ELSEVIER

Computer Standards & Interfaces 23 (2001) 325–340

COMPUTER STANDARDS & INTERFACES

www.elsevier.com/locate/csi

A unified interface for integrating information retrieval

Yue-Shan Chang [a,*], Min-Huang Ho [b], Shyan-Ming Yuan [b]

[a] *Department of Electronic Engineering, Ming-Hsin Institute of Technology, 1 Hsin-Hsing Road, Hsin-Fong, Hsin-Chu 304 Taiwan*
[b] *Department of Computer and Information Science, National Chiao Tung University, Hsin-Chu 31151 Taiwan*

Received 5 March 2001; returned for revision 28 April 2001; accepted 7 June 2001

Abstract

In this paper, we propose a unified interface and a flexible architecture for querying various information sources on the Internet and the WWW using both a popular object model and a data model. We propose an Integrated Information Retrieval (IIR) service based on the Common Object Service Specification (COSS) for Common Object Request Broker Architecture (CORBA) and apply the Document Type Definition (DTD) of eXtensible Markup Language (XML) to define the metadata of information sources for sharing the ontology between mediator and wrappers. The objective of using the IIR design is not only to provide programmers with a uniform interface for coding a software application that can query a variety of information sources on the Internet, but also to create a flexible and extensible environment that easily allows system developers to add new or updated wrappers to the system.

Keywords: Unified interface; Integrated information retrieval; Meta-Search; Object-oriented; CORBA; XML; Interoperability

1. Introduction

1.1. Problems and motivations

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 who 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 have been proposed for services to accomplish information retrieval, that is, the gathering and integration of multiple, distributed, heterogeneous, autonomous information sources on the WWW and the Internet. There are, for example, Information Manifold [1], InfoSleuth [2], TSIMMIS [3,4], OBSERVER [5,6], SIMS [7–9], WebFINDIT [10]. Most, however, share one or more of the following problems.

· Most existing systems have been cheerfully developed with proprietary technologies, a fact, which deters users from applying them.

* Corresponding author. Tel.: +886-3-5591-402; fax: +886-3-5572-930.

E-mail addresses: ysc@mhit.edu.tw (Y.-S. Chang), smyuan@cis.nctu.edu.tw (S.-M. Yuan).

0920-5489/01/$ - see front matter
PII: S0920-5489(01)00083-6

- An industrial standard and modularized programming environment helps an applications developer to create elaborate applications. Programmers using most existing systems find that the systems do not support an Applications Programming Interface (API) and are forced to develop their own programming interface. Even when an API is supported, the programmer needs to spend extra effort on information delivering over the network.
- It is important for a system to have flexibility, extensibility and scalability. There are fewer existing systems that provide the extensibility needed when their information space dynamically changes.
- The ontology of the information sources is shared. The metadata management of information sources is useful when retrieving and extracting information. So that, the system has to provide with accessing and maintaining the metadata of information sources to guarantee the consistency of the ontology and to provide a robust updating capability with information sources.
- To speed up the development of the system, it is far more preferable to have the service provider rather than the system developer implement the information source wrapper. This is because the service provider is more informed about information sources and about the interface. But given the lack of a unified API, this situation is not easily achieved with existing systems.

To transparently access heterogeneous information sources, data suppliers and consumers need to be decoupled by a unified and integrated interface. In this paper, we propose an Integrated Information Retrieval (IIR) for the Common Object Request Broker Architecture (CORBA) [11] of Object Management Group. The design of the IIR interface follows the style of Common Object Service Specification (COSS) [13]. IIR provides mechanisms for separating users from complicated semantics while federating the multiplicity of information.

In addition, certain projects for gathering and integrating information are implemented by agent technology, examples of which are seen in Refs. [10] and [14]. As may be seen in Ref. [15], the distribution of the information gathering leads naturally to a desire for cooperative software agents with distributed problem-solving mechanism for locating, retrieving and integrating information sources. We also applied agent technology to implement the IIR infrastructure. A wrapper associated with an information source is implemented and configured as an agent, and it is through the information-broking agent that all query requests are dispatched to relevant wrappers.

1.2. Objectives

This paper has the following major objectives. First, we propose a uniform interface for information retrieval and gathering in an approved standard of distributed object-oriented environment. This offers a programming interface to retrieve what applications are wanted, and uses agent technology [16] to implement the infrastructure of IIR. Since the architecture of IIR is of an N-tier client/server model and its interface is uniform, any specific application that needs to retrieve information from the Internet or the WWW needs only to initiate the query operation of an agent via IIR. Programmers need neither explore the interface of various information sources nor construct query components in their applications.

Second, each type of information source has its own query language, schema and attribute. With this approach, it is necessary to support an extensible environment that will allow integrating various information sources in the future. We propose an extensible environment that permits source providers to define their own query interface and schema in a well-known object model and language. As for application programmers, it is not necessary via the IIR framework that they explore query interfaces of information sources when querying information. Section 3 will describe the IIR interface with flexible metadata management and data representation.

Third, the Document Type Definition (DTD) announced by the World Wide Web Consortium (W3C)[1] is a popular description language of schema. We apply the DTD of eXtensible Markup Language (XML) to define the schema of information sources,

[1] http://www.w3c.org.

and to provide the interface in the IIR for managing metadata. A client may easily initiate a query operation of metadata to obtain the schema of information sources.

Fourth, due to the unity of interface, a service provider can easily implement a wrapper for their information sources on the CORBA environment and speed up the system development. This is reasonable because the service provider is far more informed about information sources. The system, therefore, has scalability.

In addition, both the object model of CORBA and data model of XML are the approved standards and are widely accepted by the industry and so will be by users and programmers.

Finally, we adopt Structured Query Language (SQL) in the IIR for transparently querying information from various sources on the WWW and the Internet. SQL has been adopted not only in the relational database but also in HTML-based [17] and XML-based [18] documents. IIR can seamlessly combine references to the Web with references to the relational database. Anyone familiar with SQL can create programs using IIR easily.

The challenge is to provide across-the-board transparency that allows clients to use Web-based or Internet-sharing data irrespective of platforms, locations, or systems. There are some difficulties [14] concerning the processing, retrieval, gathering and integration of information on the Internet. Refs. [19] and [14] depict the key challenges to providing consistently convenient access to information sources despite changes in sources and application systems. In Section 5, we explain how the IIR design resolves these challenges.

This paper is organized as follows. Section 2 describes the background to designing and implementing the IIR and surveys the related works. Section 3 mainly depicts the design of the IIR. We examine some design issues including IIR architecture, query language, data model, metadata management and programming scenario. Section 4 presents two applications using the IIR, the first is being a meta-search engine, and the second, an Information Retriever based on the Z39.50. Section 5 discusses the challenges and advantages about applying IIR to deploy the applications, and gives the future work. Section 6 presents conclusions.

2. Background

2.1. XML data model

Recommended by the World Wide Web Consortium (W3C)[2], XML is a subset of Standard Generalized Markup Language (SGML). It is a description language for representing documents, and is defined in a DTD. A DTD defines the types of elements that can be used in the document and the possible relationships between them. An element can be thought of as a kind of container for each distinct thing in a document. It is up to the person who creates the DTD to decide exactly what the "things" are. A document instance has a preferred hierarchy in which there are two kinds of relationships between elements: parent–child and peer–peer.

The DTD can also be used as a specification modeling language, which specify the specification of an information system [20]. The DTD used in this paper is to specify the specification of information source, as shown in Fig. 1. The specification is used for modeling the capability and the output of the information agent. Clients retrieving information from sources can refer to the syntax shown in the specification to formulate the expression of a query. The specification can also be viewed as metadata of information sources in IIR. The metadata will be described in Section 2.2. In addition, the specification will be also a material for generating the information agent in the future. The XML is also used for representing the data of query results from the information sources.

2.2. Related works

Except for those systems mentioned in Section 1, here, we present two CORBA-based information retrieval services: OQS [13] and ZORBA [26].

2.2.1. OQS of CORBA

To uniform the access interfaces for heterogeneous databases, OMG proposed an Object Query

[2] http://www.w3c.org/XML.

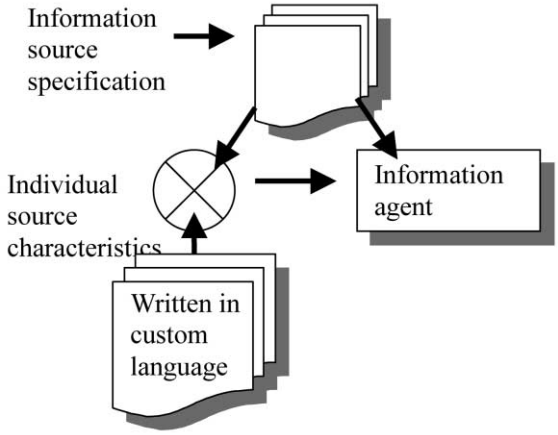

Fig. 1. Specification modeling architecture.

Service (OQS) on CORBA to provide query operations on collections of objects. OQS provides standard interfaces, which form a framework for handling queries, can be sub-typed for further functions and also for returning collections of objects. The OQS design provides the architecture for a nested and federated service that can coordinate multiple nested query evaluators. The OQS covers the following operations.

1. Prepare the query for execution.
2. Execute the query.
3. Determine the preparation and execution status of the query.
4. Obtain the result of the query.

In addition, the OQS provider must support at least one of the following two query languages: SQL (Structured Query Language) or OQL (Object Query Language). According to the above description, OQS covers basic query and result collection operations. It has not stressed the points described in Section 1.

2.2.2. ZORBA

ZORBA [26] is an information retrieval technology using distributed objects. It is intended to produce a standard re-usable interface for performing distributed search and retrieval, and to provide support for all aspects of distributed resource discovery over a variety of information sources available on the Internet using distributed object technologies. It defines a set of interfaces in CORBA IDL. The interfaces only include information query. Obviously, it is similar to the OQS. This interface does not yet deal with the following:

1. sorting query results,
2. metadata schema(s), and
3. query languages.

3. Design of integrated information retrieval

A flexible architecture and framework can improve access transparency, system scalability and extensibility. We dedicate our IIR design to such participants as data providers and information inquirers who can dynamically join the system flexibly, no matter what types of information are concerned. Any information source can at any time be joined dynamically into the system. An IIR client can obtain the information about information sources by inquiring their metadata. A service provider can also replace, access, and maintain the metadata of information sources and provide an adaptable environment. Metadata management and the extensibility and scalability of system critically enforce the IIR. In addition, we propose that the system should be capable of retaining the autonomy of a jointed local query system, that is, that the IIR and local query systems should co-exist.

The following is a detailed explanation of the design of the IIR framework and of the approaches needed to accomplish the objectives stated in Section 1, especially including IIR architecture and interface, metadata management, query language and IIR programming example, etc.

3.1. IIR architecture

IR architecture is simple and complete. From the client's perspective, the requirements are a uniform access interface as well as a unified data model for representing the results of queries. With IIR, clients use a standard query interface to acquire information based on a CORBA object model.

Fig. 2 depicts IIR architecture. It comprises *InformationRetriever*, *MetaData*, *Wrapper* and *Collector*. *InformationRetriever* acts as a mediator for dispatching query requests to the wrappers of information source and collects the results. A client program sends a query request to the information sources by first obtaining the *InformationRetriever* object from a *Factory* object. Due to the access transparency, the operations for querying all information sources using IIR are the same. A client queries information sources by mean of invoking *InformationRetriever*. The *InformationRetriever* activates corresponding wrapper(s) according to the query string involved in the parameter of query operation. It is obvious that clients accessing information sources are completely transparent by invoking an *InformationRetriever* object.

MetaData class is the management of metadata. Its purpose is to minimize the degree of complexity for federating heterogeneous information sources. With IIR, it has three following functions. First, a client can query the *MetaData*, construct a world-view and formulate the query string when it is unfamiliar with the schema and the semantics of accessed information sources. Second, the metadata is the ontology with respect to information sources. The *InformationRetriever* and the *Wrapper* shares the metadata in querying information sources and in translating the content of the query. *InformationRetriever* will refer to the metadata in judging the query string and determining the related wrapper when it receives a query request. *MetaData* provides the ability of access transparency for the IIR. 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 information source. Neither the query operations in client nor the objects in IIR are necessary to be changed. The *InformationRetriever* refers the metadata and judges the meaning of the query operation. Obviously, the IIR have extensibility and scalability. There is a need to have some supporting methods for the management of metadata in IIR.

The *Wrapper* is responsible for translating the query request into the request format associated with the information source and the results from the local system data representation to the IIR system. If the results are from multiple similar sources, they are filtered. The wrapper activates the *Filter* object according to the kind of sources. The result is packed into a standard format, for example, XML, and put into the *Collector* object. Finally, the *Collector* col-

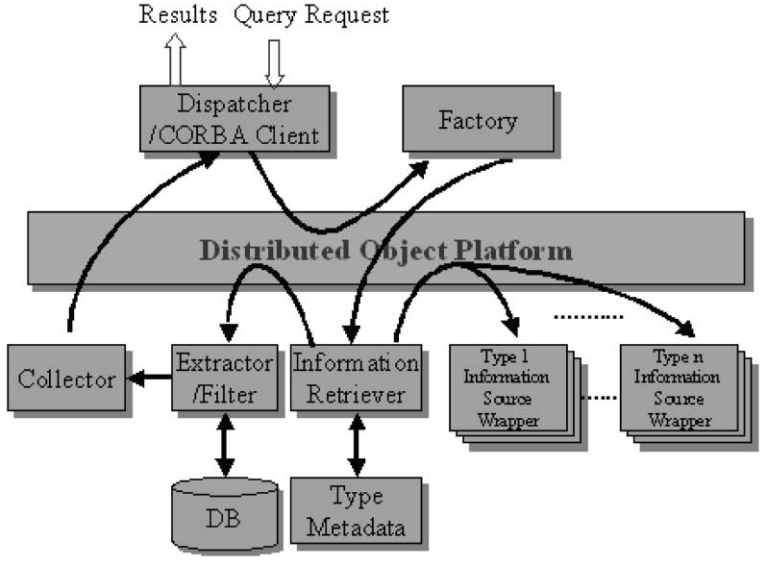

Fig. 2. Architecture of IIR.

lects the result and translates it into export view. For the client, IIR supports a unified invocation approach for querying source and obtaining results.

3.2. Query language

Query language is for formulating the expression of query to a well-developed system. The most popular query language for querying relational databases and Web-based documents [17,18,21] is the SQL. The many benefits of using the SQL as the query language in Web-based documents are listed in Ref. [17]. Owing to these benefits and the integration of various information sources that maybe configured on the WWW or the Internet, 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 information sources are stored and organized in a relational database.

As mentioned above, the purpose of IIR is to integrate the information retrieval of information sources on the WWW and the Internet. It is necessary to clarify the supported sources in IIR. It is obvious that the tied sources must be sources that can be queried in SQL, such as the DBMS (Data Base Management System), Internet- and WWW-based processing system and the Web-based documents, because the query language used in IIR is SQL. For example, the search engine on the WWW and the Z39.50 service mentioned in Section 4. Naturally, SQL can be used on the Internet for any service that can be queried.

As we know, a schema describes the structure of a relational database, i.e. the tables, fields and the relationships between them. For example, the schema for a student database might include a table with the following fields: *first_name*, *last_name*, *student_number* and *address*, where each student has a distinct *student_number*, but different students may have the same *first_name* or *last_name*.

Generally, a Web-based document or a Web-based processing system involves a table, even if it has multiple backend physical databases because it has a single interface to query inner data via the Common Gateway Interface (CGI) program. For example, search engine and biographical query system. Such the systems, we can suppose the whole system contains only a single table. The table name is defined as the service name. For example, users desire to obtain, in general, the title, description, and URL associated with a specific keyword from a search engine. In this way, a search engine can be viewed as a single table database including three fields, even if most search engines consist of many backend physical databases, and users can query specific keyword through the table. Not all information sources certainly do not simply consist of a table in their database. The service provider can identify the number and the name of table in an information source.

3.2.1. Query in search engine

Next, we show examples of how the query language can be used to query the desired data from information sources. It is assumed that the schema of the search engines generally involves *description*, *title* and *URL*. Each search engine has its own search conditions. For example, Ref. [23] shows the search conditions for Yahoo and Altavista search engines. These conditions form the WHERE clause of the SQL language when it is used to query search engines.

1. //Show the information about the "CORBA" from Yahoo.
   SELECT* FROM Yahoo WHERE Keyword = "CORBA";
2. //Show the URL about the "MP3" from Altavista that tag is "text".
   SELECT URL FROM Altavista WHERE Keyword = "MP3" and Tag = "text";
   The example shows that the query operation is invoked with a specific condition excepting the keyword condition. The condition "Tag" means the keyword is placed in the "text" of WWW document.
3. //Show the Title and URL for the "Programming Language" and "Object-oriented" from Yahoo and Altavista.
   SELECT Title and URL From Yahoo and Altavista WHERE Keyword = "Programming Language" and Keyword = "Object-oriented";

Table 1
InformationRetriever interface

```
Interface InformationRetriever{
    MetaData Get_meta(in QuerySourceType);
    Wrapper prepare (in ParameterList pl, in QuerySourceName
                                    qsName, in QueryLanguageType qlType)
}
Interface Wrapper{
    Collector Query() raises(QueryProcessingError, QueryInvalid);
}
```

The example shows that the query operation is invoked in two search engines—Yahoo and Altavista. IIR will accept the request and dispatch it to the wrappers associated with Yahoo and Altavista, respectively. When obtained, the results from two wrappers are merged.

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

Search engine is a type of WWW query system that has no manifest schemas of information sources. The query language that can be applied in the search engine can also be applied in other WWW query systems that can be abstracted of SQL syntax, such as Squeal [17], while tying these systems to IIR. Similarly, other WWW-based information sources with manifest schema, such as XML document [18], can also apply the query language to query information.

3.3. IIR interface

All IIR interfaces are structured according to the IIR architecture described in Section 3.1. In this section, we describe most IIR interfaces.

In IIR, the *InformationRetriever* interface has two methods, which are *Get_meta()* and *prepare()*, as shown in Table 1. The purpose of the *Get_meta()* operation is to obtain the *MetaData* associated with the query language type and return a *MetaData* object reference. The *prepare()* operation is for preparing a query request and obtaining a *Wrapper* object. A client uses this method to send a query string to the *InformationRetriever*. The *InformationRetriever* then makes a decision over which wrapper to invoke through comparing the passed query string with the content of metadata. The *InformationRetriever* then invokes the associated *Wrapper* and

Table 2
MetaData interface

```
Interface MetaData{
    Boolean QL_Available(in QuerySourceType qsType);
    Boolean Regsitry(in QuerySourceMeta metadata);
    Boolean Unregsitry(in QuerySourceName qsName)
    Boolean Replace(in QuerySourceName qsName, in
                                        QuerySourceMeta metadata);
    QuerySourceMeta get(in QueryLanguageType ql_type)
}
```

returns its object reference to the client. The client need not know how to invoke the wrapper. That can be done by invoking a unified information retrieval interface—*InformationRetriever*. Similarly, client can obtain the results from a *Collector* object regardless of what kind the information sources are. Of course, the *InformationRetriever* needs to parse the query string and overlook all the metadata stocked in the *MetaData* object. Section 3.4 explains the metadata management in detail. In addition, the *Wrapper* object has only one method—*Query()*, which is to execute the query request and return the result that are kept in the *Collector*.

The next interface is the *MetaData*, as shown in Table 2. As described in Section 1, the management of the metadata of information sources is useful for retrieving and extracting information. For this reason, the system must have access and maintain the metadata of information sources to guarantee the consistency of the ontology and to provide robustness with respect to updating information sources. This interface maintains the metadata of the information source, and this metadata involves learning about how to make a query and what is in the reply. When a client program wants to invoke a query request to specific source, it can examine the *MetaData* to obtain the format of the query request and the schema of the result, while the client does not need to know about the information source. *MetaData* has five methods. They are: *QL_Available()*, *Registry()*, *Unregistry()*, *Replace()* and *get()*. The *QL_Available()* operation is used to examine the *MetaData* whether a certain type of query source is available or not. The next three methods are in order to provide system extensibility. In IIR, source providers are permitted to define their own interface and schema of the information source. All definitions about the information source have to be registered in the *MetaData* of IIR using the *Registry()* operation. The query interface and schema of the result are represented in XML and presented in Section 3.5. Similarity, a source can be un-registered by the *Unregistry()* operation. Lastly, the *get()* operation is used to obtain the metadata.

The third interface is the *Collector*, as shown in Table 3. This interface collects the query results from the information source. According to the design of IIR, the *Collector* is created by the *Filter* object, which is an inner object of IIR. This interface has three methods. The first is *GetMeta()*, which has the same function as the *GetMeta()* in the *InformationRetriever* interface. Through this operation, clients obtain the schema of the query result. The second method is the *retrieve_element_at()*, which is used to obtain an arbitrary element. In addition, we also design an *Iterator* interface to access the results one-by-one. The client program invokes the *create_iterator()* operation to create an *Iterator* object. The *Iterator* has three methods: *next()*, *reset()* and *more()*. The *next()* operation retrieves the next result of the current record. The *reset()* operation resets the index of the *Iterator* object into the top. The *more()* operation is used to detect whether there are any results in the *Iterator* object.

Table 3
Collector and *Iterator* interface

```
Interface Collector{
    MetaData GetMeta(in QueryLanguageType);
    Readonly attribute long Result_size;
    Result retrieve_element_at(in long where);
    Iterator create_iterator();
}
Interface Iterator{
    Result next();
    Boolean reset();
    Boolean more();
}
```

3.4. Metadata management

Naturally, the service provider of information sources generally conceals the schema of the information source from general users. However, a system developer of information gathering and integration may obtain the schema of the sources in either an official or unofficial way and tie the source by developing a wrapper. For example, a meta-search engine [23] ties multiple search engines into the system. A developer can, by probing the user interface of search engine, obtain the schema of the search engine.

Metadata management is important for supporting an integrated interface for multiple information sources because each source has its own interfaces and attributes. When processing a request from clients, a flexible framework is necessary for knowing the interfaces and attributes of backend sources. To minimize the degree of complexity for federating heterogeneous information sources for modern application requirements, we propose to provide designers with a metadata management mechanism for developing a federated query system with dynamic scalability and extensibility.

In IIR, metadata is modeled by the DTD of XML for information representations, both for the data subscriber and the supplier. We add a *registry* method to the *MetaData* interface. All participants must register themselves in IIR. The *MetaData* interface manages the metadata according to the type of query language.

The metadata is constructed as a three-level hierarchy in IIR, which is shown in Fig. 3. The third level consists of metadata of query systems. The second level represents a collection of query systems with the same query language type. The *MetaData* manager stands in the first level. Based on metadata and specified query parameters, *InformationRetriever* knows how to plan the query path for query requests.

3.4.1. Query interface metadata

Table 4 shows an example of query definition for a search engine. It represents the interfaces and attributes of a search engine in the DTD of XML. Other type metadata of information sources can be similarly defined with the search engine. Here, we explain the query template of the search engine shown in Table 4. In general, a search engine query string consists of many *keywords* and *attributes*. The *keywords* at least contain a keyword. Each keyword might have an *Included* attribute that shown whether or not the keyword is involved in the target of queried sources. The attributes are defined by referring to Ref. [22], which lists many of the attributes used in Yahoo and Altavista search engines.

According to the definition, a client program can obtain metadata of an information source to interpret what the attributes and query string of query operation are and how to issue query a request. For example, the following is a query string that contains a keyword and an attribute named “Date”:**“SELECT**

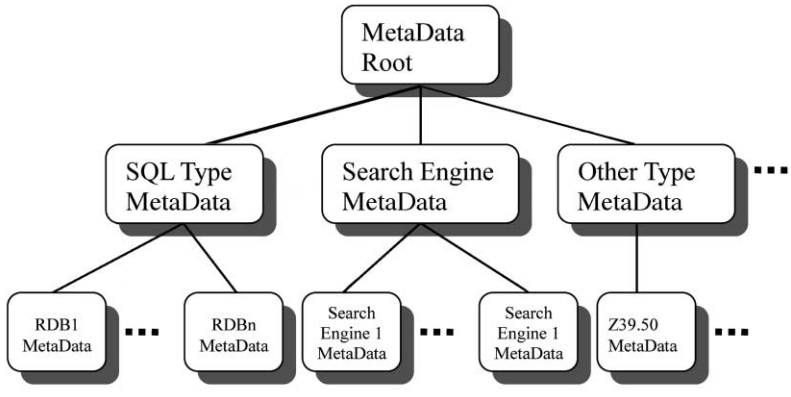

Fig. 3. *MetaData* heirarchy.

Table 4
Query definition of search engine

```
<!--This DTD includes the meta data of Search Engine query -- >
<?xml version="1.0"?>
<!DOCTYPE SEARCHENGINEINTERFACE [
        <!ELEMENT QUERYSTRING (KEYWORDS, ATTRIBUTE)>
        <!ELEMENT KEYWORDS(INCLUDED?, KEYWORD)+>
        <!ELEMENT INCLUDED (#PCDATA)>
        <!ELEMENT KEYWORD (#PCDATA)>
        <!ELEMENT ATTRIBUTE (DOMAIN | TAG | DATE | NEAR | DISPNUM |
                                                    AREA | DATABASE)?>
        <!ELEMENT DOMAIN (#PCDATA)>
        <!ELEMENT TAG (#PCDATA)>
        <!ELEMENT DATE (#PCDATA)>
        <!ELEMENT NEAR (#PCDATA)>
        <!ELEMENT DISPNUM (#PCDATA)>
        <!ELEMENT AREA (#PCDATA)>
        <!ELEMENT DATABASE (#PCDATA)>
]>
```

*** FROM Yahoo WHERE Keyword = "CORBA" and Keyword = "Object" and Date = "20001020""**

The keyword and attribute have to comply with the definition of the query. In the query string, the SELECT clause means find information from the source(s) designated in the FROM clause. The results of the SELECT clause have to be a subset of fields of the information source. The schema of the information source is defined and published by the service provider or the system developer, as shown in Table 5. The *InformationRetriever* can use the WHERE clause to compare with the metadata and judge whether the information source supported this query or not. If the equation in the WHERE clause has no equivalent ELEMENT, the *InformationRetriever* can return an error message to the client to indicate that the information source does not support the query. A *Wrapper* of the sources is also needed to check the query string based on the definition and construct the query message that will be sent to the backend information source. In addition, *Wrapper* also refers to the schema of the information source and to the query string for constructing results. An obvious merit of defining metadata by the DTD is that client can easily validate results from the *Wrap-*

Table 5
Schema of search engine

```
<!--This DTD includes the meta data of Search Engine result -- >
<?xml version="1.0"?>
<!DOCTYPE SEARCHENGINERESULT [
        <!ELEMENT RESULT(TITLE, URL, DESCRIPTION, WEIGHT)>
        <!ELEMENT TITLE(#PCDATA)>
        <!ELEMENT URL(#PCDATA)>
        <!ELEMENT DESCRIPTION(#PCDATA)>
]>
```

per. Similarly, for other information sources, source providers can also define their own query interface and register it in IIR.

Apparently, the system can integrate most of the information sources that have well-defined query syntax and semantics. We only define the metadata of the information sources and build a *Wrapper* based on IIR.

3.4.2. Query result metadata

Intuitively, there are three different views of information created in a query system: Export Views, System Views and Import Views. Import Views represent the data format and memory layout of information providers. Export Views represent the set of data that satisfies the query criteria for data inquirers. System Views are standardized collections translated from different Import Views and it will be translated into Export Views. Conventionally, there is a need for N^2 translators for N information inquirers and N information providers. To minimize the need for translation, Export Views can be represented by XML directly, instead of the vendor proprietary information format. That is, the system just needs N preprocessors to convert Import Views into System Views.

Similarly, source providers also want to define their result schema of the information source and register it in IIR. According to the result schema, a client program can interpret what information is contained in the returned result. Table 5 shows an example of the schema of a search engine. Most search engine return result to users consists of the *Title*, *URL* and *Description*. The wrapper uses the schema to organize the query result, and the client can also retrieve the content of the query result based on the schema.

3.5. Query scenario of IIR

Fig. 4 shows the IIR invocation sequence. For a client program, it first must bind to *Factory* to obtain its object reference, and execute a *create()* operation to construct a new *InformationRetriever* object. When *Factory* receives a create request, it creates an *InformationRetriever* and returns the object reference to the client. Once the client has this, other operations can be executed, such as *GetMeta()* and *prepare()*. First, the client invokes *GetMeta()* to obtain the metadata of a specified information source. Then, the client uses the *prepare()* operation to obtain a *Wrapper* object. Once the *Wrapper* object

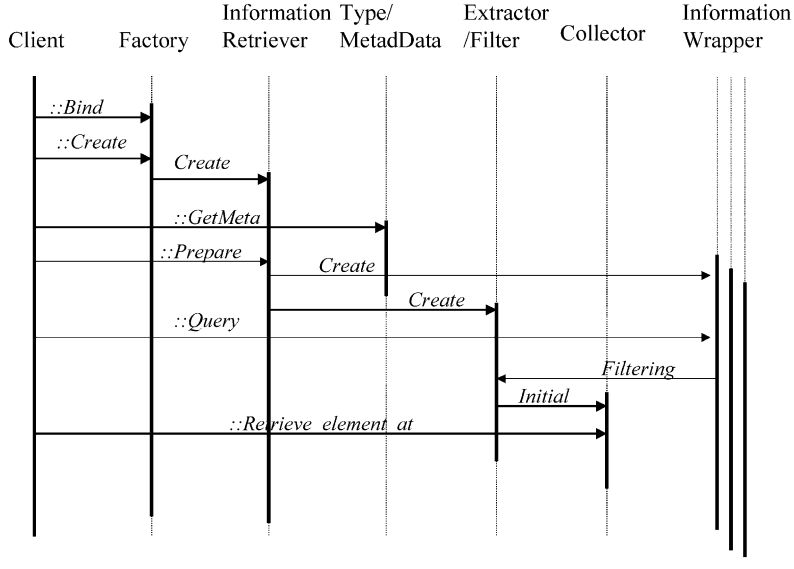

Fig. 4. The IIR invocation sequence.

is constructed, all of attributes are stored in *Wrapper*.

Then, the client program uses the *Query()* method to issue a query request. Once the *Wrapper* object receives a request, it encapsulates query strings and attributes into the query format of the information source, and sends the encapsulated query string to the corresponding source. When the *Wrapper* object receives the query result from the information source, it de-encapsulates returned messages and puts them into a *Collector* object. Then, the *Wrapper* object returns the *Collector* object reference to the client program, which then extracts the query results. By such scenario, it is obviously that an application wanting to query the Internet needs neither to have a complex query component, nor to expend much sophisticated network-accessing effort. It needs only to issue a few of invocations on the wrapper.

4. Application

We demonstrate the feasibility of IIR by integrating two prototypes of information retrieval on the WWW. The first is a meta-search engine termed Octopus [23] and the other an information retrieval facility [25]. These two prototypes are modified slightly to satisfy the imperative. Here, we state briefly how the two prototypes may be integrated into the system.

4.1. Octopus

Such as shown in Fig. 5, this system involves two search engine agents, which are implemented as CORBA objects. However, using a similar method would easily allow us to add other search engine agents to the system.

Because the system uses IIR IDL definitions, it provides a single and uniform interface for searching Web documents. On receiving a query from a WWW user, the system dispatches it to multiple search engines in parallel, and collates the returned references.

In merging Octopus into the system, the first requirement is to define the interface and the result schema of Octopus [23] in DTD. The DTD-style definition of these two materials is described in

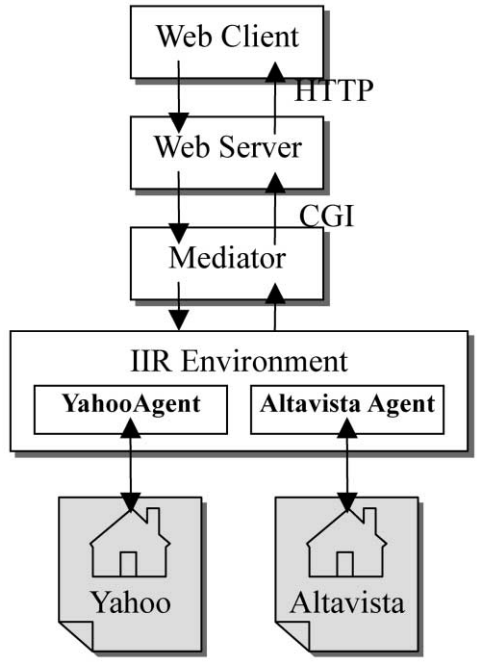

Fig.5. Meta-search engine achitecture.

Section 3.5. The *InformationRetriever* will invoke operations of the Octopus wrapper. Octopus wrapper will collect results that are the schema-compliance format.

In this system, a Web user posts a query request via the Common Gateway Interface (CGI). The CGI then forks a mediator for each request. The responsibilities of the mediator are as follows: first, it creates a thread of the *InformationRetriever* to perform the query; second, it collates the returned information from all the agents before merging and filtering them; finally, it returns the query results to the Web user.

The *InformationRetriever* is responsible for issuing the invocation of operations. All search engines have their own attributes, which are slightly different. One agent has one wrapper. The first task of a wrapper is to add keyword into the related agent, and set its attributes. Next, it will encapsulate query information into the HTTP format of the related search engine. Finally, when the client performs the *Query()* operation to invoke a search, it will send the query information to the related agent and obtains the results from the search engine. Once a search

engine returns the results, the wrapper will extract the returned information, and return the extracted information to the mediator. A further clear advantage with our system is that programmers can implement their own wrappers in application to have search service ability. Because we have a uniform interface—IIR, other agents are easily added to our system.

4.2. Z39.50 wrapper

Z39.50 is a well-known standard of information retrieval protocol defined by the National Information Standards Organization (NISO), and widely used in information retrieval applications and library systems. This protocol specifies a set of query syntaxes and attributes, and works on the basis of the protocol TCP/IP. To implement the protocol based on TCP/IP or WWW + CGI for a distributed environment, more effort on information transfer over the network is needed. The programmer must take care the session and presentation semantics of query requests cross the network.

In Ref. [25], we propose an information storage and retrieval facility based on CORBA. This facility provides information search, retrieval and storage services. In addition, an experimental digital library system based on this facility was implemented in our project to demonstrate how to use it. The system architecture of an ISRF implementation is discussed in Ref. [25]. Similarly, to merge the Z39.50 function into IIR, we define the query interface of the Z39.50 wrapper described in Ref. [25] in XML's DTD. But the interfaces are not all exported and registered into the metadata management mechanism, because only a simplified query operation is needed on the WWW. For example, in this system we do not support the check of access capability of the client. For the client, because the interface of IIR is uniform, the scenario of query processing is same as the Octopus.

5. Discussions

5.1. Challenges

This section discusses the challenges of deploying IIR for information gathering and integration, that is, the issues mentioned in Section 1. We explain how the IIR design resolves these challenges.

5.1.1. Describe information sources

"Ontology" refers to an understanding of some domain of interest. It embodies some sort of world-view with respect to a given domain. Such a world-view is often seen as a set of concepts, including entities, attributes and their inter-relations; this is referred to as a conceptualization. In IIR, the information source ontology is defined and modeled in terms of the DTD, as described in Section 3. IIR offers an interface to maintain the metadata. The interface allows the owner of sources to specify, access, and update the description and schema of their information sources. The implementation details concerning the source ontology in IIR are hidden from users in the structure of the information sources.

5.1.2. Determine relevant information sources

When a client issues a query request in terms of the proposed query language to IIR, the query string may involve the type of information sources or the name of information sources. The *InformationRetriever* will determine the relevant information sources to be queried from the query invocation issued by client. Even if the relevant information sources are vague in the query string, the *InformationRetriever* can determine the relevant sources from the query string itself, because IIR keeps the information sources ontology in *MetaData*. The *InformationRetriever* does this by mapping between the world-view ontology and information source ontology. For instance, a query string issued by a client may include a SELECT part of a SQL query, which implies the target of the query request by mapping the world-view to a certain source view. The *InformationRetriever* uses the mapping to determine the target information source.

5.1.3. Generating query plans for answering queries

As mentioned above, the *InformationRetriever* resolves relevant information sources by parsing the query string. The problem of generating query plans may have two aspects. The first concerns the client's knowledge of querying information sources and whether he will be able to generate a clear plan to

query behind the sources. For example, the client may explicitly generate a query plan as a set of generic specified SQL queries, each of which uses a specific internal sources structure, that is, a set of attributes specified in a particular relevant source ontology. The second aspect concerns generating a query plan by the *InformationRetriever*. The *InformationRetriever* retains the knowledge of the information sources and is, of course, able to generate the query plan for answering queries based on the knowledge of source.

5.1.4. Integrating / filtering query results

One of responsibilities of the *InformationRetriever* is to dispatch the request to the relevant information source wrapper that retrieves the desired information from the source. Afterwards, the wrapper delivers the results from the information source to the filtering agent that integrates the results referring to data representation of the metadata. Finally, the client may obtain a *Collector* object to retrieve the integrated results.

5.1.5. Assuring the correctness of source description

The source view of information and the mapping between the source view and world-view are both specified by the wrapper builder or the source provider. IIR also offers an interface to permit the source provider to maintain the metadata of information sources. Therefore, we presume that the source provider has a complete knowledge of their products and if so, the issue of the correctness of the sources description is determined in IIR.

5.1.6. Robustness of updating information sources

Metadata management provides a method for adding new or for updating existing information sources. When source ontology is updated, providers can update their own source ontology via the *MetaData* interface, which contains replacement and deletion operations, etc. That guarantees the ontology of information sources being the latest.

5.1.7. Robustness of security of information

Security of information sources is obviously a difficult issue with information retrieval technology on the Web because of its characteristic of openness. Security can be achieved by specifying the source ontology or limiting the access ability of an agent in IIR. The source provider can also develop the information source agent based on the unified interface of IIR in order to limit the ability of access. If necessary, source provider could choose to hide particular information from the metadata of the information source.

5.1.8. Maintaining interoperability

CORBA is an open system model that supports communication between the software components of a distributed environment as well as the dynamic location and integration of information sources. It also maintains the autonomy, so that a client can invoke remote objects transparently without having the information of server object. The agents of information sources obviously can be easily distributed at different locations in an IIR system. To replace a new version agent of information sources in the system causes no problems.

In addition, CORBA offers an IIOP (Internet Inter-ORB Protocol) for interoperating with other object-oriented model [10,12]. The wrapper of an information source can also be developed in another object models. A client issuing an invocation of an object can be mediated through IIOP bridges.

5.2. Advantages

There are many benefits of making a large client/server middle-ware based on CORBA [24]. To implement an IIR-based information retrieval system, we can raise further advantages.

First, with the progress in information retrieval technology, more useful information sources can be tied into a system by the same way. The system developer can easily tie new information sources into the system simply by creating information source agents. In our experience, most codes found in agents are the same, because the agents have the same server stub generated by an IDL compiler. To create a new agent, only a small part of a program needs rewriting. All the network operations are hidden from the CORBA's server stub. Thus, when constructing the new agent, the developer needs to only handle the interface of the information sources. The other components of the system can be changeless.

Second, programmers can easily build applications that need to be able to query information sources, such as data warehousing and data mining on the Web. Application programmers using the interface to deploy their applications that need to query information on the Internet can hide the complexity from network programming and concentrate most effort on other significant value-added services. After the information source agent returns the results, the program does not need to extract the information from the complicated and proprietary data format because, since they are all based on the same interface, applications are undiscerning when querying agents.

Third, CORBA is a distributed object-oriented environment. In IIR, agents can be easily distributed at different locations. In this way, balancing the load while the system increases in size is easy. The system manager can dynamically add other agents into the system. Thus, a system based on IIR has inherent scalability.

Finally, because this is a modularized and component-based approach, it will be easy in the future to replace certain components with new and useful algorithms, such as a weighting algorithm or a natural language processing algorithm. So, an IIR system also has inherent flexibility.

In addition, IIR clearly applies many industrial standards, including CORBA as the object model, XML as the data model, and SQL as the query language. This advantage reinforces the inclination of programmer study and application and reduces the complexity of system development. We also hope to establish a standard for IIR on the COSS of CORBA. We believe that IIR can be easily applied to other types of information sources. Thus, the same approach may be extended to most search or query services—whether or not on the Internet—since its underlying design incorporates the advantages described above.

6. Conclusion

In this paper, we have proposed an IIR facility based on CORBA to integrate existing *Information-Retrieval* techniques for providing a Web-based mining facility. IIR not only offers a programming interface for retrieving information, but also provides a flexible and extensible environment in consolidating information sources. The query definition and result schema are modeled in the DTD of XML, which allows the service provider to add new agents to the system with facility. IIR also allows source providers to define their own query interface and schema in a well-known object model and language. Since IIR architecture is of an N-tier client/server model and has a uniform interface, an application required to retrieve information or mine data from the Internet needs only to initiate the query operation of an agent using IIR. Using the IIR framework, programmers querying information need neither explore the interface of various information sources, nor construct query components in their applications.

CORBA clients can also use this interface to invoke search engine agents in their application programs. In addition, clients developed using other object models, Microsoft's COM/DCOM, for example, can in the same use these agents by means of the Internet Inter-ORB Protocol of CORBA.

Acknowledgements

We are grateful for the many excellent comments and suggestions made by the anonymous referees. We also thank the National Science Council for their financial support through the project numbered as NSC 89-2626-E-159-001.

References

[1] A.Y. Levy, A. Rajaraman, J.J. Ordille, Query heterogeneous information sources using source description. Proceedings of the 22th VLDB (1996) 251–262.
[2] R.H. Bayardo et al., InfoSleuth: agent-based semantic integration of information in open and dynamic environments. Proceedings of the ACM SIGMOD (1997) 195–206.
[3] H. Garcia-Molina, J. Hammer, K. Ireland, Y. Papakonstantinou, J. Ullman, J. Widom, Integrating and accessing heterogeneous information sources in TSIMMIS. Proceedings of the AAAI Symposium on Information Gathering, Stanford, California, USA. 1995, pp. 61–64.
[4] H. Garcia-Molina, Y. Papakonstantinou, D. Quass, A. Rajaraman, Y. Sagiv, J. Ullman, V. Vassalos, J. Widom, The TSIMMIS approach to mediation: data models and languages. J. Intell. Inf. Syst. 8 (2) (1997) 117–132.

[5] E. Mena, V. Kashyap, A. Illarramendi, A.P. Seth, Domain specific ontologies for semantic information broking on the global information infrastructure, in formal ontology, in: N. Guarino (Ed.), Information Systems, IOS Press, Amsterdam, Netherlands, 1998, pp. 269–283, June.
[6] E. Mena, A. Illarramendi, V. Kashyap, A.P. Seth, OBSERVER: an approach for query processing in global information systems based on interoperation across pre-existing ontologies. Distrib. Parallel Databases 8 (2) (2000) 223–271, April.
[7] Y. Arens, C.Y. Chee, C. Hsu, C.A. Knoblock, Retrieving and integrating data from multiple information sources. Int. J. Intell. Coop. Inf. Syst. 2 (2) (1993) 127–158.
[8] Y. Arens, C.A. Knoblock, C. Hsu, Query processing in the SIMS information mediator, in: A. Tate (Ed.), Advanced Planning Technology, AAAi Press, Menlo Park, Cliff., 1996, pp. 61–69.
[9] Y. Arens, C.A. Knoblock, W.M. Shen, Query reformulation for dynamic information integration. J. Intell. Inf. Syst. 6 (2/3) (1996) 99–130.
[10] A. Bouguettaya, B. Benatallah, L. Hendra, M. Ouzzani, J. Beard, Supporting dynamic interactions among Web-based information sources. IEEE Trans. Knowl. Data Eng. 12 (5) (2000) 779–801, Sept.
[11] Object Management Group, The Common Object Request Broker (CORBA): Architecture and Specification, vol. 2.2, 1998 February.
[12] K. Brockschmidt, Inside OLE. 2nd edn., Microsoft Press, Redmond, WA, 1995.
[13] Object Management Group, CORBA services: Common Object Services Specification1995 OMG Document Number 95-3-31, March 31, www.omg.org.
[14] Soe-Tsyr Yuan, Ontology-based agent community for information gathering and integration. Proc. Natl. Sci. Counc., Repub. China, Part A 23 (6) (1999) 766–781.
[15] D.S. Harverkamp, S. Gauch, Intelligent information agents: review and challenges for distributed information sources. J. Am. Soc. Inf. Sci. 49 (4) (1997) 304–311, April 1998.
[16] H.S. Nwana, D.T. Ndumu, An introduction to agent technology. Lecture Notes in Artificial Intelligence, Springer, Berlin, Germany, 1997.
[17] E. Spertus, L.A. Stein, Squeal: a structured query language for the Web. Int. J. Comput. Networks 33 (2000) 95–103.
[18] A. Deutsch, M. Fernandez, D. Florescu, A. Levy, D. Suciu, A query language for XML. Proceedings of the 8th International World Wide Web Conference, Elsevier, Amsterdam, 1999.
[19] C.A. Knoblock, A. Levy, Information gathering and integration. The Tutorial of the 13th National Conference on Artificial Intelligence Portland, Oregon, USA, August, 1996.
[20] M. Leventhal, D. Lewis, M. Fuchs, Designing XML Internet Applications, Prentice Hall PTR, Upper Saddle River, NJ, 1998. 07458.
[21] D. Konopnicki, O. Shmueli, Information gathering in the World Wide Web: the W3QL query language and the W3QS system. ACM Trans. Database Syst. 23 (4) (1998) 369–410, Dec.
[22] S. M. Yuan and Y.S. Chang, "Design and Implementation of Heterogeneous Full-text retrieval engine agent," Technical Report, Department of CIS, National Chia-Tung University, 1998.
[23] Y.S. Chang, S.M. Yuan, W. Lo, A new multi-search engine for querying data through internet search service on CORBA, Int. J. Comput. Networks 34 (3) (2000) 467–480, Sept.
[24] R. Orfali, D. Harkey, Client/Server Programming with JAVA and CORBA, John Wiley & Sons, New York, USA, 1997.
[25] W. Lo, Y.S. Chang, C.L. Chou, R.K. Sheu, S.M. Yuan, An Information Store and Retrieval Facility on CORBA, Lecture Notes in Computer Science (LNCS) of Springer-Verlag, vol. 1846, Springer-Verlag, Heidelberg, Germany, 2000, pp. 374–379, June.
[26] N. Ward, M. Lawley, S. Finnigan, ZORBA: information retrieval using distributed object technologies, EGOEO'98, (1998) http://www.dstc.edu.au/ZORBA.

Chang Yue-Shan was born on August 4, 1965 in Tainan, Taiwan, Republic of China. He received the BS degree in Electronic Technology from National Taiwan Institute of Technology in 1990, the MS degree in Electrical Engineering from the National Cheng Kung University in 1992, and the PhD degree from Computer and Information Science at National Chiao Tung University in 2001. Dr. Chang is a lecturer at the Department of Electronics Engineering of Ming Hsin Institute of Technology. His research interests are in Distributed Systems, Object Oriented Programming, Information Retrieval and Integration, and Internet Technologies.

Min-Huang Ho was born on February 1, 1969 in Kaohsiung, Taiwan, Republic of China. He received the BS and MS degree in Industrial Education from National Taiwan Normal University in 1993 and 1995, respectively. Currently, he is a candidate of PhD in Computer and Information Science at National Chiao Tung University. His research interests are in Distributed Systems, Internet Technologies, and Mobile Agent Technologies.

Shyan-Ming Yuan was born on July 11, 1959 in Mauli, Taiwan, Republic of China. He received the BSE.E degree from National Taiwan University in 1981, the MS degree in Computer Science from University of Maryland Baltimore County in 1985, and the PhD 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 Oct. 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.