Adaptive Networks of Smart Objects

Johan Bollen and Michael L. Nelson
Computer Science Department
Old Dominion University
Norfolk, VA
{jbollen,mln}@cs.odu.edu

Abstract

We propose the implementation of a distributed system for DL storage and retrieval which relies on two earlier developed technologies: buckets and document linking based on user retrieval patterns. Buckets are expanded with a mechanism to keep track of weighted bucket relations. Each bucket keeps track of its connections to other related buckets. A mechanism is put in place which allows buckets to change their connections to other buckets according to the registration of bucket retrieval patterns issued by users. The collection of buckets thereby dynamically updates its link structure according to user retrieval patterns.

1 Introduction

Traditional Digital Libraries (DLs) to a large extent store and distribute information in a centralized manner. Client-server architectures dominate present DL implementations due to their apparent simplicity in issues such as management of resources, DL-user interactions and the availability of unsophisticated means to control ownership and copyright issues.

However, the client-server architecture places strong limits on the scale of operation by which DL can store and manage information. Distributed architectures allow storage and management to be distributed over a large number of independently operating systems. The following sections discuss two technologies which we will employ to implement a distributed DL architecture which relies on autonomously operating information objects and adaptive information linking.

1.1 Smart Objects: Buckets

Buckets are an aggregative, intelligent construct for publishing in DLs. Buckets allow the decoupling of information content from information storage and retrieval. Buckets exist within the Smart Objects and Dumb Archives model for DLs [9] 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. A bucket is a self-contained storage unit that has data and metadata, as well as the methods for accessing both.

Buckets were developed at NASA Langley Research Center and Old Dominion University. They were first described in [11]. Based on previous NASA DL experience, buckets have a two-level structure:

1. buckets contain 0 or more packages
2. packages contain 0 or more elements

Actual data objects are stored as elements, and elements are grouped together in packages within a bucket. A two-level architecture was considered sufficient for most applications, and thus employed as a simplifying assumption during bucket implementation. Current work involves implementing 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.

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. As an example, packages can represent semantic types (manuscript, software, test data, etc.) and elements can represent syntactic representations of the packages (a .ps version, .pdf version, .dvi version, etc.). Other bucket models using elements and packages are possible.

Buckets provide user access to their content and mechanisms of content management by a set of methods. Bucket methods can be activated by user HTTP request enclosing a specific method. Each bucket can be thought of as an object which contains a set of data objects and methods to access and manage these objects.

As an example of how methods in an bucket are invoked, consider the bucket:

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

When no bucket method is specified, the "display" method is assumed. Therefore the mentioned URL is equivalent to:

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

The above mentioned URLs will induce the bucket to return an overview of the elements it contains. These elements themselves can again be URLs containing requests for bucket methods. A specific bucket method allows a bucket to redirect a request for its content to another object, which could be another bucket. For example, the request:

`http://www.cs.odu.edu/~mln/naca-tn-2509/?method=display&redirect=http://naca.larc.nasa.gov/reports/1951/naca-tn-2509/`

would request the odu.edu bucket to redirect to the nasa.gov bucket. All requests for external resources are first routed through the bucket that contains the link. The full bucket API is discussed in [10].

1.2 Distributed collections of smart objects

Bucket functionality introduces both storage overhead and management complexity overhead. Storage issues are only significant in very large numbers of buckets ($>$100,000) and can be addressed by solutions which rely on distributing storage requirement of large, connected collections of individual bucket collection.

Buckets are well suited for distributed applications because they can aggregate heterogeneous content and remain functional in low-fidelity environments. Since they are self-contained, independent and mobile, they should be resilient to changing server environments. In addition, buckets can be adapted to a variety of data types and data formats.

However, complexity management remains a serious issue. Since buckets are designed to be independent, they contain much redundant information, introducing the possibility of the buckets becoming out of synchronization with each other. Tools and models for the management of large numbers of buckets exist, but remain largely untested.

2 Adaptive linking algorithms for document collections

An adaptive hypertext mechanism which relies on temporal patterns of user retrievals to dynamically link hypertext documents is implemented in [2]. The mechanism assumes that users by repeatedly retrieving certain pairs of documents thereby indicate a degree of relation between the retrieved documents. By monitoring user retrieval sequences, collections of documents can thereby be dynamically hyperlinked into adaptive networks which respond to changing user preferences.

2.1 Hebbian learning for document collections

Hebb's law of learning [6] postulated that when two neurons fire in close temporal proximity the strength of their connection, or rather the ability of the synapse connecting the two neurons to transmit potentials, would be increased. In other words, when neuron A and B often fire in close temporal proximity the weight of their connection increases. This principle has found countless applications in machine learning and the simulation of human learning [13][14].

A similar law of learning is applied to adaptive hypertext linking in a series of experiments which generated well structured hypertext networks from user retrieval patterns [2]. Hypertext networks were initialized by generating hyperlinks for every pair of documents in a collection. Each hyperlink was assigned a small, random weight value which was held to indicate the value or relevancy of that hyperlink. User would be presented a list of hyperlinks

from each page, ranked according to their weight. Since all hyperlink weights were initialized to small, random values, from the user's perspective, the hypertext network had an initially random structure. These networks were made public and large groups of people were invited to browse the network.

When users retrieve documents from hypertext networks they do so in a sequence often determined by their interest in a particular subject. Therefore, when a hyperlink between two documents in the mentioned experimental hypertext networks was traversed by a user, the weight of this hyperlink was increased to indicate its higher validity by a set of learning rules labeled "Frequency", "Transitivity" and "Symmetry". Similar to Hebbian learning, these learning rules increased the weights of the connections between two documents as they were retrieved within temporal proximity.

The actions of the mentioned learning rules can be represented as follows. Assume we represent a collection of n documents by the set $V = \{v_1, v_2, \cdots, v_n\}$. The weight of the relation between any two documents is denoted $W(v_i, v_j)$. Each learning rules increases a different set of weights by an amount associated with the specific learning rules. The "Frequency", "Symmetry" and "Transitivity" learning rule increase link weights by amounts respectively denoted as r_f, r_s and r_t. Then, for any triplet of retrieved documents $\{v_i, v_j, v_k\}$, the learning rules change link weights as follows:

Frequency: $W(v_i, v_j) + r_f$ and $W(v_j, v_k) + r_f$

Symmetry: $W(v_i, v_j) + r_s$ and $W(v_j, v_k) + r_s$

Transitivity: $W(v_i, v_k) + r_t$

According to this definition, the "Frequency" learning rule increases the weight of connections that users have actually traversed, while the "Symmetry" and "Transitivity" learning rules increase the weight of connections that have not actually been traversed but may be relevant according to the user's retrieval path.

Figure 2 shows an example of how a set of these three learning rules shapes a network of nodes. The amounts r_f, r_s and r_t have respectively been set to 1, 0.3 and 0.5. The user traverses a path leading through the nodes "house", "table", "stool" and "bar". The nodes are connected by connections which have already acquired certain link weights. After the increases in link weights subsequent to the user's retrievals, the network has acquired a large number of new connections.

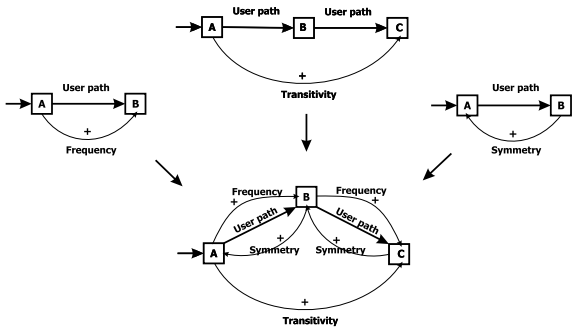

Figure 1. Graphical representation of learning rule reinforcements of document connections.

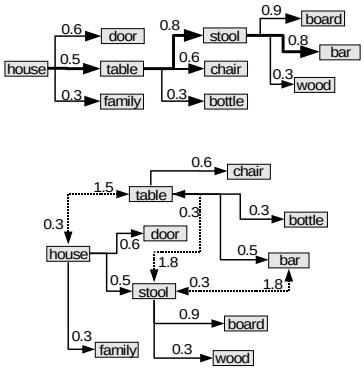

Figure 2. Example of how user retrievals of the nodes "house", "table", "stool" and "bar" leads to the generation of a large number of new connections.

When large groups of users browse such networks, their overlapping patterns of hyperlink traversal will gradually change the hypertext network's structure. A similar system is proposed by [4].

The specific experiments conducted by [2] demonstrated how such hypertext networks could self-organize from an initially random state to a well-structured state by means of user retrieval alone. Simulations have demonstrated how the generated networks provide a reliable and valid measurement of the views and perspectives of a community of users [1].

2.2 Distributed adaptive document linking

The experiments conducted by [2] show how a collection of documents can be organized by the mere adaptation

of document link weights from patterns of document retrievals. Recent applications have demonstrated the power of this paradigm to WWW [3] and DL applications [12], where document collections can be organized according to user retrieval sequences as they are reconstructed from DL server logs.

However, to the present all experiments and applications have relied on centralized mechanisms to store the document collection in question, analyze user retrieval requests and store document link weights. The applicability of such algorithms is thereby strongly limited to relatively small document collections and faces major scalability issues when applied to large document collections (DL and WWW).

3 Adaptive bucket linking

We propose to merge the bucket and adaptive hypertext approaches in a paradigm where buckets maintain a list of weighted links to other buckets which is updated according to user link retrieval patterns. Such an arrangement would enable the efficient linking of large sets of buckets into networks shaped by user retrievals, much like the mentioned dynamic hypertext networks, as well as advanced Peer To Peer (P2P) mechanisms for bucket maintenance and retrieval.

An implementation of the mentioned learning rules, without recourse to a central management mechanism, assumes each individually retrieved bucket has sufficient information to update its list of links. For example, the "Frequency" learning rule requires a bucket to update its link to the bucket a user retrieves next. The "symmetry" learning rule requires a bucket to increase the weight of the link to the bucket that was retrieved before itself. The transitivity learning rule requires a bucket to reinforce the connection from itself to a bucket retrieved as a second successor. For example, bucket b_1, b_2 and b_3 have been retrieved. The "Transitivity" learning rule stipulates an increase of $W(b_1, b_3)$. Since $b1$ maintains a list of connections to other buckets, it must therefore be informed of b_3's retrieval. The following sections outline how this functionality can be implemented with few changes to the existing bcuuetk specification.

3.1 Bucket link maintenance

Buckets already contain most of the infrastructure required to meet implement the learning rules. To accommodate the changes, we add a new default package to store the weights, and we modify the display method.

1. Package/Element addition.
   (a) A default package named "_recommendations.pkg" is created. Within that package, an element named "hebbian" is created. The format for this file is:
   package_name::element_name::weight
   (b) While this data structure will not be directly displayed to the users, it can be retrieved with the display method:
   `http://foo.org/bucket/?method=display&pkg_name=_recommendations.pkg&element_name=hebbian`
2. Display method modification.
   (a) Change the method to reorder the display of elements according to the weights that appear in _recommendations.pkg/hebbian.
   (b) Apply "Frequency", "Symmetry" and "Transitivity" learning rules by the construction of specific redirection URLs and the addition of a "referer" argument.

3.2 Example

To understand how the mentioned learning rules would be applied using the mentioned bucket methods, consider 4 buckets (b1, b2, b3, b4) which are linked together as a doubly linked list. The user somehow discovers b1 (through a DL, Google search, or enters the URL directly) and begins browsing there. In addition to whatever internal contents b1 contains (PDFs, MP3s, tar files, etc.), it contains a link to b2. However, this link is dynamically written as:
`http://foo.org/b1/?method=display&redirect=http://foo.org/b2/?method=display&referer=http://foo.org/b1/`
When the user selects this link, b1 sees the outgoing link and adjusts its weights to b2 accordingly (Frequency). When the client is redirected to b2, b2 sees that client came from b1 from the "referer" argument and adjusts its weight for b2 (Symmetry). In this case, the referer argument has the same value as the referer HTTP header, but this will not always be the case as will be shown below. Since this is the beginning of the user's browsing, there is no transitivity.

Now after interacting with b2, the user is now ready to move to b3. b2 has constructed the link to b3 as:
`http://foo.org/b2/?method=display&referer=http://foo.org/b2/&redirect=http://foo.org/b1/?method=display%26redirect=http://foo.org/b3/%26referer=http://foo.org/b2/`

This admittedly ungainly URL accomplishes all three learning rules. b2 registers an outbound link that is ultimately for b3 and adjusts its b3 weight accordingly (Frequency). But b2 links to b3 by way of b1, which allows b1 to update its weight for b3 (Transitivity). In doing this, b2 redirects to b1, but the URL that is invoked on b1 is itself a redirection to b3. To preserve the final URL, the "&" is encoded as "%26". Finally, the client arrives at b3 which checks the referer argument to know to update the weight for b2 (Symmetry). This is the case where the referer argument (b2) is different from the referer HTTP header field (b1). The link from b3 to b4 would be constructed in a similar fashion as the link from b2 to b3.

To create links in buckets where none currently exist, the bucket API is used. In the previous example, there is no link from b1 to b3 until the transitivity rule is invoked. To create this link, before b1 complies with the redirection of the client to b3, b1 first contacts b3 with:
`http://foo.org/b3/?method=metadata`
and extracts the title from the metadata. b1 then creates a link to b3 and uses b3's title as the label for this link. The presence of the referer argument signals that links might need to be made; the absence of a referer argument indicates normal bucket display operation.

3.3 Overview

The presented framework will create an infrastructure in which a network of buckets will change its links to other buckets dynamically according to user retrieval patterns. This entails an efficient manner to create large document networks from no more than registration of user retrieval patterns. Compared to other methods for information linking this method has the following advantages:

1. Format and Language independent: given that users in sufficient numbers retrieve buckets in a sequence determined by a specific interest, the system will gradually link related buckets independent of their format or language.

2. Text independent: the system will operate on buckets containing visual or auditory information as efficiently as it would on text documents. There is little or no requirement for meta-data.

3. Implicit: User interest are derived from their retrieval behavior, not from explicit statements of approval or interest. This may be a less biased manner to measure user interests.

4. Computationally undemanding: no centralized computation of document links is required. All computation is performed on the bucket level and consists of no more than the application of three simple learning rules.

The method furthermore makes no assumptions about long term patterns in user interests. It operates on highly local features or regularities of user retrieval behavior.

4 Issues and future research

The presented system has not yet been implemented, yet apart from its implementation, we foresee the following issues for future research.

First, any implemented system would have to be bootstrapped by the generation of an initial network structure. Given that the adaptive buckets system changes its structure as buckets are being retrieved by a community of users, an initial network structure needs to be provided which allow the first users to meaningfully traverse bucket to bucket links. Nelson (2000) [10] used a vector space model and cosine measures of document similarity to link a collection of buckets. This approach, however, requires the availability of meta-data or text content. Yan(1996) [16] describe a method to generate user clusters and dynamically generate according hyperlinks. This method does not require text content and can complement the proposed system for bucket link generation. A number of collaborative filtering approaches may be introduced to deal with this problem as well [15][8]. In addition, existing citation data may be used to generate an initial set of bucket links. Network adaptation subsequent to initialization may be taken as an indication of the quality of initial network structure and may be used to evaluate link generation mechanisms.

A second issue concerns the addition of new bucket collections to existing ones which have previously established a link structure. Given the propensity of user to preferably select high ranking links, new links may only with great difficulty establish sufficient traversals to be integrated into the network's existing structure. Therefore, a process of link decay may be put in place which continuously reduces the weight of links by a certain percentage of their weight value. Newly established links may thereby have experienced less decay than existing connections and may be at a lesser disadvantage.

Third, the present internal semantic structure of buckets is strongly determined by designer intuitions and existing content taxonomies. For example, in the DL implementation discussed by Nelson (1997) [11], bucket have a 2 tier structure, determined by the semantic organization of the collection for which the set of buckets was organized. The adaptive mechanisms proposed in

this article may equally be applied to the internal organization of individual buckets which may come to differ strongly from its original configuration. The implications of such a strategy need to be explored in future applications.

Fourth, Bollen et al. (2001) [1] show how the structure of adaptive networks after sufficient user retrievals will gradually converge to a valid and reliable representation of the collectively held views on document relations held by a given user community. Therefore the generated networks can be used a measurement of user preferences, and their structure as a proxy to the composition of a specific user community. A graph-theoretical analysis of the generated network may be useful to yield information about user preferences and the specific nature of a user community. Furthermore, impact measures can be derived from the generated networks which may reveal a document's or collection of documents impact among users [7].

5 Conclusion

We have proposed a system for the automated linking of information using a collection of smart objects, labeled buckets, using a set of simple learning rules which change link weights from user retrieval patterns. The bucket networks are expected to gradually change structure as users retrieves one bucket after another via a list of recommended buckets.

We believe the combination of the adaptive hypertext learning rules and smart objects solves a number of long standing problems in DL applications and the implementation of adaptive interfaces. It allows the storage of large number of buckets to be distributed across a large number of relatively small clients. By the mere occurence of large numbers of user retrievals the networks self-organize into a structure representative of its user community. Since bucket links are determined by user retrievals they are defined independently of bucket format, content or meta-data. The proposed system can therefore be applied to collections which focus on audio-visual or multi-lingual content.

Future research will focus on the development of initial prototypes, the parametrization of network development, e.g. learning rule reinforcement values, user satisfaction, link weights decay of time, the analysis of generated networks, and information retrieval mechanisms which rely on network structure such as Spreading Activation [5].

References

[1] J. Bollen. *A cognitive model of adaptive web design and navigation.* PhD thesis, Vrije Universiteit Brussel, Brussels, Belgium, 2001.

[2] J. Bollen and F. Heylighen. A system to restructure hypertext networks into valid user models. *The New Review of Hypermedia and Multimedia*, 4:189–213, 1998.

[3] J. Bollen, H. van de Sompel, and L. M. Rocha. Mining associative relations from website logs and their application to context-dependent retrieval using spreading activation. In *Proceedings of the Workshop on Organizing Webspaces (ACM-DL99)*, Berkeley, California, 1999.

[4] P. K. Chan. Constructing web user profiles: a non-invasive learning approach. In B. Masand and M. Spiliopoulou, editors, *Web Usage Analysis and User Profiling - LNAI 1836*, San Diego, CA, August 1999. Springer.

[5] F. Crestani and P. L. Lee. Searching the web by constrained spreading activation. *Information Processing and Management*, 36(4):585–605, 2000.

[6] D. O. Hebb. *The Organization of Behavior*. John Wiley, New York, 1949.

[7] J. Kleinberg, S. Kumar, P. Raghavan, S. Rajagopalan, and A. Tomkins. The web as a graph: Measurements, models and methods. In T. Asano, H. Imai, D. T. Lee, S.-I. Nakano, and T. Tokuyama, editors, *Computing and Combinatorics, 5th Annual International Conference, COCOON'99*, volume 1627 of *Lecture Notes in Computer Science*, pages 1–17, Tokyo, Japan, July 1999. Springer.

[8] J. A. Konstan, B. N. Miller, D. Maltz, J. L. Herlocker, L. R. Gordon, and J. Riedl. Grouplens: applying collaborative filtering to usenet news. *Communications of the ACM*, 40(3):77–87, 1997.

[9] K. Maly, M. L. Nelson, and M. Zubair. Smart objects, dumb archives - a user-centric, layered digital library framework. *D-Lib Magazine*, 5, 1999.

[10] M. L. Nelson. *Buckets: Smart Objects for Digital Libraries*. PhD thesis, Computer Science Department, Old Dominion University, Norfolk, VA, 2000.

[11] M. L. Nelson, K. Maly, and S. N. T. Shen. Buckets, clusters, and dienst. Technical Report TM-112877, NASA, Hampton, VA, 1997.

[12] L. M. Rocha and J. Bollen. Biologically motivated distributed designs for adaptive knowledge management. In I.Cohen and L. Segel, editors, *Design Principles for the Immune System and other Distributed Autonomous Systems*. Oxford University Press, Oxford, In Press, 2000.

[13] F. Rosenblatt. *Principles of neurodynamics; perceptrons and the theory of brain mechanisms*. Spartan Books, Washington, 1962.

[14] D. E. Rumelhart and J. McClelland. *Parallel Distributed Processing, vol I*. MIT press, Cambridge, 1986.

[15] U. Shardanand and P. Maes. Social information filtering: algorithms for automating "word of mouth". In *ACM Conference Proceedings on Human Factors in Computing Systems*, pages 210–217, Denver, CO, May 7-11 1995.

[16] T. W. Yan, M. Jacobsen, H. Garcia-Molina, and U. Dayal. From user access patterns to dynamic hypertext linking. *Computer Networkds and ISDN Systems*, 28:1007–1014, 1996.