A Metadata Repository API

Patrick Martin, Wendy Powley & Peter Zion
Dept. of Computing and Information Science
Queen's University at Kingston
(martin, wendy)@qucis.queensu.ca
peter@legasys.on.ca

.

Abstract

The Internet provides users with a wealth of data. Much of this data is maintained in passive data sources, that is sources which do not provide facilities to search and query the information. In this paper we describe an approach to querying passive data sources based on the extraction and exploitation of metadata from the data sources. We describe two situations where this approach has been used successfully. The main focus of the paper is on the metadata repository API. We describe the requirements for the repository and show how these influence the API design. We then describe the typical use of the API and explain how it differs from other standard knowledge base APIs. [1]

1. Introduction

For many of us, accessing on-line information has become a common, everyday task. From travel information to scientific data, the Internet provides users with a multitude of data in a variety of forms such as web pages, sound files, program source files, and library catalogs. This data is generally maintained not in database systems, but in what we call *passive data sources*. Unlike a database system, these passive data sources usually lack suitable Application Programming Interfaces (APIs) to search or query the data. Typically the data must be retrieved using "third-party" applications such as browsers or search engines. A topic of particular interest to the research community is how to make effective use of the data stored in these passive data sources. Research in this area includes resource discovery [Bow94, Yuw96], data mining [Etz96] and query tools [Kon95, Men96]

One approach to querying passive data sources is based on the extraction and subsequent exploitation of metadata from the data sources [Mar98]. The metadata includes descriptions of the properties of, and the relationships present in, the data and the data sources. The metadata is extracted from the data sources and stored in a Metadata Repository (MDR) which can be queried to locate data of interest.

In this approach, a metadata schema defines the organization of the metadata. This schema is stored in the MDR. Once the schema is defined, tools are built to extract the metadata from the data sources and store it in a metadatabase then query the repository to facilitate access to the data sources. By querying the metadatabase generated for a collection of passive data sources, users are able to conveniently formulate queries that specify conditions on the structure and the semantics of the data as well as its content.

Key to the metadata approach is the application programming interface (API) which the MDR clients use to access the repository. The API provides easy-to-use functions to connect to, and disconnect from, the MDR, functions to populate the MDR with metadata schemas and metadata, as well as query facilities. The API is the focus of this paper. The structure of the paper is as follows. Section 2 outlines the requirements of the MDR and its API. We briefly examine two applications of the MDR and derive the requirements for the MDR. Section 3 describes the MDR architecture. Section 4 presents the MDR API and illustrates its use. In Section 5 we discuss some related work and compare the MDR API to other approaches. Section 6 summarizes the paper.

Proceedings of the 5th KRDB Workshop
Seattle, WA, 31-May-1998

(A. Borgida, V. Chaudhri, M. Staudt, eds)
http://sunsite.informatik.rwth-aachen.de/Publications/CEUR-WS/Vol-10

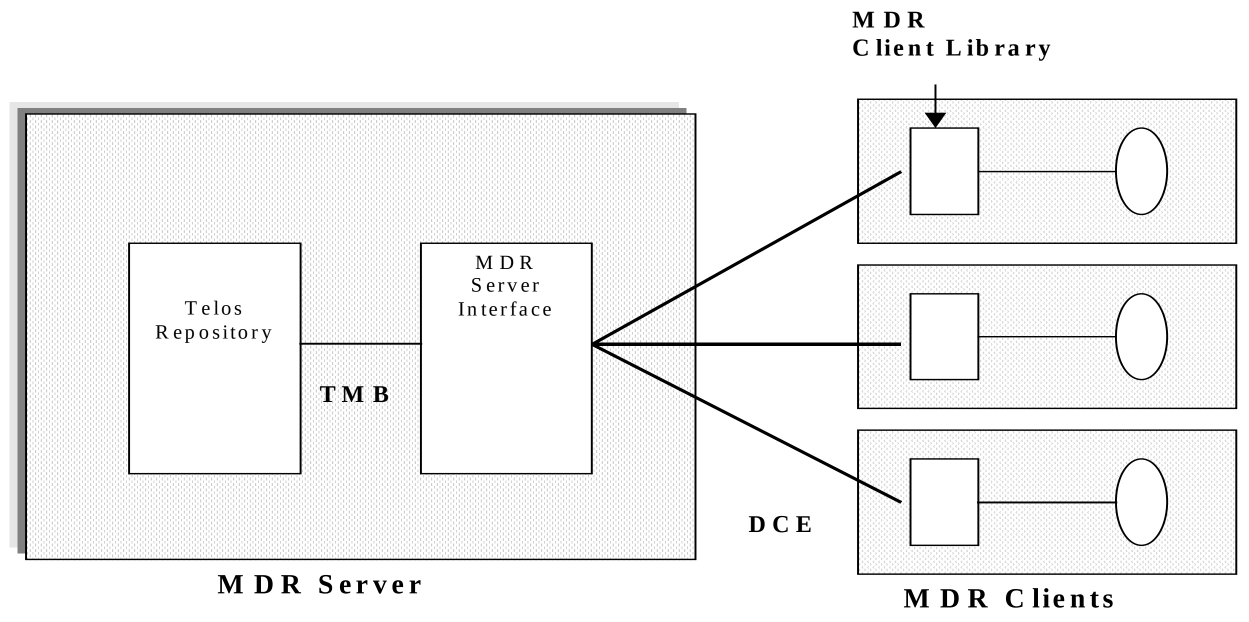

Figure 1: MDR Architecture

2. MDR Requirements

We have applied the MDR technique successfully in two different application areas; distributed application management [Bau97], and querying the World Wide Web [Wes97]. In the distributed application management system, the MDR is used to maintain configuration information. This includes the "code-view" information that describes the static organization and construction of the software making up the application. This information is embedded in the various files that make up the application and in the directory structure holding the files. Tools were built to extract the configuration information from the application files and then to browse and query the metadata that is stored in the MDR.

The second application involves querying the World Wide Web. The Web is the largest example of a collection of passive data sources. Existing query engines use either keyword or full-text indexing techniques. Neither of these techniques capture the internal structure of the Web documents, defined by their HTML tags, or the external structure of the portion of the Web containing the documents which is defined by the hypertext links connecting the documents. The MDR approach provides a tool to handle queries involving both the content and the structure of the Web documents.

When we first set out to develop the metadata repository, we felt that it had to have the following properties:

- An object-oriented data model was necessary to model the complex structures and relationships that exist between the data.
- Unlike a typical knowledge base, the MDR would consist of a small number of classes with a large number of instances. The MDR therefore had to be able to handle large data sets efficiently.
- Queries to the MDR would frequently be based on the relationships between the data (for example, finding similar items) or on attribute values. Query facilities had to be able to process these types of queries. Typical knowledge base interfaces are navigational in nature and do not support the types of queries needed for the MDR.

```
class MMO
{
  string name;                                   // the MMO name
  LinkedList<string> inherList;                  // list of  MMOs from which the MMO inherits
  LinkedList<string> localCatList;               // list of local categories
  LinkedList<string> inherCatList;               // list of inherited categories

 public:
   …
   string Name();                                // returns the name of the MMO
   boolean InherAdd(string inher);               // add a MMO to the inheritance list
   boolean LocalCatAdd(string catName);          // add a local category
   …
}

class MSO
{
  string name;                                   // the MSO name
  string instanceOf;                             // specifies which MMO the MSO is an instance of
  LinkedList<string> inherList                   // the list of MSOs from which this object inherits
  LinkedList<Category> inherCatList;             // inherited categories

 public:
   string Name();                                // returns the MSO name
   string InstanceOf();                          // returns the name of an MMO
   boolean InherAdd(string inher);               // add a MSO to the inheritance list
   boolean LocalAttrAdd(string catName, string attrName, string attrType);
                                                 // add a local attribute by specifying the category,
                                                        the attribute name, and its type
   …
}

class MDO {
  string name;                                   // the MDO name
  string instanceOf;                             //  specifies which MSO the MDO is an instance of
  LinkedList<MDO_Attribute> attrList;            // the attribute list for the MDO

 public:
  string Name();                                 // returns the name of the MDO
  string InstanceOf();                           // returns the name of the “instance of” MSO
  boolean AttrAddString(string attribute, string name );       // add an attribute of type string
  boolean AttrAddInteger(string attribute, long number);    // add a attribute of type integer
  boolean AttrAddReal(string attribute, double number);     // add an attribute of type real
  boolean AttrAddLink(string attribute, string objectname);  // add a  reference attribute
  …
}
```

Figure 2: Class Definitions for MMO, MSO and MDO Objects

- Multiple users should be able to access the repository simultaneously.

These requirements influenced the design of the MDR API. The main purpose of the MDR was to store and query large amounts of metadata. A typical knowledge base interface based on navigation would not suffice. It was necessary to provide extended capabilities to allow formulation of queries based on the contents of attributes as well as on relationships between objects. The API had to provide facilities to manage and examine large result sets.

3. The MDR Architecture

The MDR approach uses a structurally object-oriented model based on the Telos language [Myl90] to define the structures of our metadata model. Telos, being a conceptual modeling language, is particularly convenient for expressing complex relationships among the data.

Figure 1 shows the Metadata Repository architecture. The MDR consists of an MDR Server which is accessed by MDR Clients. The MDR Server has two components: the Telos Repository [Myl90] which provides the back-end knowledge base (KB) for the MDR, and the MDR Server Interface which takes requests from MDR clients and translates them into requests to the KB. An MDR Client also has two components: an application that requires access to the MDR and the MDR Client Library, which is the API to the MDR

The MDR Server communicates with the Telos repository via the Telos Message Bus (TMB). Requests and results are passed along the TMB in the form of strings called *s-expressions* that are parsed and understood by the Telos Repository. Theoretically clients could connect to the Telos Repository directly using the Telos API but we decided to build an intermediate layer, the MDR Server, for the following reasons:

- **Database Independence** – The MDR Server Interface buffers the MDR clients from the specifics of Telos and allows the MDR storage mechanism to be changed without affecting clients' code.

- **Multiplexing** – The implementation of Telos that was used for the MDR prototype supports only a single client. The MDR uses threads to service and coordinate multiple clients, sending one request at a time to the Telos Repository.

- **Extended Query Capabilities** – The Telos Repository provides limited query capabilities. The MDR Server Interface extends these capabilities to include conjunctive queries based on *instance-of* conditions, *is-a* relations, and queries by attribute value. The MDR Server Interface generates a query that can be processed by the Telos Repository. It then sends the request to the repository, and filters the result set to return only objects satisfying the client's query.

4. The MDR Client Library

As stated earlier, the two components making up an MDR client include an application that requires the services of the MDR, and the MDR Client Library, or API, which provides the routines and data structures required to access the MDR. The applications may be generic or application-specific tools. Generic tools are designed to work with any MDR application. Examples include the MDR Browser, the MDR Populator, and the MDR Administration tool. The MDR Browser presents a graphical view of the MDR and allows users to browse both the metadata schema and the metadata. It also provides query facilities. The MDR Populator takes as input a file containing a metadata schema and/or metadata and performs the necessary operations to load the information in the MDR. The administration tool is a command line interface for the MDR that provides the MDR administrator with the basic functions to test and query the repository.

Application-specific tools are designed for a particular purpose and are tailored to a specific application. To date these include the Code View Data Extractor (CDVE) [Lut98], a Web Search Tool, and a Web Query Interface Tool [Mar98]. The CDVE is part of the distributed systems management application. The CDVE automatically extracts the metadata associated with the code-view of a distributed application. The user is prompted for some vital information regarding the application and the file structure of the code making up the application. The CDVE uses key files such as the makefile(s) and IDL file(s) from a CORBA or DCE application to derive the code view metadata.

The Web Search Tool is a combined web crawler and document indexer. Its function is to gather the metadata for a collection of Web documents and store it in the MDR. The user provides the tool with an initial URL and a maximum number of documents to collect. The initial page is processed first then all links are followed and the documents processed in

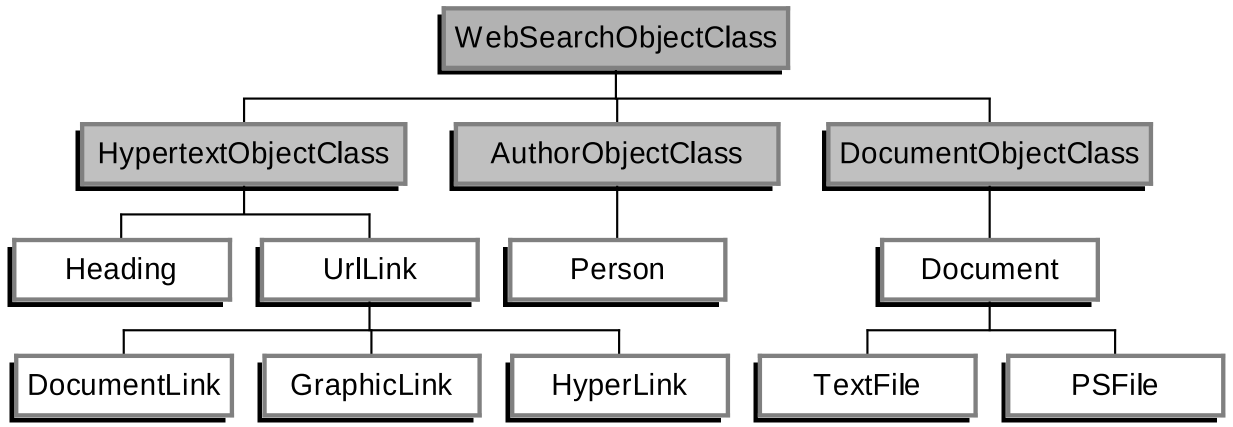

Figure 3: Metadata Schema for the WWW application

turn. Once all the metadata is collected, it is submitted to the MDR. The Web Query Interface tool provides web users with the facilities to query the repository to retrieve information about the documents collected by the Web Search Tool [Mar98, Wes97].

The API comes in two flavours: object-oriented (for C++) and procedural (for C). Clients include the MDR Client Library and header files in their compilation to enable them to invoke the MDR functions or methods. Communication between the clients and the MDR Server Interface is via DCE RPCs [Shi94].

In the API, the **Client** object class represents an MDR client. This class contains member functions for working with the knowledge bases as a whole, and for submitting new objects to the MDR. The **Client** object constructor performs the connection to the MDR, so when a new client is created, the connection to the MDR is established automatically. The destructor method severs the connection to the MDR (assuming a valid connection has been established). Existing metadatabases are listed using the *ListMDB* method. As well, they may be created (*CreateMDB*) and/or deleted (*DeleteMDB*) using the **Client** object class. The *SetMDB* method is used to choose which metadatabase the client will use. The methods to submit objects to the MDR are discussed below.

There are three basic object types used in the MDR to represent the metadata and the metadata schema. These include **Meta Meta Objects** (MMOs), **Meta Schema Objects** (MSOs) and **Meta Data Objects** (MDOs). MSOs are similar to classes in object-oriented programming languages in that they define the attributes that an MDO (data object) can contain. The attributes are further grouped into categories that are defined by an MMO. Each MSO is an instance of one or more MMOs (thus specifying the categories of the attributes) and each MDO is an instance of an MSO.

Partial class definitions are given for the object types in Figure 2. An **MMO** object contains a name and three lists: a list of MMOs from which it inherits, a list of local categories provided by the MMO, and a list of categories which the MMO provides via inheritance. An **MSO** object contains a name, an instanceOf attribute (specifying which MMO the MSO is an instance of) and four lists: an inheritance list (a list of MSOs), a list of local categories, a list of inherited categories, and an attribute list. An **MDO** object contains a name, an instanceOf attribute (specifying which MSO the MDO is an instance of) and a list of associated attributes. The similarities among the 3 types of objects are exploited in the client library interface to make creations, modifications and retrievals as simple as possible.

4.1 API Example

A small portion of the metadata schema for the World Wide Web application is shown in Figure 3. The shaded boxes represent MMOs. All others are MSOs. Sample schema definitions for the objects are given in Figure 4. We use this partial schema to illustrate the typical usage of the MDR API.

The first step in using the MDR is defining the metadata schema. This involves the creation of the MMO and MSO objects. The following shows the creation of two objects shown in the metadata schema in Figure 3: the **HypertextObjectClass** MMO and the **Heading** MSO. The **HypertextObjectClass** is created as follows:

```
// Create a new, empty MMO object called
// HypertextObjectClass
MMO mmo;
mmo.Name = “HypertextObjectClass”;
 // add the inheritance class
mmo.InherAdd(“WebSearchObjectClass”);
// define the 2 local categories
mmo.LocalCatAdd(“hypertextAttributes”);
mmo.LocalCatAdd(“classificationProperties”);
```

The MSO **Heading** is defined by:

```
//Create a new, empty MSO object
MSO mso;
mso.Name = “Heading”;
// add the MMO to the inheritance list
mso.InstanceOf  = “HypertextObjectClass”;
// add local attributes
mso.localAttrAdd(“hypertextAttributes”, “mainHeading”,
             STRING);
mso.localAttrAdd(“hypertextAttributes”,  “subheadings”,
             “Heading”);
```

```
MetaMetaClass WebSearchObjectClass
with
  descriptions: Proposition
end

MetaClass HyperTextObjectClass
 isa WebSearchObjectClass
 with
   hypertextAttributes: Proposition;
   classificationProperties: Proposition
end

class Heading
in HyperTextObjectClass
with
  hypertextAttributes
    mainHeading: String;
    subheadings: Heading
end
```

Figure 4: Sample schema definitions

Heading contains one category, *hypertextAttributes* that has two associated attributes: *mainHeading* (a string), and *subheadings* which is a link to another object of type **Heading**.

Once defined, the objects are submitted to the MDR. MMOs must be submitted before MSOs since MSO objects contain references to MMOs. Objects can be submitted alone or as a list of objects of the same type. If an object contains a reference to another object via a link, the object must either already exist in the MDR, or it must be submitted within the same list. An object is submitted via the **Client** class using the appropriate submit method. There exist two methods for each type of object: one to submit a single object and one to submit a list of objects.

Creating data objects (MDOs) is done in a similar manner. The following shows the creation of an instance of **Heading**:

```
// Create a new, empty MDO
 MDO mdo;
mdo.Name() = “PaperHeading”;
// add the MSO to the inheritance list
mdo.InstanceOf() = “Heading”;
// add the attributes
mdo.AttrAddString(“mainHeading”, “An MDR API”);
mdoAttrAddLink(“subheadings”,                    “Intro”);
mdoAttrAddLink(“subheadings”, “Req”);
mdoAttrAddLink(“subheadings”, “Arch”);
mdoAttrAddLink(“subheadings”, “CliLib”);
mdoAttrAddLink(“subheadings”, “Example”);
```

This creates an instance of the MSO class **Heading** with 8 attributes. The *Name* attribute is a user-defined name for the MDO. The MDR Server converts the name into a unique identifier for each object. *The InstanceOf* attribute indicates that the MDO is an instance of the class **Heading**. The *mainHeading* is a string attribute with the value “An MDR API”. The *subheading* attribute is a link to another **Heading** object. The referenced object is specified using the object name. In this example there are five subheadings. These objects must already exist in the MDR or they must be submitted in a list of MDOs along with the “PaperHeading” MDO.

Although information can be retrieved about particular MMOs and MSOs (such as the categories and attributes they contain), only MDOs can be queried. The **filter** object class is used for MDO queries. A filter is a list of conditions applied conjunctively to the MDOs in the MDR. There are four types of conditions:

- InstanceOf <msoName>. Return all MDOs which are instances of a given MSO.
- IsA <msoName>. Return only MDOs which are instances of an MSO which is a descendant of the named MSO.
- Name <relation> <string>. Returns only MDOs whose name is either equal to or not equal to the given string.

- Attribute <attrName> <relation> <value>. Return only MDOs which have an attribute satisfying the given relation to the given value. There are 4 variants on this type of condition, with one for each data type supported by the MDR (string, integer, real and link). A <relation> is one of {=, !=, <, >, <=, >=) with restrictions depending on the data type being examined. In addition, this filter also supports substring matches.

Consider the following query:

Find all PostScript documents about Repositories with "API" in the title written after 1994

The **Filter** object is created as follows:

```
Filter filter;
filter.AddCondIsA("PSFile");
filter.AddCondAttrString("keywords",
            SUBSTRING_EQUAL, "Repository");
filter.AddCondAttrString("title",
          SUBSTRING_EQUAL, "API");
filter.AddCondAttrInteger("year",      GREATER_THAN,
           1994);
```

Four query conditions are used to specify this particular query. The first "IsA" condition specifies that the document must be a postscript document. The 3 additional conditions are attribute based. The first two require substring matches whereas the last is an integer comparison. The conditions are applied conjunctively to the MDOs in the metadatabase. To submit the query, the user creates an instance of the **QueryMDOsRetList** class which takes as parameters the client id and the filter. Query results are returned to the client as a linked list containing all MDOs that satisfy the filter.

Linked lists are used extensively in the MDR API. Each type of list contains a set of standard functions including First(), Next(), IsDone(), Size() and CurItem(). The list is traversed forwards to allow examination of each item as follows:

```
// create the return list and submit the query
QueryMDOsRetList retlist(client_id, filter);
for (retlist.First(); !retlist.IsDone();
            retlist.Next())
  DoSomethingWithEachItem(retlist.CurItem());
```

5. Discussion

The MDR API is a different type of interface than the standard Telos Repository interface [Myl90]. The standard Telos interface is a traditional knowledge base style of interface. Its retrieval is primarily based on navigation. There is a *currency pointer* that always points to the most recently accessed object in the knowledge base and users navigate the knowledge base by following the relationships in the knowledge base, specifically the *isa* and *instance-of* relationships. Commands are composed as strings and then passed to the Telos Repository.

The MDR API, on the other hand, provides an extensive set of individual operations for interacting with the MDR. Each operation has a specific function and information is passed to and from the operation via parameters. In particular, the MDR API supports querying on the contents of attributes as well as by relationships. It provides a set of operations to build a query filter, which is a conjunction of query conditions. This filter is then applied to the repository contents and only objects that match the filter are returned. The MDR API also provides operations to manage and examine large result sets. These query-by-content features are not available in the Telos Repository and are supplied by the MDR Server Interface component of the MDR Server. The approach the MDR API uses for constructing queries and handling query results is taken from database APIs such as the Open Data Base Connectivity API, or ODBC [Gei95].

It is interesting to compare the MDR API and the Open Knowledge Base Connectivity API, or OKBC [Cha98, Ric98]. Both APIs provide a set of operations that can be used to access a repository/knowledge base system from an application program. They are both concerned with the problem of how tools interact with a system. OKBC primarily addresses tool reusability. The MDR API, on the other hand, supports the development of a wide range of tools for a specific repository system.

OKBC's emphasis on reusability means that it must be able to accommodate a range of knowledge base systems. OKBC's model, therefore, is more general than the MDR object model. OKBC also includes the concept of behaviors, which are not required in the MDR API. There is considerable overlap in the sets of operations present in the two APIs. OKBC, again because of its generality, has a wider range of operations than the MDR API. The MDR API, because of its database requirements, contains the concept of a query filter which can be constructed and then applied with a query. OKBC does not appear to have an analogous construct. It would be an interesting exercise to construct a binding for OKBC to the MDR.

6. Summary

The paper describes the API to the Metadata Repository that has been built to manage the metadata for a variety of applications such as distributed applications management and querying the WWW. The requirements for the MDR have resulted in a system that combines features from both knowledge-based systems and database management systems. The MDR, which is built around the Telos repository, provides an object-oriented API with support for multiple clients, querying based on relations and context, and managing large query result sets.

References

[Bau97] M. Bauer, R. Bunt. A. El Rayess, P. Finnigan, T. Kunz, H. Lutfiyya, A. Marshall, P. Martin, G. Oster, W. Powley, J. Rolia, D. Taylor and M. Woodside. Services Supporting Management of Distributed Applications and Systems. *IBM Systems Journal* 36(4), pp. 508 – 526, 1997.

[Bow94] C. M. Bowman, P.B. Danzig, U. Manber and M.F. Schwartz, "Scaleable Internet Resource Discovery: Research Problems and Approaches", *Communications of the ACM* 37(8), August 1994, pp. 98-107.

[Cha98] V.K. Chaudri, A. Farquhar, R. Fikes, P.D. Karp and J.P. Rice. "OKBC: A Programmatic Foundation for Knowledge Base Interoperability", *appears in the Proceedings of AAAI-98*, Madison, WI., July 1998.

[Etz96] O. Etzioni, "The World Wide Web: Quagmire or Gold Mine?", *Communications of the ACM* 39(11), November 1996, pp. 65-68.

[Gei95] K. Geiger. *Inside ODBC*, Microsoft Press, 1995.

[Kon95] K. Konopnicki and O. Shmueli, "W3QS: A Query System for the World Wide Web", *Proceedings of the 21st VLDB Conference*, 1995, ppp. 54-65.

[Lut98] H. Lutfiyya, A. Marshall, M. Bauer, P. Martin, and W. Powley. Configuration Maintenance for Distributed Applications Management. To appear in the Journal of Network and Systems Management, 1998.

[Mar98] P. Martin, W. Powley and A. Weston. Using Metadata to Query Passive Data Sources. *Proceedings of the 31st Hawaii International Conference on System Sciences*, January 1998.

[Men96] Mendelzon, G. Mihaila and T. Milo, "Querying the World Wide Web", *Proceedings of PDIS '96*, Miami FL, 1996.

[Myl90] J. Mylopoulos, A. Borgida, M. Jarke and K. Koubarakis, *Telos: A Langauage for Representing Knowledge about Information Systems (revised)*, Technical Report KRR-TR-89-1, Department of Computer Science, University of Toronto, August 1990.

[Ric98] J.P. Rice and A. Farquhar. *OKBC: A Rich API on the Cheap*, Technical Report KSL-98-09, Knowledge Systems Laboratory, Stanford University, 1998.

[Shi94] J. Shirley, W. Hu, and D. Magid. OSF Distributed Computing Environment: Guide to Writing DCE Applications. O'Reilly & Associates, Inc, 1994.

[Wes97] A. Weston, Using a Data Model to Search and Query the World Wide Web, Master's thesis, Department of Computing and Information Science, Queen's University, 1997.

[Yuw96] B. Yuwono and D.L. Lee, "WISE: A World Wide Web Resource Database System", *IEEE Transactions on Knowledge and Data Engineering* 8(4), August 1996, pp. 548-554.