BLACKBOARD-BASED INFORMATION SYSTEMS:

QUERYING AND RESOLUTION

by

DEEPAK CHINTHAMALLA

(Under the direction of Dr.Walter D. Potter)

ABSTRACT

It can be safely said that today's world is driven by information.There is a lot of information available today in various forms and sources, including databases, knowledge bases, flat file systems and the world wide web. The challenging task is to integrate the information available in these different formats. Our architecture performs integration of information that is available from such varied sources, at the heart of which lies a problem-solving technique of AI, the Blackboard technique. We model our query controller as a blackboard and perform integration using the knowledge provided to the system.

INDEX WORDS: Information Integration, Information Retreival, Querying, Query Controller , Blackboard, Wrappers.

BLACKBOARD-BASED INFORMATION SYSTEMS:

QUERYING AND RESOLUTION

by

DEEPAK CHINTHAMALLA

B.E., Osmania University, India, 1999

A Thesis Submitted to the Graduate Faculty of The University of Georgia in Partial Fulfillment of the Requirements for the Degree

MASTER OF SCIENCE

ATHENS, GEORGIA

2002

BLACKBOARD-BASED INFORMATION SYSTEMS:

QUERYING AND RESOLUTION

By

DEEPAK CHINTHAMALLA

Approved

Major Professor	Walter D. Potter
Committee:	Daniel M. Everett
	Jeffrey W. Smith

Electronic Version Approved:

Gordhan L. Patel
Dean of the Graduate School
The University of Georgia
August 2002

ACKNOWLEDGEMENTS

Firstly I would like to thank Dr. Walter D. Potter for all his encouragement and support throughout this thesis. I would also like to thank my committee members, Dr. Jeffrey W. Smith and Dr. Daniel M. Everett for being willing to preside over my defense and for providing their support. Last but not the least I would like to thank Ms. Haritha Muthyala, whose constant help made this thesis possible.

TABLE OF CONTENTS

CHAPTER 1

INTRODUCTION

1.1 Information and Integration

Today's world is driven by information, which is available in several different forms. Information can be textual, in the form of graphs and charts or in the form of pictures. But it is not just sufficient for information to be present if it cannot be accessed and used by people who need it.

One of the biggest problems faced by most organizations today is that of the integration of information that is available from disparate information sources and in repositories. This integration might be necessary because decision-makers need to access the information available in these multiple forms in a unified manner. This is a time-consuming task because the different systems cannot be accessed in a uniform manner and also the information represented in each of these forms might be inconsistent and contradictory.

1.2 Information Integration Systems

Systems that perform the integration that was described in the previous paragraphs are referred to as Information Integration Systems (IIS). When the user poses a query that requires accessing several different sources for retrieving the answer, the IIS has to combine the results returned by each source before presenting them to the user.

1.3 Issues in Information Integration

Even before the information can be integrated, one important thing that has to be done is data format conversion. That is, the data that needs to be integrated has to be translated into a common format. Also, during translation, several inherent schematic conflicts have to be considered as the formats differ. Some common types of schematic conflicts are listed here:

- Generalization conflicts - A class or attribute in one database may subsume multiple classes or attributes in another database. For example, one data source may contain first name and last name as its attributes, while another may contain the full name as a single attribute.
- Structural conflicts - An entity in one system may be modeled as an attribute in another system. For example, employee and student may be different entities in a data source, while they may be the values of an attribute in another data source.
- Naming conflicts - Semantically equivalent classes or attributes may be assigned different names. It may be the other way round also *i.e.*, attributes or entities with the same name may not represent the same thing semantically.
- Missing attributes – For example, salary of an employee may be included in one source while it may not be included in another source.
- Data type conflicts - Different types may be assigned to semantically equivalent attributes. For example, '*ssn*' can be represented either as a string or an integer.
- Scale conflicts – Salaries may be recorded in one source on a per month basis while in another they may be stored on a per hour basis.

The information system should be able to resolve these conflicts before the actual integration of information. But apart from the schematic conflicts, there are many other issues that have to be handled. Firstly, is the process of integration transparent to the user or not? That is, does the user need to know the intricate details involved in getting data from many sources and combining them or does the system do this with least interaction with the user.

Also, is the information on how the integration is to be carried out provided by the user or decided by the system itself? And, last but not least, the data in the different sources has to be converted into a pre-decided common format so they are represented uniformly. Query handling is a critical part of an Information Integration System and should address most of the issues that are discussed here.

1.4 Querying in IIS

Query handling and resolution in Information Integration Systems can be thought of as a process involving the following steps:

- Query specification - The user enters a request for data by specifying constraints on attributes. End-users pose queries against a predefined integrated view using a declarative query language, or by entering constraints in a graphical interface.
- Query modification - A query is decomposed into sub-queries, one for each component database to be accessed by the query.
- Query translation - The component sub-queries are translated into the syntax required by the corresponding local data sources. The component information system reads the query and produces a result set.

- Result translation - Each result is translated back into the common data model as necessary
- Result integration - The multiple result sets are integrated and returned to the user.

There are several issues in each of these steps and some of them are discussed below.

1.5 Issues in Querying

Querying is a means for users to communicate with the system and retrieve the necessary information. There are many different issues involved in querying a system. The querying means provided must be clear, concise and easy to understand. Providing a query language with a strict syntax can ease querying, but some systems may not have a query language at all. Sometimes, a query can be divided into several sub-queries that the system identifies and whose results are to be combined to form the final results.

When the user expresses an information request in the form of a query, the system should be able to resolve the query in order to find the correct results efficiently. This is referred to as query resolution and the process may involve many steps, wherein the query may be divided into sub-queries. The sub-queries are in turn executed to give intermediate results, which are combined into the final result.

The query provided by the user can be composed by either using a command line or a graphical interface. The command line query relies more on the expertise of the user and assumes an inherent understanding of the system on the part of the user, who should be able to write the entire query without any visual aids from the system. On the other hand, if a graphical interface is provided to the user, then the user might be able to see the type of information that is present in the system and this visualization can help in building or composing the user query.

This brings us to the issue of whether or not to present a view of the system to the user - a view showing the different kinds of information that is present in the system and that can be queried on. In the event that a view is not provided, the user would have to search all the information in the system to see if the information that is required is present in the system at all.

Another issue is declarative versus procedural querying. In the latter case, the user would have to provide all the details on how to retrieve some information, and how different data can be combined to form the final answer that is required. This reduces the workload on the system as the user provides most of the logic for resolving the query. On the contrary, in systems using declarative querying, the user just mentions the information that is required and the system figures out how to retrieve and display the results, combining intermediate results, if necessary.

It is even possible that querying be done by example. For instance, the user might provide a data example that shows the type of information required, and the system matches the information in the system to the example provided in order to deliver the final answer.

1.6 Synopsis

The motivation behind this system was the difficulties encountered in the information of integration; this becomes even more complicated if the sources involved in the integration are heterogeneous in syntax and semantics. Such problems are the reason behind the development of this system.

Though this is a relatively new area, there are already several ongoing projects that are currently conducting research in the field of information integration from

heterogeneous sources. Our project is novel because it applies a popular Artificial Intelligence (AI) problem-solving technique to the problem of information integration.

The query resolution in our system is handled by the architecture component called the Query Controller. The Query Controller resolves the query and performs integration of information, if necessary, using the Blackboard technique. We also maintain metadata about the keys, constraints and similarities between the data objects in various sources, for resolving the schematic conflicts and performing constraint mapping during information integration. Details on this are provided in the following chapters.

1.7 Organization of following chapters

Chapter 2 provides a very brief overview of some relevant Information Integration Systems that are currently available, and it contains example queries that might be posed in those systems.

Chapter 3 sets the background for the Query Controller and describes the Blackboard technique that lies at the heart of this architecture.

Chapter 4 discusses the data model being used for the system and the interface that is provided to the user to interact with the system.

Chapter 5 provides details about the Blackboard data unit and the Query Controller, which are the key query handling components of the system.

Chapter 6 describes other subsystems relevant to our architecture and finally chapter 7 provides the implementation details of the entire system.

CHAPTER 2

QUERYING IN VARIOUS INFORMATION INTEGRATION SYSTEMS

This chapter describes various popular Information Integration Systems, briefly talking about their architectures and the query languages used in those systems (along with examples to illustrate the query structure). The main purpose of this section is to provide an introduction, without much detail, as to what type of queries are posed in each of these systems, with a brief overview of the system. A discussion of these systems is relevant here to serve as a background for our Information Integration System, whose architecture is proposed in the next chapter. A brief listing of various popular Information Integration Systems is as follows:

2.1 Infomaster

Infomaster [Duschka and Genesereth 1997] is an Information Integration System developed at the Stanford University. The general architecture of Infomaster is such that it has various wrappers over all the different sources, which in turn talk with the Infomaster facilitator that uses its knowledge base to intelligently query and integrate the information. The system also has various interfaces to the clients who are using the system.

Infomaster uses a programmatic interface called Magenta, which supports ACL (Agent Communication Language). ACL consists of KQML (Knowledge Query and Manipulation Language) and KIF (Knowledge Interchange Format). The sources are visible to the user in the form of database relations. This is accomplished by providing an

abstraction over the source so that the user sees only the relational view of the source. Also, the relationships are described using special definitions, which can be used when the user is querying the system. There are three types of relations used in this system: The interface relations that are used by the User Interface, the base relations that are used by the query engine and the site relations that are used by the information sources. Hence a typical query in this system would look as follows:

> Say the user is querying for BMWs built in 1996 and whose sale price is below their average market value then the query will look as
>
> *Q(Model, Mileage, Price) ≡*
>
> *Cars (bmw, Model, 1996, Mileage, Price, Value)*
>
> *& Price < Value*

The query processing in Infomaster is a three-step process, viz., reduction, abduction and optimization. In reduction the interface relations are transformed into the base relation. In the abduction phase, the base relations are transformed into the site relations and in the final phase of optimization access planning for the sources is undertaken. Infomaster provides a simple view of the system, which can be effectively queried by the user with much ease.

2.2 Garlic

Garlic [Cody, *et al.* 1995] is an Information Integration System developed at IBM Almaden Research Center. This system is different from other systems in the sense that it is a multimedia middleware, which can handle both text and multimedia sources. Each source is called a repository and a 'repository wrapper' is present above each repository, which can be queried.

Garlic uses a query language called GQL, which is an object-oriented extension of SQL. The interfaces to each of the repository wrappers are in an object-oriented fashion and the queries posed on the system are as follows:

> The query to find the campaigns, the associated report and magazine ads for those campaigns that ran since 1989 and that had a magazine image that resembles the given image (the user drawn sketch): -
>
> *select C.campaign_name, C.report, C.mag_ads*
> *from Campaign C, C.mag_ads A*
> *where (C.report.date > "1989") and A.match_image(SKETCH) > .5*

Here in the above query .5 is the goodness of the match that can be acceptable. The advantage of Garlic compared to other systems is that it can handle all sorts of data (text and multimedia) and can be used in real world applications.

2.3 CoBase

In general databases, we might encounter situations like, the result of a query is unavailable or the query is not well formed or the data is missing. In such cases the databases return a null answer or an error. An intelligent database system would thus be very resourceful if it can permit cooperative and conceptual level querying (facilities that are not provided in conventional database schema) when the user query is not precise. The system should be able to provide some relevant information even if some data is missing. CoBase [Chu, *et al*. 1993] (developed at UCLA) is a variation of a distributed database, called co-operative database, which has these intelligent capabilities.

CoBase uses the language LOOM for representing its knowledge and uses CSQL, an extended version of SQL with cooperative features, for querying. CSQL is just like SQL but has some cooperative operators like *approximate, within, near-to, similar-to,*

relaxation-order, not-relaxable, nearer and further. An example of a query expressed in CSQL is the following:

List the airports with the parking capacity approximately equal to 200,000 square feet: -

select airport_name, parking_sq_ft from airports
where parking_sq_ft =/\ 200,000

This CSQL query is translated by CoBase into the following SQL: -

select airport_name, parking_sq_ft from airports
where parking_sq_ft >= 100,000 and parking_sq_ft <= 300,000

The approximation of 200,000 to lie between 100,000 and 300,000 is decided by the knowledge base rules specified in the system. The advantage of cooperative querying is that it can be effectively used in cases where the user is not sure of the data present in the information system.

2.4 InfoSleuth

InfoSleuth is an agent based integration system developed at Microelectronics and Computer Technology Corporation (MCC), Texas [Bayardo, *et al.* 1997]. InfoSleuth uses the concepts of agent technology, domain ontologies, brokerage and Internet computing to support integration of data. The InfoSleuth system is designed to effectively handle dynamically changing environments and is thus different from its predecessor, the Carnot information system [Finin, *et al.* 1994]. The system works in the manner that all the agents sit on the top of sources/other agents and advertise their services and process requests either by making inferences using the knowledge, by routing the request to a more appropriate agent, or by decomposing the request into a collection of sub-requests and then routing these requests to the appropriate agents and integrating the results.

InfoSleuth uses the high-level agent query language KQML and the ontologies in the system are represented using a dialect of Knowledge Interchange Format and the query language SQL. A KQML query message consists of a performative, its associated arguments and a set of optional arguments, which describe the sender and the receiver. As an example, a KQML query would be as follows:

> Say a message representing a query about the price of a share of IBM stock might be encoded as:
>
> *(ask-one*
>
> *: content (PRICE IBM ?price)*
>
> *: receiver stock-server*
>
> *: language LPROLOG*
>
> *: ontology NYSE-TICKS)*

In this message the KQML performative is *ask-one* (which asks for a single reply), the content is *(PRICE IBM ?price)*, the ontology assumed by the query is identified by the token *NYSE-TICKS*, the receiver of the message is to be a server identified as stock-server and the query there is written in the language LPROLOG. The most general query performatives are *evaluate, ask-if, ask-in, ask-one, ask-all, stream-in* and *stream-all*. InfoSleuth is an agent based Information Integration System, which effectively integrates information in dynamically changing environments.

2.5 TSIMMIS

TSIMMIS [Chawathe, *et al.* 1994] is an integration system that was developed at Stanford University. TSIMMIS is a mediator-based system with a hierarchy of mediators that resolve the queries posed by the user. Each source has a wrapper and the mediators talk with the wrappers to get the information required from the sources. The knowledge of the system lies in the mediators, which have rules defining which data to gather from

other mediators or wrappers. TSIMMIS uses the OEM (Object Exchange Model) data model, which is object oriented.

TSIMMIS uses OEM-QL as its querying language and all of the data in the system is in the form of OEM objects. A typical OEM query would look as follows:

Find each document for which "Ullman" is one of the authors:

select bib.doc.topic from Biblio
where bib.doc.authors.author-ln = "Ullman"

In the above query bib, doc and topic are the corresponding OEM objects for the mediators, bib, doc, and the wrapper is topic. Querying this system is typically easy as all the queries are posed on the object-oriented OEM view of the system, which can be directly queried upon.

2. 6 DISCO

DISCO (Distributed Information Search Component) [Tomasic, *et al.* 1996] has the typical mediator-wrapper architecture and apart from this architecture, DISCO also has a special set of mediators called the catalogs which keep track of all the other mediators and wrappers in the system. DISCO is based on the ODMG (Object Data Management Group) standard and consists of a data model, ODL (Object Definition Language), a query language (OQL) and a language binding mechanism. DISCO uses the DISCO query language for querying and it looks as follows:

A query for the names of persons who have a salary greater than 10 is as follows:

select x.name from x in person
where x.salary > 10

DISCO mainly addresses the issues of databases where sources may come in and leave at any time in a dynamic environment. DISCO also provides a flexible wrapper interface

using which wrappers can be built. Also the mediator model is quite simple and object-oriented.

2.7 Information Manifold

Information Manifold [Kirk, *et al.* 1995] is a data integration system developed at AT&T Bell Laboratories. The Information Manifold system is based on a domain model that is described by the knowledge base, describing the properties of the information sources. The user queries against the conceptual model of the domain and the knowledge base of the system, which represents the properties of the information sources, aids in the building of this conceptual model. The system contains defined ontologies using which the various aspects of the domain are represented.

The language used for representing the contents of information sources is a combination of Horn rules and concepts from the CLASSIC description logic [Brachman, *et al.* 1991]. A query in Information Manifold can be formulated as a Horn rule defining a relation Q as follows:

> A query that asks for information about travel agents in Miami, FL (area code 305) who sell tickets from Newark to Santiago for under $1000: -
>
> *quote(Ag, Al, 'Newark, NJ', 'Santiago, Chile', C, D)*
>
> */\dir(Ag, Ac, TelNo) /\ Ac = 305 /*
>
> */\C < 1000 => Q(Ag, TelNo, C)*

The above query uses information from the NY directory (dir) to retrieve phone numbers of the travel agents. The advantage of this type of querying is that the knowledge and the query are represented using the same language and also the queries are represented as Horn rules.

2.8 InterDB

InterDB is a system that provides an abstract interface for various independent databases. InterDB is developed at the University of Namur, Belgium [Thiran, *et al.* 1998]. The architecture of InterDB is such that it has a hierarchy of mediators, which transform the data into a global view and also transform the global queries into the physical queries of the system. So, on the whole, the data are assessed and queried in a global form, which is seen by the external world. The user uses the overall conceptual schema to query the system.

A typical query for this system can be posed in SQL and since it just involves the integration of data from databases, the query transformation and result integration are essential for this system. The system identifies the data conflicts that exist and resolves them. This system can be termed as an integrated database or federated database system.

2.9 OBSERVER

OBSERVER [Mena, *et al.* 1996] provides an architecture for query processing using various global information systems, which have different structure/organization, query languages and semantics of the data in them. OBSERVER was developed mainly at The University of Georgia (in the LSDIS lab) and, as all the different systems have their own ontologies, the Inter-ontology Relationships Manager (IRM) of the system relates the terms in various ontologies. The IRM is used when the results from various sources need to be integrated. The different ontology servers provide the appropriate ontologies, which can be used to represent the source, and the IRM has information about the relationships between all these ontologies.

The queries posed by the user are in the form of intentional metadata descriptions represented using Descriptive Logics (DLs) [Brachman and Scmolze 1985], here called CLASSIC [Borgida, *et al.* 1989]. The following query serves as an example for a query in this system:

The user query will be expressed in the format:

<*list-of-roles*> for <*classic-expression*>

where *list-of-roles* is a list of roles to be projected (the roles about which the user asks) and *classic-expression* is a list of constraints expressed in DL (the conditions that the answer must satisfy). For example, the query would look as follows:

[title author document pages] for (AND doctoral-thesis-ref
(FILLS keywords "metadata") (ATLEAST 1 publisher))

The query posed by the user is then translated into the queries for the individual sources and then the IRM information can be used to integrate the results in an effective manner. The query posed by the user is thus processed and the results are generated without much effort from the user.

2.10 KOMET

KOMET (**K**arlsruhe **O**pen **ME**diator **T**echnology) [Calmet, *et al.* 1997] is a system developed at the University of Karlsruhe, Germany. KOMET is a mediator-based system and the user is provided with the views of the underlying system on which the user can pose queries. KOMET uses the declarative language for its mediators, which is called KAMEL (**KA**rlsruhe **ME**diator **L**anguage). KAMEL is a language based on annotated logic. The knowledge as to how the system should retrieve and integrate the

results is provided by the information specified by this language. A typical query in this system would look as follows:

ans(Name, Date) ⟵ *stockname(Name, SID) &*
close(SID, Date, 500.00)

And the corresponding translated query in SQL for the above query would look as follows:

select r1.name, r2.date
from stockname r1, close r2
where r1.sid = r2.sid and r2.value = 500.00

The query translator part of the system translates the query from KAMEL into the source query of the system being queried upon. The rules defined in KAMEL provide information as to how the data are to be retrieved and integrated.

2.11 COIN

COntext INterchange [Bressan, *et al.* 1998] is an Information Integration System developed at MIT. This system uses the data model called COIN and an object-deductive language called COINL, which are used to define the contexts of the sources (description of the sources). The *context mediator* of the system identifies and resolves the conflicts involving various sources. Wrappers above the sources display the results in the form of a relational table format.

This system uses SQL for querying and the user views the sources as database relations and can query them directly. The context mediator would resolve any conflicts that arise while executing this query. For example a query to the system would look like:

*select Local.Ticker, Nyse.CompanyName, Local.Qty * Cnn.Last, Zacks.Rec*
from Local, Cnn, Nyse, Zacks
where Nyse.Ticker = Local.Ticker AND Cnn.Ticker = Nyse.Ticker AND

Zacks.Ticker = Nyse.Ticker;

In the above example the user queries to retrieve information of stock prices from the New York Stock Exchange website for the stocks in the portfolio, stored in a relation in the local database and also relevant information about the stocks from the CNN Financial Network and Zacks Investment Research Publishers. The advantage of using SQL for querying various information systems is that the user can easily query them and retrieve information, as the view available to the user is always in the form of database relations.

Thus, above are described some of the classical projects of information integration and their querying schemes, developed using various architectures and technologies. The next chapter describes the overall architecture of our system, describing its major components and their functions.

CHAPTER 3

SYSTEM ARCHITECTURE

This chapter provides details of our overall system, of which this thesis is just a part. It also provides details on the significance of my thesis to the whole system. Our system performs information integration from various heterogeneous sources. We propose to use a novel approach towards this by making use of a technique that has not been used for this purpose so far. A similar architecture called NED [Twery, *et al.* 1997] is being developed at The University of Georgia. We propose an IIS architecture that employs the problem-solving Blackboard technique in Artificial Intelligence. A brief description of this model and technique is provided in the next section.

3.1 The Blackboard Technique

The Blackboard technique is a popular AI technique used for problem solving. The Blackboard model has three major components – a global database, called the Blackboard, a set of logically independent knowledge sources (KS), and a set of control data structures or control modules, used for monitoring the changes on the Blackboard and for deciding what happens next. The knowledge sources contain some information, which is exclusive to each one of them, and this information is used for solving problems. This knowledge is separate and independent from the knowledge present in any other (knowledge) source.

Another duty of the knowledge sources is to collect the data that are currently available on the Blackboard and store them after encoding it. All modifications to the

Blackboard are made by the knowledge sources and all these modifications are explicit and visible. Figure 3.1 shows the framework for a Blackboard system [Engelmore, *et al.* 1988].

The knowledge sources are represented as procedures, sets of rules or logic assertions. Each of these knowledge sources has information about the conditions that have to be true so that they can contribute to a solution; they are specified as pre-conditions that indicate the condition on the Blackboard that must exist before the body of the knowledge source is triggered.

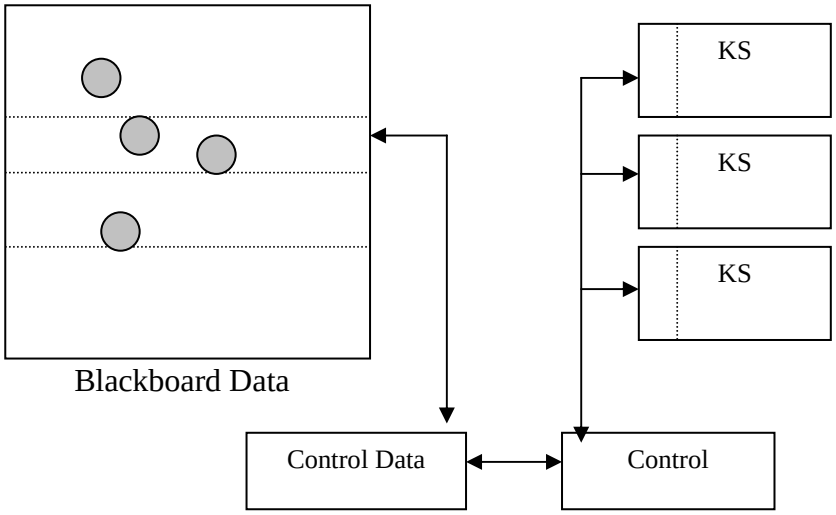

FIG 3.1: BLACKBOARD FRAMEWORK

The global database or the Blackboard is at the heart of this architecture where processes place the problems, which need a solution, on it. It is a hierarchically organized database, whose main feature is that it contains all the intermediate (and eventually the final) results to the problem at hand. Each level of the hierarchy in the Blackboard can be thought of as an abstraction level and each of these levels view a different perspective of the problem, in terms of a different set of concepts. For example, in the HEARSAY-II [Erman, *et al.* 1986] system, which is based on the Blackboard architecture, for the

speech understanding task, the different abstraction levels in the hierarchy comprised viewing the speech signal in terms of words at one level, at the phonemes at another level, and into the phrases that the words could be grouped into at another level.

The Blackboard stores information in the form of entries. An entry can be thought of as a complex data structure [Craig 1995]. The problem-solving data are available on the Blackboard. The knowledge sources produce the changes to the Blackboard that might finally lead to a solution. Modifying one of the existing entries on the Blackboard or adding a new one does this. There is no direct interaction or communication between the sources themselves, but they communicate only through the changes that are made on the Blackboard. Thus, the purpose of the global database, the Blackboard, is to hold the computation and necessary information required to find the problem solution that is produced by the sources. The following is a brief overview of the processing involved in this technique.

A knowledge source makes some changes to the current state of the Blackboard. These changes are tracked in the control structures and each source also points out how much and what it can contribute to the new Solution State. A control module then decides the approach to the processing depending on the above information. The processing can be either event-driven or knowledge-driven: it is said to be event-driven if the control module chooses to invoke appropriate knowledge rules on a particular Blackboard object. On the other hand, if the control module decides to execute a knowledge source and then chooses an object that it has to be invoked on, then the processing is said to be knowledge-driven (It is also possible that the module chooses both a knowledge source and a Blackboard object, and invokes the former on the latter). The process is terminated

when a solution is found, as a result of the changes or contributions made by the knowledge sources, or when nothing much else can be done without more knowledge.

This architecture operates on a three-phase cycle. In the first phase, the Knowledge Sources examine the Blackboard and decide if they can make a contribution, by comparing the latest happening with the pre-condition of the production rule. If a source decides that it can make a worthwhile and effective contribution, then they are considered as possible 'next-steps' by the control module. In the next phase, the control module chooses which of the sources selected in the first phase can be executed. And in the third and final phase, the knowledge source actions are finally executed.

This series of actions eventually leads to a final solution. It is even possible that once a possible solution is found, the control module tries to find alternative solutions and then chooses the most optimal and effective solution to the original problem.

The Blackboard technique has been employed in various applications and real-time systems. Some of them are ATOME [Laasri, *et al.* 1987], GBB [Corkill, *et al.* 1988], Erasmus [Baum, *et al.* 1989], and Poligon [Rice 1986].

3.2 Overall Architecture

Figure 3.2 shows the block diagram of our system. There are several different components in the system, each of which performs a specific function. They are the information sources, the wrappers, the Wrapper Generator, the Blackboard, the Query Controller and the User Interface.

The information sources in the system are heterogeneous, in that they might not use the same data model or schema to represent the information stored in them. The

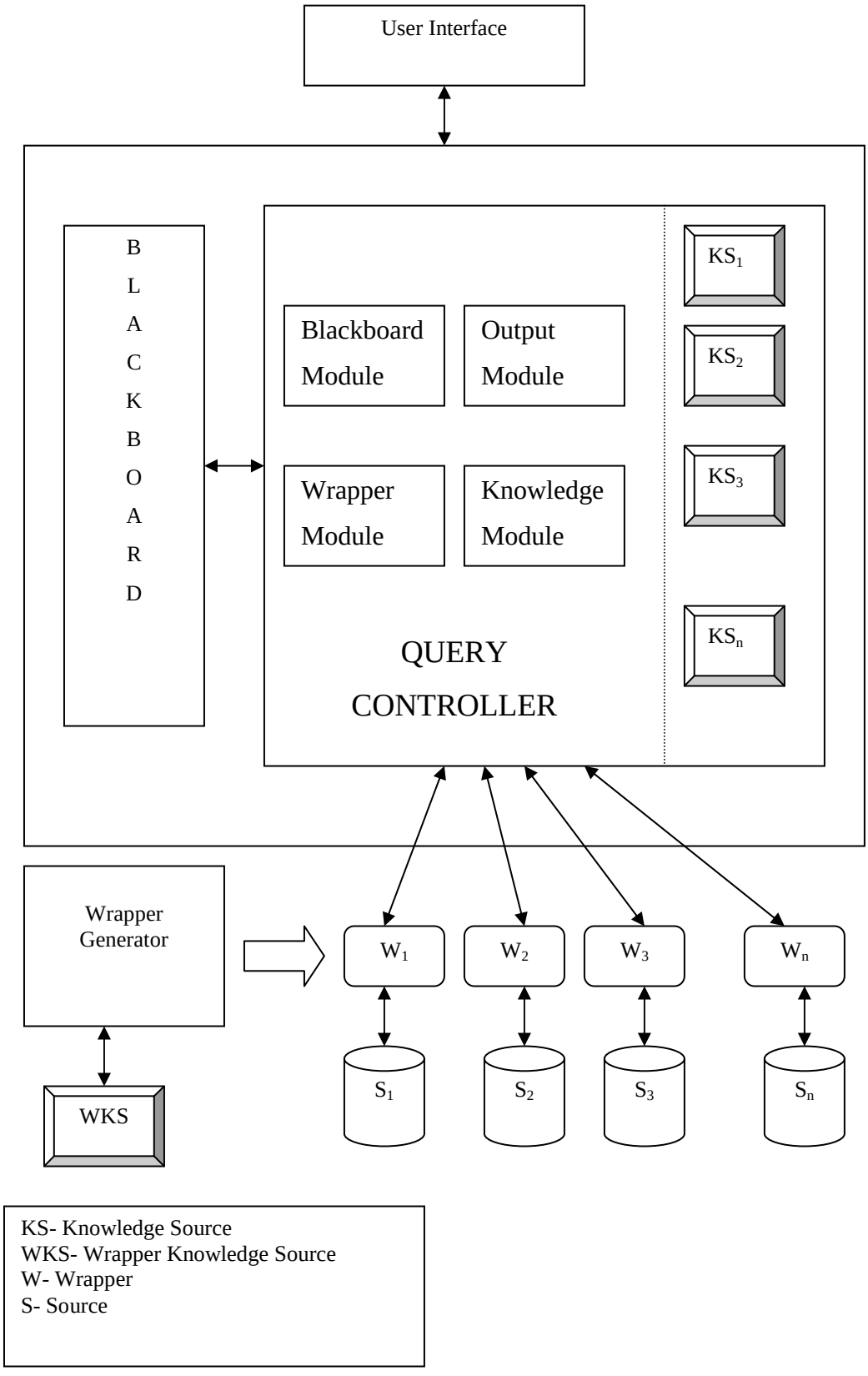

FIG 3.2 OVERALL SYSTEM ARCHITECTURE

sources may be accessed using different query languages or might not have a query language at all.

The wrappers are used to interface the system with the various heterogeneous sources. The wrappers translate the user-query, which might not be in a format understandable by the sources, into the native language for the sources. And the wrappers also translate the results returned by the source (in its native language) into a format understandable by the system. The wrappers are source-specific and one wrapper is built for each heterogeneous source that exists in the system. The Wrapper Generator generates the wrappers for the sources in our system.

The most important component of the IIS is the Query Controller. The Query Controller accepts the user query from the User Interface and performs the duty of passing it on to the relevant source(s), and refines and/or integrates the data coming from the wrappers. It divides the query into sub-queries, if necessary, depending on its knowledge of the information available in the sources. It also has the knowledge on how to integrate the information returned by the wrappers, and this knowledge is used to integrate the (intermediate) results produced by the sources. This knowledge is contained in knowledge sources that exist inside the Query Controller.

The Wrapper Generator makes easy the task of addition of a new information source and relieves the administrator of having to write a wrapper for each source that is available in the system, by automatically generating the wrappers. This is accomplished by using the specifications of the new source that is being added, which are provided by the administrator. These specifications are stored in a Wrapper Knowledge Source that is accessed for generating all the wrappers. The Wrapper Knowledge Source is populated

with information whenever a new source has to be added to the system (*i.e.*, the process of Source Registration).

We use the Object Exchange Model (OEM) [Papakonstantinou, *et al.* 1995], developed at Stanford University, as the common data model for our system. The classic <attribute, type, value> format seemed to be the best choice for the communication needs of the system. OEM is one such model and hence was used for our system. The common data model serves as a common ground for the information contained in different formats in the various heterogeneous sources that exist in our system. In other words, the data that has to be communicated to the information source by the system is converted into the native language of the source from OEM. And the data that are returned by the information source, which is in its native language, is converted into OEM so the system can understand it.

When a source is being added, details on the type of OEM object exported by the wrapper for this source are also provided. The system matches the requested attributes of the user query with the attributes of the returned object to see if this wrapper will be used for query execution to return the desired results. After specifying what a new source returns, details have to be provided as to how that information can be accessed. These details are necessary to provide information about how the system and the source communicate and how the results are translated.

The main function of the Query Controller of our system is to accept the user query and forward it to the relevant source(s), execute the query (or sub-queries) and then return the results to the user. We achieve this by using the Blackboard technique.

The Blackboard serves as the major data unit of our system. All user queries for information are initially placed on the Blackboard, from where they are read and processed by the Query Controller. The Blackboard is also the place for all the intermediate results that may need to be combined with intermediate results returned by other wrappers, or which can in turn be requests for more information. This process of reading requests and writing results on the Blackboard is continued either until the final solution is found or no more progress can be made, in which case a user alert is initiated.

The Query Controller is composed of different programming modules that perform their respective functions. It also contains the knowledge sources that contain rules for combining information from different sources. They are the Blackboard Module, the Knowledge Module, the Wrapper Module and the Output Module. Details on each of these modules are provided in the following chapters.

This chapter describes the overall details of the system and the architecture. The following chapters will provide the details specific to the thesis.

CHAPTER 4

DATA MODEL AND USER INTERFACE

The common data model being used by our system is the Object Exchange Model (OEM) [Papakonstantinou, *et al.* 1995]. For the purpose of communication between the system and the various heterogeneous sources and also within the system, a common ground becomes necessary. The classic <attribute, type, value> format seemed to be the best choice for these communication needs. OEM is one such model that has been used for information exchange in several other Information Integration Systems and so OEM was used for our system.

The user interacts with the User Interface to be able to query the system. The User Interface is graphical in nature and assists the user in formulating the query. The user query is then forwarded and written onto the Blackboard as a request. And the final results are filtered and displayed to the user via the User Interface.

Details of both the OEM model and the User Interface of the system are provided in the following sections.

4.1 Object Exchange Model (OEM)

The Stanford University Database Group developed the Object Exchange Model, popularly known as the OEM. OEM is being used as the data model for our system. The key advantage of OEM is that it allows for easy exchange of objects between various heterogeneous sources. Also, it is easy to integrate the information thus obtained from heterogeneous sources using OEM. The basic idea is that we associate a descriptive tag to

each object along with the value. For example, if we have to transfer the SSN for an employee, we can describe it as

{ssn, integer, 777777777}

where the tag “ssn” is a label, “integer” indicates the type of value and “777777777” is the value. This is the general description of an OEM object. In addition, every OEM object has a unique Object Identifier (OID), which is generally not displayed. There can also be complicated objects of the form shown below, where the object value is a set of (one or more) other objects. Here each component of the object has its own label.

The important feature of OEM that makes it very attractive is that there is no necessity to describe the structure of an object and hence it is not bound by the limitations of fixed schema or structures. In other words, we can say that each object has its own structure. And this structure is as follows:

{OID, Label, Type, Value}

The OID is a globally unique identifier that is assigned to each OEM object. The label is a tag, a variable length string that describes what the object does and also what it means in the present context. The type is the data type of the values that the object can take. For example, in the above example, the object can take any integer value. The standard types provided by OEM are int, string and set.

Sometimes, when the source is, for example, a Simulation System, it might not return textual data as results, but might instead return graphical data. For example, a graph-drawing program might return a graph when given the input variables. To take into account such scenarios, we add another type called the 'Graphical' type. Right now,

considering the types of sources that are currently available in our system, this is the only extra type that we think is needed.

And, finally, the value is the actual value of the object. For ease of representation, the OID of the objects is omitted in our examples

Example of an OEM object, that has a set of other OEM objects as its value, is as follows:

< result, set, { Employee1, set, { <ssn, integer, S_1> <first_name, string, f_1>
<last_name, string, l_1> },
{Employee2, set, {<ssn, integer, S_2> <first_name, string, f_2>
<last_name, string, l_2> } } >

The 'result' object shown in the above example is a complex object, which takes other OEM objects, 'Employee1' and 'Employee2', as its values. In turn, Employee1 and Employee2 are again complex objects having OEM objects as values.

4.2 User Interface

The User Interface displays a logical view of the system, used in query formulation. This view has all the objects (and their attributes) that can be queried and this helps the user in formulating the query. Suppose there is an internal queriable OEM object, called Employee, in the system, of the following format:

< Employee, set, { <ssn, integer, S> <first_name, string, f>
<last_name, string, l> }>

This OEM object would then be displayed as follows (logically represented):

Employee (ssn, first_name, last_name)

where ssn, first_name and last_name are attributes of the 'Employee' Object that can be queried by the user.

The attributes that the user queries are stuffed into the required resultant OEM object and this OEM object is placed on the Blackboard as a request. For example, if the user queries all Employees that have last_name 'Johnson', then the required resulting object would be of the form

< Employee, set, { <ssn, integer, S> <first_name, string, f>
<last_name, string, l> }>

The query value 'Johnson' is stuffed into this OEM object, giving the OEM object that is to be placed on the Blackboard, which looks as shown below.

< Employee, set, { <ssn, integer, null> <first_name, string, null>
<last_name, string, 'Johnson'> }>

When the query is executed and the final results are ready, the Query Controller sends them back to the User Interface, which are then displayed to the user.

The User Interface interacts with the System Knowledge Sources, which contain rules on the creation of new complex objects from objects returned by existing wrappers, and displays these complex objects to the user. The user can then choose to query them if necessary. There is an option provided to the user whether or not these complex items should be displayed.

When the user formulates the query to be submitted, the Constraint Table of the system is accessed to see if all constraints on the condition variable, if any, are being met. The Constraint Table is a system table containing a list of existing constraints on all the attributes of all the OEM objects returned by the wrappers. For example, if there is a wrapper returning an OEM object that has an attribute called 'gpa', where there is a constraint that 'gpa' has to have a numeric value that has to have a value between 2.0 and 4.0, then this constraint information is available in the Constraint Table.

A snapshot of the User Interface of the system is shown below, which comprises of the query section, where the user poses the query, the logical view section, which shows the overall logical view of the system using which the user frames the query and finally the results section, where the results returned by the system are displayed.

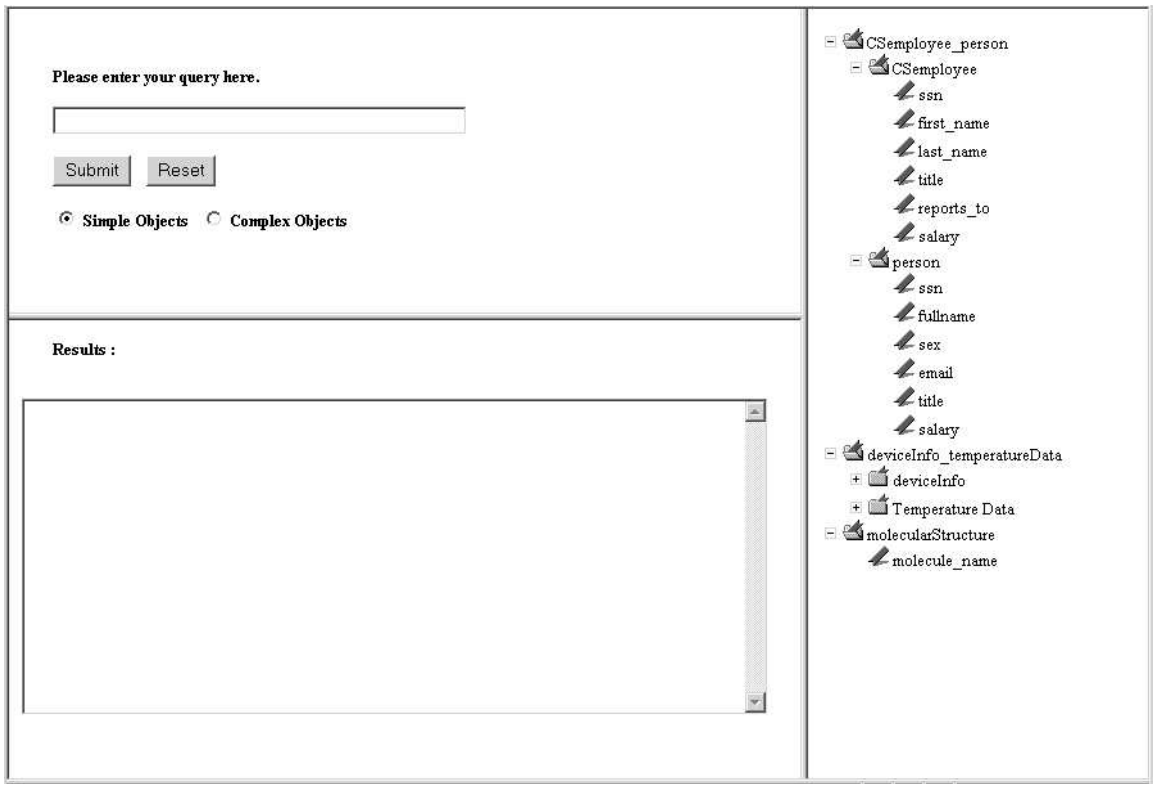

FIG 4.1 SNAPSHOT OF USER INTERFACE

The following chapter provides a detailed description on the working of the Query Controller and the Blackboard, which is the data unit of the system.

CHAPTER 5

BLACKBOARD AND QUERY CONTROLLER

The query processing is mainly carried on in the Query Controller. The Query Controller reads the request from the Blackboard and executes it and also writes any intermediate results, which can in turn be requests, onto the Blackboard again. The Query Controller is made of four different modules, each carrying out a different functionality, which enables this to happen. They are the Blackboard Module, the Knowledge Module, the Wrapper Module and the Output Module. Each of these modules has a unique functionality; details are provided in the following sections.

The Query Controller also contains the System Knowledge Sources. These knowledge sources contain all the rules on how different OEM objects, which are returned by the sources, can be integrated to form new (complex) objects that can in turn be queried on.

The Query Controller also interacts with the Key Table, the Constraint Table and the Cross-Reference Table. The Key Table contains details on the primary and foreign keys of the OEM objects returned by the sources. The Constraint Table contains constraints that need to be met by attributes of the OEM objects. And, finally, the Cross-Reference Table is a lookup on the attributes in different or similar domains that might mean the same. Each of these tables is populated during the process of Source Registration.

5.1 Blackboard

The Blackboard is the central data unit of the system. All requests to the system are placed on the Blackboard. These requests are processed and the intermediate results are again placed on the Blackboard. This process is repeated until the final solution is found or there is no other way to proceed, which initiates a user alert. The requests and intermediate results that are placed on the Blackboard are also in the form of OEM objects.

The Query Controller executes the requests that are on the Blackboard. If the request is for a complicated object that is formed by the integration of the intermediate results from several wrappers, then each of these intermediate results is posted back on the Blackboard and are considered as requests themselves, unless they are returned by the wrappers.

There are two options in implementing the Blackboard. The first alternative is to implement it as a database, which is more persistent and slow or the other alternative is to model the Blackboard as a memory unit, which is faster but more volatile. Hence a trade-off has to be made between the access times and availability.

In order to exploit the advantages of both the alternatives, a hybrid scheme is more efficient. In this scheme, a fixed size of the Blackboard is in the memory, and the rest is in the database. Whenever there is a request, the in-memory portion of the Blackboard is accessed first. And if there is no place on the Blackboard for the new request, then one of the requests that is already on the Blackboard is written out to the database to make place for the new request. The replacement scheme employed is the Least Recently Used (LRU) scheme.

5.2 Query Format and Conversion

The syntax for posing queries in the User Interface is in the form of

<Object.Attribute><Operator><Value>

*(<Connective> <Object.Attribute><Operator><Value>) **

The essential rules for posing a query that would be parsed by the system, would be as follows:

Query ***:=*** *Statement | Statement Connective Statement*

Statement ***:=*** *Entity Operator Value*

Operato ***:=*** > | >= | < | <= | =

Connective ***:=*** *And | Or ('And' has more precedence over 'Or')*

Entity ***:=*** *Object.Attribute Operator Value*

Object ***:=*** *<object name>*

Attribute ***:=*** *<attribute name>*

Value ***:=*** *<value>*

The parser of the system would then parse this query and create the OEM object that it represents and then stuff the appropriate attribute with the value mentioned, and this is posted on the Blackboard as a request. The format for this type of querying is just a suggestion so that the users can easily frame the query for the data that the user would need from the system.

5.3 Query Controller

The Query Controller resolves the query that is posed by the user and generates the final result. It is composed of different programming modules that perform their respective functions. They are the Blackboard Module, the Knowledge Module, the Wrapper Module and the Output Module. Details on each of these modules are provided in the following part of this document.

The Blackboard Module continuously monitors the Blackboard for any requests. When there is a request or intermediate result on the Blackboard, the Blackboard Module forwards this request to the Knowledge Module and goes back to monitoring the Blackboard again.

When the Knowledge Module receives a request from the Blackboard Module, it accesses the Cross-Reference Table to see if any attributes of the requested OEM objects mean the same across different wrappers. Then it compares the requested object against the heads of the rules in each of the knowledge sources of the Query Controller. If a match is found, then the objects of the tail of the matching rule are placed on the Blackboard again, along with the data structures representing the association with the parent object.

When the user makes a request for a complex object, the Knowledge Module figures out which objects need to be integrated, *i.e.*, the OEM objects in the tail of the production rule corresponding to the requested object. If there is a condition variable in the request, the Knowledge Module then accesses the Constraint Table to see and checks if there is any constraint on the condition variable and if so, if that constraint is being met. If the constraint is violated, then an alert is sent to the user with the constraint information.

Also, while integrating two or more objects to form a complex object, the Key Table is accessed to ascertain that all key constraints are being met, *e.g.*, if there is a primary key in an object that is a foreign key in another, then this information has to be utilized to ensure proper integration of objects.

This process is repeated until the request does not match the head in any of the production rules. Then it is checked to see if the object in question is being returned by some wrapper. If so the request is sent to the Wrapper Module.

When the request is on the Blackboard, the Knowledge Module has to figure out which wrappers have to be used for query execution. For this purpose, the Knowledge Module accesses the knowledge sources, which are part of the Query Controller, to see if there are one or more rules that provide knowledge for integrating the required information to result in the requested object. Then it accesses the Cross-Reference Table to see if any attributes from these wrappers mean the same. If they do, then the names of those attributes are sent to the Output Module, while the request for those attributes is sent to the Wrapper Module, specifying the names of the wrappers that are to be used, and the results returned are written back on the Blackboard.

The Wrapper Module has a very straightforward functionality. When the Wrapper Module receives a request, it forwards it to the appropriate wrapper. The wrapper does the necessary processing and forwards the request to the information source, which executes it and returns the results. The results are then translated again by the wrapper and returned to the Wrapper Module, which writes them back on the Blackboard, along with the data structures representing the association with the parent object.

Finally, the Output Module integrates all the necessary information that is available on the Blackboard, to create the final result object. For this purpose, it uses the information about identical attributes that is sent by the Knowledge Module earlier on. It also clears the Blackboard completely so that it is ready for new requests. The result is then sent to the User Interface for further processing and to be displayed to the user.

The following section provides a detailed example explaining the various steps involved in the execution of a user query and how the different modules interact to return the final result to the user.

5.4 Example

Consider the following OEM objects that are returned by the wrappers to two different sources. The latter is a simple variation of the '*person*' object that was described in the Wrappers section, the difference being the social security number being called '*socialNum*' in this object.

- *< CSemployee, set,*
  { <ssn, integer, S> <first_name, string, F> <last_name, string, L>
  <title, string, T> <reports_to, integer, R> } >
- *< $person_1$, set,*
  { <socialNum, integer, S> <email, string, E> <salary, integer, A>
  <sex, string, X> }>

Assuming that both '*CSemployee*' and '*$person_1$*' are being specified in the domain of a '*University*', the following note is made in the Cross-Reference Table when the source corresponding to '*$person_1$*' is being registered as follows:

<University, ssn> :- <University, socialNum>

Which implies that both '*ssn*' and '*socialNum*' attributes are equivalent in the '*University*' domain.

Suppose the user now poses a query that is as follows:

"Find the first names of all the employees in the University whose salary is greater than 60000".

When the query is posted on the Blackboard, the Blackboard Module, which is continuously monitoring the Blackboard, forwards it to the Knowledge Module. The

Knowledge Module then identifies that the required attributes are the '*first_name*' and '*salary*' from the '*University*' domain. The Knowledge Module of the Query Controller identifies the objects that contain these attributes. This is done by searching through the rules that exist in the knowledge sources of the Query Controller. Suppose the following rule is in one of the knowledge sources:

< CSemployee_person, set, { <ssn, integer, S> <first_name, string, F>
<last_name, string, L> <title, string, T> <reports_to, integer, R>
<emai, string, E> <salary, integer, A> <sex, string, X> } >
: -
< CSemployee, set, {<ssn, integer, S> <first_name, string, F>
<last_name, string, L>, < title, string, T>
<reports_to, integer, R>} > @ univDB
AND
< person$_1$, set, { <socialNum, integer, S> <email, string, E>
<salary, integer, A> <sex, string, X> } > @ flatfile

The Knowledge Module of the Query Controller recognizes that this rule may lead towards the result, since it has the required '*first_name'* and '*salary'* attributes in the head. It then decides that the wrappers that need to be accessed are, in this case, '*CSemployee*' and '*person$_1$*' and so a request for them is placed on the Blackboard again. The requests are then sent to the Wrapper Module for all such objects. When the results are ready, they are written back on the Blackboard.

The Knowledge Module again accesses the Cross-Reference Table and figures out the common attributes from the objects returned by the required wrappers based on whose value the integration is performed. Here, the attributes are identified as '*ssn*' and '*socialNum*' and this information is sent to the Output Module for later use.

After the results are ready on the Blackboard, the Output Module reads them and applies the query condition specified by the user, in this case, *"salary greater than 60000"* on the '$person_1$' objects. Depending on the values of the '*ssn*' and '*socialNum*' attributes, the final set of integrated result objects is generated, of which the requested '*first_name*' values are displayed to the user.

Sometime the mapping between the attribute constraints may not be direct. For example, you have ‘fullname’ in one object being mapped to a ‘first_name’ & ‘last_name’ combination in another. This problem can be generally termed as the constraint mapping problem. In such a case, the mapping rule should be defined in the Cross-Reference Table using functions that perform the appropriate partitioning or combining of attributes. Say, here, for example, we can have a function called LnFnToName (L,F). So the rule that would map these attributes would look as follows, where the mapping is done using the function, which would combine the attributes first_name and last_name so that they can be mapped:

<University, fullname> :- LnFntoName(<University, first_name>,
<University, first_name>)

The Cross-Reference Table would also have such information about objects and their mappings. The functions that are associated with this constraint mapping can be reused and a set of such functions if developed in advance can be used effectively whenever new sources are incorporated into the system.

Here it would be appropriate to mention the fact that though the Blackboard scheme looks closely like the working of the mediator-wrapper architecture, which is hierarchical in nature, the overall architecture of the Query Controller can be changed to a more parallel version where the Knowledge Module is divided into sub modules where

each of these modules can be processed in parallel by assigning them to individual processors, as shown below. The Knowledge Modules can be logically partitioned depending on the knowledge stored about the integration process *i.e.*, the Knowledge Modules can be logically partitioned in the same manner that the knowledge sources are populated. This is where the effectiveness of the Blackboard technique is evident and this parallel version clearly indicates its difference from the mediator wrapper architecture, which is more hierarchical in nature.

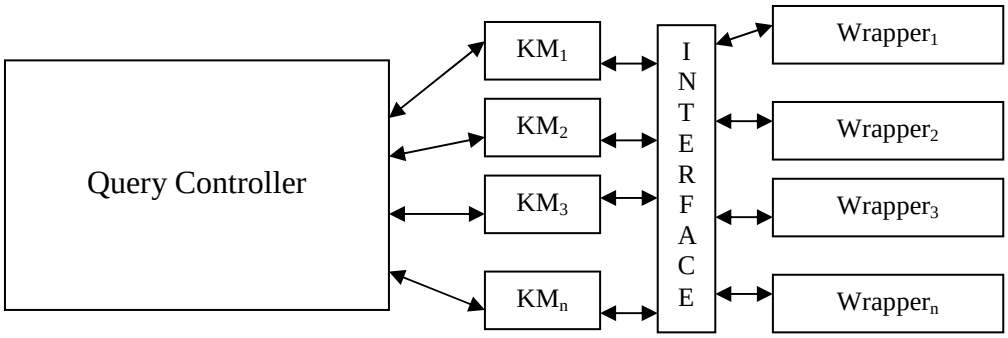

FIG 5.1 PARALLEL SYSTEM ARCHITECTURE

In the entire process, there are several places where query optimization can be applied. These details are provided in the following section.

5.5 Query Optimization

Given a query, there may be many different ways in which it can be processed and its results produced. Each of these ways is equivalent in terms of the final output, but might vary in the cost, *i.e.*, the amount of time that it takes to run. In such a case, which is the best method to run? This process of identifying the best method to execute to maximize efficiency is known as Query Optimization.

In our system, two places have been identified where query optimization can be performed. The first optimization is with regards to the query condition. According to the

current working of the system, the Knowledge Module identifies the wrappers that can return either one or more of the required attributes. Then the request is forwarded to the Wrapper Module, which in turn forwards it to the respective wrappers. The wrappers then return all the objects to the Wrapper Module to be posted on the Blackboard. Instead, if the user's query condition also can be executed directly at the wrappers, then the number of objects returned by the wrapper would be lesser if only not all of them satisfy the condition.

Another situation wherein query optimization can be performed is if the wrappers return only those attributes that they have and that are requested by the user, as opposed to returning the entire object with all the attributes each time. This way, the integration of the objects returned becomes easier, less time-consuming and more efficient.

CHAPTER 6

OTHER SUBSYSTEMS

The aim of this chapter is to give a brief insight into the other subsystems that exist in the system and that provide more functionality and effectiveness to the overall information system.

Wrappers are the interface between the system and the various heterogeneous information sources. The wrappers translate the data and queries between the data model of the system (OEM) and the native languages of the sources. The Wrapper Generator generates the wrappers for the sources in our system.

The Wrapper Generator is that component of the system that automatically generates wrappers for each of the information sources that exists in the system using the information available in the Wrapper Knowledge Source. During source registration, the Wrapper Knowledge Source is populated with information on how to handle a new source and what kind of a query request it expects, and all the other information necessary to interact with it.

The Wrapper Generator is the system component that eases the task of the addition of a new information source into the system. This entails automating the process of generating a wrapper for every new source that is added, thereby decreasing the duties of the system administrator reasonably. This automation is achieved by using pre-defined templates for various different sources, in addition with some source-specific information that is provided by the administrator while adding the new source. This information is

initially stored in the Wrapper Knowledge Source and used when the wrapper is being generated.

Every time a new information source is added to the system, it has to be 'registered', so that the system knows what kind of information is present in this source and how to access it. This requires an inherent understanding of the working of the information source. The Wrapper Knowledge Source is populated with the details of the new source that is being added into the system. Some of the information about the source that needs to be provided during the registration process includes the name, type, description and details of the source, the input and output formats, objects exported, request nature and type, and details on the connection and communication with this source. The Wrapper Generator generates the appropriate wrapper for this source depending on this information using the code templates for different kinds of sources and configuring them appropriately for the given source. And also as a part of this process the knowledge sources in the Query Controller are populated with information as to how to combine information from various sources, this process is aided also by the information from the Cross Reference Table, the Key Table and the Constraint Table.

There is a Cross-Reference Table in the system, which contains a mapping between all the terms in the different sources that mean the same, although they have different names. For example, the social security number of a person may be referred to as '*ssn*' in a database, but it might be referred to as '*social*' in a flat-file system, where '*ssn*' and '*social'* are both items of the OEM objects that are being returned by the two wrappers in question. This issue is relevant within the same domain or between different domains of reference.

This is thus an important part of the registration process. The Cross-Reference Table has to be populated with those attributes of the new source that are same as the attributes in the other (existing) sources. A typical entry in the Cross-Reference Table would look as shown below:

$<d_1, attrib_1> :- <d_2, attrib_2>, <d_1, attrib_3>$

where d_1 and d_2 are domains and $attrib_1$, $attrib_2$ and $attrib_3$ are the attributes within the domains that mean the same even though they have different names.

Similarly the Key Table and the Constraint Table are also populated during the source registration process, with the information about the keys for different OEM objects exported by the wrappers and also the constraints on the attributes of those objects.

These are the important components of the system other than the Query Controller and the User Interface, which increase the effectiveness of our Information Integration System.

CHAPTER 7

IMPLEMENTATION DETAILS

7.1 Objective

The motive behind the implementation of this system was to show that the design proposed is robust. We developed a 'prototype' system and in this chapter we discuss each component and its implementation, and the integration issues involved when trying to put the different components together. We chose to use information sources from different domains and using different technologies to further drive home the point that the design is correct and effective. In the following sections, we first provide a brief overview of the entire system and later concentrate on the individual components.

7.2 Technical Issues

The entire system was implemented using Java. We chose to use Java because it is highly platform independent and also because it can be used to make the system web-based in the future, if desired.

The mSQL database was used because it belongs to the most general class of databases used today, that is, relational. Also it is simple and freely available. It was also easy to find a JDBC driver to interface it with the Java source code. We used JDBC to connect with the mSQL database. The database runs on a remote server and this contributes to the distributed facet of the sources.

The knowledge base that we use is a fuzzy expert system that was developed using TIL Shell (fuzzy logic development tool by Togai Infralogic, Inc.) . We used this

system because the knowledge base generates rule bases that are in the C language and this was a good opportunity for us to prove that it is possible to interface our source code with a source that is non-Java. We accomplished this by using the Java Native Interface (JNI).

We use a flat file as a semi-structured data source. This was to show that, in future, if necessary, it would be possible to use the Internet as a data source, which can be considered to be the largest repository of unstructured and semi-structured data.

Our simulation system is a web-based applet. A wrapper was constructed over it that interacts with the GUI using OEM objects. The applet then returns a GUI object that is displayed on our screen. This proves that even complicated systems like simulation systems can be interfaced with our system.

7.3 Sources and Domains

The following are the systems that we used as data sources for the implementation of our project.

- Flat File System
- Database Management System
- Knowledge Base
- Simulation System

Each of these systems will be described in more detail in the following sections.

7.3.1 Flat File System

This is a semi-structured source that we created for the purpose of testing. It contains some details about University personnel, including:

Person : SSN, name, sex, title, salary and email

SSN here is the primary key and represents the Social Security Number of the person. And the name here implies that full name of the person.

We later try to integrate this information with the data available in another, structured source.

7.3.2 Database Management System

We have used the mSQL relational database as our second source. We have created two different databases – the first is a University Employee database and it contains the following information:

CSemployee - SSN, name, title, salary, reports_to

Various rules have been authored for the two sources – the flat file system and the University database – and the records can be joined on the various attributes.

The second database that we have created holds information about various storage devices. This database contains the device name and its temperature range. This database is used for integration with the Knowledge Base that we used.

deviceInfo – deviceName, minTemp, maxTemp

deviceName is the name of the device and minTemp and maxTemp together define the range of temperature that this device can be used to store within.

7.3.3 Knowledge Base

We use a fuzzy knowledge base that was created in the Biological & Agricultural Engineering Department. The system was developed using TIL Shell, which generates C files that we use. The C files are interfaced with Java, which is the language that we use, using Java Native Interface (JNI). The system has about 25 rules, and given the

'Consumption day' and the 'Degree of Ripeness', the system outputs the 'Temperature' that the fruit should be stored at.

7.3.4 Simulation System

We use the NanoCAD [Drexler, *et al.* 1991] , simulation system that was developed by Will Ware of SecureMedia, Inc. NanoCAD uses mathematical techniques of molecular modeling to simulate the behavior of molecules. With respect to our system, we can specify a molecule name as the input, and the molecular structure is displayed in a new window.

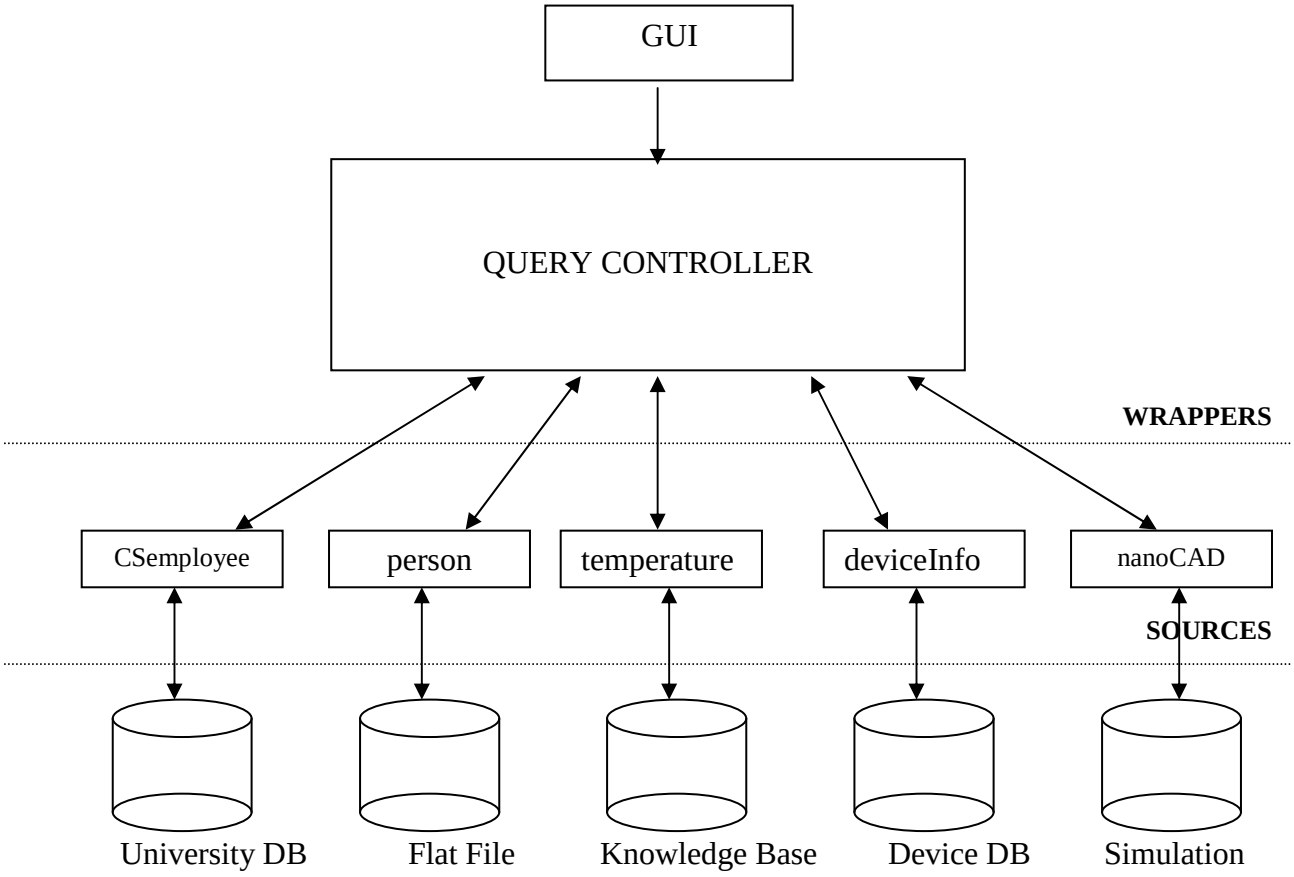

FIG 7.1 OVERALL SYSTEM STRUCTURE

7.4 Wrappers

The above was a brief description of the sources that we used for our system and their structures. The next step was to develop wrappers for each of these sources. This

was done so as to accomplish the representation of the data in each source in OEM, which is our underlying data model, since each source has a model of its own. Wrappers are source-specific software modules that are used to convert data from one model to another. Java DataBase Connectivity (JDBC) is used to access the remote databases and JNI is used to interface non-Java source code with our Java source code.

The wrappers are specified in WSL (Wrapper Specification Language), an extension of MSL.

The following are the OEM objects exported by each of the wrappers created for our information sources:

- *CSemployee – University DB – uses JDBC to connect to database*
- *< CSemployee { <ssn S> <first_name F> <last_name L> <title T> <reports_to R> <salary A> } >*
- *Person – Flat File*
  < person { <ssn S> <fullname F> <sex X> <email E> <title T> <salary A> } >
- *deviceInfo – Device DB – uses JDBC to connect to database*
  < deviceInfo { < deviceName D> <minTemp T> <maxTemp H> } >
- *molecularStructure – Simulation System*
  < moleculeStructure { < molecule_name > } >
- *temperatureData – KB – uses JNI to interface with non-Java source code*
  < temperatureData { <temperature T> <consumptionDay CD> <degreeRipeness DR> }

When each of these sources are registered these wrappers are generated and also the corresponding knowledge sources are populated with rules as to how the objects returned from various wrappers can be combined. Also the Cross-Reference, Key and Constraint Tables are populated with the appropriate information depending on the sources and the objects returned by them. A typical rule, for example, which would combine information

from the objects ‘CSemployee’ and ‘person’ would be populated in the knowledge source as a rule, which is defined as follows:

< CSemployee_person, set, { <ssn, integer, S> <first_name, string, F>
<last_name, string, L> <title, string, T> <reports_to, integer, R>
<email, string, E> <salary, integer, A> <sex, string, X> } >
: -
< CSemployee, set, {<ssn, integer, S> <first_name, string, F>
<last_name, string, L>, < title, string, T>
<reports_to, integer, R> <salary, integer, A>} > @ univDB
And
<person, set, {<ssn, integer, S> <fullname, string, FN>
<sex, string, X> <email, string, E>
<title, string, T> <salary, integer, SL>} > @ flatfile

Hence the complex object ‘CSemployee_person’ can be queried upon directly or the simple objects ‘CSemployee’ and ‘person’ can be queried if necessary.

7.5 User Interface and Querying

The User Interface reads the knowledge sources dynamically and the logical structures of all the objects in the system are shown to the user. The user has an option to choose the display of attributes for just simple objects or also complex objects. The user then uses the Interface as shown below to pose the query. The GUI takes the users query and then posts it on the Blackboard where the Query Controller reads the request and then processes it to send the results back to the User Interface, which are then displayed.

The following is a snapshot of the GUI, showing the logical view of the entire system. It has been organized hierarchically as a tree, showing the available queriable objects, depending on the option selected (simple or complex). The visible attributes that appear, as the leaf nodes in the following figure are the ones that can be queried on.

As shown below, the User Interface would contain three sections. The query section, where the user would pose the query; the system's logical view section, where the user would see the view of the system and all the queriable objects available; and finally the results section which displays the results of the query being posed. The above figure shows the snapshot of the User Interface, when a query is posed and the result is being displayed.

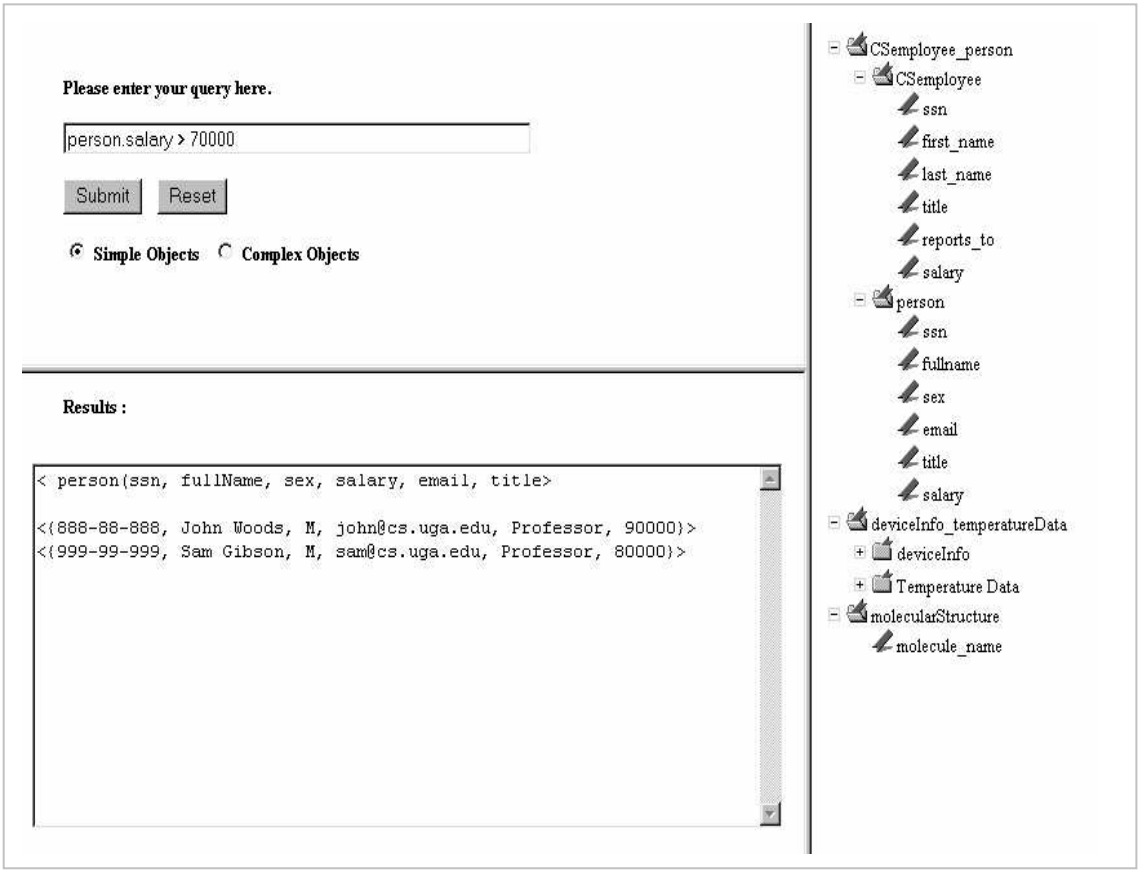

FIG 7.2 USER INTERFACE

Figure 7.2 shows the snapshot of the UI when a simple query is being posed. The system would respond in a similar manner for all the queries where the user poses the query in the query section and the results are displayed in the results section. The system will respond differently if the user queries a simulation system. In such a case, the system

returns a graphical object, which is displayed as a pop-up. Figure 7.3, shows the corresponding query and the query object returned.

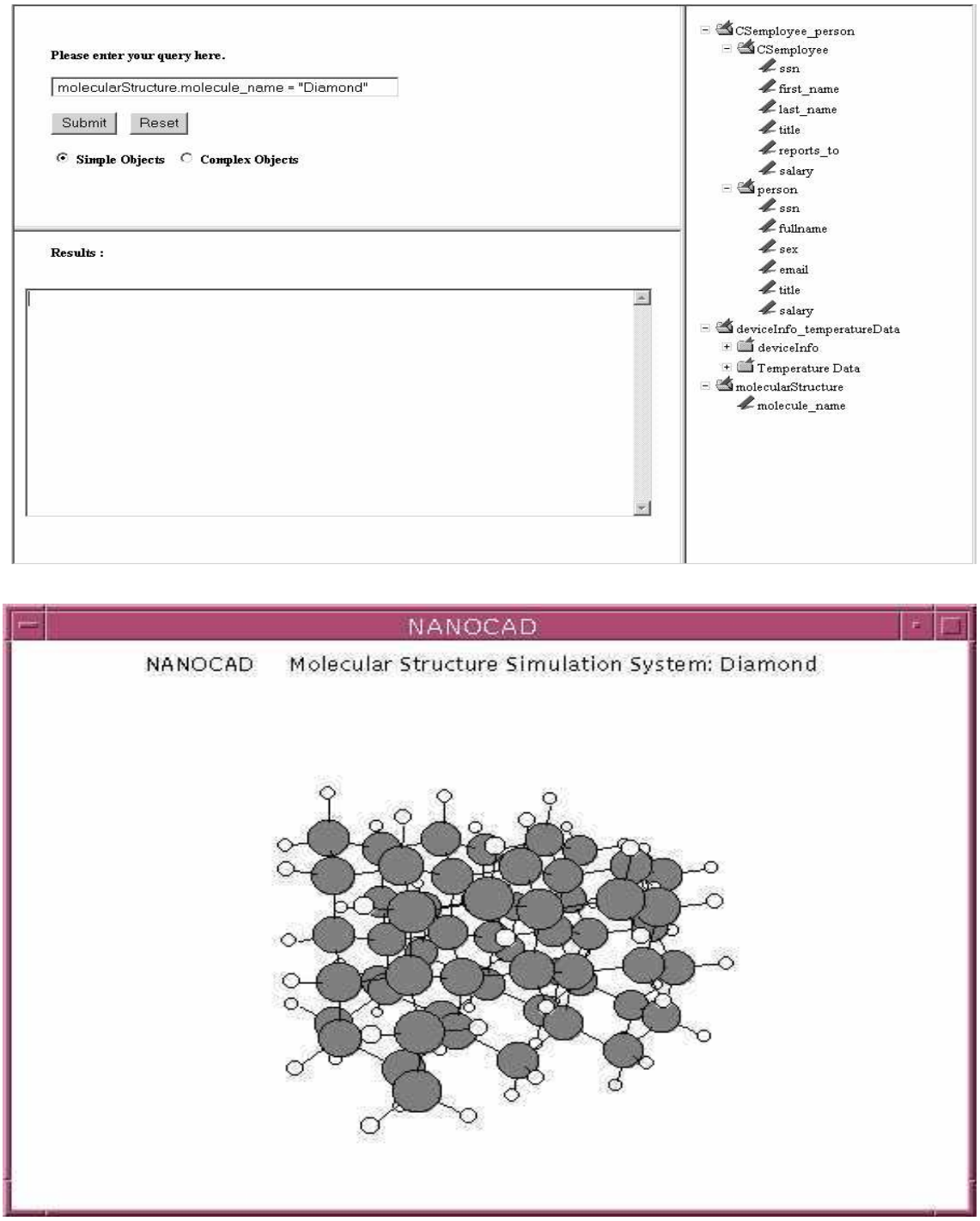

FIG 7.3 USER INTERFACE AND RESULT

This chapter was an attempt at providing information about some implementation details that went into the development of the various components of our system. In the above sections, we have tried to put forth the issues and details involved in the process of the implementation of the system. It is a working prototype of a Blackboard based IIS and we have employed five different sources to show that it can extended to include any number of sources from any domains.

CHAPTER 8

CONCLUSIONS AND FUTURE WORK

This thesis focuses mainly on the Query Controller of an Information Integration System, which is based on the Blackboard technique, and on the data modeling and interfacing for the same.

The Query Controller can rightfully be said to be the "brain" of the system, as this is the place where most of the decision-making capabilities reside. It makes use of the proven AI problem-solving technique, the Blackboard technique, and uses the knowledge generated when the new sources generated to aid in the decision-making. Hence, it can be safely said that the efficiency and correctness of the Query Controller itself depends on how well this knowledge is provided and how well a new source is incorporated into the system.

The data model we used, OEM, belongs to the classic <attribute, type, value> category. Though it can be used to represent most types of data that need to be dealt with today, the list is definitely not exhaustive. There may be some data in the future that may not be represented properly using OEM.

Though the basic architecture of the system may not vary with future additions that might come up due to the ever-growing information needs, the efficiency of the system can be further improved by providing tools that make the addition of new information sources into the system easier.

REFERENCES

Baum, L., R. Dodhiawala, and V. Jagannathan, 1989. The Erasmus System, *Blackboard Architectures and Applications, Academic Press,* pp. 347-370.

Bayardo, R., W. Bohrer, R. Brice, A. Cichocki, J.Fowler, A. Helal, V. Kashyap, T. Ksiezyk, G. Martin, M. Nodine, M. Rashid, M. Rusinkiewicz, R. Shea, C. Unnikrishnan, A. Unruh, and D. Woelk, 1997. InfoSleuth: Agent-Based Semantic Integration of Information in Open and Dynamic Environments, *Proceedings of the ACM SIGMOD International Conference on Management of Data,* Tuscon, Arizona, pp.195-206.

Borgida, A., R. Brachman, D. McGuinness, and L. Resnick, 1989. CLASSIC: A structural data model for objects, *Proceedings of ACM SIGMOD-89,* Portland, Oregon, pp.58-67.

Brachman, R., and J.Scmolze, 1985. An overview of the KL-ONE knowledge representation system, *Cognitive Science, Volume 9, Number 2,* pp.171-216.

Brachman, R. J., A. Borgida, D. L. McGuinness, P. F. Patel-Schneider, and L. A. Resnick, 1991. Living with classic: When and how to use a KL-one-like language, *Principles of Semantic Networks, Morgan Kaufmann Publishers,* San Mateo, CA, pp. 401-456.

Bressan, S., and C. Goh, 1998. Answering queries in context, *Proceedings of the International Conference on Flexible Query Answering Systems, FQAS'98,* Roskilde, Denmark, pp. 68-82.

Calmet, J., S. Jekutsch, and J. Schü, 1997. "A Generic Query-Translation Framework for a Mediator Architecture", *Proceedings of the 13th International Conference on Data Engineering ICDE,* Birmingham, UK, pp. 434-443.

Chawathe, S., H. Garcia-Molina, J. Hammer, K. Ireland, Y. Papakonstantinou, J. Ullman, and J. Widom, 1994. The TSIMMIS Project: Integration of Heterogeneous Information Sources, *Proceedings of IPSJ Conference,* Tokyo, Japan, pp.7-18.

Chu, W., M. A. Merzbacher, and L. Berkovich, 1993. The design and implementation of CoBase, *Proceedings of the 1993 ACM SIGMOD: International Conference on Management of Data, ACM Press,* Washington, D.C., pp.517-522.

Cody, W. F., L. M. Haas, W. Niblack, M. Arya, M. J. Carey, R. Fagin, M. Flickner, D. Lee, D. Petkovic, P. M. Schwarz, J. Thomas, M. Tork Roth, J. H. Williams, and E. L. Wimmers, 1995. Querying Multimedia Data from Multiple Repositories by Content: the Garlic Project, *3rd IFIP Working Conference on Visual Database Systems,* Lausanne, Switzerland, pp.17-35.

Corkill, D., K. Gallagher, and K. Murray, 1988. GBB: A Generic Blackboard Development System, *in Blackboard Systems, edited by R. Engelmore and T. Morgan, Addison-Wesley*.

Craig, I., 1995. *Blackboard Systems, Alex Publishing Corporation*.

Drexler, E., C. Petersonth, and G. Pergamit, 1991.*Unbounding the Future: the Nanotechnology Revolution, William Morrow and Company.*

Duschka, O., and M. Genesereth, 1997. Infomaster - An Information Integration Tool, *Proceedings of the International Workshop - Intelligent Information Integration - during the 21st German Annual Conference on Artificial Intelligence, KI-97.* Freiburg, Germany, pp.13-18.

Engelmore, R., A. Morgan, and H. Nii, 1988. Introduction, *Blackboard Systems, edited by R. Engelmore and A. Morgan, Addison-Wesley*.

Erman, L., F. Hayes-Roth, V. Lesser, and D. Reddy, 1986. The Hearsay-II Speech-Understanding System: Integrating Knowledge to Resolve Uncertainty, *Proceedings of AAAI-86,* Saint Paul, Minnesota, pp.58-64.

Finin, T., R. Fritzson, D. McKay, and R. McEntire, 1994. KQML as an Agent Communication Language, *Proceedings of the 3rd International Conference on Information and Knowledge Management (CIKM94), ACM Press,* Gaithersburg, Maryland, pp. 456-463.

Kirk, T., A. Levy, Y. Sagiv, and D. Srivastava, 1995. The information manifold, *Working Notes of the AAAI Spring Symposium on Information Gathering in Heterogeneous, Distributed Environments, Technical Report SS-95-08, AAAI Press,* Menlo Park, CA, pp.1312-1316.

Laasri, H., B. Maitre, T. Mondot, F. Charpillet, and J-P. Hatton. 1987. ATOME: Another Tool for Developing Multi-Expert Systems, *Proceedings of AAAI-Workshop on Blackboard Systems,* Seattle, Washington, pp.309-322.

Mena, E., V. Kashyap, A. Sheth, and A. Illarramendi, 1996. OBSERVER: An approach for Query Processing in Global Information Systems based on Interoperation across Pre-existing Ontologies, *Proceedings of the 1st IFCIS International Conference on Cooperative Information Systems (CoopIS '96),* Brussels, Belgium, pp. 223-271.

Papakonstantinou, Y., H. Garcia-Molina, and J. Widom, 1995. Object Exchange Across Heterogeneous Information Sources, *Proceedings of the Eleventh International Conference on Data Engineering,* Taipei, Taiwan, pp. 251-260.

Subrahmanian, V. S., S. Adali, A. Brink, R. Emery, J. J. Lu, A. Rajput, T. J. Rogers, R. Ross, and C. Ward, 1995. Hermes: Heterogeneous Reasoning and Mediator System, "http://www.cs.umd.edu/projects/hermes/overview/paper".

Rice, J.,1986. Poligon: A system for parallel problem solving, *Stanford Univ. Technical Report KSL-86*.

Twery, M. J., D. J. Bennett, R. P. Kollasch, S. A. Thomasma, S. L. Stout, J. F. Palmer, R. A. Hoffman, D. S. DeCalesta, J. Hornbeck, H. M. Rauscher, J. Steinman, E. Gustafson, G. Miller, H.Cleveland, M. Grove, B. McGuinness, N. Chen, and D. E. Nute. 1997. NED-1: An integrated decision support system for ecosystem management. *Proceedings of the Resource Technology '97 Meeting*. pp. 331-343.

Thiran, Ph., J-L. Hainaut, S. Bodart, A. Deflorenne, and J-M. Hick, 1998. Interoperation of Independent, Heterogeneous and Distributed Databases. Methodology and CASE Support: the InterDB Approach, *Proceedings of Coopis'98*, *IEEE*, New York, pp. 54-63.

Tomasic, A., L. Raschid, and P. Valduriez, 1996. Scaling heterogeneous databases and the design of DISCO, *Proceedings of ICDCS*, Hong Kong, pp.449-457.