The "InfoHarness" Information Integration Platform

Leon Shklar, Satish Thatte, Howard Marcus, and Amit Sheth[1]
Bell Communications Research,
444 Hoes Lane,
Piscataway, NJ 08854

The "InfoHarness" information integration platform, tools, and services being developed at Bellcore are aimed at providing integrated and rapid access to huge amounts of heterogeneous information independent of the type, representation, and location of information. InfoHarness provides advanced search and browsing capabilities without imposing the burden of restructuring, reformatting or relocating information on information suppliers or creators. This is achieved through object-oriented encapsulation of information and the associated meta-information (e.g., type, location, access rights, owner, creation date, etc.). The meta-information extraction methods ensure rapid and largely automatic creation of information repositories. A gateway that supports access to InfoHarness repositories from Mosaic and other HyperText Transfer Protocol (HTTP) compliant browsers is currently available. An HTTP compliant InfoHarness server is under construction.

1.0 Introduction

Enormous amounts of heterogeneous information have been accumulated within corporations, government organizations and universities. Such information continues to grow at ever-increasing rate. It ranges from software artifacts to engineering and financial databases, and comes in different types (e.g., source code, e-mail messages, bitmaps) and representations (e.g., plain text, binary). This information has to be accessed through a variety of vendor tools and locally developed applications. It is becoming increasingly easier to create new information due to the proliferation of sophisticated shrink-wrapped commercial authoring and office automation software. It is also becoming easier and cheaper to provide access to this information, due to the increasingly pervasive and more robust data communication technology. However, the knowledge about the existence and location of information, as well as the means of its retrieval, have become so bewildering and confusing to many users as to give rise to the widely lamented phenomenon of *write-only* databases.

Over the last few years there have been numerous attempts to address this problem by building information repositories that depend on relocating and reformatting the original information [7]. Such approaches provide a nice and uniform way to access information but require the design and maintenance of a large number of ever-changing format translators. The initial conversion of information often requires substantial human and computing resources. Maintaining the repositories presents the additional dilemma of either creating new and updating existing information in the uniform format, or continuously managing changing data in different formats. The latter problem may be partially remedied through the latest efforts in the uniform representation of heterogeneous documents [9], but this does not help with arbitrarily formatted data.

---

1. Now with the Computer Science Department, University of Georgia, 415 Graduate Studies Research Center, Athens, GA 30602-7404

Our main objective in constructing InfoHarness is to provide rapid access to huge amounts of heterogeneous information without any relocation, restructuring, or reformatting of data. InfoHarness is aimed at facilitating individual and enterprise productivity by "harnessing" existing and new information assets. Many researchers have investigated the use of meta-data to support runtime access to the original information [2,3,10,12]. Others [8,12] have investigated the use of data mining for the automatic extraction of meta-data. Our own work develops and synthesizes some of the ideas contained in these efforts to provide advanced search and browsing capabilities without imposing constraints on information suppliers or creators.

InfoHarness has been designed with a completely open, extensible, and modular architecture. It provides support for easy incorporation of new types of data, as well as new third-party indexing and browsing technologies. The InfoHarness prototype is now operational and is being trialed at Bellcore for accessing heterogeneous information about software artifacts. It supports the largely automatic generation of InfoHarness repositories, and provides access to the original information from Mosaic and other World-Wide Web (WWW) browsers through an HTTP gateway.

In Section 2 we present a high-level overview of the InfoHarness architecture, concentrating on object representation and the automatic generation of InfoHarness Repositories. Section 3 describes our experience of applying InfoHarness to building software repositories. The paper is concluded with a brief outline of our future research and development plans.

2.0 System Architecture

The InfoHarness system architecture provides a platform for integrating information in a distributed environment by encapsulating existing and new information in objects, without converting, restructuring or reformatting the information. Through this object-oriented encapsulation, the system provides an integrated view and access to diverse and heterogeneous information. The system supports and provides tools for accessing, retrieving, browsing and administering the information encapsulated in the InfoHarness repositories.

Section 2.1 provides an overview of the architecture. Section 2.2 focuses on the use of meta-information to represent the structure and organization of the original information. Section 2.3 describes the automatic generation of meta-information to create InfoHarness repositories.

2.1 Architecture Overview

As shown in Figure 1, the main components of the current implementation of the InfoHarness architecture are:

1. The InfoHarness Server that uses meta-data to traverse, search and retrieve the original information.

2. The HTTP Gateway that is used to pass requests from HTTP clients to the InfoHarness server, and the responses back to the clients.
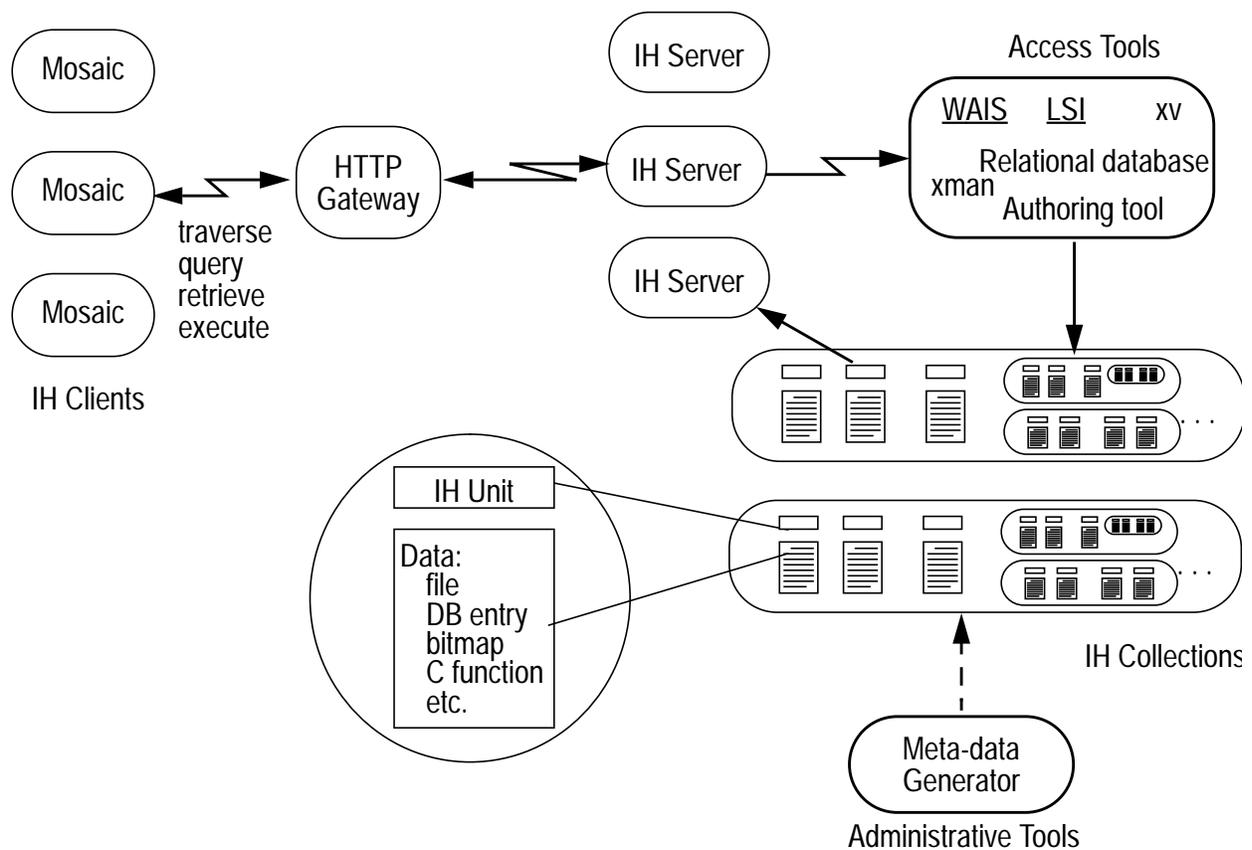
Figure 1. InfoHarness Architecture

3. The Repository Generator that is used for off-line automatic generation of meta-data that represents the desirable view on the structure and organization of the original information. This meta-data is used at run-time by the InfoHarness server to search and retrieve information.

4. Independent tools for accessing and displaying information (e.g., xv, xman, etc.), and indexing information (e.g., Wide-Area Information Server (WAIS), Bellcore Latent Semantic Indexing (LSI)).

At run-time, Mosaic users may issue query, traversal, or retrieval requests that are passed on to the Gateway, which then performs the following operations:

1. Parses the request, and reads input information when the request is associated with an HTML form.

2. Establishes a socket connection with the InfoHarness Server, generates and sends out a request, and waits for a response.

3. Parses the response, converts it to a combination of HTML forms and hyperlinks, adds the HTTP header, and passes the transformed response to the Mosaic browser.

The InfoHarness server processes requests based on the meta-information. The same information may be treated differently depending on its type. For example, consider two alternative types,

both designed to represent man pages: *man_to_html* and *xman.* In the first case, the InfoHarness server will pass the man page location to a *man_to_html* converter and send the generated HTML back to the InfoHarness HTTP gateway. In the second case, the server will pass the man page location directly to the *xman* browser and notify the gateway program.

The Gateway does not perform any format conversions of the original information. It converts to HTML those parts of the response that either contain the portion of meta-data that is currently being searched or traversed by the server, or error messages and notifications.

The InfoHarness architecture is open, modular, extensible and scalable. New types of information can be easily added by defining and registering new object types. InfoHarness implements the abstract type hierarchy that is complete in the sense that any new data type, or a new indexing technology may be added by instantiating an existing abstract type. The methods that are associated with abstract types are general enough because each method is data-driven and may invoke any third-party software. The definitions of new types are also data-driven and are not part of the InfoHarness implementation.The integration platform can support arbitrary information access and management tools (e.g., browsers, indexing methods, access methods) developed by Bellcore, universities and third parties.

The autonomy of administrators, no centralized control, support for multiple clients and multiple servers in a large-scale distributed, heterogeneous environment all support the goals of scalability and operability in a large and geographically distributed environment.

2.2 Object Representation

A meta-data entity that is associated with the lowest level of granularity of information available to InfoHarness is called the *information unit* (IU). The IU may be associated with a file (e.g., a man page), a portion of a file (e.g., a C function or a database table), a set of files (e.g., a collection of related bitmaps), or any request for the retrieval of data from an external source (e.g., a database query).

An InfoHarness object (IHO) may be one of the following:

1. A single information unit.

2. A collection of InfoHarness objects (either indexed, or non-indexed).

3. A single information unit and a non-indexed collection of InfoHarness objects.

Each IHO has a unique object identifier that is recognized and maintained by the system.An IHO that encapsulates an IU contains information about the location of data, data retrieval method, and any parameters needed by the method to extract the relevant portion of information. For example, an IHO associated with a C function will contain the path information for the .c file, the name and location of the program that knows how to extract a function from a .c file, and the name of the function to be passed to this program as a parameter. In addition, each IHO may contain arbitrary number of attribute-value pairs (e.g., owner, last update, security information, decompression method).

Each IHO that encapsulates a collection of IHOs stores unique object identifiers of the members of the collection. We refer to these members as *children* of the IHO. An IHO that encapsulates both an IU and a collection is called a *composite object*. An example of the composite object is this paper's abstract combined with the collection of postscript and HTML versions of the full paper. IHOs that encapsulate indexed collections store information about the location of both the index and the query method. Any indexed collection may make use of its own data retrieval method that is not part of InfoHarness. As a result, an InfoHarness Repository may easily be created from existing heterogeneous index structures.

An InfoHarness Repository (IHR) is a set of IHOs that are not necessarily members of the same collection. An IHO may be a member of any number of collections. We refer to IHOs that are associated with these collections as *parents*. Each IHO that has one or more parents always contains unique object identifiers of its parent objects. An IHO that does not have any parent is unreachable from any other IHO and may only be accessed if it is used as the initial starting point (or *entry point*) in the IHR traversal.

2.3 Automatic Generation of InfoHarness Repositories

Each IHO in an IHR may have multiple children, as well as multiple parents. The physical data that is associated with information units is not part of the IHR. The generation of an IHR amounts to the creation of IHOs and their relationships, and indexing the information declared to belong to an indexed collection.

The IHR generation is supported by the InfoHarness administrative tools. The top-level tool, called the Repository Generator, accepts as inputs both the location and the desired representation of data, and outputs the set of IHOs. Defining a collection of the entry points of the multiple sets of IHOs is the most straightforward way to combine them into a single Repository.

Consider a simple example of creating an indexed collection of man pages. Given the location of the man pages, their desired run-time representation, and the desired indexing technology, the Repository Generator does the following:

1. Creates IUs associated with individual man pages.

2. Creates an IHO for every IU created in step 1.

3. Invokes an independent indexing tool to index the man pages.

4. Creates an IHO associated with the index, and adds parent-child relationships for each IHO created in step 2.

5. For each IHO created in step 2 create the child-parent relationship with the IHO associated with the index.

In a more complicated example of C code, information units are associated with individual functions, and not with .c files. The IHR Generator indexes the .c files, creates the parent-child and child-parent relationships between the object associated with the index and objects associated with .c files. The latter are collections of objects associated with individual functions, which are implemented by creating the additional relationships.

In our experience, adding support for a new type of data is quite simple. Adding support for new third-party indexing technologies is slightly more complicated, primarily because of the need to map information units known to the indexing tool to those known to InfoHarness.

3.0 Applying InfoHarness to Software Reuse

The InfoHarness platform and tools are particularly suitable for constructing a variety of information-rich applications. At Bellcore, we are trialing a number of such applications, including software reuse, electronic data publishing, and accessing logical data models and software contracts. This section describes our experience on applying InfoHarness to facilitate software reuse.

There is an ever-increasing pressure on software organizations in most companies to reduce time-to-market and software development costs without sacrificing quality. The industry trend of dealing with these pressures is to promote software reuse. At Bellcore, software reuse is being advanced through its Workstation Software Factory (WSF) effort [11]. This effort includes the creation of the WSF Repository[2] of reusable software life-cycle artifacts. An artifact in the WSF Repository may be a document, a library, a software tool, etc. The long-range goal of the WSF Repository is to store all software life-cycle artifacts that are developed or used at Bellcore.

The implementation of the WSF Repository has to satisfy the following requirements:

1. Software developers do not need to be familiar with a software artifact to be able to execute or locate it in the Repository.

2. Artifact representation should provide information, which is sufficient to determine its applicability for a particular task.

3. Artifact representation should be flexible to support storing different kinds of information.

4. The creation and maintenance of the WSF Repository should not be a labor-intensive operation.

To achieve a high degree of flexibility in representing heterogeneous artifacts, we represent them as InfoHarness objects. Each artifact contains a high-level description, which is a brief synopsis of its function, and a collection of components. The synopsis information is used to index the artifacts. A component may be a textual document in any format, a bitmap, an audio recording, or a collection of lower-level artifacts. Leaf artifacts are those that do not contain a collection as one of their components.

For example (Figure 2), the WSF Repository may contain artifacts that represent vendor X libraries, Bellcore Desktop, and a vendor development tool. Each of these artifacts contains a description. The representation of the development tool does not contain any components. Both X and Desktop artifacts contain two components. X contains the user guide, and the indexed collection of man pages, while the Desktop artifact contains the collection of screens used to access

---

2. Note the difference between the WSF Repository and the InfoHarness Repository. The WSF Repository is an application, while the InfoHarness Repository is used to implement this application.
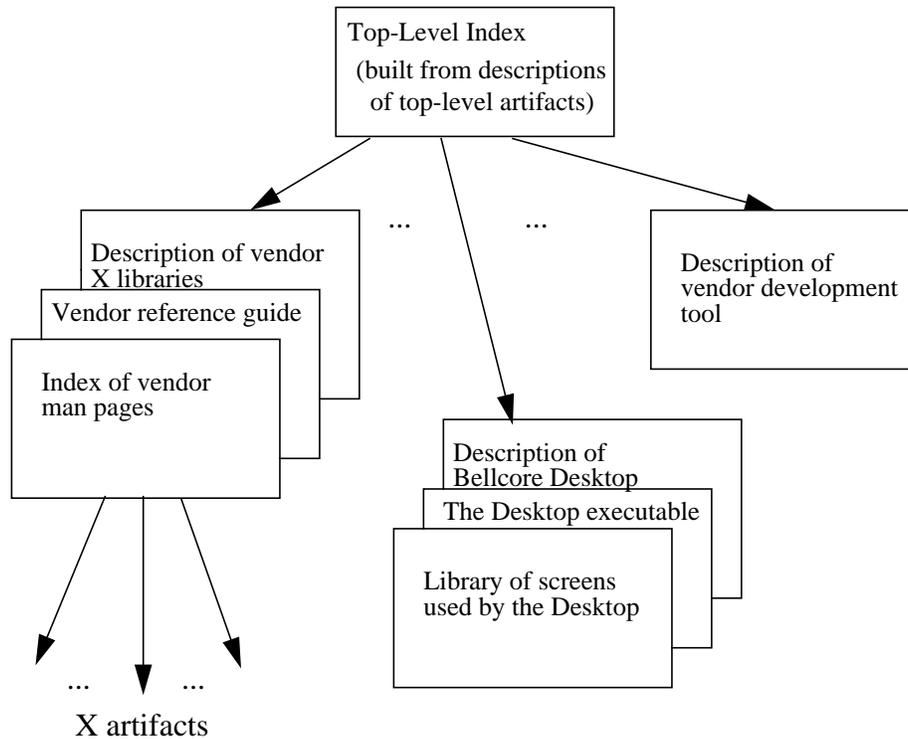
Figure 2. Structure of the WSF Repository

Bellcore-developed software, and the executable. Notice that X is a leaf artifact, while Desktop is not.

3.1 Structure of the WSF Repository

The WSF Repository has the collection of top-level artifacts as its default entry point. The top-level index utilizes general descriptions of the top-level artifacts. The words that only occur in lower-level artifacts are not represented in this index. To search for a low-level artifact, it is necessary to first find the relevant top-level artifact, and only then use one of its collection components to search through lower level artifacts.

Consider the previous example (Figure 2). One of the components of the X artifact is the collection of man pages for X tools and subroutines. Suppose that we want to find out what function returns the current numeric position of a scrollbar (a scrollbar is a widget that allows users to view data that is too large to be displayed at once). The search for *scrollbar* in the top-level collection would not produce a meaningful result, because this word is unlikely to be represented in the vocabulary of descriptions of top-level artifacts. We would first have to find the X artifact, for example, by searching for *widget* or *graphic library.*

Once the artifact is located, selecting its component that is the collection of man pages would change the search context. In the new context the search for *scrollbar* would produce a meaningful result. Selecting a component that is associated with an information unit, will invoke the data access method that is associated with this unit (e.g., open a document).

The lack of multi-level search is a serious limitation because it requires end users to understand the structure of the repository. The textual information that is contained in an artifact's component is not used for searching through artifacts. This information only becomes available once the artifact has been found.

3.2 Latent Semantic Indexing

As mentioned in the previous sections, InfoHarness is flexible enough to allow the use of any indexing technique. For this particular application we have decided to use Bellcore implementation of the Latent Semantic Indexing (LSI) technology [1,4,5]. The LSI index is a vector space of texts that is constructed by applying singular value decomposition and dimensionality reduction to the matrix of keyword-text frequencies. Closeness in this vector space represents semantic similarity.

Queries are represented in plain English text. They are not currently evaluated as sentential forms. Each query is decomposed into words. Each word is stripped of its suffixes, and words that do not occur in any of the texts are discarded. Words that occur in all descriptions (like *the*, *all*, etc.) are also discarded. The remaining words, and their combinations that occur in the texts, are the keywords which may be used to perform the search in the vector space. Queries are represented as vectors that are weighted averages of the keyword vectors. The matches are ranked in the order of the cosine of their vector representations to the query vector. The cutoff, which specifies how many of the matches should be presented to the user, is a query parameter. If the value is not defined, the system will use a default.

3.3 Scalability

The number of leaf artifacts is currently in the order of thousands, but it is likely to expand exponentially in the future. LSI will not scale up to one index that references all leaf artifacts in the Repository. Our current plan is to investigate the use of statistical and machine learning methods to combine results of running the same query against multiple indices when searching the Repository.

One possibility is to simultaneously use textual data contained in various components of artifacts. Each such component would have a separate index associated with it. Note that all indexed components need not be present (and usually will not be present) for each artifact. Following are the possible candidates for independent indexing:

1. Software Requirements

2. Design Documents

3. User Guides

4. Libraries of man pages

5. Vocabulary of keywords

Indexing the vocabularies is the most complicated. Consider some LSI index that only references leaf artifacts. A by-product of building this index is a file that contains all keywords occurring in

artifacts that are being referenced. We will consider this file to be a leaf text object and build another LSI index that references this and other similar objects that are at the same level. By recursively repeating this procedure, we will finally obtain the LSI index that references vocabularies of words, which occur in leaf artifacts. Note that the number of vocabularies is much smaller than the number of leaf artifacts. Thus, indexing the vocabularies is much cheaper because the time to build the LSI index is a function of the number of objects.

Consider the X artifact in the last example (Figure 2). Some of the words that occur in leaf artifacts do not occur in either the user guide or the brief description of the artifact. This is why we are not able to locate the X artifact when searching for *scrollbar*. Making vocabularies of keywords that occur in leaf artifacts available at the top-level helps resolve this problem.

However, using only the described vocabularies for locating top-level artifacts is not good enough for the following reasons:

1. They may have most of the words in common.

2. Words that occur in the top-level descriptions (or other textual components) are not stressed.

3. Sizes of these vocabularies may differ dramatically between artifacts.

An alternative is to also build separate indices for components that contain Requirements, Design Documents, User Guides and man page libraries, and combine the results of running the same query against these indices. The flexibility of the current version of InfoHarness is sufficient to support such approach.

4.0 Future Work

We are planning to pursue the following major directions in the future work on InfoHarness:

1. Complete the implementation of the HTTP-compliant version of the InfoHarness server. The current version of the server communicates with HTTP clients through a gateway. Our further implementation plans include utilizing OMG CORBA's Distributed Object Management for communication among IHOs.

2. Further automate the generation of InfoHarness Repositories through designing and implementing the interactive InfoHarness Web Structure Definition Language (I/WSDL).

3. Investigate the applicability of statistical and Machine Learning methods for combining the results of running the same query against different (possibly heterogeneous) indices.

5.0 Acknowledgments

References

[1] S. T. Dumais, G. W. Furnas, T. K. Landauer, S. Deerwester, and K. Harshman, "Using latent semantic analysis to improve access to textual information", *Proceedings of the 1988 CHI Conference, 1988.*

[2] G. Fischer and C. Stevens, "Information access in complex, poorly structured information spaces", *Proceedings of the 1991 CHI Conference, 1991.*

[3] F. Garzotto. P. Paolini, and D. Schwabe. "HDM - A Model-Based Approach to Hypertext Application Design", *ACM Transactions on Information Systems, 11(1), 1993.*

[4] L. A. Streeter and K. E. Lochbaum, "Who knows: a system based on automatic representation of semantic structure", *Proceedings of RIAO 88: User-oriented context-based text and image handling, Massachusetts Institute of Technology, Cambridge, MA, 1988, pp.379-388.*

[5] Y. Kane-Esrig, L. A. Streeter, W. Keese, and G. Casella, "The relevance density method in information retrieval", *Proceedings of the 4th International Conference on Computing and Information, 1992.*

[6] Y. Kane-Esrig, L. Shklar, and C. St. Charles, Using Multiple Sources of Information to Search a Repository of Software Lifecycle Artifacts, *Proceedings of the Conference on Electronic Document Delivery, May 1994.*

[7] T. Landauer, D. Egan, J. Remde, M. Lesk, C. Lochbaum, and D. Ketchum, "Enhancing the usability of text through computer delivery and formative evaluation: the SuperBook Project", *In C. McKnight, A. Dillon, and J. Richardson, (eds) "Hypertext: A Psychological Perspective", Chichester: Ellis Horwood, 1993, pp. 71-136.*

[8] C.J. Matheus. P.K. Chan, and G. Piatetsky-Shapiro, "Systems for Knowledge Discovery in Databases", *IEEE Transactions on Knowledge and Data Engineering, December 1993.*

[9] John R. Rymer, "Distributed Object Computing", *Distributed Computing Monitor, Vol. 8, No. 8, Boston, 1993.*

[10] A. Sheth and J. Larson, "Federated Database Systems for Managing Distributed, Heterogeneous, and Autonomous Databases", *ACM Computing Surveys, 22(3), 1990.*

[11] L. Shklar, "XReuse: Representation and Retrieval of Heterogeneous Multimedia Objects", *Proceedings of Bellcore Object-Oriented Symposium (BOOST), June 1993.*

[12] K. Shoens, A. Luniewski, P. Shwartz, J. Stamos, and J. Thomas, "The Rufus System: Information Organization for Semi-Structured Data", *Proceedings of the 19th VLDB Conference, Dublin, Ireland, 1993.*

Vita

Leon Shklar has been a Member of Technical Staff at Bellcore since 1990. He is a Ph.D. candidate at Rutgers University, NJ. His research and publications are in the areas of logic programming, information retrieval, and machine learning.

Satish Thatte received his Ph.D. in Electrical Engineering from the University of Illinois at Urbana-Champaign in 1979. From 1979 through 1992, he was with the Corporate Research and Development Group of Texas Instruments, Dallas, TX, where he worked as an individual contributor as well as a manager in various technical fields ranging from VLSI design and design automation, computer architecture, databases and information technologies. He joined Bellcore as the Director of Data Engineering and Technology Group in 1992, where he currently leads and manages projects on shared corporate data, information access, and object data management. He has published over 25 papers in refereed journals and conferences, and holds 11 U.S. and one international patents.

Howard Marcus is a Member of Technical Staff in Bellcore's Data Engineering and Technology group. He is currently working on the design and development of new tools to aid in the access and retrieval of highly distributed, heterogeneous information. Previously he has worked on the development of a new configuration management system to support large, heterogeneous software development. His technical interests include object oriented and logic databases, configuration management, and artificial intelligence.

Amit Sheth has led many projects at research laboratories of Bellcore, Unisys, and Honeywell. These include research and development of a heterogeneous distributed database system, federated database tools on schema integration and view update, and the BrAID system for integration of AI-database systems; design of a factory information system; and research in multi-system applications and transactional workflows, multidatabase consistency, and data quality. He has given over forty-five invited talks and eleven tutorials at major database conferences, and has authored over sixty publications. He has served as an ACM Lecturer, as the General Chair of the 1st International Conference on Parallel and Distributed Information Systems (PDIS) and a Program (co-)Chair of the International Workshop on Interoperability in Multidatabase Systems and 3rd PDIS.

leon@bellcore.com

The "InfoHarness" Information Integration Platform

Leon Shklar, Satish Thatte, Howard Marcus, and Amit Sheth[1]
Bell Communications Research,
444 Hoes Lane,
Piscataway, NJ 08854

The "InfoHarness" information integration platform, tools, and services being developed at Bellcore are aimed at providing integrated and rapid access to huge amounts of heterogeneous information independent of the type, representation, and location of information. InfoHarness provides advanced search and browsing capabilities without imposing the burden of restructuring, reformatting or relocating information on information suppliers or creators. This is achieved through object-oriented encapsulation of information and the associated meta-information (e.g., type, location, access rights, owner, creation date, etc.). The meta-information extraction methods ensure rapid and largely automatic creation of information repositories. A gateway that supports access to InfoHarness repositories from Mosaic and other HyperText Transfer Protocol (HTTP) compliant browsers is currently available. An HTTP compliant InfoHarness server is under construction.

1.0 Introduction

Enormous amounts of heterogeneous information have been accumulated within corporations, government organizations and universities. Such information continues to grow at ever-increasing rate. It ranges from software artifacts to engineering and financial databases, and comes in different types (e.g., source code, e-mail messages, bitmaps) and representations (e.g., plain text, binary). This information has to be accessed through a variety of vendor tools and locally developed applications. It is becoming increasingly easier to create new information due to the proliferation of sophisticated shrink-wrapped commercial authoring and office automation software. It is also becoming easier and cheaper to provide access to this information, due to the increasingly pervasive and more robust data communication technology. However, the knowledge about the existence and location of information, as well as the means of its retrieval, have become so bewildering and confusing to many users as to give rise to the widely lamented phenomenon of *write-only* databases.

Over the last few years there have been numerous attempts to address this problem by building information repositories that depend on relocating and reformatting the original information [7]. Such approaches provide a nice and uniform way to access information but require the design and maintenance of a large number of ever-changing format translators. The initial conversion of information often requires substantial human and computing resources. Maintaining the repositories presents the additional dilemma of either creating new and updating existing information in the uniform format, or continuously managing changing data in different formats. The latter problem may be partially remedied through the latest efforts in the uniform representation of heterogeneous documents [9], but this does not help with arbitrarily formatted data.

---

1. Now with the Computer Science Department, University of Georgia, 415 Graduate Studies Research Center, Athens, GA 30602-7404

Our main objective in constructing InfoHarness is to provide rapid access to huge amounts of heterogeneous information without any relocation, restructuring, or reformatting of data. InfoHarness is aimed at facilitating individual and enterprise productivity by "harnessing" existing and new information assets. Many researchers have investigated the use of meta-data to support run-time access to the original information [2,3,10,12]. Others [8,12] have investigated the use of data mining for the automatic extraction of meta-data. Our own work develops and synthesizes some of the ideas contained in these efforts to provide advanced search and browsing capabilities without imposing constraints on information suppliers or creators.

InfoHarness has been designed with a completely open, extensible, and modular architecture. It provides support for easy incorporation of new types of data, as well as new third-party indexing and browsing technologies. The InfoHarness prototype is now operational and is being trialed at Bellcore for accessing heterogeneous information about software artifacts. It supports the largely automatic generation of InfoHarness repositories, and provides access to the original information from Mosaic and other World-Wide Web (WWW) browsers through an HTTP gateway.

In Section 2 we present a high-level overview of the InfoHarness architecture, concentrating on object representation and the automatic generation of InfoHarness Repositories. Section 3 describes our experience of applying InfoHarness to building software repositories. The paper is concluded with a brief outline of our future research and development plans.

2.0 System Architecture

The InfoHarness system architecture provides a platform for integrating information in a distributed environment by encapsulating existing and new information in objects, without converting, restructuring or reformatting the information. Through this object-oriented encapsulation, the system provides an integrated view and access to diverse and heterogeneous information. The system supports and provides tools for accessing, retrieving, browsing and administering the information encapsulated in the InfoHarness repositories.

Section 2.1 provides an overview of the architecture. Section 2.2 focuses on the use of meta-information to represent the structure and organization of the original information. Section 2.3 describes the automatic generation of meta-information to create InfoHarness repositories.

2.1 Architecture Overview

As shown in Figure 1, the main components of the current implementation of the InfoHarness architecture are:

1. The InfoHarness Server that uses meta-data to traverse, search and retrieve the original information.

2. The HTTP Gateway that is used to pass requests from HTTP clients to the InfoHarness server, and the responses back to the clients.
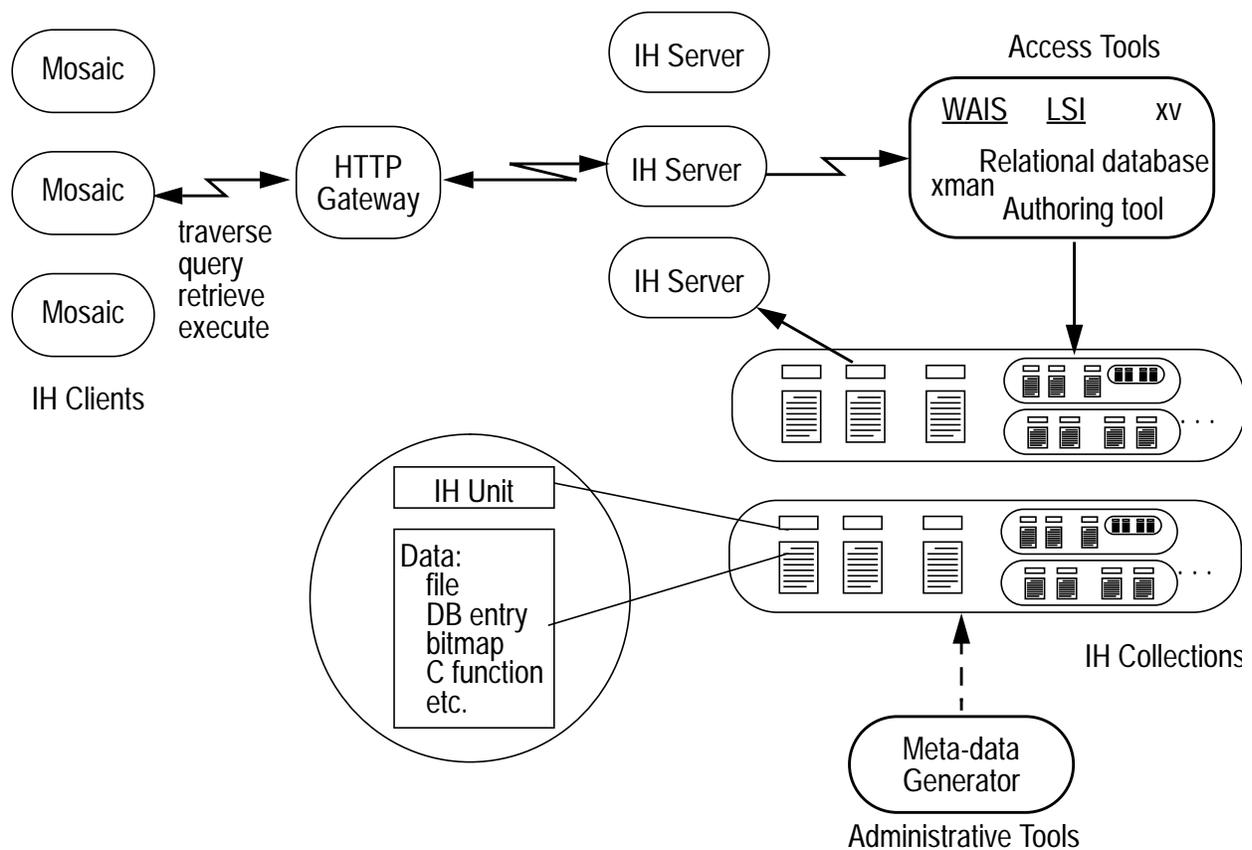
Mosaic

IH Server

Access Tools

Mosaic

HTTP
Gateway

IH Server

WAIS   LSI   xv

Relational database

xman   Authoring tool

traverse
query
retrieve
execute

IH Server

Mosaic

IH Clients

IH Unit

Data:
  file
  DB entry
  bitmap
  C function
  etc.

· · ·

· · ·

IH Collections

Meta-data
Generator

Administrative Tools

Figure 1. InfoHarness Architecture

3. The Repository Generator that is used for off-line automatic generation of meta-data that repre-sents the desirable view on the structure and organization of the original information. This meta-data is used at run-time by the InfoHarness server to search and retrieve information.

4. Independent tools for accessing and displaying information (e.g., xv, xman, etc.), and indexing information (e.g., Wide-Area Information Server (WAIS), Bellcore Latent Semantic Indexing (LSI)).

At run-time, Mosaic users may issue query, traversal, or retrieval requests that are passed on to the Gateway, which then performs the following operations:

1. Parses the request, and reads input information when the request is associated with an HTML form.

2. Establishes a socket connection with the InfoHarness Server, generates and sends out a request, and waits for a response.

3. Parses the response, converts it to a combination of HTML forms and hyperlinks, adds the HTTP header, and passes the transformed response to the Mosaic browser.

The InfoHarness server processes requests based on the meta-information. The same information may be treated differently depending on its type. For example, consider two alternative types,

both designed to represent man pages: *man_to_html* and *xman.* In the first case, the InfoHarness server will pass the man page location to a *man_to_html* converter and send the generated HTML back to the InfoHarness HTTP gateway. In the second case, the server will pass the man page location directly to the *xman* browser and notify the gateway program.

The Gateway does not perform any format conversions of the original information. It converts to HTML those parts of the response that either contain the portion of meta-data that is currently being searched or traversed by the server, or error messages and notifications.

The InfoHarness architecture is open, modular, extensible and scalable. New types of information can be easily added by defining and registering new object types. InfoHarness implements the abstract type hierarchy that is complete in the sense that any new data type, or a new indexing technology may be added by instantiating an existing abstract type. The methods that are associated with abstract types are general enough because each method is data-driven and may invoke any third-party software. The definitions of new types are also data-driven and are not part of the InfoHarness implementation.The integration platform can support arbitrary information access and management tools (e.g., browsers, indexing methods, access methods) developed by Bellcore, universities and third parties.

The autonomy of administrators, no centralized control, support for multiple clients and multiple servers in a large-scale distributed, heterogeneous environment all support the goals of scalability and operability in a large and geographically distributed environment.

2.2 Object Representation

A meta-data entity that is associated with the lowest level of granularity of information available to InfoHarness is called the *information unit* (IU). The IU may be associated with a file (e.g., a man page), a portion of a file (e.g., a C function or a database table), a set of files (e.g., a collection of related bitmaps), or any request for the retrieval of data from an external source (e.g., a database query).

An InfoHarness object (IHO) may be one of the following:

1. A single information unit.

2. A collection of InfoHarness objects (either indexed, or non-indexed).

3. A single information unit and a non-indexed collection of InfoHarness objects.

Each IHO has a unique object identifier that is recognized and maintained by the system.An IHO that encapsulates an IU contains information about the location of data, data retrieval method, and any parameters needed by the method to extract the relevant portion of information. For example, an IHO associated with a C function will contain the path information for the .c file, the name and location of the program that knows how to extract a function from a .c file, and the name of the function to be passed to this program as a parameter. In addition, each IHO may contain arbitrary number of attribute-value pairs (e.g., owner, last update, security information, decompression method).

Each IHO that encapsulates a collection of IHOs stores unique object identifiers of the members of the collection. We refer to these members as *children* of the IHO. An IHO that encapsulates both an IU and a collection is called a *composite object*. An example of the composite object is this paper's abstract combined with the collection of postscript and HTML versions of the full paper. IHOs that encapsulate indexed collections store information about the location of both the index and the query method. Any indexed collection may make use of its own data retrieval method that is not part of InfoHarness. As a result, an InfoHarness Repository may easily be created from existing heterogeneous index structures.

An InfoHarness Repository (IHR) is a set of IHOs that are not necessarily members of the same collection. An IHO may be a member of any number of collections. We refer to IHOs that are associated with these collections as *parents*. Each IHO that has one or more parents always contains unique object identifiers of its parent objects. An IHO that does not have any parent is unreachable from any other IHO and may only be accessed if it is used as the initial starting point (or *entry point*) in the IHR traversal.

2.3 Automatic Generation of InfoHarness Repositories

Each IHO in an IHR may have multiple children, as well as multiple parents. The physical data that is associated with information units is not part of the IHR. The generation of an IHR amounts to the creation of IHOs and their relationships, and indexing the information declared to belong to an indexed collection.

The IHR generation is supported by the InfoHarness administrative tools. The top-level tool, called the Repository Generator, accepts as inputs both the location and the desired representation of data, and outputs the set of IHOs. Defining a collection of the entry points of the multiple sets of IHOs is the most straightforward way to combine them into a single Repository.

Consider a simple example of creating an indexed collection of man pages. Given the location of the man pages, their desired run-time representation, and the desired indexing technology, the Repository Generator does the following:

1. Creates IUs associated with individual man pages.

2. Creates an IHO for every IU created in step 1.

3. Invokes an independent indexing tool to index the man pages.

4. Creates an IHO associated with the index, and adds parent-child relationships for each IHO created in step 2.

5. For each IHO created in step 2 create the child-parent relationship with the IHO associated with the index.

In a more complicated example of C code, information units are associated with individual functions, and not with .c files. The IHR Generator indexes the .c files, creates the parent-child and child-parent relationships between the object associated with the index and objects associated with .c files. The latter are collections of objects associated with individual functions, which are implemented by creating the additional relationships.

In our experience, adding support for a new type of data is quite simple. Adding support for new third-party indexing technologies is slightly more complicated, primarily because of the need to map information units known to the indexing tool to those known to InfoHarness.

3.0 Applying InfoHarness to Software Reuse

The InfoHarness platform and tools are particularly suitable for constructing a variety of information-rich applications. At Bellcore, we are trialing a number of such applications, including software reuse, electronic data publishing, and accessing logical data models and software contracts. This section describes our experience on applying InfoHarness to facilitate software reuse.

There is an ever-increasing pressure on software organizations in most companies to reduce time-to-market and software development costs without sacrificing quality. The industry trend of dealing with these pressures is to promote software reuse. At Bellcore, software reuse is being advanced through its Workstation Software Factory (WSF) effort [11]. This effort includes the creation of the WSF Repository[2] of reusable software life-cycle artifacts. An artifact in the WSF Repository may be a document, a library, a software tool, etc. The long-range goal of the WSF Repository is to store all software life-cycle artifacts that are developed or used at Bellcore.

The implementation of the WSF Repository has to satisfy the following requirements:

1. Software developers do not need to be familiar with a software artifact to be able to execute or locate it in the Repository.

2. Artifact representation should provide information, which is sufficient to determine its applicability for a particular task.

3. Artifact representation should be flexible to support storing different kinds of information.

4. The creation and maintenance of the WSF Repository should not be a labor-intensive operation.

To achieve a high degree of flexibility in representing heterogeneous artifacts, we represent them as InfoHarness objects. Each artifact contains a high-level description, which is a brief synopsis of its function, and a collection of components. The synopsis information is used to index the artifacts. A component may be a textual document in any format, a bitmap, an audio recording, or a collection of lower-level artifacts. Leaf artifacts are those that do not contain a collection as one of their components.

For example (Figure 2), the WSF Repository may contain artifacts that represent vendor X libraries, Bellcore Desktop, and a vendor development tool. Each of these artifacts contains a description. The representation of the development tool does not contain any components. Both X and Desktop artifacts contain two components. X contains the user guide, and the indexed collection of man pages, while the Desktop artifact contains the collection of screens used to access

---

2. Note the difference between the WSF Repository and the InfoHarness Repository. The WSF Repository is an application, while the InfoHarness Repository is used to implement this application.
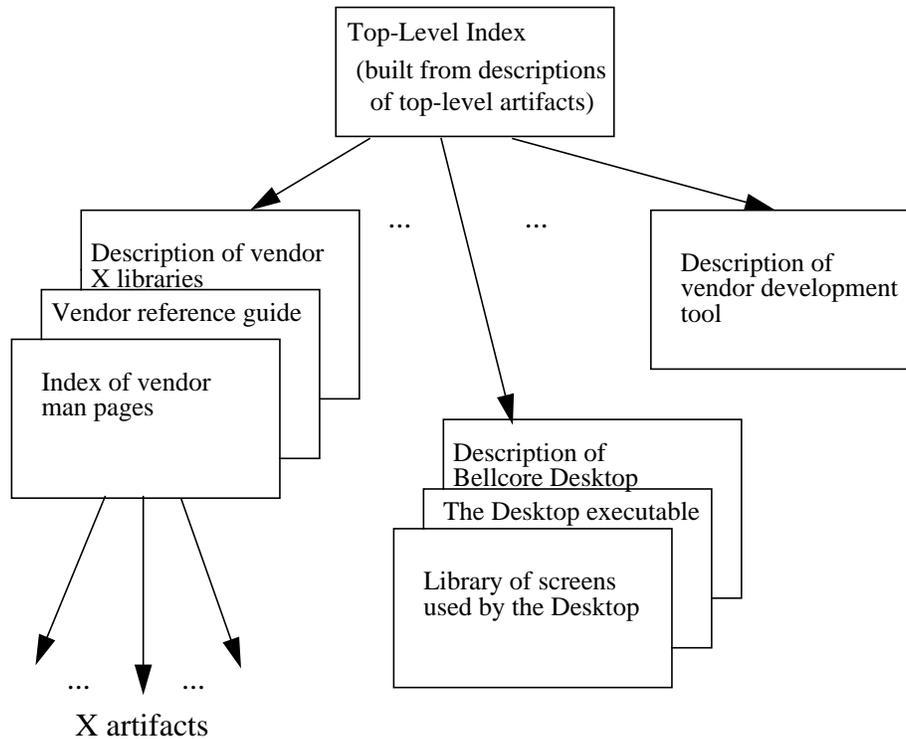
Top-Level Index

(built from descriptions of top-level artifacts)

...   ...

Description of vendor X libraries

Vendor reference guide

Index of vendor man pages

Description of vendor development tool

Description of Bellcore Desktop

The Desktop executable

Library of screens used by the Desktop

...   ...

X artifacts

Figure 2. Structure of the WSF Repository

Bellcore-developed software, and the executable. Notice that X is a leaf artifact, while Desktop is not.

3.1 Structure of the WSF Repository

The WSF Repository has the collection of top-level artifacts as its default entry point. The top-level index utilizes general descriptions of the top-level artifacts. The words that only occur in lower-level artifacts are not represented in this index. To search for a low-level artifact, it is necessary to first find the relevant top-level artifact, and only then use one of its collection components to search through lower level artifacts.

Consider the previous example (Figure 2). One of the components of the X artifact is the collection of man pages for X tools and subroutines. Suppose that we want to find out what function returns the current numeric position of a scrollbar (a scrollbar is a widget that allows users to view data that is too large to be displayed at once). The search for *scrollbar* in the top-level collection would not produce a meaningful result, because this word is unlikely to be represented in the vocabulary of descriptions of top-level artifacts. We would first have to find the X artifact, for example, by searching for *widget* or *graphic library.*

Once the artifact is located, selecting its component that is the collection of man pages would change the search context. In the new context the search for *scrollbar* would produce a meaningful result. Selecting a component that is associated with an information unit, will invoke the data access method that is associated with this unit (e.g., open a document).

The lack of multi-level search is a serious limitation because it requires end users to understand the structure of the repository. The textual information that is contained in an artifact's component is not used for searching through artifacts. This information only becomes available once the artifact has been found.

3.2 Latent Semantic Indexing

As mentioned in the previous sections, InfoHarness is flexible enough to allow the use of any indexing technique. For this particular application we have decided to use Bellcore implementation of the Latent Semantic Indexing (LSI) technology [1,4,5]. The LSI index is a vector space of texts that is constructed by applying singular value decomposition and dimensionality reduction to the matrix of keyword-text frequencies. Closeness in this vector space represents semantic similarity.

Queries are represented in plain English text. They are not currently evaluated as sentential forms. Each query is decomposed into words. Each word is stripped of its suffixes, and words that do not occur in any of the texts are discarded. Words that occur in all descriptions (like *the*, *all*, etc.) are also discarded. The remaining words, and their combinations that occur in the texts, are the keywords which may be used to perform the search in the vector space. Queries are represented as vectors that are weighted averages of the keyword vectors. The matches are ranked in the order of the cosine of their vector representations to the query vector. The cutoff, which specifies how many of the matches should be presented to the user, is a query parameter. If the value is not defined, the system will use a default.

3.3 Scalability

The number of leaf artifacts is currently in the order of thousands, but it is likely to expand exponentially in the future. LSI will not scale up to one index that references all leaf artifacts in the Repository. Our current plan is to investigate the use of statistical and machine learning methods to combine results of running the same query against multiple indices when searching the Repository.

One possibility is to simultaneously use textual data contained in various components of artifacts. Each such component would have a separate index associated with it. Note that all indexed components need not be present (and usually will not be present) for each artifact. Following are the possible candidates for independent indexing:

1. Software Requirements

2. Design Documents

3. User Guides

4. Libraries of man pages

5. Vocabulary of keywords

Indexing the vocabularies is the most complicated. Consider some LSI index that only references leaf artifacts. A by-product of building this index is a file that contains all keywords occurring in

artifacts that are being referenced. We will consider this file to be a leaf text object and build another LSI index that references this and other similar objects that are at the same level. By recursively repeating this procedure, we will finally obtain the LSI index that references vocabularies of words, which occur in leaf artifacts. Note that the number of vocabularies is much smaller than the number of leaf artifacts. Thus, indexing the vocabularies is much cheaper because the time to build the LSI index is a function of the number of objects.

Consider the X artifact in the last example (Figure 2). Some of the words that occur in leaf artifacts do not occur in either the user guide or the brief description of the artifact. This is why we are not able to locate the X artifact when searching for *scrollbar*. Making vocabularies of keywords that occur in leaf artifacts available at the top-level helps resolve this problem.

However, using only the described vocabularies for locating top-level artifacts is not good enough for the following reasons:

1. They may have most of the words in common.

2. Words that occur in the top-level descriptions (or other textual components) are not stressed.

3. Sizes of these vocabularies may differ dramatically between artifacts.

An alternative is to also build separate indices for components that contain Requirements, Design Documents, User Guides and man page libraries, and combine the results of running the same query against these indices. The flexibility of the current version of InfoHarness is sufficient to support such approach.

4.0 Future Work

We are planning to pursue the following major directions in the future work on InfoHarness:

1. Complete the implementation of the HTTP-compliant version of the InfoHarness server. The current version of the server communicates with HTTP clients through a gateway. Our further implementation plans include utilizing OMG CORBA's Distributed Object Management for communication among IHOs.

2. Further automate the generation of InfoHarness Repositories through designing and implementing the interactive InfoHarness Web Structure Definition Language (I/WSDL).

3. Investigate the applicability of statistical and Machine Learning methods for combining the results of running the same query against different (possibly heterogeneous) indices.

5.0 Acknowledgments

References

[1] S. T. Dumais, G. W. Furnas, T. K. Landauer, S. Deerwester, and K. Harshman, "Using latent semantic analysis to improve access to textual information", *Proceedings of the 1988 CHI Conference, 1988.*

[2] G. Fischer and C. Stevens, "Information access in complex, poorly structured information spaces", *Proceedings of the 1991 CHI Conference, 1991.*

[3] F. Garzotto. P. Paolini, and D. Schwabe. "HDM - A Model-Based Approach to Hypertext Application Design", *ACM Transactions on Information Systems, 11(1), 1993.*

[4] L. A. Streeter and K. E. Lochbaum, "Who knows: a system based on automatic representation of semantic structure", *Proceedings of RIAO 88: User-oriented context-based text and image handling, Massachusetts Institute of Technology, Cambridge, MA, 1988, pp.379-388.*

[5] Y. Kane-Esrig, L. A. Streeter, W. Keese, and G. Casella, "The relevance density method in information retrieval", *Proceedings of the 4th International Conference on Computing and Information, 1992.*

[6] Y. Kane-Esrig, L. Shklar, and C. St. Charles, Using Multiple Sources of Information to Search a Repository of Software Lifecycle Artifacts, *Proceedings of the Conference on Electronic Document Delivery, May 1994.*

[7] T. Landauer, D. Egan, J. Remde, M. Lesk, C. Lochbaum, and D. Ketchum, "Enhancing the usability of text through computer delivery and formative evaluation: the SuperBook Project", *In C. McKnight, A. Dillon, and J. Richardson, (eds) "Hypertext: A Psychological Perspective", Chichester: Ellis Horwood, 1993, pp. 71-136.*

[8] C.J. Matheus. P.K. Chan, and G. Piatetsky-Shapiro, "Systems for Knowledge Discovery in Databases", *IEEE Transactions on Knowledge and Data Engineering, December 1993.*

[9] John R. Rymer, "Distributed Object Computing", *Distributed Computing Monitor, Vol. 8, No. 8, Boston, 1993.*

[10] A. Sheth and J. Larson, "Federated Database Systems for Managing Distributed, Heterogeneous, and Autonomous Databases", *ACM Computing Surveys, 22(3), 1990.*

[11] L. Shklar, "XReuse: Representation and Retrieval of Heterogeneous Multimedia Objects", *Proceedings of Bellcore Object-Oriented Symposium (BOOST), June 1993.*

[12] K. Shoens, A. Luniewski, P. Shwartz, J. Stamos, and J. Thomas, "The Rufus System: Information Organization for Semi-Structured Data", *Proceedings of the 19th VLDB Conference, Dublin, Ireland, 1993.*

Vita

Leon Shklar has been a Member of Technical Staff at Bellcore since 1990. He is a Ph.D. candidate at Rutgers University, NJ. His research and publications are in the areas of logic programming, information retrieval, and machine learning.

Satish Thatte received his Ph.D. in Electrical Engineering from the University of Illinois at Urbana-Champaign in 1979. From 1979 through 1992, he was with the Corporate Research and Development Group of Texas Instruments, Dallas, TX, where he worked as an individual contributor as well as a manager in various technical fields ranging from VLSI design and design automation, computer architecture, databases and information technologies. He joined Bellcore as the Director of Data Engineering and Technology Group in 1992, where he currently leads and manages projects on shared corporate data, information access, and object data management. He has published over 25 papers in refereed journals and conferences, and holds 11 U.S. and one international patents.

Howard Marcus is a Member of Technical Staff in Bellcore's Data Engineering and Technology group. He is currently working on the design and development of new tools to aid in the access and retrieval of highly distributed, heterogeneous information. Previously he has worked on the development of a new configuration management system to support large, heterogeneous software development. His technical interests include object oriented and logic databases, configuration management, and artificial intelligence.

Amit Sheth has led many projects at research laboratories of Bellcore, Unisys, and Honeywell. These include research and development of a heterogeneous distributed database system, federated database tools on schema integration and view update, and the BrAID system for integration of AI-database systems; design of a factory information system; and research in multi-system applications and transactional workflows, multidatabase consistency, and data quality. He has given over forty-five invited talks and eleven tutorials at major database conferences, and has authored over sixty publications. He has served as an ACM Lecturer, as the General Chair of the 1st International Conference on Parallel and Distributed Information Systems (PDIS) and a Program (co-)Chair of the International Workshop on Interoperability in Multidatabase Systems and 3rd PDIS.

leon@bellcore.com