

A Deductive Database Solution to Intelligent Information Retrieval from Legacy Databases

KayLiang Ong, Natraj Arni, Christine Tomlinson, Unnikrishnan and Darrell Woelk
Microelectronics and Computer Technology Corporation (MCC),
West Balcones Center Drive, Austin, Texas, USA,
{kayliang, arni, tomlic, unniks, woelk}@mcc.com

Abstract

In this paper, we will report on the experience of building a successful industrial application using the LDL++ deductive database as part of the technology transfer process to our sponsor company, Eastman Chemical Company. We will describe the nature of the problems faced by Eastman Chemical Company and show how the LDL++ deductive database technology helps to build an Intelligent Information Retrieval System (IIRS) to solve their problems. We will also elaborate separately on the specific benefits contributed by the IIRS, Deductive Database technology and the LDL++ system. Lastly, we will share some of the invaluable lessons learned from the experience.

1 Introduction

The field of deductive database has received much attention as a form of new technology for integrating knowledge-based systems with relational database technology. Much of the research has focused on theories and techniques for efficient computation. The fruits of such efforts have brought about the release of several prototype systems such as Glue/Nail [6], LDL/LDL++ [2, 3, 13, 1], Coral [7], Aditi [8]. The LDL system was released for use and experimentation by MCC shareholders and university researchers in 1989. This was followed by the release of the LDL++ system, a second generation system with an industrial-strength robust implementation and with many new and improved features based on feedback from users of the LDL system.

Despite the availability of the deductive database technology, little has been focused on the application aspect of the technology to real-world problems faced in the industry. There were research efforts to explore data dredging applications [12] such as [4, 10, 15] that attempted to find patterns in atmospheric and bio-genome data. However, there has been very little evidence of the value of deductive databases in the commercial/business arena. In the case of LDL/LDL++, several commercial applications are being explored right now. In particular,

LDL++ is being tested as a tool for Data Cleaning and Purification [5, 9] on large telecommunication databases in collaboration with Bell Communication and Pacific Bell. In addition, LDL++ also serves as a Knowledge Discovery/Mining tool for integrating inductive Machine Learning algorithm with deductive querying capabilities [14]. More detailed results of these efforts will be reported in the future once they have matured.

In this paper, we will share our experience in building an industrial application using deductive database technology. The application is termed an Intelligent Information Retrieval System (IIRS) because it is a system for retrieving legacy information driven by a domain-specific knowledge base. The users of the system are novices, the system provides a very simple and user-friendly graphical form-based user interface for specifying a request. As there is a significant knowledge gap between the users and the underlying data format, the system has to be intelligent enough to utilize the knowledge base to process the query. The application is built as part of the technology transfer process to Eastman Chemical Company, a sponsor of the LDL/LDL++ research at MCC. The IIRS is used in their research division for analysis of chemical compounds. As we will describe later, the whole effort is not a simple one way process such that there is only technology transfer to the users. Instead, it is a two way transfer process where much useful feedback was received from the users that prompted the revision and improvement of the technology itself.

2 The Problem

Eastman Chemical Company has been producing chemical products for the past 30 years. Before chemical products are manufactured, properties of these chemical compounds are examined and tested in the laboratory. Once the tests are performed, the information is recorded in a database. Thus, after 30 years, much valuable information has been accumulated. The latest version of the database has been migrated to a relational database system.

A *chemical product* is usually made up of a combination of compounds, which is also referred to as the *composition* of the chemical product. Each compound has its own spe-

cial encoding and different chemical products are created by mixing different compounds, with different quantity of different units of measurement plus other information. Compounds are also divided into different categories and there are domain-specific constraints depending on the categories of the compounds used to create the chemical product. The actual information about how each chemical product is composed is proprietary and will not be discussed in the paper.

Each chemical product is tested against a suite of different types of laboratory tests and each type of laboratory test determines the *property* of the chemical product. Based on these properties, one can determine how the chemical products can be used. The information stored in the database includes the compounds, composition and the various properties of the chemical products. The number of compounds, chemical products and properties is relatively large. This information provides significant competitive advantage to the company in terms of cost and response to new products in the market place. In particular, the information is being used repeatedly as follows:

- (1) Given a composition, determine if a chemical product of the same or similar composition has been manufactured or investigated before.
- (2) Given a set of properties, determine if a chemical product of the same or similar properties has been manufactured or investigated before.
- (3) Given a chemical product, determine what tests have been performed before and also determine if a particular test has been performed before.

(1) is necessary to evaluate a competitive product manufactured by a rival company. First, the composition of the chemical product is determined. Then, the information from the database can help to determine if the same or similar chemical product has been manufactured or investigated before. If an identical or similar product is found, (3) can be used to retrieve prior tests on these products. The process to test a chemical product is normally very expensive and (3) will help to avoid unnecessary tests. Furthermore, if sufficient information about the chemical product can be accumulated, the company can promptly begin manufacturing the product. Thus, this will give the company the benefits of lower costs and faster response time.

(2) is used when a customer requests the manufacturing of a chemical product based on some requirements on its properties. Again, from these constraints on the properties, the database could potentially have information about chemical products that have the same or similar property profile as the requested product. Even if only similar products are found, the information is useful to help chemists narrow down the choice of compositions. The composition can then be modified to fit into the requirements of the product requested.

Unfortunately, the retrieval of information for these

queries is too complex to be carried out directly by the chemists themselves. Historically, a chemist will have to consult the in-house database experts, who have the knowledge of both the database configuration and domain-specific knowledge about chemicals, in order to get the necessary information. When the database experts finally get the results, it will be sent back to the chemists. If the results are not what the chemists want, another request will be necessary until the chemists get what is needed. This poses problems in the following ways:

- *Long Turn-Around Time* It could take several weeks for the chemists to receive the results. The results returned by the queries may not be specific enough and thus, too much results could be generated in hundreds of pages of output that are useless to the chemists. Furthermore, subsequent queries are normally necessary to probe further into the result. As a result, the turn-around time is extremely long, tedious and frustrating for both the chemists and the database experts.
- *Knowledge and Representation Gap* There is a significant knowledge and representation gap between the users/chemists and the information in the underlying database. Information about the chemicals and laboratory tests are represented in a canonical form very different from the terms understood by the chemists. As a result, a chemist normally submits a request based on a set of conversation codes (sometimes informally on a piece of scratch paper) which the database experts can understand. The conversation codes are essentially a list of mappings between the encodings understood by the chemists and the actual canonical representation in the actual databases and they are used for conversation between the users and the experts. Following that, the database expert creates a SQL query that converts the terms specified by the chemists into the canonical representations. Furthermore, it is also the responsibility of the database experts to include additional domain-specific constraints in order to retrieve the information correctly.

The long turn-around time frustrates the chemists. Often, the time required to retrieve the necessary information is so long that it does not justify their time waiting for it. Thus, they end up re-performing the laboratory tests and may incur unnecessary cost. Furthermore, requests for new products from customers must be answered in a specific time-frame and if the time for information retrieval takes too long, it is simply not acceptable and requests may not be answered on time. The current setup for information retrieval also relies heavily on the experience and knowledge of the database experts. Eastman Chemical Company will face tremendous problems when these database experts retire. Their roles are not easily replaceable because each of them has accumulated a wealth of domain-specific knowledge about the conversation codes and the chemical domain. They also know how these chemicals are represented in the database sys-

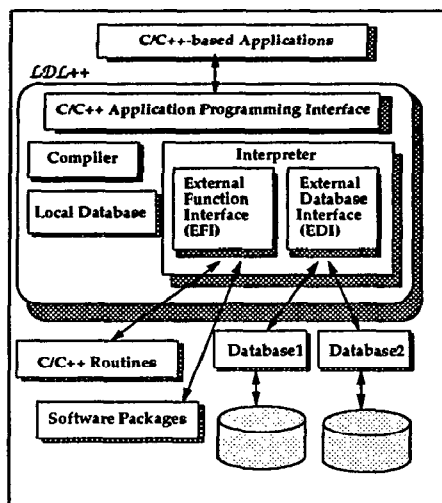


Figure 1: LDL++ Open Architecture
terms as well as how to pose queries to retrieve information.

These problems have prompted Eastman Chemical Company to look into commercial products as well as MC-C technologies for a solution to these problems. There was no commercial product that has both the capability to represent and perform inference on domain-specific knowledge and to query against legacy databases. The LDL++ Deductive Database technology was investigated as a possible tool for building a system to facilitate their information retrieval needs.

3 The LDL++ System

The LDL++ system is a deductive database system based on the integration of logic programming system with relational database technology. It provides a logic-based language that is suitable for both database queries and knowledge representation. More details on the LDL++ system and language can be found in [1, 3, 13, 18]. In this section, we will briefly describe some of the salient aspects of the LDL++ system as relevant to the building of the intelligent information retrieval system.

The LDL++ query language is based on Horn clause logic and an LDL++ program is essentially a set of declarative rules. For example, the following rules

```
ancestor(X,Y) ← parent(X,Y).
ancestor(X,Y) ← ancestor(X,Z),parent(Z,Y).
```

specify that a new relation ancestor/2 can be defined based on the relation parent/2. X, Y and Z are variables and ancestor and parent are predicate symbols. By declarativeness, we mean that the ordering between the rules is unimportant and will not affect the results returned. Deduction of all values of ancestor/2 is achieved

through an iterative bottom-up execution model.

The LDL++ language supports a rich set of complex data types such as complex objects, lists and sets in addition to other basic types such as integer, real and string. Examples of these complex types are *rectangle(1,2)*, *[1,2]* and *1,2* respectively. Thus, the LDL++ language is ideal for representing the domain knowledge in IIRS. Furthermore, rule-based inference capability allows generation of complex domain-specific constraints at run-time. The language also supports the meta-query facility which plays a very significant role in the IIRS. Based on run-time data, this facility first allows construction of rules, followed by the compilation of query form before invoking the query. This empowers the IIRS to handle queries that are not originally specified in the rule base.

The open architecture of the LDL++ system, shown in Figure 1, meets many of the demands of the IIRS. It is "open" to the C/C++ procedural languages in two ways: It provides an Application Programming Interface (API) that allows applications to drive the system and, an External Function Interface (EFI) that allows C/C++ routines to be imported into the inference engine. It is also "open" to external databases such as Sybase, Oracle, Rdb, Ingres, and DB2¹, through its External Database Interface (EDI). Both tables in the external databases and C/C++ interface routines are modeled as predicates through the EDI and EFI respectively. As a result, these external resources are transparent to the inference engine and the IIRS can plug into different database and procedural routines without having to make any changes to the overall implementation. This empowers IIRS to have transparent access to data from different sources and the front-end portion of the application does not have to change for different data sources. The EDI and EFI are also convenient for gathering data from multiple, heterogeneous databases or files.

4 The IIRS Implementation

We begin by first examining the configuration of the IIRS system. This will give an overview of software components, how they fit together and their implementation platform. The configuration is shown in Figure 2. As shown, there is a three-tier design to the configuration. They are:

- *The Client Process* Each individual user will have his/her own client process and can reside on any platform.
- *The Server Process* The server process provides the ability to process and dispatch concurrent queries.
- *The Data Repository* The data could reside at any other place transparent to the client process.

¹Sybase is a trademark of Sybase Inc., Oracle and Rdb are the trademark of Oracle Inc. Ingres are the trademark of Computer Associates Inc. and DB2 is a trademark of IBM Inc.

The purpose of this three-tier design is to ensure data independence. The underlying mechanisms in the client process do not need to know where the data comes from. In the long run, it will allow for transparent migration of the data from the old repository to new ones.

The client process is a single process that includes a graphical form-based user interface (GUI) and the LDL++ engine. Additional C++ routines are imported into the inference to support customized predicates. The LDL++ backend communicates with the server through a SQL Access Group Call Level Interface (SAGCLI). The server is based on the Extensible Services Switch (ESS) technology developed at MCC [16]. It serves as the multiplexer to receive queries from multiple client processes and dispatches them to the appropriate data repositories. In the current setup, there are two relational databases and both reside on DEC-supported RDB relational database on VAX mini-computers. These repositories are likely to be migrated to a new relational database on a new platform in the long run.

The chemists, who are novice users, manipulate graphical objects such as buttons, menus, forms, etc. provided by the graphical form-based user interface. It is implemented using Motif graphical user interface toolkit. The underlying implementation and configuration are completely hidden from the users. They do not know where the data comes from, how the queries are composed and how results are assembled before returning to them. This is a critical design decision that is planted consciously because the thought of having to use logic-based system or language will impede their desire to use the system. Queries are automatically formulated as they choose the options in the menus and buttons. More specific information is entered by filling slots in electronic forms.

When the IIRS client process is brought up, LDL++ schema, rules, facts (facts that represents some domain-specific knowledge) are automatically loaded into the client process. Query forms are then compiled once and ready for querying. The C++ routines are loaded into the process on a per-demand basis if used by the queries selected by the users. These routines are loaded once and subsequent queries do not require re-loading of the loaded routines.

When a user has completed formulating his/her query, an LDL++ query form is instantiated with the appropriate bindings or values. The knowledge base, encoded as LDL++ rules and facts, will perform inference and generate a meta-query expression represented as ground data. This meta-query expression is then fed into the LDL++ meta-query facility that generates new rules and new query forms. These newly created query forms are subsequently compiled at run-time. The compiler will perform the necessary optimization that will compress and collapse as many rules and literals and generate one or more SQL expression as compactly as possible. These

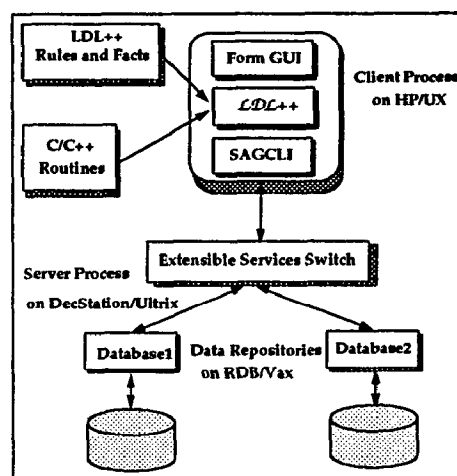


Figure 2: IIRS Configuration

SQL expressions are then dispatched to the server process through the SQLCLI interface. Results returned from server process is then propagated as tuples back to the GUI. Some results are filtered to provide more mnemonic and meaningful presentation to the users. The server process accepts the SQL expression and dispatches it to the appropriate database repository. This server process can accept queries from various client processes concurrently. Thus, multiple LDL++ client processes can be executing at the same time, each serving a different user.

5 Illustration of the Application

Even though the IIRS application is made up of different components, the core of it lies in the rule-based component driven by the deductive database engine. In this section, we will show and discuss a reduced simplified subset of the actual application and illustrate using one of the many query forms.

All the tables, regardless of the database in which they reside, are viewed transparently as LDL++ predicates. The database schema is declared as follows ²:

```
ess :: property1(SampleId: integer,
                PropValue11: string, PropValue12: string)
ess :: property2(SampleId: integer,
                PropValue21: float, PropValue22: string,
                PropValue23: float)...
```

There are about 100 properties, each represented as a separate table. Each record in a table represents one test on a particular property. Each property may have a different number of attributes. SampleId, PropValue11, etc

² As the information in the application is highly proprietary, table names, attributes names and values are modified sufficiently in order not to disclose too much information. However, the made-up description shall be adequate to illustrate the points.

are attribute name declarations while `integer`, `float` are column type declarations. The attribute `SampleId` is an index on the sample on which tests have been performed. The `ess::` prefix denotes the database server where the table is residing. Each test sample is also a composition which is made of a set of compounds. There is a primary table that stores the composition of the test sample. The schema for this table is:

```
ess :: composition(SampleId: integer,
                  CompoundCode: string,
                  CompoundQuantity: float)
```

Thus, a composition sample with `SampleId` of value 1000 could have the following entries in the composition/3 table:

```
composition(1000,'X..1',20.0).
composition(1000,'X..2',15.0).
composition(1000,'X..3',65.0).
composition(1000,'X..4',100.0).
composition(1000,'X..5',40.0).
composition(1000,'X..6',35.0).
```

'X..1', 'X..2', ... are canonical encoding of the compound. If a test on property1 has been performed, there will be an entry in property1/3 as follows:

```
property1(1000,15.22,'XYZ').
```

Various query forms are pre-compiled and available for the GUI to access. Here, we will illustrate using one query form. This objective of this query form is to find if there is a sample, in this case finding the sample id will suffice, that satisfies a set of composition constraints. The query form is denoted in LDL++ as:

```
export find_sample_id_from_composition(
    $CompConstraints,SampleId).
```

In LDL++, arguments prefixed with a '\$' in the query form provides hints to the compiler that the argument will be bound with some values at query time. Hence, the query form indicates that `$CompConstraints` will be supplied with a value while the result will be bound to `SampleId` after the query is evaluated.

There are various important considerations. First of all, the number of composition constraints are unknown when the rule base is loaded and query forms are compiled. The chemists can input any number of constraints when filling in the form. Thus, the query form must be ready to handle variable number of constraints. This is represented as a list of functors shown as follows:

```
[ compound('A1',range(75.0,45.0),
  compound('B2',range(84.0,50.0),
  compound('C3',range(10.0,5.0),
  compound('D4',range(100.0,10.0),
  compound('E4',range(100.0,10.0),...]
```

Each functor represents a compound with a compound code and a range of values. In this way, by having a list structure, an arbitrary number of constraints can be specified. The GUI is responsible for transforming the

electronic form entries into the list structure. Secondly, rules must be written to verify that the input compound codes, i.e. 'A1', 'B2', ..., etc are indeed valid. They are checked against a knowledge base of conversation codes represented as facts. In addition, they are transformed into whatever the canonical forms represented in the database. Thirdly, each of the compound codes is analyzed to determine to which category of chemicals they belong. Thus, several string processing routines specialized for this application are written in C++ and imported into the rule base.

Once the various categories of chemicals are identified, the rule base validates the domain specific constraints against the input constraints by the rule base. More importantly, additional domain specific constraints are generated and appended to the original constraints before the query is dispatched. For instance, consider the domain specific constraint that all compounds of a given chemical category must be summed to 100%. If 'B2' and 'C3' are the only compounds of the same chemical category, then we know that the query generated based on the constraints will never produce any result. Thus, the query is not evaluated any further and the query returns no solution. On the other hand, if 'A1' and 'D4' are the only compounds of the same chemical category, then an additional constraint is generated that specifies that the values selected for compound 'A1' and 'D4' must be summed to 100%. This will become clear once the generated rule is shown.

Once the constraints are filtered, transformed, verified and enhanced, they are processed through a meta-query facility that takes these constraints as a form of data and generates new rules that implements these constraints. Assume that 'A1' and 'D4' are the same chemical category while 'B2', 'C3' and 'E5' are the same chemical category. Assume that each compound code is transformed by adding '..' to the original encoding, then, conceptually, the following LDL++ rule is generated at run time:

```
meta_pred(SampleId) ←
  composition(SampleId,'A..1',V1),
  composition(SampleId,'B..2',V2),
  composition(SampleId,'C..3',V3),
  composition(SampleId,'D..4',V4),
  composition(SampleId,'E..5',V5),
  V1 <= 75.0, V1 >= 45.0, V2 <= 84.0,
  V2 >= 50.0, V3 <= 10.0, V3 >= 5.0,
  V4 <= 100.0, V4 >= 10.0, V5 <= 100.0,
  V5 >= 10.0, V1 + V4 = 100,
  V2 + V3 + V5 = 100.
```

The LDL++ compiler will then rewrite this new rule and through the SQL compression and collasation algorithm, transform it into a compact SQL statement and dispatch it to the backend database server. Many of the system features of the meta-query facility, the SQL compression and collasation algorithm, external procedural

interface, external database interface, etc. cannot be covered within the scope of this paper but will be covered in future publications [1]. In addition, this illustration is also by no means comprehensive. Many details about how the conversion codes are represented and implemented, how the constraints are processed, validated, enhanced and eventually transformed into a form suitable for meta-query facility are too tedious to be discussed here.

6 Evaluation of the IIRS

In this section, we will discuss how the IIRS has contributed to solving the problem faced by Eastman Chemical Company and the impact and differences that have been made before and after its implementation. We will focus on the benefits realized by the IIRS users. In addition, we will discuss the role deductive database technology plays in realizing this solution. In particular, we would like to answer the question of why the deductive database approach is essential and why other approaches are less suitable. Lastly, we will highlight some of the specific contributions of the LDL++ system and we will examine some of the useful features in the LDL++ system that lead to a superior implementation of IIRS.

6.1 Benefits of the IIRS

The specific benefits brought about by the installation of IIRS are:

- *Direct Access by Novice Users* One of the major differences that the IIRS has introduced is to provide the users with the ability to access the information directly. This gives them a sense of control as well as flexibility. This includes the flexibility to access information at a time convenient to them as well as the flexibility to experiment with different queries in the way they want to. Prior to the IIRS, the information gathering process was time-consuming and users had to go through an expert. In short, the IIRS actually eliminates the knowledge gap between the users and the underlying information.
- *Capturing Domain-Specific Knowledge* The building of the IIRS required acquisition of domain knowledge. One of the problems that prompted the building of IIRS was that some of the in-house experts who have been handling the queries for the chemists will be retiring. Thus, the IIRS, to a certain extent, replaces these experts. More significantly, information retrieval from the databases by the chemists does not depend on the availability of these experts any more.
- *Increase Use of Legacy Information* The GUI represents a significant step to encourage better use of the legacy information. Before IIRS, each request for a query relied on the availability of the experts. Such bottlenecks discourage request of queries from the users. As mentioned before, re-performing laboratory experiments rather than searching for previous experimental results can be expensive. More importantly, chemists cannot

take advantage of the valuable information and knowledge about manufacturing a chemical product that may be available in the legacy data. The easy-to-user GUI certainly makes the legacy information more accessible and helps to prevent unnecessary cost.

- *Information Filtering and Augmentation* One of the roles of the IIRS is to provide filtering of both input and output information. Input specifications (also referred to as conversation code) entered and understood by the users are mapped to the canonical representation in the data repositories. Furthermore, values returned from the database are filtered into a more presentable format and if necessary, augmented with more mnemonic information. For example, the canonical representation of a compound could be replaced or augmented with the actual chemical name understood by the users.

In short, IIRS represents a technology leap for Eastman Chemical Company as an organization. But more importantly, it improves the environment for conducting a business with a higher productivity and a lower cost.

6.2 Benefits of the Deductive Database Technology

Why is deductive database technology better able to solve the problems faced by Eastman Chemical Company than other technologies such as the relational database or logic programming technology? The IIRS could certainly be implemented using C or C++ with embedded SQL on top of a relational database. Or it can be implemented by extending logic programming system such as Prolog with the interface to databases. However, the time, cost and efforts would be tremendously larger. This is due to the following benefits made available by deductive databases:

- *Knowledge-Driven Database Querying* The IIRS requires the ability to close the knowledge gap between the users and the canonical representation in the data repositories. Thus, there must be a representation language for capturing the domain specific knowledge. The logic-based representation of deductive database language fits this requirement very well. Secondly, inference on this knowledge is required to perform information filtering and augmenting, analysis of input specifications and generation of domain-specific constraints. Furthermore, the application also does some form of constraint checking by pre-evaluating and pre-validating the input queries. Queries that are deemed to fail will be intercepted, cancelled immediately and not dispatched to the server at all. Once the input specifications are processed by the knowledge base, the same specifications is also expressible as a query to the database. Thus, deductive database satisfies these multiple needs of the IIRS very well in supporting a database querying environment driven by a knowledge base. Such capabilities to represent knowledge structures and perform inferences and constraint checking are not inherent in procedural languages such as C/C++ which they are never designed for. Thus, rebuilding such capabilities from scratch using C/C++ is time-consuming and it makes no sense spending all the

resources developing yet another knowledge-based system. Furthermore, there is also the tedious job of dealing with the impedance mismatch between the C/C++-based manipulation language and the SQL querying language. Many details have to be implemented to handle the generation of the query and the conversation of data returned from the database server. On the other hand, logic programming systems can handle this knowledge representation and inference requirement equally well but they lack the ability to query against databases and generate optimized SQL expressions from the rules. Deductive databases can perform the two tasks equally well in a seamless fashion because they have the knowledge representation and inference capability as well as the built-in facilities to perform database querying.

- *Easy Maintainability and Extensibility* As the IIRS is slowly emerging as a tool used by the chemists on a daily basis, the requirements are evolving as the users are exposed to its various functionalities. Thus, continuous modifications are necessary and new functionalities are constantly being requested. The IIRS has proven to be very flexible with respect to upgrades. The experience has shown that the time taken to build the rule-based portion of the application is short with a high-level declarative language. However, frequent iterations of revisions are necessary. Often, incomplete or incorrect knowledge is captured in the knowledge base due to misunderstanding in the knowledge acquisition process. Furthermore, since the rules determine the format of the results returned, users often request changing of the format for better presentation. Fortunately, as most of the applications are coded in high-level declarative rules, they can be easily revised, maintained and extended. Such tasks would require significantly more effort if the implementation were done in C/C++.

6.3 Benefits of the LDL++ Technology

Many of the features in the LDL++ system have been found to be critical in the implementation of the IIRS. This has confirmed the benefits of many of the design decisions when the LDL++ system was being developed. In particular, the open architecture of LDL++ makes it possible to integrate the various implementation components, each of which has its own strengths and merits. The second important design decision was the transparent access to multiple heterogeneous legacy databases. More importantly, the experience in building the IIRS provided feedback that resulted in improvements to many aspects of the system. In particular, the meta-query facility were developed due to the IIRS effort. The roles of these LDL++ features in the IIRS are discussed in detailed below:

- *Open Architecture* As shown in the Figure 1, the open architecture of the LDL++ system offers various channels to access external resources in addition to the schema, rules and facts. The IIRS demands many capabilities. First, it has a GUI, which has to be implemented using the C-based Motif toolkits. Secondly, it also requires

string manipulation and certain customized lower level processing that can only be done efficiently in a procedural language such as C/C++, not in declarative rule-based language like LDL++. In addition, the IIRS has to be able to interface with external legacy databases. The API of the LDL++ system offers the interface for the C-based implementation of the GUI. In fact, through the API, the GUI serves as the master application that drives the LDL++ engine. The C++ routines are imported into the system through the EFI and thus, new customized predicates based on these routines can be defined. Lastly, external legacy databases can be accessed through the EDI. Without this ability to interface with various different types of components, building of the IIRS would have been very difficult, may be impossible.

- *Transparent Access to Heterogeneous Legacy Databases* One of the constraints when building the IIRS is that it requires no reconstruction of the legacy databases, at least in the first few years of operation. As shown in the configuration, the client process makes no assumption about the locale of the data repositories. Thus, other applications that are using the data repositories are not affected by the installation of the IIRS. Furthermore, when the data repositories are eventually migrated to a different locale on a different vendor database on a different platform, the implementation of the facilities on the client process, i.e. the GUI, the rules and facts, does not need to change at all. Another ability brought about by the IIRS is the ability to integrate information from different legacy databases in a transparent manner. The users are not aware of whether the query sent involves many databases and which part of the results come from which database.
- *Meta-level Facility* The development of the meta-query facility originated from the need to generate SQL constraints based on input data and execute them at run-time. It was later generalized to generate rules and query forms and compile them at run-time. The meta-query facility is unique in the sense that it offers flexibility at run-time. New rules and query forms are created and deleted dynamically and the choice of which predicate to query can be delayed and can be decided based on the input data values, rather than fixed to the predicate of the query form. Due to the nature of the IIRS where queries are driven by the knowledge base and the input data, the meta-query facility is absolutely essential for the successful implementation.

7 Conclusions

In developing the IIRS, we have learned some very critical lessons. The first is that further development of industrial applications are essential in order for deductive database to mature into the commercial arena. The experience gained from these industrial applications is invaluable as a form of feedback that is not available from academic research and applications. More practical features can be discovered and designed to meet real needs from the eventual users. Secondly, a deductive database must be

designed with an open architecture in mind. It is recognized that a declarative rule-based language is more expressive for specifying queries more complex than SQL queries. However, to develop a complete solution for the users, the deductive database technology itself does not meet all the requirements such as procedural processing, GUI etc. Thus, by having an open architecture, it serves the glue with the freedom to tap into other resources.

Thirdly, access to legacy databases is important. Existing systems should be allowed to take advantage of deductive database capabilities quickly and directly without having to migrate all the data into deductive database format. The migration process is cumbersome and takes time and it presents the risk of data corruption. An extensible approach is more suitable by gradually enhancing existing legacy databases with new functionalities in an incremental manner using deductive database techniques. Lastly, we would like to remark on the issue of performance in the case of IIRS. In almost all cases, the time taken by the inference engine is insignificant relative to the time taken to process the SQL queries dispatched to the server. Thus, query processing at the data repositories and communication cost of data transfer represent the largest bottlenecks. Thus, it is critical that optimization in the deductive database compilers and engines should pay more attention to such factors. As a result, to minimize communication cost, the LDL++ compiler attempts to push as many joins and selections as possible to the server by generating the SQL statement as compactly as possible.

In view of the successful completion of the first usable IIRS system, further applications are being explored. In particular, the LDL++ deductive database technology could be used to integrate heterogeneous databases from different departments or divisions into one uniform view and present to the users for various decision making. Some databases are mainframe-based such as DB2 and IMS while others could be relational databases on Unix platforms. The LDL++ system can tap into whatever information resources accessible by the ESS component of the IIRS. Furthermore, the LDL++ system could also be used for migration of legacy databases into newer databases on newer platforms such that old or erroneous data are filtered or corrected using the rule base. There is also an on-going pilot effort to perform knowledge mining on the same databases based on an approach that combines inductive learning with deductive database technology. The purpose is to attempt to discover new knowledge hidden in the huge data pool and such discoveries may return substantial future business values in the long run.

In short, the potential of deductive database technology is tremendous and the IIRS is one step closer to demonstrate that the technology is indeed useful in real-world applications.

References

- [1] Arni, Ong, Tsur, and Zaniolo, *LDL++: A Second Generation Deductive Database System*, Working Paper, 1994.
- [2] Chimenti, D. et al., *The LDL System Prototype*, *IEEE Journal on Data and Knowledge Engineering*, vol. 2, no. 1, pp. 76-90, March 1990.
- [3] Naqvi and Tsur. 1989. *A Logical Language for Data and Knowledge Bases*, W. H. Freeman Company.
- [4] Muntz, R.R., E.C. Shek and C. Zaniolo, *Using LDL++ for Spatio-temporal Reasoning in Atmospheric Science*, Vancouver, Canada, 1993.
- [5] Ong, KayLiang, Sheth, Amit and Wood, Christopher, *LDL++ and Q-Data: a Practical Deductive Database in Action*, Working Paper (1994).
- [6] Phipps, G., M.A., Derr and K. A. Ross, *Glue-Nail: a Deductive Database System*, *Proc. 1991 ACM-SIGMOD Conference on Management of Data*, pp. 308-317 (1991).
- [7] Ramakrishnan, R., Srivastava, D. and Sudarshan, S., *CORAL: A Deductive Database Programming Language*, *Proc. VLDB Int. Conf.*, pp. 238-250, 1992.
- [8] Ramamohanarao, K. *An Implementation Overview of Aditi Deductive Database Systems*, *Procs. Third Int. Conference on Deductive and Object-Oriented Databases*, Dec. 6-8, 1993, Scottsdale, Arizona.
- [9] Tsou, E., et al., *Improving Data Quality Via LDL++, ILP'93 Workshop on Programming with Logic Databases* Vancouver, Canada, 1993.
- [10] Tsur, S., F. Olken and D. Naor, *Deductive Databases for Genomic Mapping*, *Proc. NACLIP Workshop on Deductive Databases*, Ed. J. Chomicki, Nov. 1990.
- [11] Tsur S., *Deductive Databases in Action*, *Proc. 10th, ACM SIGACT-SIGMOD-SIGART Symp. on Principles of Database Systems*, pp. 205-218, 1990.
- [12] Tsur S., *Data Dredging*, *Data Engineering*, Vol. 13, No. 4, IEEE Computer Society, Dec. 90.
- [13] Tsur, Arni, and Ong. 1993. *The LDL++ User Guide*, *MCC Technical Report*, Carnot-012-93(P).
- [14] Shen, W. Mitbender, B., Ong, K. and Zaniolo, C., *Using Metaqueries to Integrate Inductive Learning and Deductive Database Technology*, *AAAI Workshop on Knowledge Discovery from Databases*, 1994.
- [15] Wing-Kwong Wang, *Logic Programming and Deductive Databases for Genomic Computations: A Comparison between Prolog and LDL*, *Proceedings HICSS*, 1993.
- [16] Woelk, D., Huhns, M., Jacob, N., Ksiezzyk, T., Ong, K., Shen, W., Singh, M., and Tomlinson, C., *Carnot Prototype*, to appear *Object Oriented Multidatabase Systems*, Ed. Omran Bukhres and Ahmed Elmagarmid, 1994.
- [17] Zaniolo, C., *Design and Implementation of a Logic Based Language for Data Intensive Applications*, *Proc. of the 5th Int. Conf. and Symp. on Logic Programming*, pp. 1666-1687, MIT Press, 1988.
- [18] Zaniolo, C. 1992. *Intelligent Databases: Old Challenges and New Opportunities*. *Journal of Intelligent Information Systems*, 1. 271-292. Kluwer Academic.