A Powerful Wide-Area Information Client

Tak W. Yan*
Hewlett-Packard Laboratories
1501 Page Mill Road
Palo Alto, CA 94304

Jurgen Annevelink
Hewlett-Packard Object Access Program
19119 Pruneridge Avenue
Cupertino, CA 95014

Abstract

Wide-area information sharing is rapidly gaining momentum. However, current information systems provide inadequate functionality for users to efficiently and effectively search and use the information made available. Users are faced with a vast information space apparently in disarray. Assuming current client-server information system architecture, we argue in this paper that more functionality is needed on the information client side. We describe an information client prototype that supports (1) the use of object technology to model the information space, (2) declarative querying as a complement to navigational search, (3) uniform access and integration of data from diverse sources, and (4) user-specified and transparent caching that is integrated with permanent local storage of remote data. We also identify the needed support from the server side to better achieve these capabilities.

1 Introduction

Technological advances have made wide-area information sharing commonplace. A number of systems, notably Wide-Area Information Servers (WAIS) [7] and World-Wide Web (WWW) [3], have emerged to allow ordinary information owners to offer information online for others to search and retrieve across wide-area networks. The countless information providers together make up a global information system that is rich in data content, but at the same time, extremely large-scale, dynamic, and diverse.

These current systems are typically client-server oriented, and each system is largely defined by its underlying communication protocol (e.g., Z39-50 for WAIS and HTTP for WWW). An information source is set up as a server, and a client generally has a sophisticated graphical user interface for remotely accessing these servers. While these systems have made a great step toward global information sharing, they are nonetheless unsatisfactory in certain aspects. One key point is that current implementations of information clients make inadequate use of local computing and storage resources. In this paper we focus on the following limitations.

- *Inadequate support for modeling of information space* Current systems provide users with the ability to discover new information sources through querying and browsing, but have little support for storing, managing, and using meta information about the sources after discovery. What is needed is a local view of known, relevant, and reliable sources in the vast information space; a view that the user can expand, prune, manipulate, and query.

- *Procedural browsing and searching* One of the main complaints about current systems is the difficulty in searching for information. Any search is highly procedural. Typically, the user first searches a directory or "meta index" server that stores information about other servers to locate potential target servers. Then the user has to query each target server, and browse through all the results. After much searching and browsing, the user easily becomes disoriented and goes astray in the information space. In the case where the user has complicated searching criteria, say only searches sources that are located in the U.S., the searching will be even more tedious.

- *Difficulty in information integration* In current systems, the client interface is typically a standalone program, without an application programming interface. It is difficult for the user to make use of data that he may have stored in a local file system or a database to assist in searching (typically the only means to utilize the data is to "cut-and-paste"). After finding relevant data, the client program normally allows the user to store it locally; but to further process the data, say organize and store them in a database, often requires many extra steps.

*Also, Department of Computer Science, Stanford University, Stanford CA 94305

- *Inadequate caching support* With the overwhelming number of information providers and users, traffic generated by information sharing is rapidly pushing the limits of existing long-haul networks. Local caching is recognized as a good way to reduce the traffic [1], but current systems do not provide full caching support. Since adding caching to wide-area information clients is not a challenging problem, we expect future client programs to provide a greater degree of caching. However, there are a number of issues that need to be resolved; these include support for user-specified caching vs. transparent caching, cache invalidation policies, and interaction between caching and permanent local storage of remote files.

The proliferation of high-performance workstations and personal computers means that the typical client machine on the network has considerable processing and storage power, which can be utilized to provide more efficient and effective information sharing. In addition, this should reduce the growing demand on information servers and communication networks. Using the popular wide-area information system WAIS as our target system, this paper presents our approach to address the above issues. We use an extensible, object-oriented database system, Odapter[6], to encapsulate the querying mechanisms supported by WAIS. This results in an information client that inherits the database system's powerful modeling, querying, and information integration capabilities. The prototype also provides full caching support, and addresses issues such as cache coherence and interaction with persistent storage.

The rest of the paper is organized as follows. We briefly summarize the WAIS and Odapter systems in Section 2. In Section 3 we describe the coupling of Odapter and the WAIS client and show how this prototype addresses the issues raised above. In Section 4 we discuss some related work, and finally we conclude in Section 5.

2 WAIS and Odapter

Our target wide-area information system is WAIS. Although we are focusing on one system, we believe the ideas presented in the following sections apply to other systems as well. One notable example is the increasingly popular WWW system. WWW is a global hypermedia system and, unlike the query-based WAIS, following hyperlinks constitutes a major part of the search process. However, researchers and implementors have long recognized the limits of a navigation-only system, and querying has been added to speed up the search process [4,5]. Thus, the issues discussed in this paper are important and applicable to wide-area hypermedia systems as well.

The following is a brief overview of the WAIS and Odapter systems. For detailed coverage, the reader is referred to [7] and [6] respectively.

2.1 WAIS

WAIS was developed by Thinking Machines to facilitate network information sharing applications. Information owners are supplied with software to set up their WAIS servers, and information users are equipped with client programs to access the servers. WAIS has kept growing since its start in 1990, and presently, there are over 500 WAIS sources, providing topics from poetry to computer science literature, from government documents to weather information.

A WAIS server may offer a number of databases, called sources. Documents in a source are full-text indexed. The user searches a source with a query, which is simply some plain English text. To process a query, a WAIS server uses a certain weighted matching algorithm; those documents with the most occurrences of query words are returned.

There are some special directory sources available that store meta information about other sources. The user queries one of these directories to locate sources that may contain relevant information and subsequently searches the located sources. The directory sources are accessed in the same way as other sources.

The version of WAIS client and server we refer to in this paper is freeWAIS-0.3 from the Clearinghouse for Networked Information Discovery and Retrieval.

2.2 Odapter

Odapter [6], developed by Hewlett-Packard, is an object-oriented database access layer integrating access to relational databases and other types of external data sources. OSQL [2] is a database programming language for Odapter. It is centered around three basic concepts: *objects*, *types*, and *functions*. Objects represent the real-world entities and concepts from the application domain that the database is storing information about. Each object has a unique object identifier (OID). Types are used to classify objects on the basis of shared properties and/or behavior. Types are related into a subtype/supertype hierarchy that supports multiple inheritance.

Functions are used to model attributes of the object, interobject relationships, and arbitrary computations. One of the key distinctions of OSQL as compared to other object languages is this unifying notion of a function. An OSQL function takes one or more objects as arguments and may return an object as a result. OSQL functions can be overloaded, i.e., there can be multiple functions with the same name but different argument types. A function mapping can be explicitly stored, or it can be computed. A computed function can be implemented as a procedure written in a general purpose programming language (e.g., C or C++). This is called an external function which gives OSQL a unique form of extensibility by allowing the encapsulation of (entry points in) external libraries to access external data and functionality. We made extensive use of external functions in our prototype.

OSQL supports a query language whose semantics is based on domain calculus. The OSQL select function provides the basic query facilities of OSQL and closely resembles the SELECT statement of SQL. For example, suppose we have a type Employee in the database with function (attribute) JobTitle. Then the following returns the OIDs of all software engineers in the database.

```
select e for each Employee e
  where JobTitle(e) = 'Software Engineer';
```

3 A Powerful Wide-Area Information Client

In this section we describe our wide-area information client prototype according to the four issues raised in the introduction. Figure 1 shows an overview of our approach. We use the external function capability of Odapter to encapsulate the WAIS querying and retrieval mechanisms. Odapter provides an object modeling layer, enabling the user to model WAIS sources as objects and use object supertype/subtype hierarchies to categorize sources. The user can use the database query language OSQL to express his search needs with more declarative queries. Further, as Odapter encompasses local and remote data uniformly, information integration becomes easy. Local caching is layered on top of the WAIS routines. Below we elaborate on each feature in turn.

3.1 An Object-Oriented View of External Information Space

In WAIS, the user can maintain a list of sources. However, the means to organize them or query meta information about them is lacking. The user has to rely on memory, or browsing, to determine their usefulness in subsequent searches. Mosaic (version 2.0 for X Window), one of the most advanced client interfaces for WWW, allows the user to keep a "hot list" of Universal Resource Locators (URLs), which are pointers to remote hypermedia documents. No meta information about resources is stored.

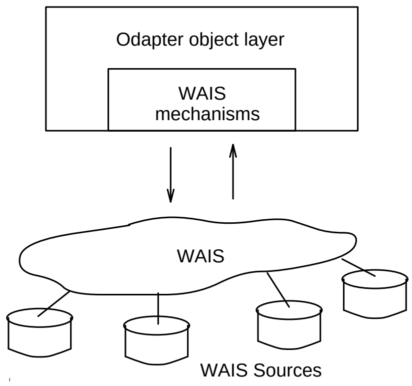

Figure 1: Encapsulation of WAIS search and retrieval mechanisms

The user of wide-area information systems needs a local view of the global information space that is pertinent and relevant to him. The local model should adequately characterize the sources as well as allow the user to add notes or annotations to them. The model should allow the user to organize, manipulate, and query local view to assist in searching. We achieve these goals with an object layer provided by Odapter.

We model information sources as objects of type WAISSource. Attributes of a WAISSource object store meta information exported from the source, including ServerName, which is the IP name for the server site, DatabaseName, the name of the WAIS database, Cost, the cost for accessing the source, CostUnit, the unit in which the cost is expressed, and Description, a textual description of the content of the source (maintained by the source). We have an additional attribute, Annotation, a field for holding textual user annotations.

To categorize different information sources, the user may create subtypes of WAISSource and build a complex supertype/subtype hierarchy. The categorization can be based on the content of the information sources (e.g., grouping sources offering computer science literature into a subtype), an organizational boundary (e.g., all NASA sources), or any other arbitrary criteria (e.g., language, or geographical location). Figure 2 shows a portion of a sample categorization hierarchy. Here, to group sources by content, we create the types ImageArchive and MedDocumentBase. To group sources by geographical location, we create the types USSource and ForeignSource. Multiple inheritance allows us to create a type such as MedImageArchive that is a subtype of both ImageArchive and MedDocumentBase. It also allows a particular source instance to belong to multiple types, such as MedImageArchive and USSource.

As a tool for the user to populate the database with instances of WAISSource, the prototype provides the OSQL function CreateWAISSource. This function

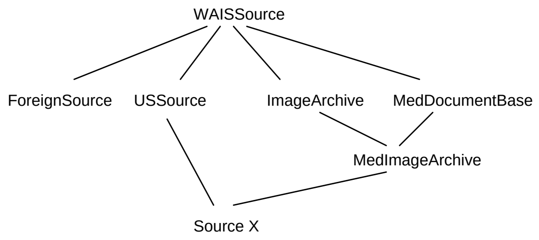

Figure 2: Portion of a sample WAIS sources type hierarchy

parses structured documents retrieved from WAIS directory sources that contain information about other WAIS sources. It extracts attributes such as server name and so on from the documents and then creates a WAISSource object with the attributes. In a typical session, the user searches certain directory sources to retrieve documents describing other sources. After an interesting source is discovered, the user invokes CreateWAISSource with the structured description to create a WAISSource object. The user may then add some annotations to the object and classify it as desired.

With this local categorization of sources, the user may direct a search to a certain portion of the information space. He may further query the characterizing meta information to narrow the search to more appropriate sources. In the following section we will elaborate on these points.

3.2 Declarative Queries

Current client programs provide a very simple mode for querying external sources. The user queries one source at a time, types in a query, and waits for the reply. He then browses through the documents returned one by one. If the search is not satisfactory, he picks another source and repeats the process.

In our prototype information client, we encapsulate the querying of a single WAIS source by a function, SearchWAISSource. It takes a variable number of arguments; the first argument is the WAISSource object to be queried, and the rest are character strings to be concatenated to form the query string. As this function is embedded in the general query language OSQL, the user is able to express far more complex search declaratively. For example,

```
select t for each tuple of WAIS keys t, ImageArchive s
where
    t occurs in SearchWAISSource(s, 'Jupiter');
```

In this simple query, the user specifies a search of image archives known to him for an image of Jupiter. SearchWAISSource is invoked with each source in turn. The results are combined and a set of tuples of keys that identify matched WAIS documents is returned. The user may subsequently request to retrieve the actual documents using these keys. To ask the same query with the current WAIS client, the user would have to select image archives from a stored list of WAIS sources, and submit the query to the sources one by one. He would then look through all the results, merge them together in his mind to identify the most relevant documents.

Although the OSQL query above provides much convenience to the user, an objection that may be raised is that it would generate many queries in case the number of image archives in the local model is large. A more efficient and maybe more effective query would be:

```
select t for each tuple of WAIS keys t, ImageArchive s
where
    Description(s) like 'astronomy' and
    Cost(s) = 0 and
    t occurs in SearchWAISSource(s, 'jupiter');
```

Here the user narrows the search to only free archives that hold astronomy images. To do this in the WAIS client, the user would have to choose sources manually and then submit the query to each source separately.

Sophisticated users may find it flexible to be able to express search needs in a general query language like OSQL. However, for novice users, it is helpful for the database administrator to identify a selected set of useful queries, parametrize them, and develop a graphical user-interface accordingly.

The reader may wonder why the result of the function SearchWAISSource is a set of tuple of keys and not a set of OIDs. To support the latter, when a WAIS document is returned, we would create an object to reference it. However, this is very expensive in terms of both time and storage. Firstly, object creation is an expensive operation and would slow down the retrieval time. To avoid creating objects referring to the same WAIS document we also need to do index look-ups over existing references. This adds to the overhead in retrieval. Secondly, the number of documents returned is generally large, but many of which may be irrelevant. Keeping permanent references to these documents would waste system resources and storage. Thus, the tuple of keys acts as some sort of "transient OID" that references an external object.

On the other hand, to make a reference permanent, we need a function to promote such a transient OID to a normal OID. After identifying an interesting WAIS document, a user can invoke the function CreateWAISDocument to create the permanent link. This function takes a tuple of keys as argument and returns the OID of the WAISDocument object created, whose attributes store the key information. A copy of the document is also permanently stored in the local file system, until the object is deleted by the user. We will further discuss this in Section 3.4.

3.3 Uniform Information Access

In Odapter, the object layer may encompass a number of data sources, both local and remote. Being able to query multiple sources in the same language, the user can uniformly access them and integrate the data. For example, he may make use of data stored in the local database system to form queries submitted to external WAIS resources. To illustrate this, consider the following example. Suppose we have some structured data about employees in the local database. Suppose we also have a subtype PatentDatabase of WAISSource that groups all databases holding patent information. We may ask a query like

```
select t for each tuple of WAIS keys t, Employee e,
        PatentDatabase p
where
    t occurs in SearchWAISSource(p, 'look-and-feel',
        Name(e)) and
    JobTitle(e) = 'Software Engineer';
```

This query searches for patent documents that mention "look-and-feel" and also the name of an software engineer in the local database. Besides joining local and remote data, we may also extract information from a remote source, structure and store it in the local database. We have implemented an OSQL function RetrieveWAISDocument that returns the actual content of a textual WAIS document as a character string, which can then be processed and assimilated into the database. The function is an overloaded function, taking either a tuple of WAIS keys or a WAISDocument object as argument. Details about the implementation of the function is further discussed in the Section 3.4. (A similar function DisplayWAISDocument displays a WAISDocument object using the appropriate application program.)

3.4 Caching Support

In current clients, caching support is limited. For example, in Mosaic, a cache of hypertext documents are kept in visited order. If the user backtracks, the cached documents are displayed. However, the cache is not indexed by URLs (pointers to documents), and thus if the user requests a document by its URL instead of backtracking, the document has to be refetched, even though it may be in the local cache. Further, in both WAIS and Mosaic clients, the user may download a remote file and keep it in the local file system. However, no bookkeeping of such storage is done by the client, and when the user requests the exact same file later on, the local copy is ignored and the file is retrieved from the remote server again.

In our prototype, we have implemented a document cache with Least-Recently-Used replacement policy indexed by WAIS keys. We first discuss below how this cache is integrated with the persistent storage of external documents through the function CreateWAISDocument. Then we look at how cache coherence is maintained.

3.4.1 Interaction with Persistent Storage

Recall that the function CreateWAISDocument is used to promote a temporary WAIS document reference to a permanent Odapter object. When CreateWAISDocument is invoked, it first checks that an object has not already been created for the document. Next it checks the cache for a local copy. If that exists, it is removed from the cache and deposited as a permanent copy in a place we call the repository. Finally a WAISDocument object is created that references (among other information) the local copy. If there was no cached copy, the document was first retrieved from the remote source (via an underlying external function that encapsulates the WAIS client retrieval procedure). CreateWAISDocument provides user-specified caching: the copy in the repository is maintained until deleted explicitly by the user.

Now when the function RetrieveWAISDocument (or DisplayWAISDocument) is called with an OID, the object is retrieved from the repository. If called with a tuple, first it checks if there is a WAISDocument object in the database that has the same keys. If so, the object is retrieved from the repository; otherwise, the local cache is looked up, and if the document is not cached, it is retrieved from the remote source.

3.4.2 Cache Coherence

As in any caching scheme, it is important to maintain consistence or coherence of the cached copy with the actual remote copy. If we made the assumption that a WAIS document was immutable, i.e., the same set of keys always referred to the same document, or the same version of a document, then any cached copy would always be up-to-date and valid. However, there is no constraint on WAIS server administrators to enforce this. We need a way to detect any out-of-date cache content. One exact way is to maintain timestamp or version information about documents. That is, in the key information returned by a WAIS source, there should be a timestamp associated with the document. This information is stored with the local copy. Now when the user searches the source and attempts to retrieve the document using the key information returned, this information (including the timestamp) is checked against the local information. If it is not an exact match, then the new version is retrieved from the source.

This is a good solution, but unfortunately WAIS key information does not contain a timestamp. In our implementation, we use a heuristic to invalidate cache copies. We check for consistency using the document size information, which is in terms of both number of lines and number of bytes; i.e., if the requested document has a size different from the size of the cached version, then we know that it is invalid. Of course, in cases where the new version has the exact same number of lines and bytes, our heuristic fails. However, we believe this is a reasonable solution given the underlying practical limitations.

Note that the above applies when a WAIS document is requested by its tuple of keys. If a WAISDocument object is requested by its OID, then the permanent local copy is returned. The rationale is that, by specifying its OID, the user is requesting to retrieve the particular version of the document that he has downloaded. In case he discovers later that there is a new version of the document, he may delete the old WAISDocument object and create a new one.

To summarize, our scheme is in fact quite simple: we include as key information the size of the document. If the keys of a requested document exactly match those of a cache copy, then the local copy is valid and is returned; otherwise it is requested from the remote source.

4 Related Work

In a previous work [9], we describe the integration of a structured-text retrieval system (TextMachine) with OpenODB/Odapter. There, we focused on issues such as the integration of the text retrieval language into OSQL, schema mapping from the text retrieval system to the database system, and the use of algorithmic OIDs to reduce the overhead of accessing external data sources. Paepcke [8] describes a prototype that provides an object view onto the commercial information service Dialog. It addresses issues such as modeling and schema integration of heterogeneous databases within the service. Our work focuses on areas not addressed there, such as modeling of wide-area resources, uniform access to local and remote data, and caching in wide-area information systems. Hence the two pieces of work can be seen as complementary. In [1], Alonso et al. investigate caching issues in wide-area information retrieval systems. Ideas such as user-specified caching, quasi-copies are proposed and analyzed.

5 Conclusions

Universal access to information is becoming a reality. However, to be better able to use and manage the information accessible, more functionality is needed for the user to map the information space and conduct searches. In this paper we have described several capabilities that the future wide-area information client should provide. We also identify the support needed on the server side, including: (1) the exportation of meta data to characterize sources that are important for source modeling, searching, and query processing; and (2) the enforcement of immutable documents, or timestamping and versioning of documents for maintaining cache coherence.

Acknowledgements Thanks to Umesh Dayal, Wolfgang Demmel, and Thomas Wang for reading drafts of this paper and providing invaluable comments.

References

[1] R. Alonso, D. Barbara, and H. Garcia-Molina. Data caching issues in an information retrieval system. *ACM Transactions on Database Systems*, 15(3):359–84, 1990.

[2] J. Annevelink, R. Ahad, A. Carlson, D. Fishman, M. Heytens, and W. Kent. Object SQL – a language for the design and implementation of object database. In W. Kim, editor, *Modern Database Systems*. ACM Press, 1994.

[3] T. Berners-Lee, R. Cailliau, J.-F. Groff, and B. Pollermann. World-Wide Web: The information universe. *Electronic Networking: Research, Applications, and Policy*, 1(2):52–8, 1992.

[4] C. Clifton and H. Garcia-Molina. Indexing in a hypertext database. In *Proc. VLDB Conference*, pages 36–49, 1990.

[5] National Center for Supercomputing Applications. *Internet Resources Meta Index*. URL http://www.ncsa.uiuc.edu/SDG/Software/Mosaic/MetaIndex.html, 1994.

[6] Hewlett-Packard. *OpenODB/Odapter Reference Manual B3185A*, 1994.

[7] B. Kahle and A. Medlar. An information system for corporate users: Wide Area Information Servers. *Connexions – The Interoperability Report*, 5(11):2–9, 1991.

[8] A. Paepcke. An object-oriented view onto public, heterogeneous text databases. In *Proc. Data Engineering Conference*, pages 484–93, 1993.

[9] T.W. Yan and J. Annevelink. Integrating a structured-text retrieval system with an object-oriented database system. In *Proc. VLDB Conference*, pages 740–9, 1994.