Available online at www.sciencedirect.com
SCIENCE DIRECT®
The Journal of Systems and Software 72 (2004) 281–294
The Journal of Systems and Software
www.elsevier.com/locate/jss

Prototyping an integrated information gathering system on CORBA

Yue-Shan Chang [a,*], Kai-Chih Liang [b], Ming-Chun Cheng [b], Shyan-Ming Yuan [b]

[a] Department of Statistics, National Taipei University, 151 Da Xue Rd, San Xia Town, Taipei County 237, Taiwan, ROC
[b] Department of Computer and Information Science, National Chiao Tung University, Hsin-Chu 31151, Taiwan, ROC

Received 6 November 2001; received in revised form 8 April 2002; accepted 12 July 2002

Abstract

The sheer volume of information and variety of sources from which it may be retrieved on the Web make searching the sources a difficult task. Usually, meta-search engines can be used only to search Web pages or documents; other major sources such as data bases, library corpuses and the so-called Web data bases are not involved. Faced with these restrictions, an effective retrieval technology for a much wider range of sources becomes increasingly important. In our previous work, we proposed an Integrated Retrieval (IIR), which is based on Common Object Request Broker Architecture, to spare clients the trouble of complicated semantics when federating multiple sources. In this paper, we present an IIR-based prototype for integrated information gathering system. It offers a unified interface for querying heterogeneous interfaces or protocols of sources and uses SQL compatible query language for heterogeneous backend targets. We use it to link two general search engines (Yahoo and AltaVista), a science paper explorer (IEEE), and two library corpus explorers. We also perform preliminary measurements to assess the potential of the system. The results shown that the overhead spent on each source as the system queries them is within reason, that is, that using IIR to construct an integrated gathering system incurs low overhead.

Keywords: Information gathering and integration; Search service; Information retrieval; CORBA; XML

1. Introduction

With the advances in the Internet and the World Wide Web (for short Web), many more types of information sources are becoming available. Legacy systems are increasingly moving to the Web: finance, education, travel, business, and digital libraries. The sheer volume of information and its variety of sources makes retrieval very difficult. Many well-known search engines, for example Yahoo, AltaVista, Lycos, and WebCrawler, can be helpful but have their limitations (Selberg and Etzioni, 1995; Chang et al., 2000; Meng et al., 2002). None of them individually is sufficient and they suffer poor scalability in searching. Researchers have thus constructed meta-search engines (MSE) for querying more than on search engine at the same time (Selberg and Etzioni, 1995; Dreilinger and Howe, 1997; Chang et al., 2000; Sato et al., 2001; Meng et al., 2002).

1.1. Problems and motivations

In general, MSEs only search Web pages or documents but not other sources, such as Web data bases (Bouguettaya et al., 2000) or library corpuses. The reason is that these other sources are designed using proprietary technology, and general search engines cannot retrieve from them. Searches may encounter query failures or returns of unrelated results when the requested information is not in the Web page of a particular domain client. In addition, an expert user cannot query certain related sources in a unified environment. For example, a student or a scientist can find information about particular keywords from web pages, science paper explorers, and library corpus systems simultaneously. Obviously, general MSEs or Web data bases cannot achieve this objective. Furthermore, not all search engines and query systems support a unified interface and a flexible environment to allow expert

* Corresponding author. Tel.: +886-2-86746782; fax: +886-2-86715907.
E-mail addresses: ysc@mail.ntpu.edu.tw (Y.-S. Chang), smyuan@cis.nctu.edu.tw (S.-M. Yuan).

0164-1212/$ - see front matter © 2003 Elsevier Inc. All rights reserved.
doi:10.1016/S0164-1212(03)00086-4

clients to select and query heterogeneous sources. An effective retrieval technology has thus become increasingly important.

The retrieval of information dispersed among multiple and heterogeneous sources requires a general familiarity with their contents and structures, with their query languages, and with their location on existing methods. In addition, each type of information source has its own proprietary protocol over standard transport protocol. A client that wants to retrieve information from those information sources has to follow their protocols as well as the standard transport protocols. Therefore, the development of software to ease the integration and interoperation of existing information sources is one of the most significant challenges currently facing computer researchers and developers.

Many systems, such as Manifold (Levy et al., 1996), InfoSleuth (Bayardo et al., 1997), TSIMMIS (Garcia-Molina et al., 1995, 1997), OBSERVER (Mena et al., 1998, 1999), SIMS (Arens et al., 1993, 1996a,b), and WebFINDIT (Bouguettaya et al., 2000), gather and integrate sources, no matter distributed, heterogeneous, or autonomous. Most, however, share one or more of the following problems:

- They were developed with proprietary technologies, which is a deterrent.
- They do not support an Applications Programming Interface (API), or, when they do, the programmer needs to spend extra effort on information delivery over the network. (An industrial standard and modularized programming environment helps an applications developer to create elaborate applications.)
- They do not provide flexibility, extensibility, and scalability whenever their space dynamically changes.
- They do not provide metadata management of sources during retrieval and extraction. (Systems should provide for accessing and maintaining the metadata of sources to guarantee the consistency of the ontology and to provide a robust updating capability for sources.)
- They lack a unified API. (Being much more informed about sources and interfaces, the service provider is better placed than the system developer to use source wrappers.) This is not easily achieved with existing systems.

1.2. Objectives

In a previous paper, to solve the above-mentioned problems, we proposed an Integrated Information Retrieval (IIR) framework (Chang et al., 2001) based on Common Object Request Broker Architecture (CORBA) (Object Management Group, 1998). This helps a client avoid complicated semantics when handling a number of sources in parallel.

Mediator/wrapper architecture is a popular means for handling heterogeneous web documents. A query picked up by a mediator is conveyed to a target by a wrapper, which then translates the result into a structure specified by the client. Every source connected to the system has its own wrapper. The codes of a wrapper are tightly coupled with the format or structure of a specified source.

In this paper, we present an integrated information gathering system (InGa) based on IIR for retrieving information from heterogeneous sources. The prototype offers a unified interface for querying sources having their own interfaces or protocols, and uses SQL compatible language to query heterogeneous backend targets. In addition, it can easily link with other sources because IIR provides a flexible, extensible, and scalable framework. We link two general search engines (Yahoo and AltaVista), a science paper explorer (IEEE), and two library corpus systems.

Its major advantages are as follows. First, it is based on a CORBA that is a distributed object-oriented environment of an approved standard. We integrate the three above heterogeneous sources with a unified client and application programming interface.

Second, the CORBA model is of a standard widely accepted by the industry and, therefore, by clients and programmers. This fact recommends it to application developers looking for an integrated retrieval system. The unified programming interface easily allows service providers, who are far more informed about sources, to use wrappers in the CORBA environment and speeds up system development. The system, therefore, is scalable.

Third, each of the sources mentioned has its own query language, schema, and attribute, which means that InGa must be extensible and integrate any future variety of sources. Our extensible environment permits source providers to define their own query interface and schema in a well-known object model and language, so that application programmers using the IIR framework need not query the source interface.

Finally, we adopt structured query language (SQL) for transparently querying heterogeneous sources. We do this not only in the relational database but also in HTML-based and XML-based documents. InGa can combine references seamlessly to the Web with references to the relational database.

The paper is organized as follows. Section 2 describes other gathering and integrating systems. In Section 3, we examine query language and design issues relating to IIR architecture. Section 4 evaluates describes the use and performance of the system. Section 5 discusses the application of IIR and the future work. Section 6 presents conclusions.

2. Related works

InGa is basically a gathering system. Broking systems are solutions for data sharing in a large and dynamic space. We now discuss a range of other gathering and integration systems.

The Information Manifold (IM) is a project based at AT&T Bell Laboratories (Levy et al., 1996). It provides uniform access to collections of heterogeneous sources on the WEB and a high-level query system that describes the content and capabilities of the sources. In addition, it uses declarative descriptions of the sources, and applies algorithms that prune them for a given query, generating executable query plans. IM also applies a conjunctive query that specifies which data of the global schema are represented by the source to query the backend sources. The query processing algorithms that allow such transformations to be accomplished take into account the query processing capabilities of the different sources. The main components of IM are the domain model, the plan generator, and the execution engine. The domain model is the knowledge base. The contents and capabilities of the sources are actually described as queries over a set of relations and classes.

InfoSleuth is a Microelectronics and Computer Technology Corporation (MCC) research project (Bayardo et al., 1997) that draws on work carried out by the Carnot project (Singh et al., 1997). It is designed for retrieval and processing in a dynamic environment such as the Web. Its functionalities include gathering from databases and semi-structured (Suciu, 1998) sources distributed across the Internet, polling and notification for monitoring changes in data, and analyzing gathered information. The system has a collection of communicating agents, each taking responsibility for a particular aspect of the system, for instance, the client agent to control interaction and the resource agent to wrap a source. A broker agent controls the integration of sources and a resource agent advertises their particular content. Infosleuth uses a frame-slot data model with standard data types: integer, float, string, date, frames, and relationships. Its emphasis is on being able to add new sources, which have local autonomy. A so-called dynamic binding of resources allows the broker agent to invoke any currently available source.

OBSERVER is associated with the InfoQuilt project at the University of Georgia (Mena et al., 1998, 1999). It handles broking in global systems. It supports queries over existing sources, where each source is associated with an ontology that describes its contents. Hence, each data repository is attached to one or more ontologies, and clients formulate queries using terms relating to the selected ontology. The data are stored in a repository that comprises several sources. OBSERVER can thus be used either to interoperate existing ontologies or to integrate sources, first by using the ontology to describe the contents of the source, and then by relating one ontology to another. Its approach for broking or for mapping between ontologies and underlying sources is not straightforward.

The University of Southern California SIMS project (Arens et al., 1993; Arens et al., 1996a; Arens et al., 1996b) has a mediator that supports declarative querying of heterogeneous sources with the purpose of integrating them. The integration is based on Loom knowledge representation language. Its architecture is similar to tightly coupled federated databases.

The Stanford University TSIMMIS project (Garcia-Molina et al., 1995, 1997) has a framework and a collection of tools to assist the integration of sources that are expected to have not regular schemes but volatile and highly variable content, structure and accessibility. Its architecture belongs to a hierarchy of simple mediators using substantial wrappers. The automated generation of both mediators and wrappers is from simple high-level descriptions of their functions. In TSIMMIS, queries are expressed through a web-based object browser, MOBIE, or directly in LOREL, an OQL-based query language. In addition, TSIMMIS uses a common "lightweight object model" termed the Object-Exchange Model (OEM) to express queries and to communicate between mediators and wrapper. The objects they export are not required to produce uniform objects. The declarative Mediator Specification Language (MSL) used for queries is the specification language for mediators and the query language for wrappers. TSIMMIS primarily focuses on the semi-automatic generation of wrappers and mediators that allows integration and access to underlying sources when processing OEM-based queries.

WebFINDIT project (Bouguettaya et al., 2000) is used to describe, locate, and access data in large networks of databases, especially for a World Wide Database (WWD). It aims at making the Web a friendly platform for accessing and managing heterogeneous and autonomous databases. Data organization, discovery, and sharing are facilitated through coalitions and service link concepts. In WebFINDIT, a codatabase is linked to and stores relevant metadata on sources. In addition, it offers a variety of tools to locate the sources of interest. The WWD-QL language proposed in the project, whose purpose is to educate clients about the available space, enables a database to know what other databases contain without violating their autonomy, to locate and access data in large networks of databases, and to maintain the codatabases.

Below, we summarize and compare the above systems in detail. The matters we are concerned with are Data Model, Query Language, Client Interaction, Source Integration, Wrappers, and Sources. Table 1 shows the comparison.

Table 1
Comparison of selected systems

	Data model	Query language	User interaction	Source integration	Wrappers	Information sources
Information manifold	Relational model with some object-oriented features	Conjunctive queries	Query	Rewriting	Database	Structured and semi-structured data
InfoSleuth	Frame-slot data model with standards data types	SQL or graphical interface for a particular domain	Query	Model matching, rewriting	Relational database, flat file	Structured and semi-structured data
OBSERVER	OBSERVER'S domain model with some object-oriented futures	Description logic expressions	Query	Rewriting	Database, flat file, process system	Structured and semi-structured data
SIMS	SIMS' domain model (object-oriented)	Loom query language	Query	Rewriting	Database, process system	Databases or knowledge bases
TSIMMIS	Object exchange model	LOREL for OEM objects	Browse, query	View creation	Database, flat file, process system	Structured and semi-structured data
WebFINDIT	CORBA object model	World wide database query language (WWD-QL)	Browse, query	View creation	Database, process system	Structured

We deduce the following:

1. The ends and means for achieving data sharing are distinct, so that wrappers and sources are developed for heterogeneous sources.
2. The data model in Table 1 shows that all the systems except WebFINDIT have proprietary approaches.
3. Query languages are diverse, with no system using a well-known standard.
4. All systems support data sharing for browsing and querying of interested sources only.

In addition, most systems support GUI (Graphical Client Interface) for querying rather than API programming serviceable applications. In the next section, we present API-supported retrieval based on CORBA.

3. Overview of integrated information retrieval

A flexible architecture and framework improves transparency of access, and the scalability and extensibility of a system. A system developer needs a uniform access interface as well as a unified data model to represent query results. The design of our IIR architecture and interface follows the style of the Common Object Service Specification (COSS) of CORBA. All the IIR components are used as CORBA objects. With IIR, by invoking a specific object based on the CORBA object model, a standard query interface is quite enough for heterogeneous sources. They can be wrapped into a CORBA object. Any source can at any time be dynamically linked into such a system. The IIR framework is briefly described next. For further details, please refer to (Chang et al., 2001).

3.1. IIR architecture

IIR architecture is simple but complete. Fig. 1 shows that it comprises *InformationRetriever*, *MetaData*, *Wrapper*, and *Collector*. A client program first obtains an *InformationRetriever* object from a *Factory* object and then sends a query request to the source. The client queries the source by invoking *InformationRetriever*, which acts as a mediator that sends the query to the source wrapper and then retrieves the result. *InformationRetriever* activates one or more corresponding wrapper(s) according to the query string involved in the parameter of the request.

Metadata management and the extensibility and scalability of the system are important supporting features. The purpose of *MetaData* is to make the federation of heterogeneous sources as simple as possible. The client can obtain a description of sources by checking their metadata in IIR, as follows. First, the client queries *MetaData*, constructs a world view and formulates a query string if it is not familiar with the schema and the semantics of the source. Second, metadata is the ontology for the source. *InformationRetriever* and the *Wrapper* share metadata in querying the source and in translating the content of the query. *InformationRetriever*, on receiving the query, refers to metadata to judge the query string and determine the relevant wrapper. Finally, IIR is needed to enable management of the metadata when the source dimension changes, that is, for example, for adding or deleting a wrapper of a source. Obviously, using IIR to query sources means that neither the query sequence in the client program nor the wrappers in IIR need to be changed. *InformationRetriever* refers the metadata to translate query string and judges the meaning of the query sequence. IIR therefore has extensibility and scalability.

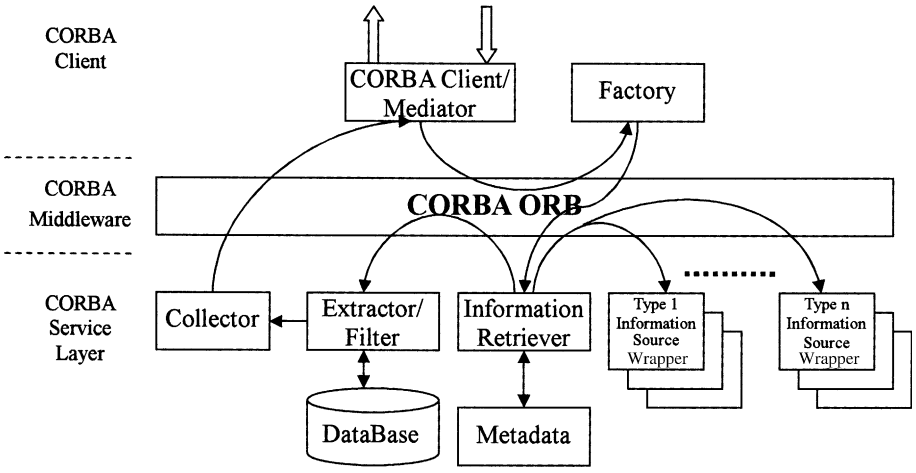

Fig. 1. IIR architecture.

Wrapper is responsible for translating the query request into a format supported by the source, and local system data results to the IIR system. The wrapper is used as an agent that acquires demanded information from those sources, which, if heterogeneous, are filtered. The wrapper activates the *Filter* object according to the source. The result is packed into a standard format, for example XML, and put into *Collector*, which then translates it into an export view. Thus, IIR supports a unified invocation approach for querying sources and obtaining results.

A further clear advantage to our system is that programmers can use the wrappers in their own applications and enable the applications to provide a search service. Because we have a uniform interface-IIR, adding other agents into our system is made easy.

3.2. Query language

SQL is the most popular language for querying relational databases and Web-based documents (Spertus and Stein, 2000; Deutsch et al., 1999; Konopnicki and Shmueli, 1998). Spertus and Stein (2000) demonstrate its many merits. We decided to adopt the SQL as the query language rather than invent a new one. In this way, the query language of IIR provides programmers with the illusion that the sources are stored and organized in a relational database.

Generally, because they have a single interface to query inner data via a Common Gateway Interface (CGI) program, Web-based documents or processing systems involve a table. This is so even if they have heterogeneous backend physical databases. Examples are a search engine or a biographical query system. We can assume that all such systems contain only one table. The table name is defined as the service name in the InGa. Usually, wish to obtain the title, description, and URL associated with a specific keyword. In this way a search engine can be seen as a single table database with three fields, even if most search engines consist of many backend physical databases, and clients can query specific keyword through the table. Not all sources comprise only one table in their database. The service provider identifies the table number and name in a source.

We now show examples of how the query language is used to make a query. It is assumed that the engine schema involves *description*, *title*, and *URL*. Each engine has its own search conditions, which form the WHERE clause of the SQL language.

1. //Show the content about the "CORBA" from Yahoo

   SELECT * from Yahoo Keyword = "CORBA";

   Here, query is invoked to Yahoo's wrapper to obtain all useful (*description, title*, and *URL* that is represented "*") about the keyword "CORBA". IIR accepts and dispatches the request to the Yahoo wrappers.
2. //Show the URL about the keyword "MP3" that is in the tag "text" from Altavista

   SELECT URL from Altavista WHERE Keyword = "MP3" and Tag = "text"

   The query here is invoked with a specific condition excepting the keyword condition. The condition Tag means that the keyword is placed in the "text" of the Web document.
3. //Show the Title and URL for "Programming Language" and "Object-oriented" from Yahoo and Altavista.

   SELECT title URL from Yahoo and Altavista WHERE keyword = "Programming Language" and keyword = "Object-oriented";

The query here is invoked in Yahoo and Altavista in parallel. IIR accepts and dispatches the request to the Yahoo and Altavista wrappers and the results, when obtained from the wrappers, are merged.

In addition, combining the second and third examples creates a problem in which a condition might conform to the rule of only one of the search engines. For example, a client may query a certain keyword placed in the "Anchor" tag from Altavista, but may not be able to do so with Yahoo.

A search engine is a type of Web query system that has no manifest schemas of sources. The query language used in one can also be used in other Web query systems that can be abstracted by SQL syntax, such as Squeal (Spertus and Stein, 2000), while tying these systems to IIR. Similarly, other Web-based sources with manifest schema, such as XML document (Deutsch et al., 1999), can also use the query language.

4. InGa system implementation

This section describes the InGa in use and evaluates its performance. The description includes the development environment and the use of *InformationRetriever*, *Wrapper*, *Metadata*, and *Collector*. Finally, measurements of the heterogeneous components of the system for querying sources are made.

4.1. System overview

Our InGa involves three source agents implemented as CORBA objects, but by the similar way easily allows us to add other agents to the system.

Fig. 2 illustrates its architecture. The system can tie a number of sources into the meta-search engine because it uses IIR IDL (Interface Definition Language), a single, unified interface. It receives and dispatches a query to a number of search engines in parallel, and collates the returns.

In this system, a Web client posts a query request via Common Gateway Interface (CGI). The CGI then forks a dispatcher for each request. The dispatcher does the following: First it creates a number of threads to perform the query. Second, it collates, merges and filters the returns from the agents. Finally, it returns the results to the client.

4.2. Query support

InformationRetriever is responsible mainly for dispatching a query to wrappers to query backend sources. It also determines the sources from the query request issued from client, invokes relevant wrappers, retrieve the object references of wrappers, and retrieve the result from wrappers.

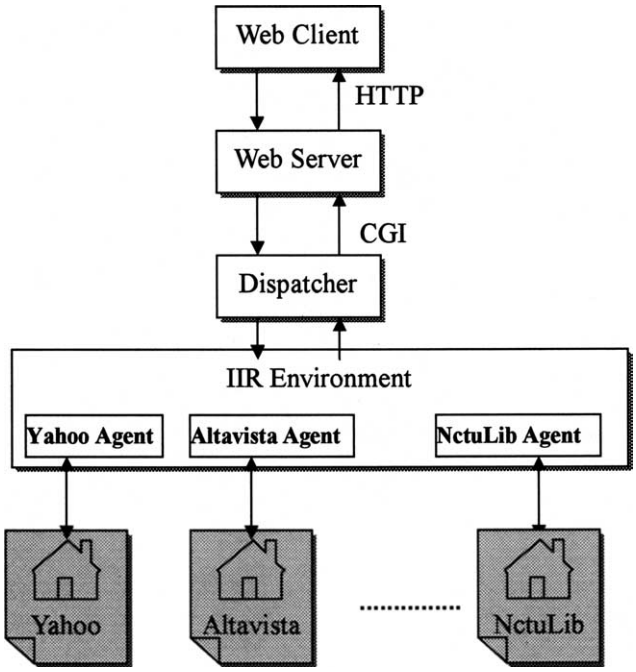

Fig. 2. InGa system architecture.

It is also possible that a request may include an implied query that hiding the target from the query context, in which case *InformationRetriever* cannot determine the query target from the query context directly. It needs to parse the query content to determine which wrappers are invoked.

Clearly, *InformationRetriever* has important tasks. The following shows two ways in which *InformationRetriever* can retrieve the object reference of a wrapper and directly invoke the query sequence. In the first, a client designates an explicit target; we call this a "single query". In the other, the client may not be explicit about a target, or the query may contain a number of targets; we call this a "compound query". The following explains how the two are handled.

4.2.1. Single query

A client designates a target for which a target wrapper is retrieved and invoked. Fig. 3 shows how *InformationRetriever* prepares the object reference of the wrapper. The sequence in the figure does not influence the client query sequence. First, *InformationRetriever* parses the query string from the client and builds the parse tree of a query string. The details are described in the Section 4.3. It uses the tree to determine the wrapper target and retrieves its definition and schema. Next, it parses the target definition and schema and compares them with the previously built ones to determine the correctness of the query content. If the content now agrees with the said definition and schema, *InformationRetriever* then invokes the target's wrapper and returns it to the client. If not, an error message is returned.

```
Algorithm for preparing a wrapper in InformationRetriever
Begin:
    parse query string and build parse tree;
    determine the backend information sources;
    get definition of information source;
    get schema of information source;
    Parse the schema and definition of information source;
    If (the parse tree of query string conforms to the definition and schema of information source)
        invoke object and get the object reference of wrapper of I.S.;
        return object ref. of wrapper of information source;
    else
        return error message;
    endif
endbegin
```

Fig. 3. Algorithm for preparing a wrapper in *InformationRetriever*.

4.2.2. Compound query

In the case of a compound query, a client issues the *invoke_query()*. *InformationRetriever* parses the query string to determine which targets are to be queried, receives the query string and takes the necessary steps to invoke the wrappers of the relevant targets. Fig. 4 shows the algorithm for invoking the query sequence.

The top of algorithm has the same sequence as in Fig. 3. These first five steps are used to find the wrapper of a source and build the desired data structure for it. Then *InformationRetriever* uses a comparison to judge whether the query content conforms to the definition and schema of target wrapper. Next, *InformationRetriever* invokes all the relevant wrappers and retrieves the results from them. Finally, *InformationRetriever* packs the result into an export view and returns that to the client. A detailed description is given in the following section.

4.3. Query planning

In the previous section, we showed *InformationRetriever* handling two types of queries. Now we discuss the use of the query plan of the source. This entails parsing the query content, building the parse tree, determining the target and retrieving the source. We demonstrate the capability of IIR by showing its query sequence and by using search engines as backend sources. This can be best done by the use of examples.

When *InformationRetriever* is used for a request by invoking *prepare() or invoke_query()*, the query string is as follows:

"**SELECT URL**, **Title FROM Yahoo WHERE Keyword** = "**MP3**"

Fig. 5 shows that *InformationRetriever* first parses the query string, then builds the parse tree and retrieves the metadata from the Metadata object. Fig. 6 shows that the definition, schema and parse tree building are included. The "Condition" subtree in Fig. 5 is a subset of the "Attribute" subtree in Fig. 6, and the "Construction" subtree is a subset of the "Schema" also in Fig. 6. The content conforms to the Yahoo metadata. Next, *InformationRetriever* retrieves the object reference of the Yahoo wrapper and then invokes the query sequence.

InformationRetriever first follows the query plan phase. Then it accepts a request, initiates the sequence for a compound query, invokes the relevant wrappers, retrieves the object reference and returns it to client. In IIR, all wrapper objects have the same interface. *InformationRetriever*, regardless of which type of query it

```
Algorithm for invoking the query operation in InformationRetriever
Begin:
    parse query string and build parse tree;
    determine the backend information sources;
    get definition of information source;
    get schema of information source;
    Parse the schema and definition of information source;
    If (the parse tree of query string conforms to the definition and schema of information sources)
        invoke object and get the object reference of wrapper of all information sources;
        invoke the query operation of all wrappers;
        get result form all wrappers;
        extract the content ;
        if (multiple targets)
            view integration;
        endif
        packed the view and put it into collector;
        return collector;
    else
        return error message;
    endif
endbegin
```

Fig. 4. Algorithm for invoking the query operation in *InformationRetriever*.

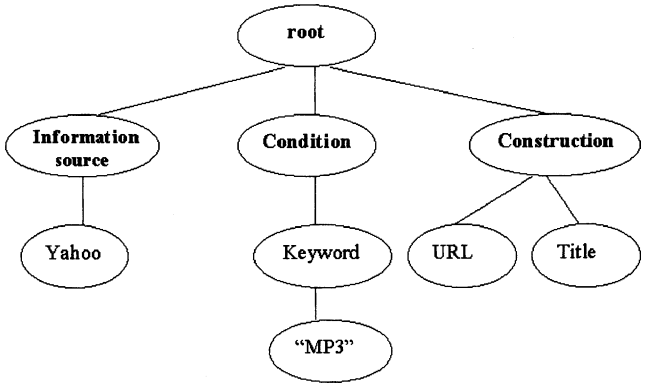

Fig. 5. Parsing tree of query string.

initiated, needs to know which wrapper is in use, i.e. *InformationRetriever* needs to know the relationship between the source and the wrapper in use.

4.4. The development environment

There are now several commercial *ORBs* that are CORBA-compliant: *IONA's Orbix*, *IBM's SOM*, *Sun's NEO*, *and Borland's VisiBroker*. Most support C and C++ mapping. In addition, Orfali and Dan (1997) list some benefits using Java-implemented CORBA objects. Fortunately, some ORB manufacturers define and use Java-IDL mapping to invoke CORBA objects from Java objects, or example, Joe for *NEO*, *OrbixWeb for Orbix and VisiBroker for Java*.

In our system, components are used like in the Borland's (Visigenic) VisiBroker for Java (version 4.5) that is a CORBA client and server ORB written in Java. All *VisiBroker ORBs* make full use of the IIOP, which makes it easy for C++ objects to invoke methods on Java objects, and vice versa. *VisiBroker for Java* supports both static and dynamic CORBA method invocations. Methods on a server can be invoked by Java client applications or by applets within a browser. *Visigenic* intends to fully comply with the *OMG CORBA* binding for Java as soon as it becomes available. The *VisiBroker* IDL compiler generates skeleton code for the server objects in C++ or Java; it also generates Java stubs for the client side.

The version 4.5 Release has been tested on Java 1.3.0 Java Virtual Machines and has been certified for Windows NT 4.0 with both Service Packs 3 and 5, Windows 98, Windows 2000, and Solaris 2.6, 2.7, 2.8. Our choice is Windows 2000.

4.5. Wrapper

A wrapper is an agent that acquires desired data from corresponding sources. It is responsible for translating the form of a client query into the form of the source and form of a result (import view) into the form of the export view. Fig. 7 shows the algorithm for the wrapper. First, the wrapper parses the query string to verify the correctness of its content. This step is same as the action in *InformationRetriever*. Then it changes the form of

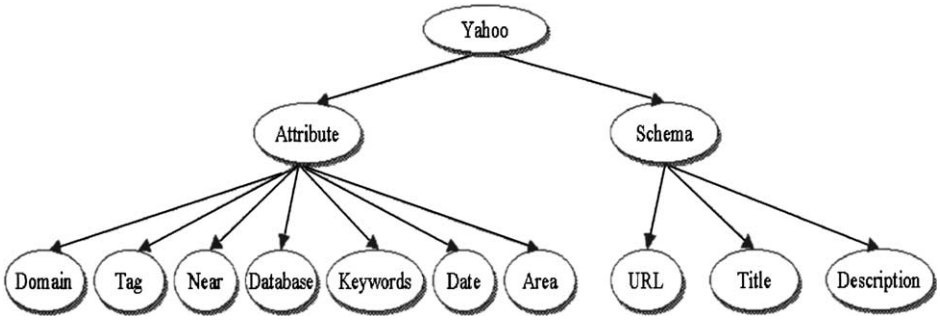

Fig. 6. Example of metadata of Yahoo search engine.

```
Wrapper algorithm for querying the information source
Begin:
    parsing query string;
    transform the query;
    send the transformed query to information source;
    wait for the return and get the result;
    if(time expired or some error occurred )
        return error message;
    else
        view translation;
        packing the result;
        return result;
    endif
endbegin
```

Fig. 7. Wrapper algorithm for querying the information source.

content into the form of the source, sends the change to the source, and waits for the return. Finally, it does the view translation and returns the result to client. If a source crashes, the wrapper returns an error message.

InGa can also link other types of sources, such as relational databases, flat files, and processing systems, by developing and registering corresponding wrappers. The wrapper translates the request into the target's format and invokes the exported interfaces provided by source providers. For example, Z39.50 is a well-known standard retrieval protocol defined by the National Standards Organization and widely used in corpus retrieval applications. InGa thus ties a Z39.50 server by developing a Z39.50 wrapper, which translate client requests into a Z39.50 server and invokes an exported Z39.50's interface (Lo et al., 2000).

4.6. Metadata

Metadata stores metadata for source, similar to the way that a files directory stores files. We can store metadata in an off-the-shelf directory service, such as a Lightweight Directory Access Protocol (LDAP) server, and on that basis use an LDAP adapter in a CORBA object, such as (Burghart, 1998). LDAP is well suited as a gateway between directory clients and distributed objects. Combining it with CORBA provides a secure means for standard off-the-shelf applications to transparently access data in system-specific object schemas at relatively low cost. Because the metadata is formulated in a string, its content is easily stored in a flat file. For speedy use, we store the metadata as flat files in local storage. A future task is to store it in a LDAP server.

4.7. Collector

Collector gathers returns from sources in IIR. It is created by *Wrapper* object when source accomplishes the query request. The *Filtering* object filters the results to avoid repetition of results.

In *Collector*, to speed up coding and for greater simplicity, JavaSoft's [1]-JAXP parses the XML files gathered by *Collector* from all the are *Wrappers* into a W3C's [2] DOM (Document Object Model) tree. The wrappers return results are all formatted as XML files. The DOM tree generated by JAXP provides methods for manipulating and traversing the tree. The sequence in *Collector* is used to traverse the DOM tree and retrieve the content of tree. The system is thus reduced due to the simplified and unified document object model.

Clients can use *Collector* to create an *Iterator* object to retrieve results successively. When a client retrieves a *Collector* object, it can then invoke *createiterator( )* to retrieve *Iterator*. Then, *Collector* builds *Iterator*, *to which it* conveys the root node of the DOM tree. That allows *Iterator* to re-parse the XML file returned from the *Wrapper*. Invoking the JAXP method is simply done, either by *getNextSibling( )* or *getPreviousSibling( )*. Other methods in *Iterator* are same as in *Collector*, i.e., traverse the DOM tree and retrieve its content one by one.

Finally, we explain the use of *Filtering* object. Because all of query results of a *Wrapper* are inserted into a DOM tree, it is necessary to parse throughout the nodes of DOM tree while checking if a URL duplicated or not. The approach of checking duplication we used is to compare the URL in the node of DOM tree is existence in *hash table* or not. If not so, put the URL into *hash table*. Otherwise, delete the node because it is a duplicated result. The method we used to delete a node is *removeChild( )* in DOM API.

4.8. User view

In order to improve the system performance, the system assigns a dedicated thread of a mediator to each query request. Since the mediator sends the request to a number of search engine agents it must create a number of client threads.

The sources here are Yahoo and AltaVista, the IEEE and the two library corpus engines. As Fig. 8 shows, the query forms for them are; Fig. 8(a) Alta Vista with the string **"SELECT TITLE, URL, DESCRIPTION FROM YAHOO, ALTAVISTA WHERE KEYWORD = XML";** Fig. 8(b) IEEE with the string **"SELECT TITLE, PROCEEDING, FULLTEXT_URL FROM IEEE WHERE KEYWORK = XML";** Fig. 8(c) the corpus engines with the string **"SELECT BOOKNAME, AUTHOR, INDEX FROM NCTU_LIB, NTHU_LIB WHERE KEYWORD = database"**. Fig. 8 lists the results in an organized and unified form.

4.9. Performance evaluation

The system uses CORBA/Java and Web technology. The CORBA ORB is VisiBroker for Java version 4.5 and the components use Java language. All the components are run on Windows 2000 with Celeron-800 CPU, 128 M Bytes RAM, and 100 Mbps Ethernet.

In the absence of benchmarking, we did certain experiments to measure critical components. The latency of the components is measured by determining the time each spends on a query, separately measured from Client View, which queries the system, and System View, which relates to Server. Client View measurements are: *Bind Factory*, *Create InformationRetriever*; *Get Metadata*, *Prepare Wrappers*, and invoke query to *remote source* (Yahoo) and to *local source* (National Chiao-Tung University library). The sequence is invoked by

[1] Web.javasoft.com.
[2] Web.w3c.org.

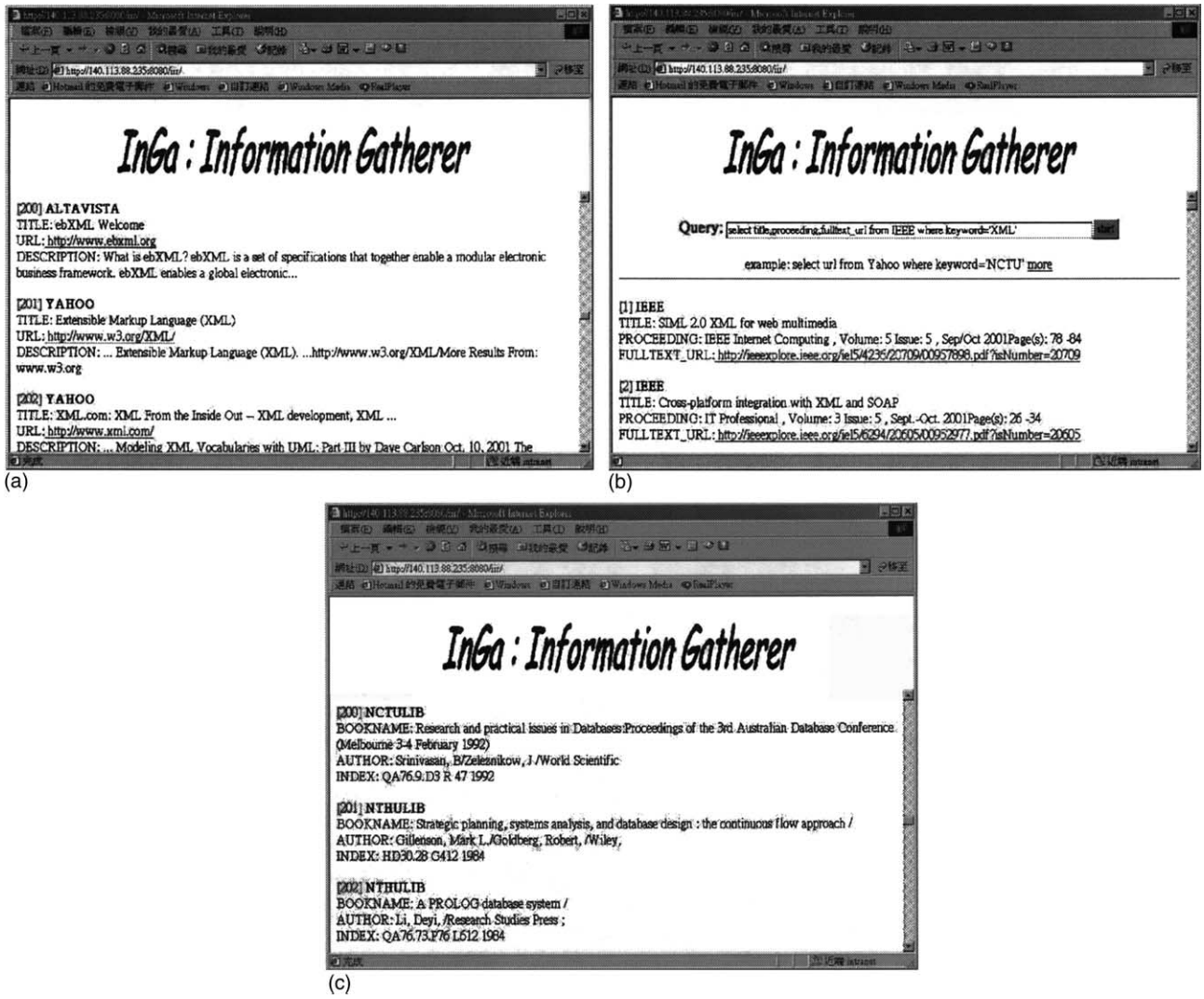

Fig. 8. Search result of our system: (a) the result of meta-search from our system, (b) the result from IEEE explorer, (c) the result of library search from our system.

the client. System View measurements are: *Information-Retriever Parsing*, *Get Metadata*, *Create Wrappers*, *Wrappers execution*, the time spent in *remote and local Source*, *Collector*; and *Filtering*. All the components are used as CORBA objects and are invoked in turn when a request is submitted. The time taken by the components is the overhead of InGa. Data relating to 200 executions of each query were collected. The measurement included the time ORB takes to transmit messages over the network. We evaluate the total overhead in InGa.

Fig. 9 shows the result for Client View, which ties the sources of the Yahoo search engine and the National Chiao-Tung University library, when a request to them was submitted. This measurement for both sources is just 200 searches, although the search engine sometimes exceeded this number. Obviously, when querying a source with IIR in such as a search engine, maximum overhead occurs when a request is submitted and a result obtained from the source.

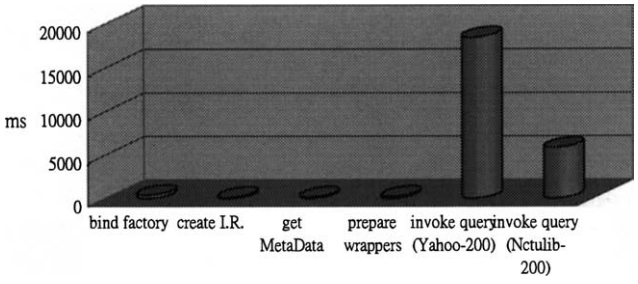

Fig. 9. Client view and one information source.

The result clearly shows that less than 5.2% total overhead is spent invoking certain components, such as binding to *Factory*, creating *InformationRetriever*, getting *Metadata*, and preparing *Wrapper*; in the course of invoking remote sources. The figure for local sources is less than 16.3%. Thus, client programs spend slightly longer invoking remote sources and reasonable extra time invoking local ones.

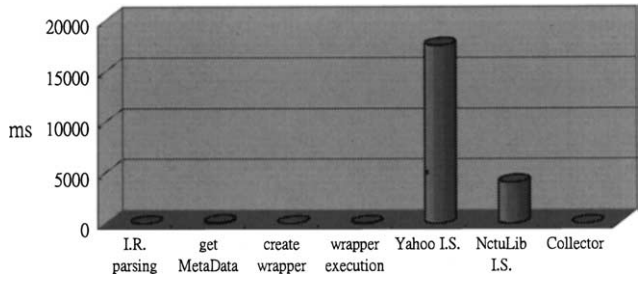

Fig. 10. System view and one information source.

Fig. 10 shows the time that system components spend on executing requests. The overhead spent in querying a source is major on invoking query on the source. It is less than 1% when invoking other components when querying a remote source in the server side. Less than 4.6% is spent on local sources. It is obvious that the overhead is slight. Figs. 9 and 10 shows that an integrated gathering system using IIR incurs very low overhead for both client and server.

Fig. 11 shows the client submitting a query request to Yahoo and Altavista in parallel with the following string: **"SELECT Title, URL, Description FROM Yahoo, Altavista WHERE Keyword = "XML"**. Fig. 12 shows the time that the system components spent executing request and that the difference between the returns is only slight. The record for Yahoo is 200 and that for Altavista 1000. Despite that difference, the records for the different sources do not affect performance. A comparison of Figs. 10 and 12 shows that the only component affected is the I.S., whereas *InformationRetriever Parsing*, *Get Metadata*, *Create Wrappers*, *Wrappers execution*, *Collector*; and *Filtering* do not differ very much. The major overhead is in querying backend targets. In short, the overhead for all InGa components is the same, irrespective of the number of linked sources.

In our system, performance is seriously affected by many components working in parallel. We improve matters by the following strategies: multi-threading and a number of wrappers configured on a number of hosts.

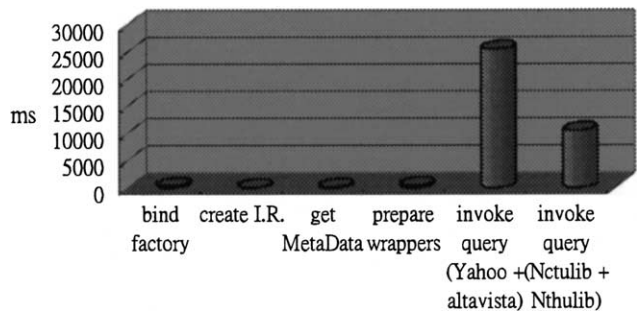

Fig. 11. Client view and two information source.

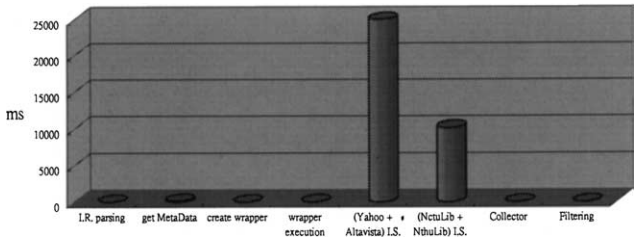

Fig. 12. System view and two information source.

Multi-threaded programming is a well-known technology for improving server performance. In our system, the *InformationRetriever* component and all the *wrappers* are multi-threaded. When an *InformationRetriever* thread is created, it immediately creates a multi-threaded wrapper. We assign each wrapper to a dedicated host. This is easily done in the CORBA environment. Thus, the strategy balances the overhead of our system. We also consider object migration techniques to balance any future system load.

We perform the preliminary measurements shown in Fig. 13 to assess the latency of the system that tying Yahoo and AltaVista search engines, IEEE paper explorer, and two library corpus systems respectively. In the figure, the *XXX* shows the access time of source and the *XXX_OH* shows the round trip time for the packet from system to source. For example, Yahoo shows the access time of the Yahoo search engine, and *Yahoo_OH* shows the packet round trip time for Yahoo. The *Yahoo_* + *Yahoo_OH* show the response time for querying search engines. Clearly, when the system queries those targets, the overhead for the sources is within a reasonable range.

5. Discussions and future works

5.1. Extensions

Although the InGa presented here successfully links the three information sources, the requirements of experts who have different application domains are not satisfied. InGa must be able to tie more. Connecting the sources so that they can be queried by InGa users is easy because we provide a flexible, extensible, scalable framework for the system developer. The first task is to extend the capability of InGa to develop a source-related wrapper object based on the wrapper's IDL of IIR. In general, similar sources have similar query interfaces and protocols. Wrapping the sources needs only the development of corresponding wrappers by changing their source codes. When wrapping different sources, the system developer needs only to follow the query sequence of a source, translate a query into a source-accepted format and invoke a wrapped source

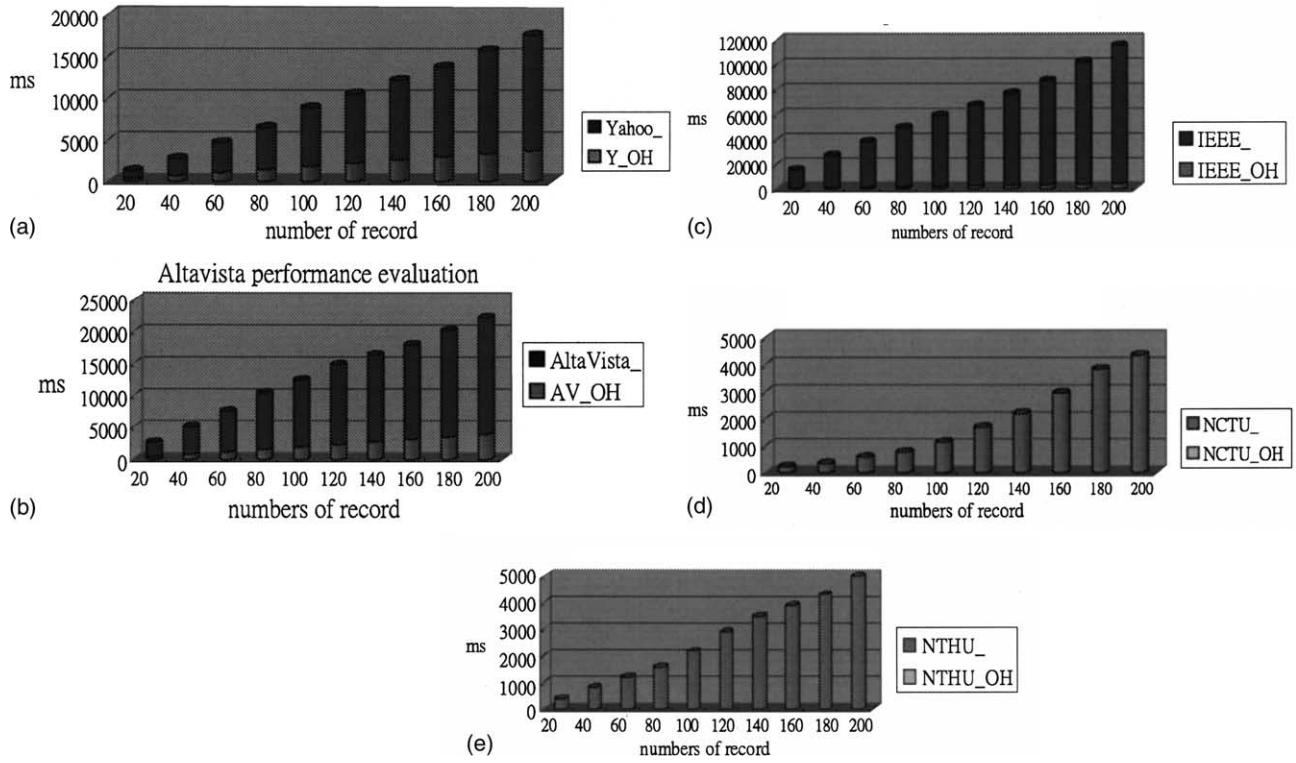

Fig. 13. Performance measurement of InGa system: (a) Yahoo performance evaluation, (b) AltaVista performance evaluation, (c) IEEE performance evaluation, (d) Nctu_Lib performance evaluation, (e) Nthu_Lib performance evaluation.

by a specific interface of a protocol, such as the relational Database. For detail of how to use the wrapper, please refers to Section 4. For the second task, the system developer also needs to define the meta-data of the source in XML's DTD and register it in InGa. The meta-data comprises the interface and schema of the source. Of course, being much more informed about sources and interfaces, the service provider is better placed than the system developer to use source wrappers.

5.2. Automatic wrapper generation

The codes of a wrapper are tightly coupled with the format or structure of a specified source. That is, if the format of the web document changes, the client usually has to modify the codes of the wrapper accordingly. Most web documents change format frequently, and new sources are added apace, so an automated framework for wrapper generation or modification is very helpful.

Mediator-based architecture for retrieval can be used in many ways. Most focus on analyzing and translating results. Many researchers have proposed wrapper generation, focusing on how to translate the structure of sources into representing queries from the mediator, but this is not enough for a retrieval application developer. The developer must have a framework for quick and easy development of wrappers, which must not only be XML-compliant for easy data exchange but also CORBA-compliant for easy heterogeneous communication. Moreover, the client or the retrieval application should be able to extract the useful data from different sources, using a mediator/wrapper framework in high-level query scheme such as SQL.

For most of these approaches, defining a good and structured set of extraction rules is difficult and tedious work for an application developer and requires a good knowledge of web documents. In addition, the returned result is designed for a human user, not for a program. Communication with a program by these approaches increases the workload the wrapper programmer. The solution is a framework for automated XML-based wrapper generation with a unified interface based on CORBA. Here, the XML data model is used to express the metadata of sources, and the output file of the results is in XML format too. CORBA is used to define the uniform interface for the gathering system. With this framework, not only a human user but a retrieval application can communicate with heterogeneous wrappers through the CORBA standard, acquiring results in a popular and structured data model. Furthermore, because XML is a popular standard for data representation and exchange nowadays, the wrapper programmer need not learn a new extraction language for wrapper generation.

6. Conclusions

In this paper, we have presented a prototype based on IIR for integrated gathering (InGa) from three kinds of sources. The system uses CORBA/Java and Web technology. InGa offers a unified interface for querying heterogeneous interfaces or protocols of sources. SQL compatible query language is used to query a number of backend targets. InGa tied two general search engines, IEEE document explorer, and two library corpus systems, can easily tie extra sources, since IIR provides a flexible, extensible, and scalable framework. We demonstrated how to base the system on an approved standard of distributed object-oriented environment.

Compared to the systems listed in Section 1, ours has the following major advantages. First, the CORBA object model is the widely accepted standard, which recommends it to most application developers for integrated retrieval systems. Service providers are far more informed about sources and, because of the unity of the programming interface, they can easily source use wrappers on the CORBA environment and thus speed up system development. The system, therefore, has scalability. Second, it is easy for programmers to build applications that need search ability. Application programmers utilizing the interface to search for information in their application can hide the complexity from network programming and concentrate most effort on other significant value-added services. After the search engine returns the results, the program does not need to extract the information from the complicated format. Because they are all based on the same interface, applications are undiscerning when querying wrappers. Third, each type of source has its own query language, schema, and attribute. With this approach, it is necessary to support an extensible environment that allows integrating a number of sources must be supported. That environment permits source providers to define their own query interface and schema in a well-known object model and language. As for application programmers, in the IIR framework they need not explore query source interfaces when querying. Finally, for transparently querying a number of sources in parallel, we adopted SQL, not only in relational databases but also in HTML-based and XML-based documents. InGa combines references seamlessly to the Web with references to the relational database.

We then performed preliminary measurements to assess the potential of the system. The results shown in Section 4 verifies that the overhead spent on each source as the system queries them is within reason, that is, that using IIR to construct an integrated gathering system incurs low overhead.

Acknowledgements

We are grateful for the many excellent comments and suggestions made by the anonymous referees. This work was supported in part by the Nation Science Council of Republic of China under Grant no. NSC90-2213-E-159-005 and the Ministry of Education's Program of Excellence Research under Grant no. 89-E-FA04-1-4.

References

Arens, Y., Chee, C.Y., Hsu, C., Knoblock, C.A., 1993. Retrieving and integrating data from multiple information sources. International Journal of Intelligent and Cooperative Information Systems 2 (2), 127–158.

Arens, Y., Knoblock, C.A., Hsu, C., 1996a. In: Tate, A. (Ed.), Query Processing in the SIMS Information Mediator Advanced Planning Technology. AAAI Press, Menlo Park, Cliff.

Arens, Y., Knoblock, C.A., Shen, W.M., 1996b. Query reformulation for dynamic information integration. Journal of Intelligent Information System 6 (2/3), 99–130.

Bayardo, R.H., et al., 1997. InfoSleuth: agent-based semantic integration of information in open and dynamic environments. In: Proceedings of the ACM SIGMOD, pp. 195–206.

Bouguettaya, A., Benatallah, B., Hendra, L., Ouzzani, M., Beard, J., 2000. Supporting dynamic interactions among web-based information sources. IEEE Transactions on Knowledge and Data Engineering 12 (5), 779–801.

Burghart, T., 1998. CORBA as an LDAP Server Datastore: an architecture for intranet directory services. http://idm.internet.com/features/corba-ldap.shtml.

Chang, Y.-S., Yuan, S.-M., Lo, W., 2000. A new multi-search engine for querying data through internet search service on CORBA. International Journal of Computer Networks 34 (3), 467–480.

Chang, Y.-S., Ho, M.-H, Yuan, S.-M., 2001. A unified interface for integrating information retrieval. Computer Standards and Interfaces 23 (4), 325–340.

Deutsch, A., Fernandez, M., Florescu, D., Levy, A., Suciu, D., 1999. A query language for XML. In: Proceedings of Eighth International World Wide Web Conference, Elsevier, Amsterdam, 1999.

Dreilinger, D., Howe, A.E., 1997. Experience with selecting search engines using metasearch. ACM Transactions on Information Systems 15 (3), 195–222.

Garcia-Molina, H., Hammer, J., Ireland, K., Papakonstantinou, Y., Ullman, J., Widom, J., 1995. Integrating and accessing heterogeneous information sources in TSIMMIS. In: Proceedings of the AAAI Symposium on Information Gathering, Stanford, California, pp. 61–64.

Garcia-Molina, H., Papakonstantinou, Y., Quass, D., Rajaraman, A., Sagiv, Y., Ullman, J., Vassalos, V., Widom, J., 1997. The TSIMMIS approach to mediation: data models and languages. Journal of Intelligent Information Systems 8 (2), 117–132.

Konopnicki, D., Shmueli, O., 1998. Information Gathering in the World Wide Web: The W3QL Query Language and the W3QS System. ACM Transactions on Database Systems 23 (4), 369–410.

Levy, A.Y., Rajaraman, A., Ordille, J.J., 1996. Query heterogeneous information sources using source description. In: Proceedings of the 22th VLDB, pp. 251–262.

Lo, W., Chang, Y.-S., Chou, C.-L., Sheu, R.-K., Yuan, S.-M., 2000. An Information Store and Retrieval Facility on CORBA. In: Lecture Notes in Computer Science (LNCS), vol. 1846. Springer-Verlag, Heidelberg, Germany, pp. 374–379.

Mena, E., Kashyap, V., Illarramendi, A., Seth, A.P., 1998. Domain Specific Ontologies for Semantic Information Broking on the Global Information Infrastructure. In: Proceedings of the International Conference on Formal Ontologies in Information Systems (FOIS'98).

Mena, E., Illarramendi, A., Kashyap, V., Seth, A.P., 1999. OBSERVER: an approach for query processing in global information systems based on interoperation across pre-existing Ontologies, Parallel and Distributed Databases, 1999.

Meng, W., Yu, C., Liu, K.L., 2002. Building efficient and effective metasearch engines. ACM Computing Surveys 34 (1), 48–89.

Object Management Group, Inc., 1998. The Common Object request Broker (CORBA): Architecture and Specification. v2.2, February 1998.

Orfali, R., Dan, H., 1997. Client/Server Programming with JAVA and CORBA. John Wiley, New York.

Sato, N., Uehara, M., Sakai, Y., Mori, H., 2001. A distributed Search Engine for Fresh Information Retrieval. In: Proceedings of 12th International Workshop on Database and Expert Systems Application, 3–7 September, Munich, Germany, pp. 211–216.

Selberg, E., Etzioni, O., 1995. Multi-engines search and comparison using the MetaCrawler. In: Proceeding of the Fourth World Wide Web Conference'95, Boston, USA.

Singh, M.P. et al., 1997. The carnot heterogeneous database project: implemented applications. Distributed and Parallel Databases Journal 5 (2), 205–227.

Spertus, E., Stein, L.A., 2000. Squeal: a structured query language for the Web. International Journal of Computer Networks 33, 95–103.

Suciu, D., 1998. An overview of semistructured data. In: Vianu, V. (Ed.), Database Theory Column. Sigact News 29 (4), 28–38.

Yue-Shan Chang was born on August 4, 1965 in Tainan, Taiwan, Republic of China. He received the B.S. degree in Electronic Technology from National Taiwan Institute of Technology in 1990, the M.S. degree in Electrical Engineering from the National Cheng Kung University in 1992, and the Ph.D. degree from Computer and Information Science at National Chiao Tung University in 2001. Dr. Chang joined the Department of Electronic Engineering of Ming Hsing Institute of Technology (MHIT) as a lecturer in August 1992. Since from August 2001, he became an associate professor. He now is the Director of Computer Center of MHIT. His research interests are in Distributed Systems, Object Oriented Programming, Information Retrieval and Integration, and Internet Technologies.

Kai-Chih Liang received his BS and MS degrees in computer and information science from National Chiao Tung University, Taiwan, in 1994 and 1996 respectively. He is now the PhD candidate in computer science of the same school. His current research interests include Web technology, distributed object computing architecture, high confidence middleware, enterprise application integration and software engineering.

Ming-Chun Cheng received the B.S. degree in computer and information science from National Chiao Tung University, Taiwan, in 1999. Currently, he is a Ph.D. candidate in the Institute of Computer and Information Science, National Chiao Tung University, Taiwan. His research interests include web technology, distributed system and mobile computing.

Shyan-Ming Yuan was born on July 11, 1959 in Mauli, Taiwan, Republic of China. He received the B.S.E.E degree from National Taiwan University in 1981, the M.S. degree in Computer Science from University of Maryland Baltimore County in 1985, and the Ph.D. degree in Computer Science from University of Maryland College Park in 1989. Dr. Yuan joined the Electronics Research and Service Organization, Industrial Technology Research Institute as a Research Member in October 1989. Since September 1990, he had been an Associate Professor at the Department of Computer and Information Science, National Chiao Tung University, Hsinchu, Taiwan. He became a Professor in June 1995. His current research interests include Distributed Objects, Internet Technologies, and Software System Integration. Dr. Yuan is a member of ACM and IEEE.