Information Organization Using Rufus

Allen Luniewski, Peter Schwarz, Kurt Shoens, Jim Stamos, John Thomas

(luniew, schwarz, shoens, stamos, jthomas)@almaden.ibm.com

IBM Almaden Research Center, 650 Harry Road, San Jose, CA 95120

Computer system users today are inundated with a flood of semi-structured information, such as documents, electronic mail, programs, and images. Today, this information is typically stored in filesystems that provide limited support for organizing, searching, and operating upon this data, all operations that are vital to the ability of users to effectively use this data. Database systems provide good function for organizing, searching, managing and writing applications on structured data. Current database systems are inappropriate for semi-structured information because moving the data into the database breaks all existing applications that use the data. The Rufus system attacks the problems of semi-structured information by using database function to help users manage semi-structured information without requiring that the user's information reside in the database.

OVERVIEW OF RUFUS

Rufus brings traditional database function to the problems of semi-structured information. Rufus uses an object oriented database (OODB) to store descriptive information about a user's data and a full text retrieval database to provide access to the textual content of that data. We refer to these two databases as "the Rufus database." When user data is to be described in a Rufus database, Rufus automatically examines that data to determine its class. A class-specific import method extracts and indexes structured information about the user data, placing the results into the Rufus object oriented database. The import method also extracts and indexes the textual content of the file (indexing of non-textual content such as images could be added). The underlying databases support fast querying. The OODB supports fast access to its objects. Rufus provides various ways of structuring the objects to support browsing. This object infrastructure is made available through a client-server interface and forms the basis for a family of applications that manage semi-structured information.

The Rufus data model represents, as an object, structured information extracted from user data. The structured view describes what the data is, interesting values determined about the data, and what operations can be performed on it. We chose a single inheritance, object-oriented data model supporting substitutability, as in Smalltalk[5]. Thus Rufus supports both subclassing and object composition. The classes in the hierarchy are recognizable to users as types of files and information that they use. At the root of the hierarchy is a generic *Object* class. At intermediate nodes of the hierarchy are broad classes such as *Text*, *Binary* and *Mail*. At the leaves are specific classes such as *C-Source*, *RFC-822 Mail*, and *Usenet Articles*.

Methods associated with Rufus classes encapsulate actions that a user can take on the data represented in the Rufus database. There is no need for the user to know the details of how to operate on the data; the user need only tell Rufus what operation is desired.

When Rufus is presented with user data, it must determine its type within the class hierarchy. Given the volumes of semi-structured data, it is unreasonable to expect users to manually perform this classification. Thus, Rufus examines the user data and automatically determines its class. This is the first piece of information that Rufus supplies a user about a piece of data. The classifier is reasonably accurate, but if it makes a mistake the user may override its decision. Classification is fast enough to not be a bottleneck during import.

Once some user data has been classified, Rufus calls the import method for that class to extract and index information from the data. Interesting class-specific attributes of the object are placed into the OODB. Any textual data is placed into the Rufus text index. Rufus could be extended to support databases of other contents (*e.g.*, images). The import process leaves the user data unchanged, thus preserving the functionality of existing applications.

The Rufus query language provides content-based access to semi-structured data. Rufus queries look for objects based both upon their extracted attributes and their unstructured content (*i.e.*, text in Rufus today). Attribute queries look for objects with certain attribute values. Currently Rufus only permits the comparison of attribute values to a constant. Text queries look for objects that have certain words in their text content and feature tests for word adjacency and nearness. Text queries may also take into account word stems and capitalization. Arbitrary boolean combinations of text and attribute query terms are possible in a single query.

Rufus provides collections, object composition, and hypertext links[2] to represent inter-object structure. Collections are sets of objects of mixed classes, and may include other collections. An object can be in several collections at once. For each class, Rufus maintains a *class extent*, a collection of all instances of the class. When an object is created, Rufus automatically adds it to its class extent and to the extent of each of its superclasses. Collections are also used to store the results of queries. Since collections can be used as the domain for a query, a limited form of query refinement is possible. Rufus will discover some hyper-links between a user's objects (*e.g.*, a chain of mail messages connected via their "In-Reply-To" fields). Collections and hyper-links assist the user in browsing the objects in a Rufus database.

SIGMOD /5/93/Washington, DC,USA
© 1993 ACM 0-89791-592-5/93/0005/0560...$1.50

Since users can access much of their data through distributed file systems, the Rufus capabilities described in the foregoing are implemented by a client/server architecture to provide the same connectivity to data. Each Rufus server mediates access by multiple simultaneous clients to a single Rufus database. A Rufus database may be public or private. We anticipate an environment in which Rufus databases that describe information of interest to the community are made available by members of the community. Current client/server functionality is limited in several important ways. Client applications can connect to only a single Rufus server at a time. Collections that cross server boundaries are not supported nor are queries across multiple databases.

A Rufus library is provided that permits the creation of applications that use and modify the information in one or more Rufus databases. The library provides client access to Rufus servers. An application might simply provide access to the information in a Rufus database or it might use Rufus to provide additional functionality beyond its pre-existing services.

RUFUS APPLICATIONS

The Rufus project has built three applications using the Rufus toolkit: *xrufus*, *rufustrn* and *rufusxrn*. The Rufus demonstration will feature these three applications.

The xrufus application provides general purpose access to a Rufus database. With it, users can cause data to be imported into a Rufus database, browse the data in an object specific manner and search for information of interest. In addition to a general purpose query interface, xrufus also has a simple interface for queries on the textual content of objects, the most common queries. Finally, xrufus provides access to the methods on Rufus objects, thus giving the user the ability to access and manipulate the underlying objects with their native applications.

The Rufus project has modified two newsreaders to make use of Rufus capabilities. The Internet Usenet news feed provides a great deal of information organized in one particular way. Users who browse Usenet are often looking for information on a particular subject. This information frequently will not fall neatly into one of the many "buckets" provided by Usenet. The user is faced with either manually browsing this rather large newsfeed (over 7.5 megabytes of new data each day) or missing information of interest. By placing the contents of the news feed into a Rufus database, queries can be used to find desired information. As has been seen before [3, 4], this is of great assistance to the user. The rufustrn application is an adaptation of the trn newsreader that allows the user to use queries to define *virtual newsgroups* on topics of interest. The rufusxrn application brings virtual newsgroups to the X[1]-based xrn news reader.

STATUS and FUTURE WORK

Rufus is currently available to the Computer Science Department at the Almaden Research Center. The Rufus project is supporting two public databases: one for the Usenet newsfeed and one for the internal IBM newsfeed. In addition, a number of private databases have appeared on our local network. Our user community is small but growing.

Work is currently underway in the Rufus project in a number of areas. This work will improve both the functionality and performance of Rufus.

The current Rufus data model models many kinds of user data very well. However, some data, such as PostScript[2] files, which are simultaneously text, image and programs, do not fit into the single inheritance data model. A new data model based upon conformity[1] and the Melampus data model [6] is being implemented. The new data model will enable Rufus to better model the data actually found in computers as well as making it easier to add new types to Rufus.

The current Rufus classifier is based upon decision trees. Although it works very well, it is difficult to add new types due to the interdependencies among the weighted answers returned by various nodes in the tree. A more extensible classifier is being designed and built, based upon statistical classification. Preliminary results indicate that the new classifier will be much more extensible than the old one and have better classification accuracy.

The current text database has very good query and update performance up to about 15 megabytes of data. Up to about 45 megabytes of data performance remains adequate. After that, performance becomes increasingly unacceptable. New data structures and algorithms are being developed that should significantly improve the space and time performance of the text database.

We are also working towards supporting a more general client-server environment. We would like clients to be able to have collections that span server boundaries. Clients should also be able to execute queries against multiple databases simultaneously.

We would also like a more general hyperlink facility. Finally, we would like to have rules and agents in Rufus.

Rufus is using traditional database technologies to address the important problem of managing semi-structured information. Initial experiences with this approach are very positive. With the changes outlined here, Rufus will be even more valuable.

REFERENCES

[1] A. Black, et. al., Object Structure in the Emerald System. In *Proceedings of ACM Conference on Object Oriented Programming Systems, Languages and Applications*, September 1986.

[2] J. Conkin, Hypertext: An introduction and survey, *IEEE Computer*, 20(9), 1987.

[3] D. Gifford et. al., An Architecture for large scale information systems. In *Proceedings of the 10th ACM Symposium on Operating Systems Principles, pages 161-170, December 1985.*

[4] D. Goldberg et. al., Using collaborative filtering to weave an information tapestry. *Comm. ACM*, 35(12):61-70, December 1992.

[5] A. Goldberg, D. Robson, *Smalltalk 80: The Language and Its Implementation.* Addison-Wesley Publishing Company, 1983.

[6] J. Richardson, P. Schwarz, MDM: An object-oriented data-model. In *Proceedings of the Third International Workshop on Database Programming Languages*, August, 1991.

1. X Window System is a trademark of the Massachusetts Institute of Technology.

2. PostScript is a trademark of the Adobe Corporation.