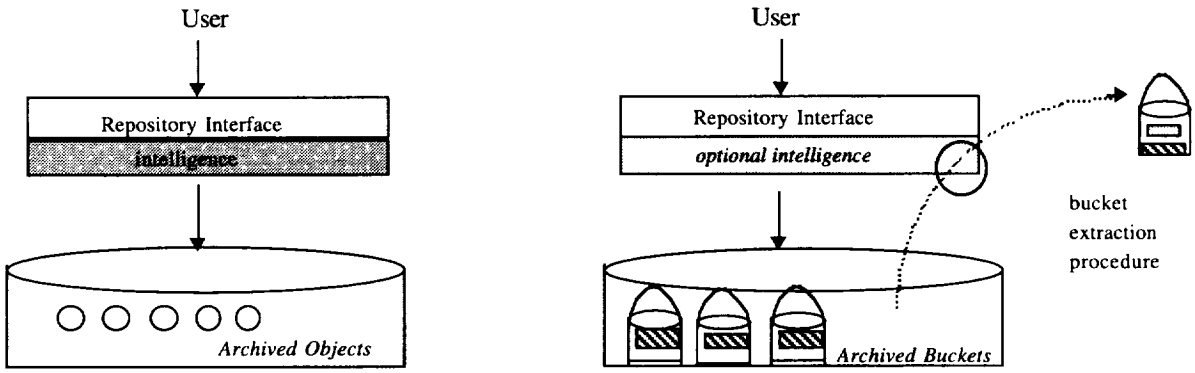


Figure 3: Traditional and Bucket Repository Architectures

Bucket Tools

There are two main tools for bucket use. One is the *author tool*, which allows the author to construct a bucket with no programming knowledge. Here, the author specifies the metadata for the entire bucket, adds packages to bucket, adds elements to the packages, provides metadata for the packages, and selects applicable clusters. The author tool gathers the various packages into a single component and parses the packages based on rules defined at the author's site. Many of the options of the author tool will be set locally via the second bucket tool, the *management tool*. The management tool provides an interface to allow site managers to configure the default settings for all authors at that site. The management tool also provides an interface to query and update buckets at a given repository. Additional methods can be added to buckets residing in a repository by invoking `add_method` on them and transmitting the new code. From this interface, the manager can halt the archive and perform operations on it, including updating or adding packages to individual buckets, updating or adding methods to groups of buckets, and performing other archival management functions.

Bucket Implementation

Our bucket prototypes are written in Perl 5, and make use of the fact that Dienst uses hypertext transfer protocol (HTTP) as a transport protocol. Like Dienst, bucket metadata is stored in RFC-1807 format [12], and package and element information is stored in newly defined optional and repeatable fields. 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. Thus a Dienst administrator can cd into the appropriate directory and access the contents. However, access for regular users occurs through the WWW. The bucket is accessible through a Common Gateway Interface (CGI) script that enforces terms and conditions, and negotiates presentation to the WWW client.

Table 1: Default Public Bucket Methods

Method	Argument	Currently Implemented	Description
metadata	format	Yes	with no argument, returns the metadata in the default format; with an argument, derives and returns the desired format
display	--	Yes	default method; bucket "unveils" itself to requester
id	--	Yes	returns the bucket's unique identifier (handle)
list_tc	--	No	describes the nature of the publicly visible terms and conditions
list_methods	--	Yes	list all public methods known by a bucket
list_owners	--	Yes	list all principals that can modify the bucket
add_owner	owner	No	add to the list of owners
delete_owner	owner	No	delete from the list of owners
add_package	package	Yes	adds a package to an existing bucket
delete_package	package	Yes	deletes a package from an existing bucket
add_element	element	Yes	adds an element to an existing package
delete_element	element	Yes	deletes an element from an existing package
get_package	package	No	capability to get an entire package, including all elements
get_element	element	No	get an element from a package in a bucket; currently direct URLs are used for element extraction
add_method	method	Yes	"teaches" a new method to an existing bucket
delete_method	method name	Yes	removes a method from a bucket
copy_bucket	destination	No	export a copy of a bucket, original remains
move_bucket	destination	No	move the original bucket, no version remains

Table 2. Default Private Bucket Methods

Method	Argument	Currently Implemented	Description
tc	method name	Yes	all public methods pass through this terms and conditions method
derive_metadata	format	No	converts from the default metadata format to the desired format

The philosophy of Dienst is to minimize the dependency on HTTP. Except for the User Interface service, Dienst does not make specific assumptions about the existence of HTTP or the Hypertext Markup Language (HTML). However, Dienst does make very explicit assumptions about what constitutes a document and its related data formats. Built into the protocol are the definitions of PostScript, ASCII text, inline images, scanned images, etc. To add a new file format, such as the increasingly popular PDF, Dienst configuration files have to be changed. If the protocol was resident only at one site, this would be acceptable. However, Dienst servers are running at nearly 100 sites – protocol additions require a coordinated logistical effort to synchronize versions and provide uniform capability.

We favor making Dienst less knowledgeable about dynamic topics such as file format, and making that the responsibility of buckets (Figure 4). In NCSTRL+, Dienst is used as an index, search, and retrieval protocol. When the user selects an entry from the search results, Dienst would normally have the local User Interface service use the `Describe` verb to peer into the contents of the documents directory (including the metadata file), and Dienst itself would control how the contents are presented to the user. In NCSTRL+, the final step of examining the directories structure is skipped, and the directory's `index.cgi` file is invoked. The default method for an `index.cgi` is generally the display method, so the user should notice little difference. However, at that point the bucket, not Dienst, determines what the user sees.

RELATED WORK

There has been a lot of research in the area of redefining the concept of "document." In this section we examine some of these projects and technologies that are similar to buckets.

Digital Objects

Buckets are most similar to the digital objects first described in the Kahn/Wilensky Framework [8], and its derivatives such as the Warwick Framework containers [11] and the more recent Flexible and Extensible Digital Object Repository Architecture (FEDORA) [4]. In FEDORA, *DigitalObjects* are containers, which aggregate one or more *DataStreams*. DataStreams are accessed through an *Interface*, and an Interface may in turn be protected by an *Enforcer*. Table 3 is a continuation of Table 1 from [4], with the fourth column added to show the bucket equivalents of concepts from the Kahn/Wilensky Framework, the Warwick Framework, and FEDORA.

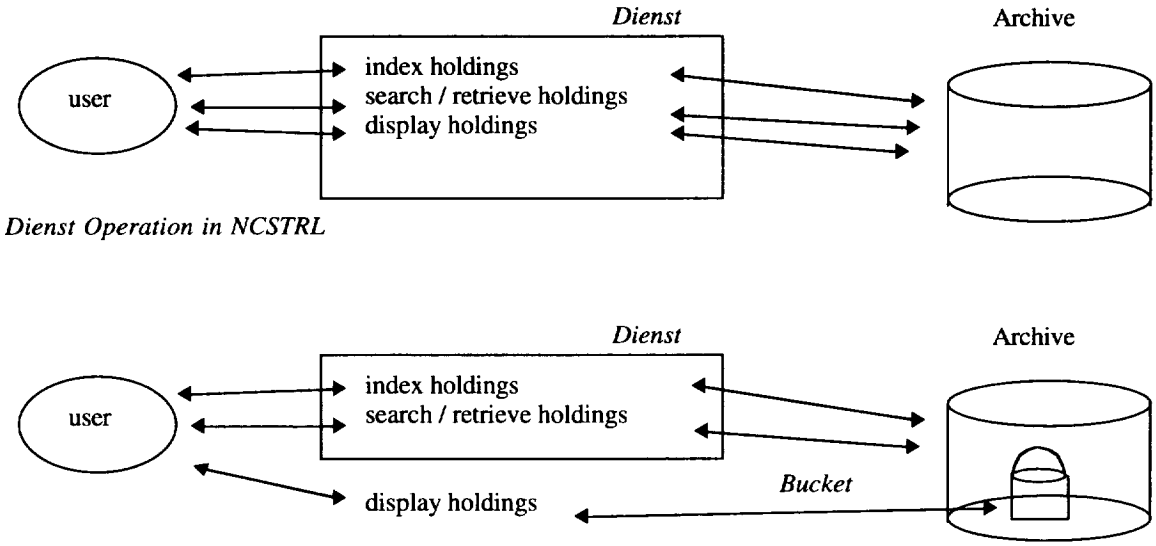


Figure 4: Buckets, Not Dienst, Control Display in NCSTRL+

Table 3: Bucket Concepts Added to Table 1 from [4]

The Kahn/Wilensky Framework Concepts...	map to the FEDORA classes...	which are implemented using the Extended Warwick Framework concepts...	and the Bucket equivalent is...
Data and Metadata	DataStream	Package	Packages, Elements
Digital Object	DigitalObject	Container	Bucket
Dissemination	Interface	Distributed Active Relationship	get_package, get_element
Terms and Conditions	Enforcer	Distributed Active Relationship	tc
Repository	Repository	Container	Repository, or standalone (URL)

FEDORA has not been completely implemented at this point, and it is unknown what repository or digital library protocol limitations will be present. Also, it is unknown if FEDORA plans to allow DigitalObjects to be intelligent agents, similar to the Bucket Matching System described below.

Multivalent Documents

Multivalent documents [17] appear similar to buckets at first glance. However, the focus of multivalent documents is more on expressing and managing the relationships of differing "semantic layers" of a document, including language translations, derived metadata, annotations, etc. There is not an explicit focus on the aggregation of several existing data types into a single container.

Open Doc and OLE

OpenDoc [13] and OLE [1] are two similar technologies that provide the capability for compound documents. Both technologies can be summarized as viewing the document as a loose confederation of different embedded data types. The focus on embedded documents is less applicable to our digital library requirements than that of a generic container mechanism with separate facilities for document storage and intelligence. OpenDoc and OLE documents are more suitable to be elements within a bucket, rather than a possible bucket implementation.

Digibox

The DigiBox [18] technology is a container construct designed for electronic commerce. The goal of DigiBox is "to permit proprietors of digital information to have the same type and degree of control present in the paper world [18]. As such, the focus of the DigiBox capabilities are heavily oriented toward cryptographic integrity of the contents, and not so much on the less stringent demands of the current average digital library. There also appear to be no hooks to make a DigiBox an intelligent agent. DigiBox is a commercial endeavor and is thus less suitable for the for our NCSTRL+ prototype.

CURRENT AND FUTURE WORK

We are using the author tool to populate NCSTRL+ to gain insight on how to improve its operation. We are starting with buckets authored at Old Dominion University and NASA Langley Research Center and are choosing the initial entries to be "full" buckets, with special emphasis on buckets relating to NSF projects for ODU and for

windtunnel and other experimental data for NASA. Until NCSTRL+ becomes a full production system, we are primarily seeking rich functionality buckets that contain diverse sets of packages.

Alternate Implementations

We are planning to also implement buckets using Lotus Domino, a Web server integrated with a Lotus Notes database server, in addition to the current CGI and Perl implementation. The bucket API as defined in Tables 1 & 2 will remain unchanged. In experimenting with Domino, we also plan to investigate implementing NCSTRL+ components without using Dienst. We plan to evolve NCSTRL+ to support a generalized publishing and searching model that can be implemented using Dienst or other DL protocols.

Bucket Matching System

The premise of the Bucket Matching System (BMS) is that the archived objects (buckets) should handle as many tasks as possible, not humans. Toward this end, we are designing the BMS as a communication mechanism for buckets to exchange information among themselves. The "tuple-space" communication of the Linda programming language [3] is the model for BMS.

The following example illustrates a usage of the BMS. Consider a technical report published by the CS department which is also submitted to a conference. The report appears under the server maintained by the department and publishing authority which is: ncstrl.odu.cs. If the conference paper is accepted, it will eventually be published by the conference sponsor, say the ACM. The publishing authority would be ncstrl.acm. Although the conference paper will surely appear in a modified format, the tech report and the conference paper are clearly related, despite being separated by publishing authority, date of publication, and revisions. Two separate but related objects now exist, and are likely to continue to exist. How best to create the desired linkage between the objects? "ncstrl.acm" may have neither the resources nor the interest to spend the time searching out previous versions of a manuscript. "ncstrl.odu.cs" cannot link to the conference bucket at the creation time of the ODU bucket, since the conference bucket did not exist at the time. It is unrealistic to suggest that the relevant parties will go back to the ncstrl.odu.cs collection and create the linkage correctly after several months have passed.

The solution is to have both buckets publish their metadata, or some subset of it, in the BMS. When a match, or near match, is found, the buckets can either 1) automatically link to each other; or more likely 2) bring the possible linkage to the attention of a person, who will provide the final approval for the linkage. There are a number of ways that a "match" can be found, but most likely it will be similar metadata within some definable threshold (e.g., 90% similar). Other uses for the BMS could include:

Find similar works by different authors. The exact values would have to be determined by experimentation, but it is possible to envision a similarity ranking that is slightly lower being an indication of a similar work by different authors. For example, a similar work by a different author would be: 70% < similarity < 90%.

Arbitrary selective dissemination of information (SDI) services. When a user's profile is matched, a notification can be sent immediately or a digest sent at every defined time interval (i.e., weekly). This method can be used to track different versions of a report, not just inter-genre (technical report vs. conference paper) or inter-institution (the author moves to a different university) issues. If version 2.0 of a bucket comes out, it can "find" all previous versions, and the appropriate actions can be taken (i.e., create a fully connected graph between the buckets, delete previous buckets, etc.)

Metadata scrubbing. The issues of maintaining consistency and quality of metadata information is an increasingly important concern in digital libraries [6]. Part of the BMS coud also include a metadata scrubber that, based on rules and heuristics defined at the scrubber, could automatically make or suggest updates to metadata. For example, the scrubber could have all references to "Hampton Institute" indicate the name change to "Hampton University", or handle an author's name change (for example, if someone changes their name upon marriage), or correct errors that may have been introduced, etc.

The BMS could be implemented on multiple workstations, and would be primarily batch processing. Given that some of the operations would be computationally expensive, it can be done with loose time guarantees, perhaps even done on stolen idle cycles (from "hallway clusters" of workstations).

CONCLUSIONS

Buckets provide a mechanism for logically grouping the various semantic data objects (manuscript, software, datasets, etc.) and the various syntactic representations (PostScript, PDF, etc.). The ability to keep all the data objects together with their relationships intact relieves the user from having to reintegrate the original information tuple from many separate archives. Buckets also provide a more convenient method for describing the output of research projects, and provide a finer granularity for controlling terms and conditions within an archive. The aggregative aspects of buckets have already been implemented. The tools to make buckets easy to use and manage are being created. The Bucket Matching System will allow buckets to be intelligent agents, and allow inter-bucket communication as well as communication and action with arbitrary network resources.

REFERENCES

1. K. Brockschmidt, "Inside OLE 2", Microsoft Press, Redmond, WA, 1995.

2. S. Browne, J. Dongarra, E. Grosse, S. Green, K. Moore, T. Rowan, & R. Wade, "Netlib Services and Resources," University of Tennesse Technical Report UT-CS-93-222, 1993.

3. N. Carriero & D. Gelernter, "Linda in Context," *Communications of the ACM*, Vol. 32, No. 4, 1989, pp. 444-458.

4. R. Daniel & C. Lagoze, "Distributed Active Relationships in the Warwick Framework," *Proceedings of the 2nd IEEE Metadata Conference*, Silver Spring, MD, September 16-17, 1997.

5. J. R. Davis, D. B. Krafft, & C. Lagoze, "Dienst: Building a Production Technical Report Server," *Advances in Digital Libraries*, Springer-Verlag, 1995, pp. 211-222.

6. J. C. French, A. Powell & E. Schulman, "Automating the Construction of Authority Files in Digital Libraries: A Case Study," University of Virginia Technical Report CS-97-02, January 1997.

7. R. Kahn, "The Handle System Version 3.0: An Overview," http://www.handle.net/docs/overview.html

8. R. Kahn & R. Wilensky, "A Framework for Distributed Digital Object Services," cnri.dlib/tn95-01, May, 1995. http://www.cnri.reston.va.us/home/cstr/arch/k-w.html

9. C. Lagoze, E. Shaw, J. R. Davis, & D. B. Krafft, "Dienst: Implementation Reference Manual," Cornell Computer Science Technical Report TR95-1514, 1995.

10. C. Lagoze & D. Ely, "Implementation Issues in an Open Architectural Framework for Digital Object Services," Cornell University Computer Science Technical Report, TR95-1540, June 1995.

11. C. Lagoze, C. A. Lynch & R. Daniel, "The Warwick Framework: A Container Architecture for Aggregating Sets of Metadata," Cornell Computer Science Technical Report TR96-1593, July 1996.

12. R. Lasher, & D. Cohen, "A Format for Bibliographic Records," Internet RFC-1807, June 1995.

13. C. Nelson, "OpenDoc and its Architecture," *The X Resource*, Vol. 1, No. 13, 1995, pp. 107-126.

14. M. L. Nelson, G. L. Gottlich, D. J. Bianco, S. S. Paulson, R. L. Binkley, Y. D. Kellogg, C. J. Beaumont, R. B. Schmunk, M. J. Kurtz & A. Accomazzi, "The NASA Technical Report Server," *Internet Research: Electronic Networking Applications and Policy*, Vol. 5, No. 2, 1995, pp. 25-36.

15. M. L. Nelson, "Building Multi-Discipline, Multi-Format Digital Libraries Using Clusters and Dienst," Old Domion University Computer Science MS Thesis, August 1997. (Also Available as NASA TM-112876)

16. M. L. Nelson, K. Maly & S. N. T. Shen, "Buckets, Clusters and Dienst," Old Dominion University Computer Science Technical Report 97-30, July 1997. (Also available as NASA TM-112877)

17. T. A. Phelps & R. Wilensky, "Multivalent Documents: Inducing Structure and Behaviors in Online Digital Documents," *Proceedings of the 29th Hawaii International Conference on System Sciences*, Maui, HI, January 3-6, 1996.

18. O. Sibert, D. Bernstein & D. Van Wie, "DigiBox: A Self-Protecting Container for Information Commerce," *Proceedings of the 1st USENIX Workshop on Electronic Commerce*, New York, NY, July, 1995.

19. D. G. Roper, M. K. McCaskill, S. D. Holland, J. L. Walsh, M. L. Nelson, S. L. Adkins, M. Y. Ambur & B. A. Campbell, "A Strategy for Electronic Dissemination of NASA Langley Technical Publications," NASA TM-109172, December 1994.

20. J. Sobieszczanski-Sobieski, "A Proposal: How to Improve NASA-Developed Computer Programs," NASA CP-10159, 1994, pp. 58-61.

REPORT DOCUMENTATION PAGE

Form Approved
OMB No. 0704-0188

1. AGENCY USE ONLY (*Leave blank*)	2. REPORT DATE	3. REPORT TYPE AND DATES COVERED
	May 1998	Technical Memorandum

4. TITLE AND SUBTITLE
Buckets: Aggregative, Intelligent Agents for Publishing

5. FUNDING NUMBERS

6. AUTHOR(S)
Michael L. Nelson, Kurt Maly, Stewart N. T. Shen, and Mohammad Zubair

7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES)
NASA Langley Research Center
Hampton, VA 23681-2199

8. PERFORMING ORGANIZATION REPORT NUMBER
L-17753

9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES)
National Aeronautics and Space Administration
Washington, DC 20546-0001

10. SPONSORING/MONITORING AGENCY REPORT NUMBER
NASA/TM-1998-208419

11. SUPPLEMENTARY NOTES
Also available as Old Dominion University Computer Science Technical Report 97-41.

12a. DISTRIBUTION/AVAILABILITY STATEMENT
Unclassified-Unlimited
Subject Category 82 Distribution: Standard
Availability: NASA CASI (301) 621-0390

12b. DISTRIBUTION CODE

13. ABSTRACT (*Maximum 200 words*)
Buckets are an aggregative, intelligent construct for publishing in digital libraries. The goal of research projects is to produce information. This information is often instantiated in several forms, differentiated by semantic types (report, software, video, datasets, etc.). A given semantic type can be further differentiated by syntactic representations as well (PostScript version, PDF version, Word version, etc.). Although the information was created together and subtle relationships can exist between them, different semantic instantiations are generally segregated along currently obsolete media boundaries. Reports are placed in report archives, software might go into a software archive, but most of the data and supporting materials are likely to be kept in informal personal archives or discarded altogether. Buckets provide an archive-independent container construct in which all related semantic and syntactic data types and objects can be logically grouped together, archived, and manipulated as a single object. Furthermore, buckets are active archival objects and can communicate with each other, people, or arbitrary network services.

14. SUBJECT TERMS
Digital library architectures, agents, archiving, multi-format, bucket, container, package

15. NUMBER OF PAGES
12

16. PRICE CODE
A03

17. SECURITY CLASSIFICATION OF REPORT	18. SECURITY CLASSIFICATION OF THIS PAGE	19. SECURITY CLASSIFICATION OF ABSTRACT	20. LIMITATION OF ABSTRACT
Unclassified	Unclassified	Unclassified	