Studies on Integration of Heterogeneous Information Sources

Koichi Munakata

February 2000

Doctor's Thesis
submitted to Graduate School of Information Science,
Nara Institute of Science and Technology
in partial fulfillment of the requirements for the degree of
DOCTOR of ENGINEERING

Koichi Munakata

Thesis committee: Shunsuke Uemura, Professor
Minoru Ito, Professor
Hiroyuki Seki, Professor
Masatoshi Yoshikawa, Associate Professor

Studies on Integration of Heterogeneous Information Sources*

Koichi Munakata

Abstract

The spread of computer networks enables us to obtain and utilize various kinds of information from remote sites maintained by local administrators. In order to make the most use of the existing, rich assets of information, it is important to provide the users with the integrated view of information sources.

This dissertation addresses the problems in integration of information sources. As a system architecture, we assume the *mediation architecture*. In this architecture, in response to a query request from an application program, the integration module, or a *mediator*, issues subqueries to heterogeneous information sources, acquires the subquery results, integrates them, and returns the query result back to the application program.

The first goal of this dissertation is to provide the users with the maximum information obtained from multiple information sources. The data obtained from information sources are, by nature, incomplete in their contents and data structure, producing irregularity of data instances. The key to provide the integrated view of such incomplete data is shown to be the outerjoin, predicate and foreign function operators. Algorithms that incorporate the key operators are described

* Doctor's Thesis, Department of Information Systems, Graduate School of Information Science, Nara Institute of Science and Technology, NAIST-IS-DT9961025, February 7, 2000.

to create a query processing plan for producing the maximum information that satisfies the conditions imposed by a conjunctive query.

The second goal is to produce and execute the query processing plan based on a *self-descriptive object model*, which is a nested structure of subobjects, each consisting of an object identifier, a label which describes the meaning of the subobject, a value, and the type of the value. This data model is simple but flexible to absorb the structural discrepancies among various data models, even allowing the structure of each data instance to be different from one another. Several query processing techniques inherent in handling the data irregularities in the data model are described.

The third and final goal is to mediate the temporal heterogeneity of *periodically generated data sequences* (**pgds**'s), each having a constant period of data generation. In developing complex real-time systems by integrating various sensors and sub-systems, it is important to acquire a synchronous data combination (or a snapshot) that consists of data each obtained from different **pgds**'s. For the purpose of determining the optimum data combination, an evaluation function is proposed based on novel criteria: *freshness* and *synchronousness* of the data combination. Then, query processing frameworks are presented based on the evaluation function. Finally, as a concrete example, it is shown that the proposed framework can be applied to a road obstacle and accident detection system with multiple sensors, enabling the accurate detection of the objects nearly half the size as compared with a naive algorithm.

Keywords:

database, heterogeneity, integration, outerjoin, self-description, periodic data

Contents

List of Figures

List of Tables

Acknowledgments

I am grateful to my advisor, Professor Shunsuke Uemura, for his advice and guidance while at Nara Institute of Science and Technology.

I wish to thank Professor Minoru Ito and Professor Hiroyuki Seki for their numerous comments and suggestions to improve this dissertation.

I would like to express my sincere gratitude to Associate Professor Masatoshi Yoshikawa for his continuous guidance and support throughout the research at Nara Institute of Science and Technology.

I wish to thank Professor Hector Garcia-Molina and Professor Gio Wiederhold at Stanford University and Assistant Professor Yannis Papakonstantinou at University of California, San Diego, for numerous discussions and advice while I visited the Database Group at Stanford University.

I am grateful to Professor Yahiko Kambayashi at Kyoto University for his precious advice in initiating my research activities in the field of computer science.

I would like to thank Dr. Ichiro Nakahori, Dr. Shigeru Abe, Mr. Kenji Shima, Dr. Yoshihiko Ozaki, Dr. Makoto Tsukiyama and Group Manager Seiichi Shindo at Mitsubishi Electric Corporation for giving me the opportunity of the research and for their advice and encouragement.

Finally, my thanks go to all the members of Uemura Laboratory for discussions and help.

Chapter 1

Introduction

1.1 Integration of Heterogeneous Information Sources

Information is one of the most important assets of humans. In the 15th century, Gutenberg invented the printing press, achieving drastic improvement in availability of information. Again, at the end of the 20th century, the expansion of computer networks, such as the Internet, initiated an information revolution, enabling us to obtain various kinds of information generated everywhere in the world within a few seconds. As a result, we have access to huge amount of information maintained by different administrators. Consequently, on one hand, we are given opportunities to acquire the high quality information of interest out of the rich assets of information sources; on the other hand, it has become increasingly difficult to find the necessary information from the flood of huge data. Under such circumstances, it is important to offer the tailored, integrated view of existing information sources to users, helping to focus on the domain of interest.

Integration of existing information sources is important not only in the above

mentioned open network environments, but also in closed network environments, or the *Intranet*, for organizational entities such as private enterprises. As an alternative to the approach of integrating existing information sources, there is an approach to design and develop the whole system from scratch in a uniform fashion. In the latter approach, the system is designed carefully to provide the functionalities exactly needed and to yield the best performance. A typical commercial product thus designed is SAP/R3, which is one of ERP (Enterprise Resource Planning) systems and a product of SAP AG in Germany. In this system, all the data are stored in a single relational database. Compared with such systems, the approach of integrating existing information sources has the following advantages:

1. It enables us to make use of the existing systems.

   It is common that an enterprise which is planning to develop a new information system has some existing systems. By integrating the existing systems, the enterprise can minimize the cost and time for developing the new system, avoiding the high risk of investing huge amount of budget at a time. The use of the existing systems also enables us to establish a reliable system easily since they have stood the test of time.

2. It allows gradual expansion of the system.

   The enterprise can expand the system easily over years, incorporating the state-of-the-art technology. This aspect is of particular importance today, when the information technology is making rapid progress. Also the enterprise can migrate from the existing, independent systems to the integrated system gradually, making use of the yearly budget. Another practical aspect is that the enterprise can avoid the risk of depending on a single vender, by being able to replace a part of the information sources with the products

of more promising venders.

3. It enables local administrators to manage the information autonomously.

   As the amount of information becomes larger, information generated at local sites can be better managed by the local administrators: a single section or organization is not capable of administering huge amount of information of various kinds.

In this dissertation, the approach of integrating existing information sources is taken. In integrating existing information sources, problems arise: autonomy of and heterogeneity among information sources. In general, information sources are developed by different venders at different times with different technologies. What is more, they are maintained by local administrators. Thus information sources are heterogeneous: they have different data models, data structures, schemas, interfaces, semantics/vocabularies of data, constraints on data, and discrepancies of data generation times. Also, information sources are autonomous: the administrators and the users may change arbitrarily the schema, interface, and the contents of the information sources.

The goal of the study in this dissertation is to provide the users with the integrated view of information sources: not only relational databases or object oriented databases, but any kind of information sources including information servers (such as World Wide Web), simulators, and sensors. The first focus of the study is on integration of maximum information obtained from information sources with various aspects of heterogeneity. Algorithms are proposed for integrating relations with the outerjoin operators, preserving the maximum information that meets the given query condition. Then the procedures of query processing for integration of autonomous, heterogeneous information sources are described in detail. Some techniques inherent in handling semistructured data

are proposed for coping with the autonomy of and the heterogeneity among information sources. Lastly, data acquisition frameworks are proposed for obtaining data from periodically generated data sequences, aiming at resolving temporal heterogeneity among real time information sources. The study proposed in this dissertation makes it possible to flexibly integrate a wide variety of information sources, making the most use of the existing assets of information sources.

In order to integrate heterogeneous information sources, *mediator architecture* [Wie92] is incorporated throughout the study in this dissertation. In the architecture, *mediators* are established between the application programs and the information sources. Mediation technologies needed for integrating heterogeneous information sources are studied and presented.

1.2 Contributions

To achieve the goal of integration of maximum information from autonomous, heterogeneous information sources, this dissertation proposes the following issues:

1. Integration of maximum information

   In response to a conjunctive query posed by the user to the mediator, it is important to gather information as much as possible from relevant information sources. The problem is that the information sources, in general, are incomplete in their contents and have semantic discrepancies. In order to prevent information loss due to incomplete contents, outerjoin operators are shown to be essential. It is also shown that predicates and foreign function are required to resolve semantic discrepancies. First of all, in order to obtain the maximum information, algorithms for creating a query plan that includes outerjoin operators are presented. Then, an algorithm for creat-

ing a query plan that includes predicate and foreign function operators, in addition to outerjoin operators, is described.

2. Query Processing based on a Self-Descriptive Object Model

   To deal with semistructured data, which have incomplete data structure, *self-descriptive object model* is a basic and important tool. In this data model, each data instance has the label that describes its own meaning. This data model is simple but flexible to absorb the structural discrepancies among various data models, even allowing each data instance structure to be different from one another. Being different from the flat structure of relational data model, the proposed self-descriptive object model requires query processing techniques not observed in relational database management systems. Several query processing techniques inherent in handling the data irregularities in the data model are described.

3. Data Acquisition from Periodically Generated Data Sequences

   In dealing with real time systems, *periodically generated data sequences* (**pgds**'s) play an important role. For the purpose of determining the optimum data combination that approximates a snapshot of the data generated at a point of time, an evaluation function is proposed based on novel criteria: *freshness* and *synchronousness* of the data combination. Some properties on synchronousness are studied. Then, query processing frameworks are presented based on the evaluation function. Finally, as a concrete example, it is shown that the proposed framework can be applied to a road obstacle and accident detection system with multiple sensors.

1.3 Organization

The remainder of this dissertation is organized as follows. In Chapter 2, the background in integration of heterogeneous information sources is surveyed. Chapter 3 presents algorithms for creating query plans for obtaining the maximum information sources, using outerjoin, predicate, and foreign function operators. Chapter 4 describes query processing techniques for integration of semistructured data based on a self-descriptive object model. Chapter 5 proposes novel criteria for choosing the optimum data combinations from periodically generated data sequences, and query processing frameworks are described. Lastly, Chapter 6 presents final discussion and concludes the dissertation.

Chapter 2

Background

In this chapter, the background of the research on integration of heterogeneous information sources is surveyed.

2.1 Research History

The research on integration of heterogeneous information sources [Wie92, PGMU96, Wie96a, Munse, Mun96b, Mun99] has made progress as an extension to the research on integration of databases. At the first stage, the focus of the research was on integrating homogeneous, distributed databases, so that the user can query and manipulate the data as if handling a single, centralized database. Such a tight integration of databases is based on the premise that all the distributed databases have common interface and functionalities, such as two phase commit. Such a premise holds for the organizations that build a sole, standardized information system. This, however, is not always possible in the real world, since a single organization may have a lot of different kinds of information systems developed at different times by different vendors, using different technologies. Moreover, replacement of all the systems at a time requires huge budget and high risk.

Thus the focus of the research has moved to loose integration of existing databases. Such systems are called multi-database or federated database management systems, depending on the system structure. Comprehensive surveys on such systems are available [SL90, TTC+90, LMR90, BLN86, Kamse]. The databases dealt with in such systems are relational databases. At the beginning of 1990's, object oriented databases emerged [OOse, Makse], achieving strong data modeling capabilities. Consequently, research was conducted to integrate object oriented databases and relational databases [SL90]. Also some commercial products that integrate both of the databases were developed [KCGS93]

In reality, information sources are not limited to relational databases and object oriented databases, but there are numerous kinds of other sources: proprietary databases, flat files, various information servers, such as the World Wide Web servers, simulators, and sensors, just to name a few.

Under such circumstances, the notion of *mediator architecture* [Wie92] was proposed by Gio Wiederhold in 1992, aiming at extraction of useful knowledge by abstraction and at integration of numerous information sources of various kinds. Based on this architecture, the Tsimmis project [CGMH+94, PGMW95, PGGMU95, PGMU96, PGM96] at Stanford University started. The goal of this project is integration of information sources with structural and semantic discrepancies. The Tsimmis project and prevalence of the World Wide Web initiated the study on semistructured data [Abi97, Bun97]. This has been accelerated by the spread of XML (eXtensible Markup Language), which is now considered a promising framework for exchange of information.

In the remainder of this chapter, the taxonomy of heterogeneity and autonomy, research issues in integration of information sources, and the concepts of the mediator architecture are described [Mun96a].

2.2 Taxonomy of Heterogeneity and Autonomy

Information sources developed and managed independently are inherently distributed, heterogeneous, and autonomous. Survey by Sheth and Larson [SL90] on integration of databases is expanded here with a focus on heterogeneity and autonomy of general information sources.

2.2.1 Heterogeneity

Heterogeneity among information sources can be classified into structural and functional heterogeneity, semantic heterogeneity, and temporal/spatial heterogeneity. Each of them is described below.

Structural and Functional Heterogeneity

This type of heterogeneity stems from the following:

1. Difference in structure

   Data models are different depending on information sources. For example, the relational data model and the object oriented data model yield heterogeneity. Even among homogeneous data models, data structure may differ: the entity represented in one information source as "name" can be expressed as a pair of "first name" and "last name" in another source. Also in a relational database, records of both managers and employees may be contained in a single relation, being distinguished by the value of "manager" or "employee" in an attribute "title", whereas in another database, managers and employees may be contained in separate relations whose names are "manager" and "employee."

2. Differences in constraints

Different data models, in general, yield different constraints of data. For example, relational DBMS supports referential integrity constraints, whereas object oriented DBMS does not.

3. Differences in data manipulation interface

   There are a number of data manipulation interfaces for defining/updating/deleting schema, querying, inserting/updating/deleting data, and defining access control:

   (a) Function call

   Common interfaces to information sources are provided in the form of function calls in some programming languages, such as C, C++, Java, and COBOL.

   (b) Embedded data manipulation language

   For instance, relational DBMS has a standardized data manipulation language, SQL. This language can be embedded in common programming languages, such as C and C++. In a common approach, a special purpose preprocessor interprets the embedded parts of the source program first, converting them to function calls.

   (c) Network communication protocol

   Some information sources offer the interface only as a network communication protocol, such as HTTP (Hypertext Transfer Protocol) of the World Wide Web server.

   (d) Remote method invocation

   As frameworks for network programming, remote method invocation frameworks, such as Remote Procedure Call and CORBA have been developed and used. After defining a Data Definition Interface, client

programs can invoke functions at remote sites as if calling local functions.

(e) Mobile agents

There are *mobile agents* such as Aglet [TK99] and Concordia [WPW+97]. When the client program calls a special function, a mobile agent, which is actually a bunch of source codes of a program with some data, is sent to remote sites and the program is executed. After the agent travels across some of the remote sites, it disappears or returns to the original client program with the resultant data of the program executions. This approach has an advantage that the network connection between the original site and the remote sites can be disconnected while the agent is traveling.

4. Schema/data structure evolution

Because of the autonomy of information sources, schema and data structure may change as time passes by. What is worse, the administrator who manages the integrated view may not be notified of the fact of the change itself. It is desirable that the integration system can automatically deal with such situations with as little efforts of the administrator as possible.

Semantic Heterogeneity

Semantic heterogeneity stems from disagreement of the meaning, interpretation, or intended use of the same or related data. This kind of heterogeneity can occur even among homogeneous information sources. Examples of semantic heterogeneity are listed below:

1. Vocabulary

Interpretation of vocabulary depends on the system designers and the users. For instance, the term "name" may indicate both a person name and a product name. Also "family name" and "last name" may indicate the same entity. One of the approaches to resolving such semantic heterogeneity is development of *ontology*. Ontology consists of a *conceptualization* of the entities and their relationship in the domain of interest and of the *terminology* to express the conceptualization. Efforts have been made to establish ontologies [Nis94, GTW92, Gru93].

2. Data type

   The same entity can be represented by different data types: an age can be expressed either as a string or an integer value.

3. Data unit

   Length can be expressed by the units of centimeter and inch. Also grades can be expressed by a set of values {A, B, C, D, E} in one system, and by integer values 0 through 10 in another.

Temporal/Spatial Heterogeneity

This type of heterogeneity includes the following:

1. Periodic versus aperiodic data

   In some information sources, data are generated at exactly the same intervals along either temporal or spatial axis; in other sources, data are generated arbitrarily at any point. Thus data generation points, in general, do not meet.

2. Data generation intervals

Even if each information source generates data at constant periodic intervals, the intervals may differ for multiple information sources. Thus it is not possible to acquire and compare the values generated at the same point of time or space. Also, even if multiple systems generate data at the same constant periodic intervals, their data generation points may have lags, and the data are never generated at the same point.

3. Difference of scale

   The scales of data along temporal or spatial axis sometimes differ. For example, two sheets of maps that represent neighboring areas may not match exactly, partly due to the approximation of mapping the area on the sphere onto a flat surface and partly due to the low accuracy of the map itself: in order to create a wide area map by putting together small area maps, the heterogeneity of scale should be resolved. Another example is a comparison of 3 dimensional motions. In order to compare a golf swing of an amateur player with the one of a professional player, the adjustment of time scale is necessary taking into account the speed change of the swing.

2.2.2 Autonomy

Information sources managed by different organizational entities are autonomous, meaning that they are locally controlled regardless of the data integration modules. The autonomy can be classified into the following:

1. Design autonomy

   This refers to the ability of each information system to choose its own design with respect to any matter, including:

   (a) The data being managed,

(b) The representation (data model, query language) and the naming of the data elements,

(c) The conceptualization or semantic interpretation of the data (which greatly contributes to the problem of semantic heterogeneity),

(d) Constraints (e.g., semantic integrity constraints and the serializability criteria) used to manage the data,

(e) The functionality of the system (i.e., the operations supported by the system), and

(f) The implementation (e.g., record and file structures and concurrency control algorithms).

2. Communication autonomy

   This refers to the ability of each information system to decide whether or not to communicate with other components.

3. Execution autonomy

   This refers to the ability to execute local operations without interference from external operations, and decide the order in which to execute external operations.

4. Association autonomy

   This refers to the ability to decide whether and how much to share its functionality (i.e., the operations it supports) and resources (i.e., the data it manages) with others.

In addition to the above mentioned autonomy and heterogeneity, there are those classified in terms of transaction processing; but these are beyond the scope of this dissertation.

2.3 Research Issues

In this section, research issues regarding the integration of information sources are classified and discussed, and some of the past and ongoing researches are described.

Selection of Information Sources

In case it is not known exactly which information source contains what sort of data, it is important to select the appropriate information sources to search for the information. In particular, it is a major issue when the number of information sources is large, as in the case of World Wide Web. The criteria for selection of information sources are listed below:

1. The selected information source is likely to have the expected information.

   First of all, it is essential that the information source is likely to have the requested information. GlOSS (Glossary Of Servers Server) project [GGMT94] focuses on text information sources. Each information source keeps statistical information on the text strings. Given a query, the GlOSS server refers to the statistical information to compute the *ranks* of the information sources, and selects the information sources with high ranks to actually issue the query.

2. The information source is active.

   The system as a whole should cope with the situation where the information source is out of order, stopped for maintenance, or deleted,

3. Query results are fast returned.

   The faster, the better. This is enabled by query processing optimization described later.

4. Cost is inexpensive.

   Some commercial databases require fee for data access. Also data communication through the network may generate fee. It is desirable to choose the information sources that minimize the cost.

Schema Integration

It is necessary to resolve the discrepancies of data models and schemas. This issue can be classified as follows. Survey is also available for schematic heterogeneity focusing on relational databases and object oriented databases [KCGS93].

1. Discrepancies of data models

   For relational DBMS, the logical data storage structure is in the form of tables, and for object oriented DBMS, in the form of classes with inheritance and their aggregations. Like this, information sources are based on a particular data model, and the differences of data models should be resolved for the integration of information sources.

2. Discrepancies of schemas

   As stated in Section 2.2.1, discrepancies of schema can occur even among homogeneous information sources. The resolution of such discrepancies is an important research issue. Recently *semistructured data* has been studied intensively [Abi97, Bun97]. There seems to be no strict definition in academia of the term, semistructured data, but the term is usually used to indicate:

   (a) A set of data with irregular structures which have no explicit schema;

   (b) A set of data not stored in a database, and no explicit schema is defined for the data in advance;

(c) A set of data with a *loose* schema such that each data instance in the set does not strictly conform to the schema.

Query processing techniques for resolving schematic discrepancies are proposed in Chapter 4 of this dissertation.

Resolution of Interface Discrepancy

A common approach to resolving the data manipulation interface discrepancy described in Section 2.2.1 is to establish a software module, called a *wrapper*, for each of the information sources [PGMW95]. The wrapper is responsible for converting the query issued by the mediator into the native interface to the information source, and converting the query result obtained from the information source into the common data structure to send back to the mediator.

Query Processing

A lot of research efforts have been made on query processing and its optimization, traditionally for relational DBMS's [Ull89]. Many of the techniques developed so far can be used for integration of heterogeneous information sources. There are issues, however, that are inherent in the heterogeneity of information sources. In particular, optimization of query processing is hard, since query processing capabilities of information sources differ drastically from one another. Some of the research issues are listed below:

1. Finding and selecting information sources

   Finding and selecting information sources is an important research issue, particularly in the environment where information sources participate in or retreat from the information integration system arbitrarily.

2. Identifying query processing capabilities of information sources

   It is necessary for the integration module to understand the query processing capabilities of each information source. Some information sources have very naive capabilities for query processing; for example, to fetch the contents only by its identifier. Others may have sophisticated capabilities, as seen in SQL3. There is an approach to having the wrapper offer the descriptions of its capabilities to the mediator when requested [VP97].

3. Data conversion

   In order to resolve the structural and semantic discrepancies among the information sources, it is necessary to convert the native data into desirable form.

4. Data filtering

   Since some of the information sources have poor query capabilities, it is often necessary to filter out the unnecessary data at the wrapper or the mediator. Generally, better performance is gained when unnecessary data are eliminated at an early stage of the query processing. This strategy is known well as "to push selection as far down to the source as possible."

5. Outerjoin

   Due to the discrepancies and incompleteness of the data contents, some entities in one information source may not match entities in another information source. In such a case, the natural join eliminates all the data entities. Instead, the *outerjoin* [Dat83] should be used if we are to obtain the maximum information [LW94]. The outerjoin retains the partial information that would be eliminated by the inner join. By the outerjoin of two relations, tuples from one relation that do not match tuples from the second

relation are added to the result of the inner join of the two relations; for the unmatched tuples, the vacant attributes are padded with nulls. The query optimization in the presence of outerjoin operators is hard, since associativity does not hold for the outerjoin, being different from the inner join. This issue is studied in detail in Chapter 3 of this dissertation.

6. Data acquisition

   In real time systems, data are generated either periodically or randomly. In either case, it is important to obtain an approximation of the snapshot of the data generated at a point of time. Selection and acquisition of the optimum data combination that best approximates the snapshot is a research issue. The problem for acquiring data combinations from periodically generated data sequences is addressed in Chapter 5 in this dissertation.

Access Control

Access control of the data is essential in dealing with important information. When native information sources do not have common capabilities with regard to access control, the data integration module is responsible for this. There is research for enabling the mediator to have access control capabilities in the field of medical record management [Wie96b].

Management of Constraints

As described in Section 2.2.1, constraints on data differ from one information source to another. For a simple example, in relational DBMS's, any tuple in the table with a key is guaranteed to be identified by the key. This, however, does not hold for some information sources: multiple data entities may exist for the same identifier, making the query processing difficult. Also imposing and

controlling constraints on data scattered across multiple information sources is a hard problem, especially when local administrators and users can change the data autonomously. What is worse, there are cases where multiple information sources have the data of the same entity, but data in one information source and in another have discrepancies. Some facilities to mediate such cases should be offered by the data integration module.

Transaction Processing

Databases, by definition, have some level of transaction processing capabilities, called ACID (Atomicity, Consistency, Isolation, and Durability); but this does not always hold for information sources. Thus in general, it is impossible to achieve strict serializability over all the information sources. Achieving some form of weak serializability, depending on the transaction capabilities of the information sources, is a research issue.

Graphical User Interface

The query results obtained from heterogeneous information sources consist of different types of data. If such data are to be displayed to human users, flexible GUI should be provided.

Administration Support

It is not possible to resolve all kinds of heterogeneity and to manage all the constraints on data in information sources automatically. Consequently, the system administrator plays an important role in management of the data integration system. It is necessary to provide the system administrator with powerful support tools for practical deployment of the data integration system.

Ad-hoc versus Materialized Approach

Approaches to offering the integrated view of information sources can be classified into two groups: to create a view in an ad-hoc manner whenever a query is issued by the user; and to store (or *materialize*) the necessary information in a data warehouse and maintain it constantly. The ad-hoc approach thus saves the efforts for maintaining the data warehouse, and is more suitable to cope with the local autonomy of the information sources. Throughout this dissertation, the former approach of ad-hoc query processing is taken.

2.4 Mediator Architecture

The role of the mediator is to mediate the heterogeneity of the information sources and to provide the application program with the useful knowledge extracted from the huge amount of information [Wie92]. Databases are meant to be used for various purposes by a number of users, whereas application programs are meant to serve for a specific purpose in a domain. The mediator is aimed at bridging the gap between the universal purpose of databases and the specific purpose of applications.

An example of the mediator architecture proposed by the Tsimmis project at Stanford University is shown in Figure 2.1. In this architecture, a wrapper is established for each information source in order to conceal the interface inherent in each information source from the mediators. The mediators and the wrappers have the same interface to their clients. Thus it is not necessary to distinguish between the mediator and the wrapper. This enables flexible construction of a hierarchy of the mediators. For example, in Figure 2.1, Mediator 1 obtains information from both Mediator 2 and Wrapper 1. The hierarchical structure of the mediators enables the local system administrators to focus on the local

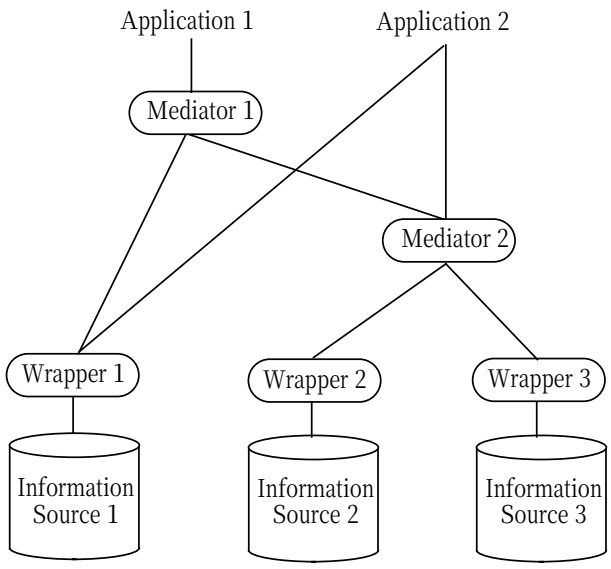


Figure 2.1. The Tsimmis mediator architecture.

information of the domain, keeping the amount of information in a manageable size. This is particularly important when the whole system is so large that the information to deal with is beyond the manageable size for a single group of local system administrators. The hierarchical structure of the mediators also achieves modularity of domains, enhancing reuse of the mediators by different applications.

The mediator architecture is assumed as the framework of the integration system described in this dissertation.

Chapter 3

Integration of Maximum Information

3.1 Introduction

To make use of the abundant information available via a computer network, such as the Internet, it is desirable to offer integrated views of information sources. The goal of this chapter is to create optimized query processing plans for retrieving the maximum information from multiple relations. The algorithms presented in this chapter can be used for the query processing described in Chapter 4.

Given a set of relations, the *maximum information* can be defined as the *full disjunction* [GL94] of the relations. The full disjunction contains every piece of *connected* tuples of the base relations and has no redundancy. In order to obtain the maximum information, *outerjoin* [Dat83, Dav92] operators are essential. Suppose that one relation contains attributes for person names and their telephone numbers, and another relation has tuples of a subset of the person names in the first relation and their addresses. We do not want to drop any data in

the first relation, but want to include the address data from the second relation if available. In such a case, we need to merge the two relations with a *left* (or *one-sided*) *outerjoin* , preserving all the tuples of the first relation. By the left outerjoin, tuples in one relation that do not match tuples in the second relation are added to the result of the inner join of the two relations; for the unmatched tuples, the attributes for the second relation are padded with null values.

Another kind of operators that are useful for integrating relations is *predicates*. Suppose that in a relation, person's full names are contained in an attribute, and in another relation, person's first and last names are contained in separate attributes. In order to join the two relations on person's names, we can define a predicate that takes three variables: the first name, the last name, and the full name. If the concatenation of the first name and last name matches the full name, the predicate returns true; otherwise it returns false. Using this predicate, we can join the two relations. In this way, predicates can be used to mediate discrepancies among relations.

We can also use foreign functions to mediate discrepancies among relations. Suppose we have a foreign function that takes a first name and a last name as input and concatenates them to produce a full name as output. After converting the first and last names in the relation of the above example with the foreign function into a full name, we can join the two relations with the natural join. Foreign functions can also be used for creating arbitrary views. For instance, if the character strings in an attribute of a relation are all in upper case letters, we can create a foreign function to convert the letters except for the first ones into lower case letters. Thus we can convert the native data into another format, making the strings easier to read.

This chapter describes algorithms for creating an optimized query plan for retrieving the maximum information using outerjoins, in response to a conjunctive

query. First, algorithms are presented for creating a query processing plan for merging relations with the natural join. Then, an algorithm is proposed for creating a query processing plan in the presence of predicates and foreign functions. A query plan of relational algebra is optimized traditionally by reordering operators in a query expression tree. A typical strategy is to "push selection as far down the expression tree as possible" [Ull89]. Such a reordering of operators partly depends on *associative identity* [Ull89] of the inner join. The outerjoin, however, is not associative, and imposes restrictions on free reordering of operators. Foreign functions also impose restrictions: suppose that first names and last names are contained in two different relations. For a foreign function to merge them into full names, the two relations should have been already joined with a join operator. Thus foreign functions impose *dependencies* among operators. Consequently, the query processing that involves outerjoins, predicates, and foreign functions is significantly more complex than the ones without them, and has not been presented in the literature.

The rest of this chapter is organized as follows. In Section 3.2, first of all, motivating examples are described for the integration of the maximum information with outerjoins, but without predicates and foreign functions. Next, a naive algorithm, then optimized one, are presented for the query processing that generates the maximum information. In Section 3.3, a more complex case is discussed where predicates and foreign functions are involved. First, motivating examples are shown. After defining basic notions, a rigorous algorithm is presented for creating a query processing plan in detail. Section 3.4 discusses related works. Section 3.5 summarizes the chapter.

3.2 Integration Using Outerjoins

3.2.1 Motivating Examples

As an example, let us examine the following Datalog [Ull89] rule:

$$view(R, N, G, F, RM, B) \text{ :-}$$
$$WHOIS(R, N) \ \&\ CS(N, G, F, RM) \ \&\ GUIDE(G, B). \qquad (3.1)$$

The term preceding the symbol ":-" is called the head of a rule and the terms following the symbol the tail of the rule. In this rule, each term (also called a subgoal) of the tail connected with symbol "&" is an extensional database predicate. The capital letters enclosed by parentheses are variables. An extensional database predicate has a corresponding relation: for example, corresponding to $WHOIS$, there is an actual relation whose schema is (R, N). In Rule (3.1), R stands for relation, N for name, G for group name, F for floor, RM for room number, and B for building. Table 3.1 shows example relations for each of the extensional databases that appear in the tail of the rule.

In this dissertation, we deal with only conjunctive queries that consist of the terms of conditions connected with *and*'s. Each term of the query conditions is restricted to an equation that holds true if the attribute value in the extensional database predicates matches exactly with the given constant value. Thus the query is of the form

$$answer \ \text{:-} \ view\ (A_1,\ A_2, \ldots, A_n) \ \&\ A_k = c_k \ \&\ A_l = c_l \ \&\ \ldots \ \&\ A_m = c_m, \qquad (3.2)$$

where A_i $(1 \leq i \leq n)$ is an attribute variable and c_j $(j = k, l, \ldots, m$ and $1 \leq j \leq n)$ is a constant value. In this section we assume the *view* relation is created by merging the extensional database predicates with the natural join.

Table 3.1. Example relations of extensional database predicates.

(a) *WHOIS(R, N)*

R	*N*
"faculty"	*"Gio Wiederhold"*
"faculty"	*"Arthur Keller"*

(b) *CS(N, G, F, RM)*

N	*G*	*F*	*RM*
"Gio Wiederhold"	*"database"*	4	415
"Ben Kao"	*"database"*	4	413
"Arthur Keller"	*"logic"*	4	425

(c) *GUIDE(G, B)*

G	*B*
"database"	*"MJH"*

Table 3.2. The inner join result of relations $WHOIS$, CS, and $GUIDE$.

R	*N*	*G*	*F*	*RM*	*B*
"faculty"	*"Gio Wiederhold"*	*"database"*	4	415	*"MJH"*

According to the normal interpretation of the Datalog, for a given rule as in Rule (3.1), the view is constructed by joining each term in the tail with the inner join. The *view* relation expressed in Rule (3.1) for the relations in Table 3.1 is shown in Table 3.2. This relation consists of a single tuple: any tuple of the relations in the tail is discarded at the inner join if the join attributes of the relations do not match exactly.

The goal of the study in this chapter is to obtain from the information sources

Table 3.3. The full disjunction of relations $WHOIS$, CS, and $GUIDE$.

R	N	G	F	RM	B
"faculty"	*"Gio Wiederhold"*	*"database"*	4	415	*"MJH"*
"faculty"	*"Arthur Keller"*	*"logic",*	4	425	-
-	*"Ben Kao"*	*"database"*	4	413	*"MJH"*

the maximum information that meets the conjunctive query condition. Given a set of relations, the exact meaning of the "maximum information" can be formally defined as the *full disjunction* proposed in [GL94]. The full disjunction of relations is conceptually the maximum information without redundancy, and can be computed in the next algorithm [RU96].

Algorithm 3.1

1. Take the union of all the join-consistent tuples of subsets of the relations in addition to the tuples that are not join-consistent with any other tuples. We say tuples are join-consistent when the values agree on all the common attributes for any pair of the tuples that come from distinct relations.

2. Pad vacant attributes with null values.

3. Delete all the tuples that are *subsumed* [Ull89] by any other tuple.

   We say tuple t subsumes tuple u if t and u agree in every attribute where u is not null. That is, t is obtained from u by replacing zero or more null values with concrete values. □

Example 3.1

Table 3.3 shows the full disjunction of the relations in Table 3.1. In order to calculate the full disjunction, we gather tuples obtained by merging all the combi-

nations of the relations with the inner join: $WHOIS$, CS, $GUIDE$, $WHOIS \bowtie CS$, $CS \bowtie GUIDE$, $GUIDE \bowtie WHOIS$, and $WHOIS \bowtie CS \bowtie GUIDE$. Then, take the union of the tuples thus gathered. At this time, the vacant attributes are padded with null values: for example, when we join $WHOIS$ and CS, all the values in attribute B are padded with null values. Thus we obtain tuples in a relation of schema (R, N, G, F, RM, B). In the step of tuple subsumption, for example, the tuple (*"faculty"*, *"Gio Wiederhold"*, -, -, -, -) obtained from $WHOIS$ is subsumed by the tuple (*"faculty"*, *"Gio Wiederhold"*, *"database"*, 4, 415, -) obtained from $WHOIS \bowtie CS$. □

As proved in [RU96], the full disjunction is unique given a set of relations. Thus when a query is issued, we can find the maximum information by selecting from the full disjunction such tuples that meet the query condition.

Concrete examples in the balance of this section illustrate that the full disjunction is appropriate as the definition of the maximum information of multiple relations. It is also shown that different query processing plans should be created, depending on the conditions imposed by the query.

Example 3.2

Suppose we retrieve tuples where $R =$ '*faculty*' and $F = 4$ based on the head of Rule (3.1). Following the normal interpretation of Datalog, this query can be expressed with SQL as

$$
\begin{aligned}
&\mathit{select}\ WHOIS.R,\ WHOIS.N,\ CS.G,\ CS.F \\
&\quad CS.RM,\ GUIDE.B \\
&\quad \mathit{from}\ WHOIS\ \mathit{join}\ CS\ \mathit{on}\ WHOIS.N = CS.N \\
&\quad \mathit{join}\ GUIDE\ \mathit{on}\ CS.G = GUIDE.G \\
&\quad \mathit{where}\ WHOIS.R = \text{'}faculty\text{'}\ \mathit{and}\ CS.F = 4.
\end{aligned} \tag{3.3}
$$

Table 3.4. Result of SQL (3.4).

R	N	G	F	RM	B
"faculty"	*"Gio Wiederhold"*	*"database"*	4	415	*"MJH"*
"faculty"	*"Arthur Keller"*	*"logic"*	4	425	-

The result of this query is identical to Table 3.2. This result, however, does not include the maximum information that meets the condition that $R =$ '$faculty$' and $F = 4$: relations $WHOIS$ and CS both have tuples for Arthur Keller and they meet the query condition if jointed, but the information on Arthur Keller is not included in Table 3.2. The reason why this information is missing in Table 3.2 is that the joined tuples of $WHOIS$ and CS for Arthur Keller dropped in the joining process with $GUIDE$. □

In order to avoid such loss of information, we need the outerjoin.

Example 3.3

SQL-92 enables the use of the outerjoin, and we can describe the appropriate query for finding the maximum information as

$$
\begin{aligned}
&select\ WHOIS.R,\ WHOIS.N,\ CS.G,\ CS.F \\
&\quad CS.RM,\ GUIDE.B \\
&\quad from\ WHOIS\ join\ CS\ on\ WHOIS.N = CS.N \\
&\quad left\ outer\ join\ GUIDE\ on\ CS.G = GUIDE.G \\
&\quad where\ WHOIS.R = \text{'}faculty\text{'}\ and\ CS.F = 4.
\end{aligned}
\tag{3.4}
$$

This query merges $WHOIS$ and CS first with the inner join, generating tuples for Arthur Keller as well as Gio Wiederhold. The result of the join meets the imposed condition that $R =$ '$faculty$' and $F = 4$. Next, these tuples are merged

Table 3.5. Result of SQL (3.5).

R	N	G	F	RM	B
"faculty"	*"Gio Wiederhold"*	*"database"*	4	415	*"MJH"*
-	*"Ben Kao"*	*"database"*	4	413	*"MJH"*

with $GUIDE$ with the left outerjoin, preserving all the resultant tuples of the inner join. In this process, the tuple for Gio Wiederhold is joined with the tuple in GUIDE, but the tuple for Arthur Keller is not: there is no value in $GUIDE.G$ whose value is "*logic*." As a result, we obtain the relation shown in Table 3.4. Note that the value of attribute B for Arthur Keller is null. This query result is identical to the one obtained by selecting from the full disjunction in Table 3.3 the tuples that meet the query condition. □

Example 3.4

We consider a different query with the condition that $F = 4$ and $B =$ 'MJH'. The appropriate SQL is

$$\begin{aligned} &select\ WHOIS.R,\ WHOIS.N,\ CS.G,\ CS.F \\ &\quad CS.RM,\ GUIDE.B \\ &\quad from\ CS\ join\ GUIDE\ on\ CS.G = GUIDE.G \\ &\quad left\ outer\ join\ WHOIS\ on\ CS.N = WHOIS.N \\ &\quad where\ CS.F = 4\ and\ GUIDE.B = \text{'}MJH\text{'}. \end{aligned} \tag{3.5}$$

This query merges CS and $GUIDE$ with the inner join, generating tuples for Gio Wiederhold and Ben Kao. The result of the join meets the imposed condition that $F = 4$ and $B =$ 'MJH'. Next, in order to add additional information of relation $WHOIS$, the result is merged with $WHOIS$ with the left outerjoin,

preserving all the resultant tuples of the inner join. In this process, the tuple for Gio Wiederhold is joined with the tuple in $WHOIS$, but the tuple for Ben Kao is not: there is no value in $WHOIS.N$ whose value is Ben Kao. As a result, we obtain the relation shown in Table 3.5. Note again that the value of attribute R for Ben Kao is null. This query result is identical to the one obtained by selecting from the full disjunction in Table 3.3 the tuples that meet the query condition.

□

As suggested by the above examples, we modify the normal interpretation of the Datalog, as in Rule (3.1): in our interpretation, the view of a set of relations indicates the maximum information, or full disjunction. In order to retrieve the maximum information from multiple relations, the outerjoin is required. Note that we have to select the appropriate join methods (i.e., inner join, left outerjoin, or full outerjoin) depending on the query. In Query (3.3), the conjunction of the query condition is imposed on attribute R of relation $WHOIS$ and attribute F of relation CS. In this case, relations $WHOIS$ and CS should be merged with the inner join in order to meet the conjunction of the conditions that $WHOIS.R$ is *'faculty'* and $CS.F$ is 4. By merging $GUIDE$ with the left outerjoin into the inner join result, we can add additional information to the tuple of Gio Wiederhold. On the other hand in Query (3.4), the conjunction of the query conditions is imposed on attribute F of relation CS and attribute B of relation $GUIDE$. In this case, relations CS and $GUIDE$ should be merged with the inner join, then, $WHOIS$ should be merged with the left outerjoin in order to add additional information to the tuple of Gio Wiederhold.

As our examples indicate, in order to retrieve the maximum information from multiple relations using SQL, (i) we should have the knowledge of the schema of each relation, and (ii) we should specify explicitly the appropriate join method

for each join operation. It is desirable, however, that we have knowledge of only the view expressed by the head of the Datalog rule for posing conjunctive queries: once a conjunctive query is cast, the database management system should automatically create a proper query processing plan with the appropriate join methods for merging the relations.

3.2.2 Algorithms

A Naive Algorithm

First, a naive algorithm is presented for the query processing of conjunctive queries against Datalog rules under our interpretation of obtaining the maximum information. Theoretically we can find the maximum information that meets a conjunctive query condition by the following algorithm.

Algorithm 3.2

1. Find the full disjunction for all the relations according to Algorithm 3.1.
2. Select only such tuples from the full disjunction that meet the query condition. □

Obviously this algorithm is too expensive to be desirable. We can retrieve more efficiently the maximum information for conjunctive queries by the algorithm described in the rest of this section.

An Optimized Algorithm

In order to explain the algorithm, we make use of the notion of *hypergraphs* that appear in [Ull89]. Figure 3.1 shows the hypergraph representation of three relations X, Y, and Z: relation X has two attributes A and B; relation Y has two attributes B and C; and relation Z has three attributes A, B, and C. Each

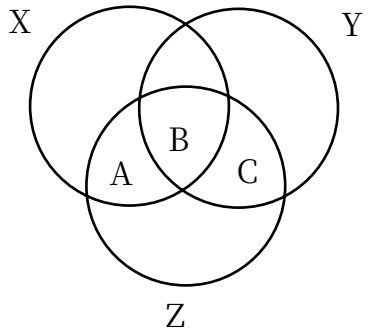


Figure 3.1. A γ-3-cycle.

of the attributes is called a *hypernode*. The circles that enclose the hypernodes are called *hyperedges* and represent relations: for example, the circle that encloses hypernodes A, B, and C represents relation Z. Hyperedge Z can be expressed using the hypernodes as ABC. From now on, we sometimes refer to a hypernode just as a node.

In this dissertation, we deal with only a *connected* hypergraph. Let us define the exact meaning of "connected":

1. A hypergraph is *disconnected* if we can partition its hyperedges into two non-empty sets such that no node appears in members of both sets; otherwise the hypergraph is *connected*.

2. Hyperedges and hypernodes are connected if all the hyperedges and hypernodes are contained in a connected hypergraph.

Several degrees of acyclicity for hypergraphs are defined in [Fag83]. Among them is γ*–acyclicity*. If a hyperedge is γ –acyclic, it is known that we can calculate the full disjunction easily as described in detail later in this section. In order to define γ–acyclicity, we define a *pure cycle* and a γ*–3–cycle* first.

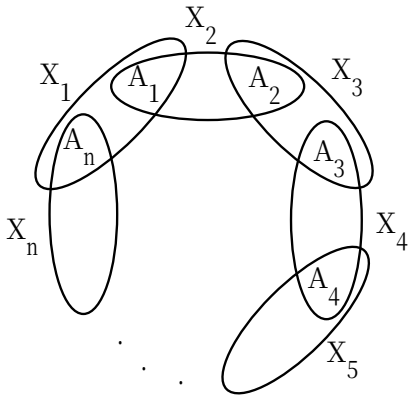


Figure 3.2. A pure cycle.

Definition 3.1 (Pure cycle)

A pure cycle is a collection of $n \geq 3$ hyperedges and nodes suggested by Figure 3.2. Each pair of hyperedges X_i and X_{i+1} (and also X_n and X_1) have at least one node A_i (or A_n in the case of X_n and X_1) in common. Moreover, none of the shared nodes A_j appears in more than the two hyperedges of the pure cycle suggested by Figure 3.2. These nodes may appear in hyperedges that are not part of the pure cycle, however. There may be more than one node in the intersections shown, but those nodes must not appear in any other hyperedges of the pure cycle. □

Definition 3.2 (γ–3–cycle)

A γ–3–cycle is a set of three hyperedges in the configuration of Figure 3.1. That is, the nodes A, B, and C must exist; any other regions of the diagram may or may not be empty, and the regions containing A, B, and C may also contain other nodes. Put another way, there must be at least one node in all three hyperedges, at least one node that is in only X and Z, and at least one node that is in only Y and Z. □

For example, the smallest γ–3–cycle consists of the three hyperedges $\{ABC$, AB, $BC\}$.

Using the definitions of the pure cycle and the γ–3–cycle, a *γ–cyclic hypergraph* is defined as follows.

Definition 3.3 (γ–cyclic hypergraph 1)
A hypergraph is γ–cyclic if it contains either a pure cycle or a γ–3–cycle. □

This definition of γ–cycle is proved in [Fag83] to be equivalent to the next definition.

Definition 3.4 (γ–cyclic hypergraph 2)
A hypergraph is γ–cyclic if it has a pair E, F of incomparable, nondisjoint hyperedges such that in the hypergraph that results by removing $E \cap F$ from every hyperedge, what is left of E is connected to what is left of F. E and F are said to be incomparable if $E \not\subseteq F$ and $F \not\subseteq E$. □

A γ–acyclic hypergraph is defined as follows.

Definition 3.5 (γ–acyclic hypergraph)
A hypergraph is *γ–acyclic* if it is not γ–cyclic. □

In [RU96], Algorithm SOJO is presented for creating a *sound outerjoin ordering* that guarantees to obtain the full disjunction if the hypergraph representation of the relations is γ–acyclic. A sound outerjoin ordering for a database schema $R = \{R_1, \ldots, R_n\}$ is defined to be an expression in which

1. Each relation appears exactly once,
2. The only operator is the full outerjoin, and
3. The result is the full disjunction of R.

Algorithm SOJO, however, does not take query conditions into account, and retrieves all the information available from the relations, using the full outerjoin. The rest of this chapter deals with only a set of connected, γ–acyclic relations.

As stated earlier, a set of relations can be expressed as hyperedges and the attributes of the relations as hypernodes. When a conjunctive query is issued to a set of relations, we say the hypernodes on which the query condition is imposed are *initially indispensable* in this section. In Rule (3.2), hypernodes A_k, $A_l, \ldots, A_m$ are initially indispensable nodes.

In what follows, we sometimes treat hyperedges as a set of hypernodes. Thus we use the normal set operators and notations. For sets X and Y, $X \cap Y$ indicates the set of elements that are in both X and Y; $X - Y$ indicates the set of every element that is in X but not in Y; $X = \phi$ indicates X is an empty set; set S is a subset of set T if every element in S is also in T; S is a proper subset of set T if S is a subset of T and $T - S \neq \phi$.

In the situation where a conjunctive query is issued to a set of relations, we have the following proposition.

Proposition 3.1

As illustrated in Figure 3.3, let H be a connected, γ–acyclic hypergraph representation of extensional database predicates. Let P and Q be any hyperedges in H, where Q is a subset of P. Let I be the set of all the initially indispensable nodes. Let S be $Q \cap I$ and let T be $(P \cap I) - S$. If both S and T are non-empty, then, removal of Q does not affect the conjunctive query result that represents the maximum information. □

Proof: We let U be $Q - S$. We prove the proposition holds true by dividing the possible situations into different cases as follows:

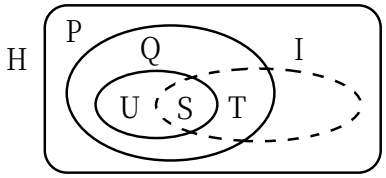


Figure 3.3. Configuration of hypergraph H in Proposition 3.1.

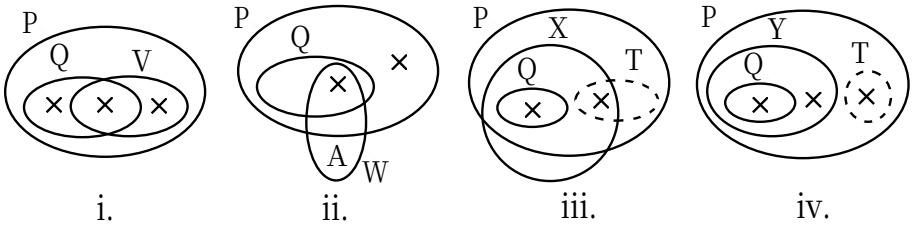


Figure 3.4. Configurations of the hypergraphs corresponding to cases i., ii., iii, and iv. in the proof of Proposition 3.1.

1. The case where $U = \phi$.

   In this case, the tuples in Q are subsumed by some tuples in P that meet the query condition, *or* the tuples in Q do not meet the query condition: remember we consider only equality for each term of the query condition in this dissertation. Hence removal of Q does not affect the query result.

2. The case where $U \neq \phi$.

   (a) The case where all the tuple in Q are subsumed by some tuples in P.

   In this case, also removal of Q does not affect the query result.

   (b) The case where some tuples in Q are *not* subsumed by any tuple in P.

   In this case, the tuples in Q that are subsumed by any tuple in P do not participate in the query result just as case (a). Thus we now

assume that no tuples in Q are subsumed by P. Let H' be the hypergraph that consists of all the hyperedges in H except for hyperedge P. We assume the hypernodes in H are also in H' unless the hypernodes are disconnected from Q after removal of P. For the tuples in Q to participate in the query result, Q should be connected to every node in T in hypergraph H'. Thus suppose in H', Q is connected to some hypernodes in T. We show this hypothesis causes a contradiction in any case listed below, proving that Q is not connected to any hypernodes in T and that removal of Q does not affect the query result. Refer to Figure 3.4 for the configurations of the hypergraphs for each case. In the figure, the existence of one or more hypernodes is indicated by symbol "$\times$."

i. Suppose there exists a hyperedge V such that $V \cap (P - Q) \neq \phi$, $Q \cap V \neq \phi$, and $Q - (Q \cap V) \neq \phi$.

   In this case, in H, hyperedges Q, P, and V constitute a γ–3–cycle, causing a contradiction to our premise.

ii. Suppose there exists a hyperedge W such that $Q \cap W \neq \phi$ and at least one hypernode A exists that is not contained in P. Suppose also in H', A is connected to some hypernode in T by a series of hyperedges excluding Q and W.

   In H, if we let P be E in Definition 3.4 and let W be F in Definition 3.4, then after removal of $E \cap F$, what is left of E is connected with what is left of F, indicating the original hypergraph is γ–cyclic. This is again a contradiction.

iii. In H, suppose that there exists a hyperedge X that includes all the hypernodes in Q and at least one hypernode in T.

In this case, we let H be H', X be P, and apply this proof recursively until there are no more hyperedges that include all the hypernodes in Q and at least one hypernode in T. The recursion of the proof terminates, since the number of the hyperedges that include all the hypernodes in Q and at least one hypernode in T is finite.

iv. In H, suppose that there exists a hyperedge Y such that Y is a proper subset of P, Q is a proper subset of Y, and $Y \cap I = \phi$.
In this case we let Y be Q and apply this proof recursively to prove that either the tuples in Y are subsumed by some other tuples or Y is not connected to T in H'. Thus connection with Y does not enable Q to be connected to T. The recursion of the proof terminates, since the number of the hyperedges that are proper subsets of P, contain Q as a proper subset, and do not include any initially indispensable node is finite. □

Now we are ready to show the precise algorithm.

Algorithm 3.3

[input]

- A Datalog rule whose tail consists of a conjunction of extensional database predicates that can be joined with the natural join and whose hypergraph representation is connected and γ–acyclic.
- A conjunctive query that imposes a conjunction of conditions on the variables that appear in the head of the Datalog rule.

[output]

- The query result consisting of the tuples in the full disjunction that meet the query condition.

[method]

1. Let the set of the relations (or hyperedges) that correspond to all the extensional database predicates be U. Let the set of all the initially indispensable hypernodes be I.

2. Let P and Q be any hyperedges in U, where Q is a subset of P. Also let S be $Q \cap I$ and let T be $(P \cap I) - S$. If $S \neq \phi$ and $T \neq \phi$, then, remove Q from U.

3. From any relation in the set of all the hyperedges that include any hypernode in I, remove the tuples that do not meet the query condition.

4. Apply Algorithm SOJO in [RU96] to relations (or hyperedges) U to yield the sound outerjoin ordering.

5. According to the sound outerjoin ordering, join the relations. In joining two relations, say X and Y, use the following join methods depending on the cases:

   (a) If $(X-Y) \cap I \neq \phi$ and $(Y-X) \cap I = \phi$, join the relations with the left outerjoin, preserving X. If $(Y - X) \cap I \neq \phi$ and if $(X - Y) \cap I = \phi$, join the relations with the left outerjoin, preserving Y; otherwise,

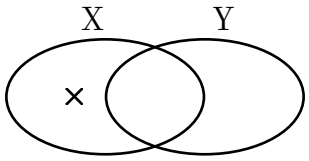
   (b) If $(X - Y) \cap I \neq \phi$ and $(Y - X) \cap I \neq \phi$, join the two relations with the inner join; otherwise,

   (c) Join the two relations with the full outerjoin.

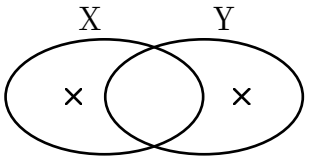
   After each join process, remove the tuples whose attribute values corresponding to the initially indispensable nodes are null, if any. □

In Step 2, removal of Q from U is justified by Proposition 3.1: the removal never affects the query result.

Refer to Figure 3.5 for the configurations of the relations X and Y in Step 5. In the figure, the existence of at least one initially indispensable node is indicated by symbol "×." In case (a) of Step 5, we dropped by the left outerjoin the tuples in one of the relations if the dropped tuples do not join with the other tuples; in case (b) of Step 5, we dropped by the inner join the tuples in both of the relations if the tuples do not join with each other; also at the end of Step 5 we removed the tuples whose attribute values corresponding to the initially indispensable nodes are null. These operations are safe since preservation of the dropped tuples would make some attributes in I be null, dissatisfying the query condition: remember that null values in any of the join attributes never allow its tuple to join with any tuple in other relations.



(a) Left outerjoin, preserving X.



(b) Inner join.

Figure 3.5. Join methods for Steps 5 (a) and 5 (b) of Proposition 3.1.

Example 3.5

Figure 3.6 shows an example of the hypergraph representation of relations. Suppose a conjunctive query condition $A = a$, $C = c$, and $D = d$ is imposed. In the figure, we use the convention of marking the initial indispensable hypernodes by "$*$." In this case, in Step 2 of Algorithm 3.3, hyperedge BC is removed. In Step 3, for each of relations AB, $BCDEF$, and $BCDEG$, only the tuples that meet the query condition are selected, and the other tuples are discarded. For example, in relation AB, only such tuples whose value of A is a are selected. In Step 4, Algorithm SOJO is applied to create a sound outerjoin ordering:

$$AB \underset{\leftrightarrow}{\bowtie} (BCDEF \underset{\leftrightarrow}{\bowtie} ((BCDEG \underset{\leftrightarrow}{\bowtie} GL) \underset{\leftrightarrow}{\bowtie} (GHI \underset{\leftrightarrow}{\bowtie} GHJK))).$$

In this expression, $X \underset{\leftrightarrow}{\bowtie} Y$ indicates the full outerjoin. This is one of the several possible orders created by SOJO. Then, according to this order, in Step 5, relations are joined with the appropriate join methods as:

$$A^*B \bowtie (BC^*D^*EF \underset{\leftrightarrow}{\bowtie} ((BC^*D^*EG \underset{\leftarrow}{\bowtie} GL) \underset{\rightarrow}{\bowtie} (GHI \underset{\leftrightarrow}{\bowtie} GHJK))).$$

In this expression, the attribute with a superscript "*" indicates the corresponding hypernode is an initially indispensable node. Also for hypernodes X and Y,

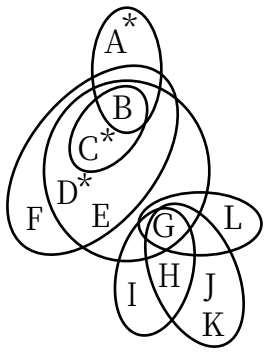


Figure 3.6. An example of a hypergraph.

$X \bowtie Y$ indicates the inner join, $X \underset{\leftarrow}{\bowtie} Y$ the left outerjoin preserving X, and $X \underset{\rightarrow}{\bowtie} Y$ the left outerjoin preserving Y. After each join process, remember to discard redundant tuples whose initially indispensable attributes include a null value. The relation thus obtained is the query result. □

3.3 Incorporation of Predicates and Foreign Functions

In the previous chapter, we dealt with extensional database predicates and showed algorithms that yield the tuples in the full disjunction that meet the given, conjunctive query condition. In this chapter we introduce predicates and foreign functions to realize more flexible integration of information sources.

3.3.1 Motivating Examples

From a practical view point, it is desirable that we can deal with intentional database predicates, in addition to extensional database predicates. As stated earlier, the extensional database predicate has a corresponding, actual relation. The intentional database predicate, in contrast, does not have a corresponding, actual relation, and is used to create a view relation by joining or selecting tuples from other relations. In this chapter, we assume all the intentional predicates are strong, meaning that the predicates evaluate to false if one or more input variables are null. An example of the Datalog rule that includes an intentional database predicate is

$$\begin{aligned} view(R, N, G, F, RM, B) \text{ :- } & WHOIS(R, FN, LN) \\ & \& \ CS(N, G, F, RM) \ \& \ GUIDE(G, B) \ \& \\ & examine\_name(LN, FN, N), \end{aligned} \tag{3.6}$$

where FN stands for a first name, LN for a last name, and N for a full name. The last term in the rule, *examine_name(LN, FN, N)* , is an intentional database predicate and returns true if the concatenation of FN and LN with a space in between is the same string as N; otherwise it returns false. In this chapter, predicates with upper case letters denote extensional database predicates and ones with lower case letters, intentional database predicates. From now on, we assume that relations are always joined using predicates, and not joined with the natural join.

It is also desirable if we can deal with functions that convert input variables into output variables. Such functions make it possible to create an arbitrary view. We extend Datalog by introducing terms of the following form to its rules:

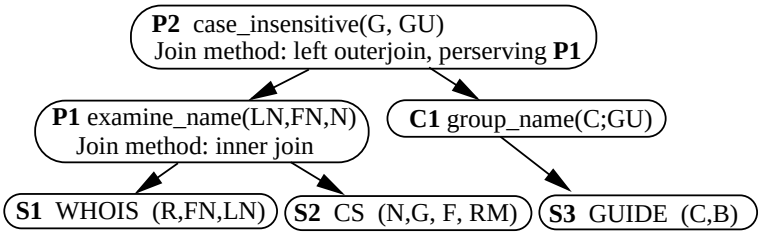
$$func(I_1, I_2, ..., I_m;\ O_1, O_2, ..., O_n),$$

where $I_1, I_2, ..., I_m$ are input variables and $O_1, O_2, ..., O_n$, output variables.

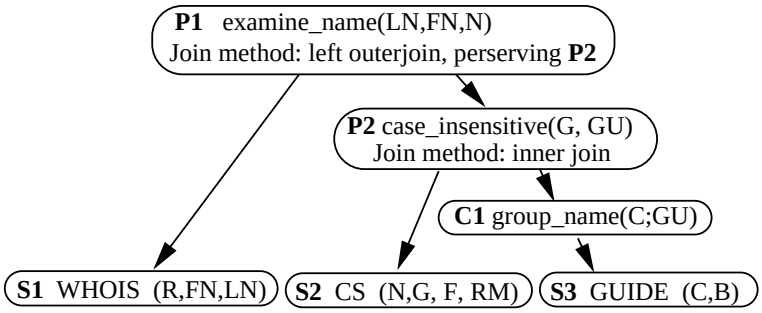
Example 3.6

Suppose that $GUIDE$ in Rule (3.6) has a group code number C instead of group name G and the code number C should be converted into a group name GU with a function $group\_name(C;\ GU)$. Suppose also that GU consists of upper case letters and predicate $case\_insensitive(G, GU)$ should be used to compare GU with G in CS. In this case, Rule (3.6) should be rewritten as

$$\begin{aligned} view(R, N, G, F, RM, B) \text{ :- } & WHOIS(R, FN, LN) \\ & \&\ CS(N, G, F, RM)\ \&\ GUIDE(C, B)\ \& \\ & examine\_name(LN, FN, N)\ \&\ case\_insensitive(G, GU)\ \& \\ & group\_name(C;\ GU). \end{aligned} \tag{3.7}$$



(a) The expression tree for Rules (3.7) and (3.8).



(b) The expression tree for Rules (3.7) and (3.9).

Figure 3.7. Expression Trees.

Suppose that the following query is issued against Rule (3.7):

$$answer(R, N, G, F, RM, B) :\text{-}$$
$$view(R, N, G, F, RM, B) \ \&\ R = \text{“}faculty\text{”} \ \&\ F = 4. \qquad (3.8)$$

Figure 3.7 (a) shows the appropriate query plan in the form of an expression tree for Rule (3.7) and the query in Rule (3.8). In the figure, each node represents a basic unit of execution, and has a unique label denoted by bold letters: the label for relation nodes starts with **S**, indicating *Source node*; predicate nodes with **P**, indicating *Predicate node*; foreign function nodes with **C**, indicating *Converter node*. The output of $WHOIS$ and CS should be merged with the inner join first, using predicate $examine\_name(LN, FN, N)$, since the conjunction of

conditions is imposed on $WHOIS.R$ *and* $CS.F$. The output relation of the predicate node **P1** meets the query conditions, but we can add additional information that originally come from $GUIDE$. Attribute C of relation $GUIDE$ is first converted by foreign function $group\_name(C;\ GU)$ into GU. Then, predicate $case\_insensitive(G, GU)$ merges the output relations from **C1** and **P1** with the left outerjoin, preserving the tuples in **P1**. □

Example 3.7

Figure 3.7 (b) shows the appropriate query plan when the following query is given against Rule (3.7).

$$\begin{aligned} &answer(R, N, G, F, RM, B) : - \\ &\quad view(R, N, G, F, RM, B)\ \&\ F = 4\ \&\ B = \text{“}MJH\text{”}. \end{aligned} \tag{3.9}$$

In this case, the outputs of CS and $GUIDE$ should be merged with the inner join, after $GUIDE.C$ is converted into GU with **C1**, since the conjunction of conditions is imposed on $CS.F$ *and* $GUIDE.B$. Then, **P1** should merge the result of the inner join and $WHOIS$ with the left outerjoin, preserving the output of the inner join in order to add additional information. □

As the above examples illustrate, different query plans should be created for the same extended Datalog rule, depending on different queries. In creating query plans, the execution order of the nodes in the expression tree is constrained by foreign function nodes: the output variables of foreign functions are available only after the execution of the functions. This fact makes automatic creation of the query processing plan significantly more complex. The rest of this paper is focused on describing the algorithm for creating the query processing plan.

3.3.2 A Model for the Problem

Basic Definitions

Before describing the algorithm, basic terms are defined here.

A directed graph G is defined to be a pair (N, E), where N is a non-empty set of *nodes* and E is a set of *edges*, and an edge is an ordered pair $(n_i, n_j) \in N \times N$, for $i \neq j$.

Successor of node n is defined as:

$$successor(n) = \{s_i \mid s_i \in N, \ ^{\exists} e \in E, e = (n, s_i)\}.$$

Predecessor of node n is defined as:

$$predecessor(n) = \{p_i \mid p_i \in N, \ ^{\exists} e \in E, e = (p_i, n)\}.$$

Path is a sequence of nodes $[n_1, n_2, ..., n_k]$, such that there is an edge (n_i, n_j) for $i = 1, 2, ..., k-1$ and $j = i + 1$.

Connection is a sequence of nodes $[n_1, n_2, ..., n_k]$, such that:

1. $n_i \neq n_j$ for $1 \leq i, j \leq k$ and $i \neq j$, and

2. There exists an edge either (n_i, n_j) or (n_j, n_i) for $1 \leq i \leq k-1$ and $j = i+1$.

When there exists a connection that includes n_p and n_q, n_p and n_q are *connected*.

A directed acyclic graph is defined to be a directed graph (N, E), such that no nodes appear more than once on any path on the graph.

A *root node* of directed acyclic graph (N, E) is an element of the set $\{n_i \mid n_i \in N, predecessor(n_i) = \phi\}$. A *leaf node* of a directed acyclic graph (N, E) is an element of the set $\{n_i \mid n_i \in N, successor(n_i) = \phi\}$. A *reachable node* of node n in a directed acyclic graph (N, E) is an element of the set $\{r_i\}$ for $i = 1, 2, ...$ such that a path exists from n to r_i.

Let each of S, P and C be a set of nodes in directed acyclic graph G'$= (N, E)$, such that each node in N is an element of exactly one of S, P, and C.

Join connection in G' is either $[s]$, where $s \in S$, or an alternate sequence of nodes $[s_1, p_1, s_2, p_2, ..., p_k, s_{k+1}]$, such that:

1. $s_i \in S, p_i \in P$,
2. $s_i \neq s_j$ for $1 \leq i, j \leq k+1$ and $i \neq j$, and $p_i \neq p_j$ for $1 \leq i, j \leq k$ and $i \neq j$, and
3. There exists an edge (p_i, s_i) for $1 \leq i \leq k$, and there exists an edge (p_i, s_{i+1}) for $1 \leq i \leq k$.

Merge connection in G' is a connection $[n_1, n_2, ..., n_k]$, such that:

1. $n_1, n_k \in S \cup C$, and $n_i \in C \cup S \cup P$ for $2 \leq i \leq k-1$,
2. If $n_i \in C$ for $2 \leq i \leq k-1$, there exists a pair of edges either $((n_{i-1}, n_i)$ and $(n_i, n_{i+1}))$ or $((n_{i+1}, n_i)$ and $(n_i, n_{i-1}))$, and
3. If $n_i \in S$ and $n_{i+1} \in P$, there exists an edge (n_{i+1}, n_i), and if $n_i \in P$ and $n_{i+1} \in S$, there exists an edge (n_i, n_{i+1}).

Example 3.8

In Figure 3.10(b), $[\mathbf{S1}, \mathbf{P1}, \mathbf{S2}, \mathbf{P2}, \mathbf{S3}]$ is a join connection, and $[\mathbf{S1}, \mathbf{P1}, \mathbf{C1}, \mathbf{P3}, \mathbf{C2}]$ is a merge connection. □

Creating an Initial Expression Graph

Given an extended Datalog rule as in Rule (3.7) and a conjunctive query as in Rule (3.8), first of all, we create an *initial expression graph*.

First, we create nodes for each of extensional database predicates, intentional database predicates, and foreign functions in the extended Datalog rule:

1. For an extensional database predicate, create a source node. All the variables in the parentheses are output variables. This node has no input variables.

2. For an intentional database predicate, create a predicate node. All the variables in the parentheses are input variables. This node has no output variables.

3. For a foreign function, create a converter node. The variables that precede symbol ";" are input variables, and the variables that follow ";" output variables.

Then, we create an edge from node n_i to n_j if both output variables of n_j and input variables of n_i share a non-empty set of variables. (Note that the direction of edges is the opposite to the flow of data.)

If the query imposes conditions on output variables of source or converter nodes, we define such nodes as *initially indispensable nodes*[1] . We create a predicate node for each of the initially indispensable converter nodes, and create an edge from the predicate node to the converter node. We let the predicate node thus created have the predicate that selects only such tuples that meet the query conditions.

Example 3.9

Figure 3.8 shows the initial expression graph for Rules (3.7) and (3.8). Nodes **S1** and **S2** that correspond to $WHOIS$ and CS are marked by symbol "$*$" to indicate these are initially indispensable nodes, since Rule (3.8) imposes conditions on variables R and F that are outputs of **S1** and **S2**. □

[1] Note that this definition of initially indispensable node is different from the one in Section 3.2.

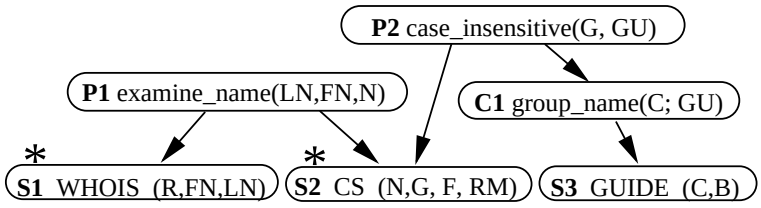


Figure 3.8. Initial expression graph for Rules (3.7) and (3.8).

Problem Statement

Now that we have created an initial expression graph, our goal is to convert the graph into an expression tree that represents an optimized query plan. Note that the notion of full disjunction introduced in Section 3.2 is not applicable to our current premise, where the extended Datalog rule includes foreign functions. Thus the algorithm presented in the rest of this chapter aims at obtaining as much information as possible, in as short time for the query processing as possible. In this dissertation, we focus on algebraic optimization that can be achieved with the information on the given rules.

Our basic policy of optimization is as follows:

1. Push selection operators (predicate nodes with a single successor) as far down to sources as possible.

   By this policy, we can reduce the number of tuples at early stages of the query processing, and save computation time. Note that even though this policy is appropriate for most of the cases, there are exceptions: when the cost of the selection operation is expensive, there are cases where it is better to apply the operation at a later stage of the query processing. In such cases, the designated optimization stages in the algorithm proposed below should be modified.

2. Pull conversion operators (converter nodes) as far up to view as possible.

   By this policy, selection and join operators can reduce the number of tuples before they are processed by the conversion operators.

3.3.3 An Algorithm for Creating a Query Plan

Outline

An algorithm is presented for converting an initial expression graph into an optimized expression tree in this section. Figure 3.9 shows the outline of the algorithm. The first step is to convert the given initial expression graph into a *dependency expression graph*. Figure 3.10 (a) shows an example of the initial expression graph, and (b) shows the corresponding dependency expression graph. The dependency expression graph indicates "dependency" among the nodes in the graph. Node **C1** in (a), for example, "depends" on **P1** and **P2** in that **C1** should be executed after **P1** and **P2**: for **C1** to convert attributes that come from **S1** and **S3**, the outputs from **S1**, **S2**, and **S3** should be already joined with **P1** and **P2**. Thus in Figure 3.10 (b), edges from **C1** to **P1** and from **C1** to **P2** are added. The dependency expression graph indicates the *necessary condition*

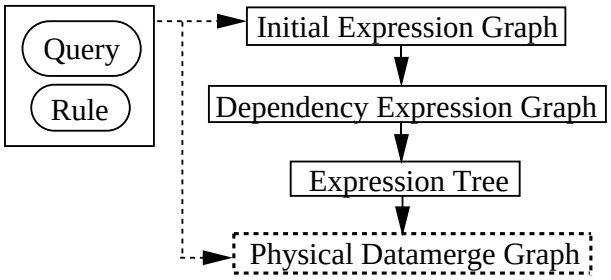
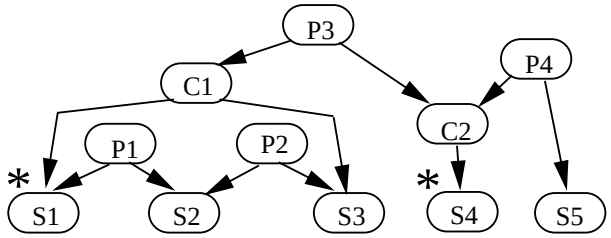
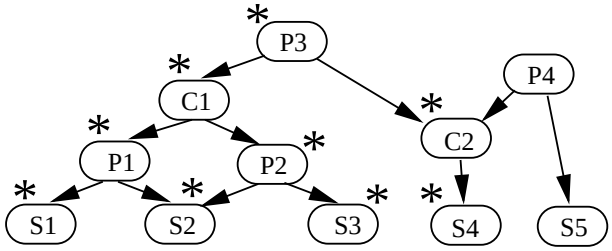


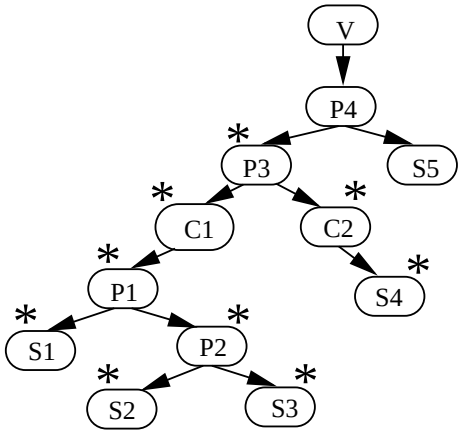
Figure 3.9. The outline of the query processing.



(a) Initial expression graph.



(b) Dependency expression graph.



(c) Expression tree.

Figure 3.10. An example of the query processing.

for a valid expression tree. The conversion from the initial expression graph into a dependency expression graph does not involve any optimization.

The next step is to convert the dependency expression graph into an executable, optimized expression tree, as shown in Figure 3.10 (c). This conversion involves algebraic optimization.

The final step is to create a physical datamerge graph mentioned in [PGMU96]. Each of source, predicate, and converter nodes creates the full procedures. For example, a source node creates a procedure to send a query to the native information source, extract attributes from the query result, and set the attributes in a relation. The final step will be described in Chapter 4 of this dissertation.

Each step is explained as follows.

Conversion into a Dependency Expression Graph

At this stage, we convert an initial expression graph into a dependency expression graph. During this process, first, edges that denote the dependencies are created among source, predicate, and converter nodes. Then, the nodes whose output should be "preserved" in outerjoin operations are marked "*indispensable nodes*." This process is achieved by procedure **settle_all()** in Figure 3.11, which calls procedure **settle(N,I)** in Figure 3.12.

In procedure **settle_all()**, the merge connection that connects all the initially indispensable nodes is searched on line (2). In Figure 3.10 (a), the indispensable nodes **S1** and **S4** are marked by symbol "*." The merge connection that connects **S1** and **S4** is [**S1, C1, P3, C2, S4**]. All the nodes in the merge connection are "indispensable" for the initially indispensable nodes to join. Then for each node **N** on the merge connection, procedure **settle(N, True)** is called on line (4). The second argument indicates whether **N** is an indispensable node or not. On line (6), **settle(C, False)** is called for each node **C** of converters that are not initially

procedure settle_all();

```
    begin
(1)   Let I be the set of all the initially indispensable nodes;
(2)   Search for the merge connection that connects all the nodes in I, and
      let M be the set of the nodes on the searched merge connection;
(3)   for each node N in M do
(4)     settle(N, True);
(5)   for each converter node C that is not an initially indispensable node do
(6)     settle(C, False);
(7)   reduce();
    end.
```

Figure 3.11. Procedure **settle_all**.

procedure settle(N,I);

```
     begin
(1)    if I is true, mark node N "indispensable";
(2)    if N is already marked "settled" then return;
(3)    if node N is not a converter node then
(4)      for each successor S of node N do
(5)        settle(S,I);
(6)    else
         begin
(7)        C := all the nodes on the merge connections among N's successors;
(8)        for each node M in C do
           begin
(9)          settle(M, I);
(10)         if M is a predicate node then create an edge from N to M;
(11)         if M is a source node then delete the edge from N to M;
           end;
         end;
(12)   mark N "settled";
     end.
```

Figure 3.12. Procedure **settle**.

indispensable nodes. The procedure **reduce()** on line (7) reduces edges in the graph so that the graph represents the skeleton, or the transitive closure [Ull88] of the relations denoted by the edges, eliminating redundant edges.

Procedure **settle(N, I)** is called recursively unless **N** is already "settled." On line (1), node **N** is marked "indispensable" if **I** is true. If **N** is not a converter node, for each successor **S** of node **N**, **settle(S, I)** is called on line (5). If **N** is a converter node, **settle(M, I)** is called on line (9) for each node **M** of all the nodes on the merge connections among **N**'s successors. In our example, when **settle(M, I)** is called for **M**=**C1** and **I**=True, the merge connection among **C1**'s

procedure optimize();

```
     begin
(1)    Let S be the set of indispensable source nodes;
         // S keeps the root nodes of the clusters.
(2)    do forever
       begin
(3)      C := S;
(4)      do forever
         begin
(5)        if C is empty, goto line 11;
(6)        Pick up an arbitrary node N from C;
(7)        cluster(N, G, T);
             // T is the new top node, and
             // G is a set of all the nodes in the cluster.
(8)        S := S - G;
(9)        add node T to S;
(10)       C := C - G;
         end;
(11)     reduce();
(12)     if S has just 1 element then;
         begin
(13)       U := the element in S;
(14)       goto line 23;
         end;
(15)     C := S;
(16)     do forever
         begin
(17)       if C is empty, goto line 3;
(18)       Pick up an arbitrary node M from C;
(19)       mergeIndispensable(M, T);  // T is the new top node
(20)       Remove node M from S;
(21)       Add node T to S;
(22)       Remove node M from C;
         end;
       end;
(23)   do forever
       begin
(24)     mergeOthers(U,Q);
(25)     if node Q is null or has no predecessors, goto line 26;
       end;
(26)   Create a view node and connect it to Q;
     end.
```

Figure 3.13. Procedure **optimize**.

successors is [**S1**, **P1**, **S2**, **P2**, **S3**]. On line (10), edges are created from **C1** to **P1** and from **C1** to **P2**, since **C1** "depends" on **P1** and **P2** in that **P1** and **P2** should be executed prior to **C1**. Redundant edges from **C1** to **S1** and from **C1** to **S3** are deleted on line (11). On line (12), node **N** is marked "settled," meaning **settle()** has already been called for all the nodes that are "below" node **N**.

Conversion into an Expression Tree

At this stage, we convert a dependency expression graph into a valid expression tree. We assume that the marks "settled" are all cleared after the above procedures.

Procedure **optimize()** in Figure 3.13 converts a dependency expression graph into an expression tree by calling three main procedures:

1. **cluster(N, G, T)** (Figure 3.14)

   This procedure is called from procedure **optimize()** on line (7). First of all, on line (1), **cluster(N, G, T)** searches for the longest join connection that consists of indispensable nodes only, starting from node **N**. In our example, **cluster (N, G, T)** is called twice: first, with **N** = **S1**, and second, with **N** = **S4**. In the first call, join connection [**S1, P1, S2, P2, S3**] is found on line (1). We call the group of the nodes in a join connection a *cluster*. Then the nodes on the join connection are reconnected to form a tree structure with only one root node. Join connection [**S1, P1, S2, P2, S3**], for example, is reconnected as shown in Figure 3.10 (c). On line (5), procedure **reconnectToTop(T)** is called. If edges are connected from the nodes outside the cluster to the nodes inside the cluster, then, this procedure changes the destinations of all such edges to root node **T**. Procedure **cluster(N, G, T)** returns set **G** of all the nodes in the cluster and the new root node **T**. When **cluster (N, G, T)** is called for the second time, join connection [**S4**] is found. The structure of join connection [**S4**] remains the same. After this procedure is called, we regard the nodes in **G** as "settled" and "indispensable," and the root node **T** as a new source node.

   In the process of forming a tree structure on line (2) of **cluster(N, G, T)**,

procedure cluster(N,G,T);

```
    begin
(1)   Search for the longest join connection G that includes N;
(2)   Reconnect the nodes in G to form a tree structure;
(3)   Let the root node of the tree structure be T;
(4)   Mark all the nodes in G "settled" and "indispensable";
(5)   reconnectToTop(T);
    end.
```

Figure 3.14. Procedure **cluster**.

procedure mergeIndispensable(N,M);

```
    begin
(1)   If N's predecessor nodes include one or more nodes that are
      indispensable predicate nodes, set M to N and return;
(2)   Pick up a node M with the highest precedence out of N's
      predecessors. The precedence is as follows:
        1. Predicate node that has just 1 successor
        2. Converter node:
          • that has 1 successor, and;
          • each of whose ancestors has 1 successor, and;
          • whose top-most ancestor is a predicate node.

        3. Converter node whose ancestors include indispensable predicate nodes;
(3)   If no node satisfies the above condition, set M to N and return;
(4)   reconnectToTop(M);
(5)   reduce();
    end.
```

Figure 3.15. Procedure **mergeIndispensable**.

there is room for further optimization. For instance, we can change the order of join operations based on the statistical information on the relations. For a parallel system, balancing the tree would be a good optimization technique.

2. **mergeIndispensable(N, M)** (Figure 3.15)

   The clusters searched by procedure **cluster(N, G, T)** are separated due to the existence of converter nodes: in Figure 3.10 (b), if it were not for converters **C1** and **C2**, the clusters [**S1**, **P1**, **S2**, **P2**, **S3**] and [**S4**] could be merged into one cluster with predicate node **P3**.

   In order to group together separate clusters, procedure **optimize()** calls procedure **mergeIndispensable(N, M)** on line (19). **N** is the root node

of a cluster. This procedure picks up a node out of **N**'s predecessor nodes, according to the precedence indicated in the pseudo code. Then, this procedure "absorbs" the picked-up node into the cluster, and the absorbed node becomes the new root node.

After procedure **mergeIndispensable(N, M)** is called, the root node of this cluster is regarded as a new source node. The new source node, thus created, may be merged with other clusters by procedure **cluster(N, G, T)**. In our example, after **C1** is merged into cluster [**S1, P1, S2, P2, S3**] and **C2** is merged into cluster [**S4**], these two clusters can constitute a join connection together with **P3**, forming a new cluster [**S1, P1, S2, P2, S3, C1, P3, C2, S4**]. Procedure **optimize()**, therefore, calls procedures **mergeIndispensable(N, M)** and **cluster(N, G, T)** in turn until all the clusters are merged into one.

3. **mergeOthers(N,M)** (Figure 3.16)

   After all the indispensable clusters are merged into one, **optimize()** calls procedure **mergeOthers(N,M)** on line (24) to absorb into the cluster a node immediately above the cluster. This procedure is called repeatedly until all the nodes are merged into the cluster.

   Procedure **mergeOthers(N,M)**, on line (1), picks up a node out of **N**'s predecessor nodes, according to the precedence indicated in the pseudo code. Then, this procedure "absorbs" the picked-up node into the cluster and the absorbed node **M** becomes the new root node.

Lastly, on line (26), procedure **optimize()** creates a view node, and connects it to the root node of the expression tree with an edge. The final result of the expression tree is shown in Figure 3.10 (c).

procedure mergeOthers(N,M);

```
     begin
(1)    Pick up a node M with the highest precedence out of N's predecessors.
       The precedence is as follows:
         1. Predicate node that has just 1 successor
         2. Converter node:
          • that has 1 successor, and;
          • each of whose ancestors has 1 successor, and;
          • whose top-most ancestor is a predicate node.

         3. Any predicate node
         4. Converter node whose ancestors include predicate nodes
         5. Any converter node;
(2)    If no node satisfies the above condition, set M to null and return;
(3)    reconnectToTop(M);
(4)    reduce();
     end.
```

Figure 3.16. Procedure **mergeOthers**.

Procedures **mergeIndispensable(N,M)** and **mergeOthers(N,M)** can be further optimized in choosing a predecessor node **M** if additional information, such as execution cost of each operation, is available. For example, as mentioned in Section 3.3.2, if the cost of selection with a particular predicate is known to be expensive, the priority of the "predicate node that has just 1 successor" should be modified to be lower. We can also project out columns at converter and predicate nodes, if the columns are not used later.

After the expression tree is created, for each predicate node with two successors, we can select the appropriate join method from the alternatives of the inner join, left outerjoin, and full outerjoin according to the following rule. If only one of the two successor nodes of a predicate node is marked indispensable, the join method should be the left outerjoin: the output from the marked node should be "preserved," and the output from the other node should be discarded unless a matching tuple is found. If both of the successors are marked indispensable, the join method should be the inner join. If none of the successors are marked indispensable, the join method should be the full outerjoin. For example, in Figure 3.10 (c), **P4** should be a left outerjoin, and **P2** an inner join.

3.4 Related Work

The outerjoin operation has been included in SQL-92, and is supported by some of the commercial RDBMS products. There are pioneering works for optimization of query processing with outerjoins [RR84, RGL90, GLR92, GL94]. [BGI95] uses the hypergraph [Ull89] and defines a *modified generalized outerjoin* to allow duplicates in relations for reordering of wide range of queries. They do not, however, take foreign functions into account. [CS93] handles query optimization in the presence of foreign functions, using *rewrite rules*, but this does not consider outerjoins.

Efforts have been made to integrate databases with the outerjoin. [LW94] gives a rigorous algorithm to instantiate object views from relational databases, using the outerjoin. Its focus is on creation of static views, and it requires more detailed specification expressed by relational algebra on how to instantiate a view than our approach. Also it makes use of integrity constraints of the underlying RDBMS's. [Che90] integrates multidatabase systems with the outerjoin, but its focus is on sharing algebraic operations among databases. Universal relation [Ull89, MRa86, MUV84] also aims at integration of underlying RDBMS, but it does not deal with foreign functions and its emphasis is on exploiting dependencies among the data for eliminating bogus connections.

3.5 Summary

In this chapter algorithms were presented for creating query processing plans that organize maximum information in response to conjunctive queries. The contributions are summarized as follows:

1. Given a set of relations and a conjunctive query, the maximum information was defined as the tuples in the full disjunction that meet the query

condition. It was illustrated with concrete examples that the outerjoin is essential to calculate the full disjunction, preventing information loss. It was also shown that different query processing plans are required, depending on different queries. A naive algorithm was presented for creating query processing plans, using the notion of full disjunction. This algorithm is applicable to any kind of relations whose hypergraph representation is connected.

2. An optimized algorithm was shown for creating query processing plans in the above situation, using a hypergraph. This algorithm is applicable only to a connected, γ–acyclic hypergraph. In the proposed algorithm, we first drop some relations whose tuples are guaranteed not to participate in the final query result. Next, we select only such tuples in each relation that meet the query condition. Then, using Algorithm SOJO presented in [RU96], we create a sound outerjoin ordering. This ordering is guaranteed to produce the full disjunction of the relations. In each join process, appropriate join methods are chosen to improve efficiency.

3. Predicates and foreign functions were shown to play important roles in integrating multiple relations for creating a view. Datalog was extended to incorporate foreign functions that can take an arbitrary number of input and output variables. Then, a detailed algorithm was presented for creating query processing plans for conjunctive queries in the presence of predicates and foreign functions, aiming at obtaining as much information as possible. In the algorithm, the stages were identified where the plan can be further optimized, for instance, with the knowledge of the statistical information on the relations.

Algorithm 3.3 is guaranteed to produce the tuples in the full disjunction that meet the given query condition. This algorithm, however, is not necessarily optimum: for example, in some cases, we can remove more relations before joining the relations in Step 5. The improvement of the algorithm is a future research issue.

The algorithm in Section 3.3 was implemented for use in the mediation system described in Chapter 4. The program was tested for a number of different pairs of an extended Datalog rule and a query, and the program works for the test suite. The proof of the correctness of this algorithm, however, seems to be hard, and is an open research issue.

Chapter 4

Query Processing Based on a Self-Descriptive Object Model

4.1 Introduction

In this chapter, a query processing method is proposed for obtaining self-descriptive objects from heterogeneous information sources and generating a view by converting, joining, and selecting them. In creating the query processing plan, the algorithm described in Chapter 3 is employed.

Traditional query processing premises that the schema of all the data is specified in advance. In integration of heterogeneous information sources, however, the schema is not necessarily known to the mediation module in advance. The data obtained from heterogeneous information sources are, in general, semistructured: they may have a different structure from one another and are not always structured completely. For example, in some information sources, there are cases where the same attribute values can be set plurally, some data are missing, and users can append attributes and their values. In addition, each section of an organization may change the schema of the local information source autonomously. Thus it is

not possible to deal with semistructured data with the query processing methods for traditional data models, such as relational data model, object-oriented data model, and predicate logic.

A promising data structure to integrate heterogeneous information sources is a self-descriptive object model. This is a model in which each data instance has a label that expresses its own meaning. The Tsimmis project at Stanford University [CGMH+94, PGMW95, PGMU96, PGGMU95, PGGMU95, VP97] aims at integrating heterogeneous, distributed information sources by using *OEM* (Object Exchange Model) object model. The OEM object is composed of a nested structure of self-descriptive subobjects. Data of various structures obtained from heterogeneous, distributed information sources can be converted into OEM objects and integrated by the mediator. This allows flexibility in dealing with the semistructured data and allows partial evolution of the schema, without any system change [PGMU96]. No other research in which the self-descriptive object model is applied to the integration of information sources has been found, and the implementation method of the query processing has not been clarified. In this chapter, query processing techniques inherent in handling the semistructured data are described based on the OEM data model.

The rest of this chapter is organized as follows: In Section 4.2, first, the outline of the Tsimmis project is described. Then a new query language *OEM-QL* (OEM Query Language) is proposed. In addition, *RMSL* (Revised MSL) is proposed as a revision of *MSL* (Mediator Specification Language), which is used for defining a view in the Tsimmis project. In Section 4.3, a method for creating a query processing plan is proposed. Section 4.4 describes how to execute the query processing plan. Section 4.5 describes the implementation of the proposed method. Section 4.6 discusses the proposed method. Related work is described in Section 4.7. Finally Section 4.8 summarizes the chapter.

4.2 Data Integration System

4.2.1 System Architecture

The mediator architecture [PGMU96] for integration of heterogeneous, distributed information sources in the Tsimmis project is shown in Figure 4.1. Mediators are located between information sources and application programs. A wrapper is established for each information source and plays the role of concealing the interface inherent in each information source from the mediators. An application program issues queries to a mediator with a query language called *MSL-QL* (MSL Query language). The mediator refers to the data integration specification described by *MSL* (Mediator Specification Language), and creates a query processing plan. Based on this plan, subqueries in MSL-QL are issued to wrappers. Each wrapper converts the subquery into the native query of each information source and

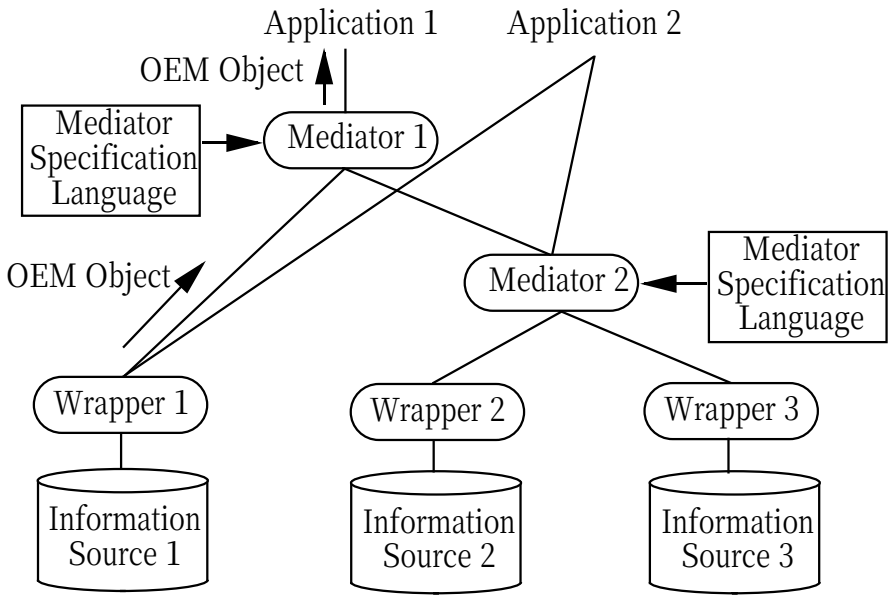


Figure 4.1. The Tsimmis architecture of the data integration system.

issues this to the information source to receive the result. The result is converted into the OEM object and returned to the mediator. The mediator receives the query results from the wrappers, then it converts, joins, and selects the results to generate a view expressed as the OEM object to return it to the application program.

4.2.2 Object Model OEM

In the Tsimmis project, the query processing result is delivered from the wrapper to the mediator and from the mediator to the application program as the OEM object. The OEM object has a nested structure of self-descriptive subobjects of the following form:

< OID, LABEL, TYPE, VALUE >.

OID is an object identity; LABEL is a label which describes the meaning of the value of this object; TYPE is a type of VALUE; VALUE is a value. The type of the VALUE is either an atomic value or a set of object identities. An atomic value includes a character string, an integer, and binary data. If VALUE is a set of object identities, the OEM object has a nested structure. For example, the following is a nested OEM object:

< o_1, person, set, {o_2, o_3}>
< o_2, name, string, "Jeff Ullman">
< o_3, phone, integer, 7720>.

In this example, the subobject with identity o_1 has a label "person" and a value which is a set {o_2、 o_3} of the identities of the subobjects; the subobject with identity o_2 has a label "name" and a value "Jeff Ullman" of the string type; the

subobject with identity o_3 has a label "phone" and a value 7720 of the integer type.

In the following, relative to a subobject (e.g., o_1), we consider the subobject (e.g., o_2 and o_3) of an element of a set specified as a value to be subordinate by one level, and the subobjects (e.g., o_2 and o_3) within a set to be all at the same level. We call the set of subobjects, the atomic value, and the label generically an object element. In addition, in the following, we represent the above mentioned OEM object as follows, omitting the object identities and the value types except for the topmost subobject:

o_1:<person, {<name, "Jeff Ullman"> <phone, 7720>}>.

4.2.3 Query Language

In the Tsimmis project, retrieval of an OEM object is requested with MSL-QL from the application program to the mediator and from the mediator to the wrapper [PGMU96]. It is, however, impossible for MSL-QL to specify in detail which subobjects are to be obtained. Therefore a revised query language OEM-QL is proposed based on MSL-QL. An example is shown as follows:

? <cs_person, {+ <phone, $*$ > <name, "Jeff Ullman">}>

Each subobject is expressed with a pattern of [s]< L [, V]>. [ ] denotes an optional item and represents itself as omissible. A label or symbol "$*$" is specified to L; symbol "$*$" means an arbitrary label and can be specified only when no other object of the same level exists. V is a query condition of an atomic value, "$*$", or a set of subobjects, and is omissible. If "$*$" is specified, it is regarded as an imposition of a query condition "a subobject with the label L exists." Symbol s may be "?," "+," or "−," and is omissible. A maximum of one "?" can be

specified in a query, and "?" denotes obtaining a subobject, with its subordinate subobjects occurring after the designation of "?." If "?" is not specified, the designation is considered to have been made just before the topmost subobject. If "+" is specified, only the subobjects with "+" are obtained in the set of the subobjects at the same level. If "−" is specified, neither the subobject nor its subordinate subobjects are obtained. If V is not specified, s should be specified to the subobject.

In the above mentioned example, a retrieval request of such OEM object is specified which consists of a topmost subobject with a label 'cs_person' and the value of a set of subobjects, including a subobject with a label 'phone' and an arbitrary value and a subobject with a label 'name' and a value 'Jeff Ullman.' The subobject with a label 'name' is, however, not obtained.

4.2.4 Mediator Specification Language

In the Tsimmis project, the processing of the mediators is declaratively described by MSL [PGMU96]. MSL is a language used to specify from which wrapper to obtain the data and how to combine the obtained data to make a view. However, there exist some problems in using the MSL for our purposes, which will be mentioned later in Section 4.6. Hence a revised MSL (RMSL) is proposed. An example is shown in Figure 4.2. Hereafter, this is called **RMS1**. RMSL is, in general, composed of multiple rules. **RMS1** has only one rule. In the description of RMSL, prior part [(1)st line] is called the head, and the posterior part the tail. The meaning of each part is explained below.

1. Head. ((1)st line in **RMS1**)

   The mediator expresses a view of the OEM object which is returned to the application as a retrieval result. The pattern < L, V > represents a

(1) <cs_person, {<name, $N> <relation, $R>
<group, $GL><phone,$P> <location, $L> $Rest1 $Rest2>}>:-
(2) <$R, {<first_name, $FN> <last_name, $LN> $Rest1}>@WHOIS
(3) <person, {<name, $N> <group, $GL> <phone, $P> $Rest2}>@CS
(4) <group, {<code, $C> <location, $L>}>@GUIDE
(5) examine_name($LN, $FN, $N)
(6) case_insensitive($GU, $GL)
(7) group_name($C; $GU)

Figure 4.2. **RMS1**, an example of RMSL.

subobject of the OEM object with label L and value V. The value is an atomic data item or a set of subobjects. The character string beginning with $ is a variable, and the actual data are bound to it according to the designation in the tail.

The tail is composed of each of the following description sections.

1. Description section of information sources. ((2)nd, (3)rd, and (4)th lines)
   A description is given of a wrapper (or other mediator) and a pattern of the OEM object sent from there. In the tail, for example, @**WHOIS** on the (2)nd line indicates that the OEM objects of the pattern specified before @ are obtained from the wrapper **WHOIS**. In each of the obtained OEM objects, the object elements corresponding to the variables such as $R, $FN, $LN, and $Rest1 are bound to these variables.

   For example, as a result of processing corresponding to the query

   <cs_person, {<relation, "faculty"> <phone, 7720>}>, (Q1)

WHOIS:

o_1 :<faculty, {<first_name, "Gio"> <last_name, "Wiederhold">
<room,415>}>

CS:

o_2 :<person, {<name, "Gio Wiederhold"> <group, "database">
<phone,7720>}>

GUIDE:

o_3 :<group, {<code, "CS-LOGIC"> <location, "Gates">}>

o_4 :<group, {<code, "CS-DB"> <location, {<building, "MJH">
<floor,4>}>}>

Figure 4.3. OEM Objects obtained from wrappers.

suppose that the OEM objects shown in Figure 4.3 are obtained from each wrapper. The object obtained from the wrapper **WHOIS** is collated with the following pattern on the (2)nd line in **RMS1**:

<$R, {<first_name, $FN> <last_name, $LN> $Rest1}>

Consequently, "faculty," "Gio," and "Wiederhold" are bound to the variables $R, $FN, and $LN, respectively. The variable $Rest1 is bound to the set ({<room, 415>}) of all subobjects other than those bound to the pattern (< first_name, $FN>< last_name, $LN>) specified before $Rest1. Such variables are called the Rest variable. In the same way, regarding the objects obtained from wrapper **CS** and **GUIDE**, the object elements are bound to each variable. Thus *variable tables* shown in Table 4.1 are created.

2. Predicate description section ((5)th and (6)th lines)
   The predicate is described. For example, examine_name($LN, $FN, $N) on

Table 4.1. Variable tables of object elements obtained from each wrapper.

(a) Object elements obtained from **WHOIS**

$R	$Rest1	$FN	$LN
"faculty"	{<room, 415>}	"Gio"	"Wiederhold"

(b) Object elements obtained from **CS**

$N	$GL	$P	$Rest2
"Gio Wiederhold"	"database"	7720	ϕ

(c) Object elements obtained from **GUIDE**

$C	$L
"CS-LOGIC"	"Gates"
"CS-DB"	{<building, "MJH"><floor,4>}

the (5)th line and case_insensitive($GU, $GL) on the (6)th line in **RMS1** are predicates. Either a "true" value or a "false" value is returned according to the input arguments specified in the parenthesized variable list. For example, in examine_name($LN, $FN, $N), if "Wiederhold," "Gio," and "Gio Wiederhold" are input to each of $LN, $FN, and $N, a "true" value is returned. The processing of the predicate described here is to be implemented with a programming language separately.

3. Conversion description section ((7)th line)

   The function to convert values is described. The group_name($C; $GU) on the (7)th line in **RMS1** is a function that converts a value given as an input argument and returns the result as an output argument. In the parenthesized variable list, the variables before ";" are input arguments and the variables after it are output arguments. The conversion processing

described here is also to be implemented, similarly to the predicate, with a programming language separately.

In RMSL, obtained data are bound to variables in the description section of information sources in the tail and they are assembled freely to make a view as described in the head. This enables flexible resolution of heterogeneity among information sources:

1. Difference of schema

   The same entity may be expressed by different schemas in different information sources. For example, an attribute expressed as 'name' in **CS** is represented as two attributes 'first_name' and 'last_name' in **WHOIS**. Such discrepancies can be resolved with a predicate by binding the values to its input arguments.

2. Correspondence of data and meta data

   Data of an information source (e.g., a value stored in a table) may be dealt with as meta data (e.g., table name) in another information source. In **RMS1** the label ($R) of the topmost subobject of **WHOIS** is used as a value of a 'relation' subobject in the head. In this way we can deal with data and meta data without discrimination.

3. Schema evolution

   Sometimes a schema of information sources is changed. For example, an attribute "secretary" may be added in **WHOIS** and <"secretary", "Siroker"> may be added to the 'faculty' subobject of object o_1. Even in this case, the added subobject is bound to the Rest variable, $Rest1, and the addition of an attribute is reflected in the view without changing the RMSL.

4. Incomplete data structure

   As seen in **GUIDE**, the data structure may be different from instance to instance in some information sources. Even in this case, it is possible to bind the data structure to a normal variable or the Rest variable and absorb the difference of the structure.

5. Difference of vocabulary/data structure

   There are some cases where the corresponding vocabulary of data, the type/ accuracy/unit of data, and so forth vary depending on information sources. For example, there may be a case where the value corresponding to $N in **CS** is not "Gio Wiederhold," but "Wiederhold, G." In this case, even if "Wiederhold," "Gio," and "Wiederhold, G." are bound to $LN, $FN, and $N in examine_name ($LN, $FN, $N), it is possible to deal with the case by implementing the function to return "true."

The mediator, on receiving a query, first obtains the OEM objects from the information sources in compliance with the description section of information sources of the RMSL and extracts the values, then creates variable tables such as shown in Table 4.1. In compliance with the conversion description section, it converts the object elements in the tables, and in compliance with the predicate description it joins the tables and selects tuples. Finally, the mediator makes the OEM object of the pattern shown in the head of the RMSL and returns it as a query result.

4.3 Planning of Query Processing

The procedure of the query processing by the mediator is shown in Figure 4.4. The details of the processing are described in this section.

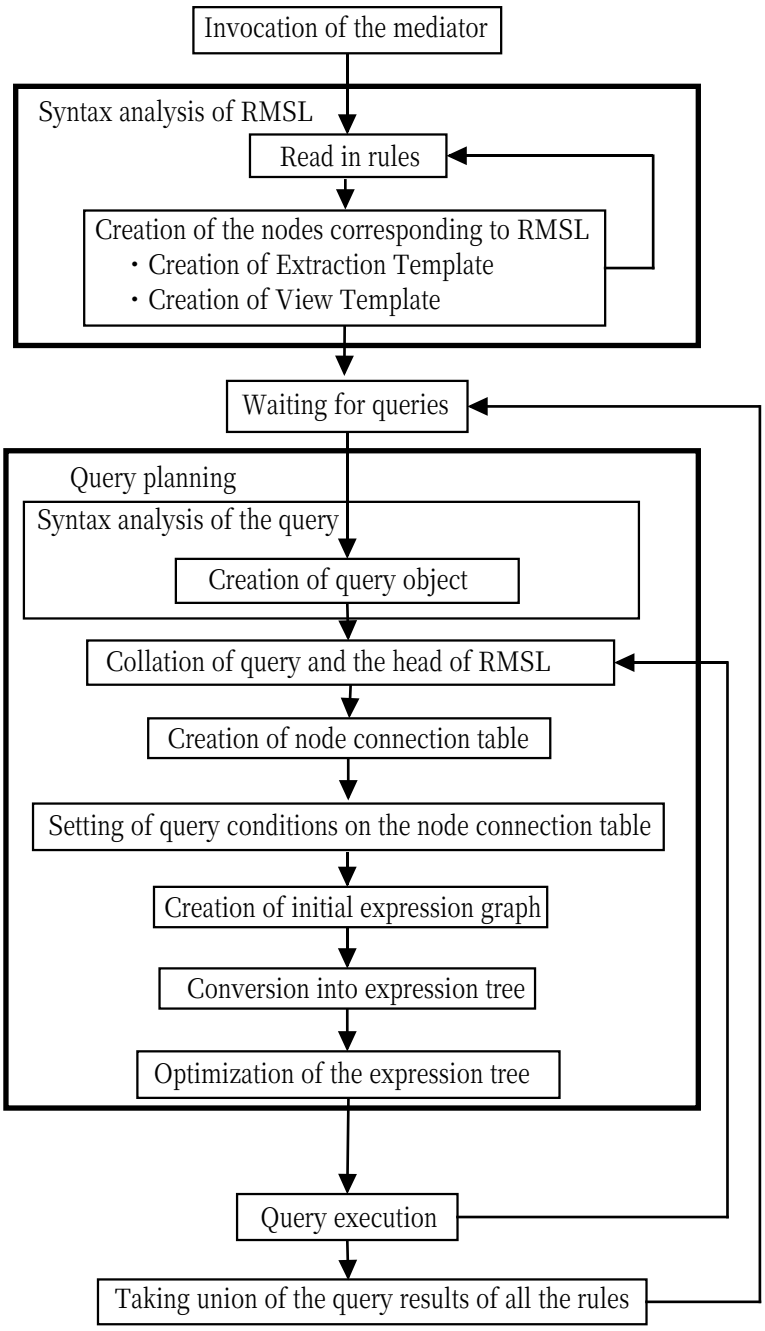


Figure 4.4. The outline of the mediator's query processing.

4.3.1 Preprocessing

At the time of activation of the mediator, the syntax of the RMSL is analyzed, and in the process, the following four kinds of nodes are created as the units of the query processing, corresponding to the head of each rule, the description of the information sources, the predicate description, and the conversion description.

1. View node. For each description of the head, a view node with the following attributes is created:

   (a) Input variables. Variables in the head become input variables to this node.

   (b) View template. A view template is created according to the structure of the OEM object described in the head. This is composed of a nested structure of subobjects. Each subobject has the form [s] <vlabel, vvalue>. Here, vlabel is a label or an index indicating a table column where labels are stored. Also, vvalue is a value, an index indicating a table column where values are stored, or a set of subobjects. An element of the set of subobjects may include an index which indicates a table column where a set of subobjects are stored, instead of the subobject of a normal view template. The value of [s] is set later at the time of comparison with a query.

Example 4.1

The following view template is created for the head of **RMS1**.

<cs_person, {<name, [$N]> <relation, [$R]> <group, [$GL]>
<phone, [$P]><location, [$L]> [$Rest1] [$Rest2]>}> (V1)

Here, [$N], [$R], [$GL], [$P], and [$L] are indexes indicating variable table

columns corresponding to the variables $N, $R, $GL, $P, and $L. [$Rest1] and [$Rest2] are indexes indicating variable table columns where sets of subobjects denoted by $Rest1 and $Rest2 are stored. □

2. Source node. For each description of information sources, a source node with the following attributes is created:

   (a) Output variables. The variables in the description of information sources become the output variables of this node.

   (b) Extraction template. According to the structure of the information source, an extraction template is created. This has the form <elabel, evalue>, just as a view template.

   (c) Information source. According to the description after @ in the description of the information source, an information source (a wrapper or another mediator) to which the subquery is issued is held.

Example 4.2

The following extraction templates are created for **WHOIS**, **CS**, and **GUIDE**.

<[$R], {<first_name, [$FN]> <last_name, [$LN]> [$Rest1]}> (E1)

<person,{<name, [$N]><group, [$GL]> <phone, [$P]>
[$Rest2]}> (E2)

<group, {<code, [$C]> <location, [$L]>}> (E3)

□

3. Predicate node. For each predicate description, a predicate node with the following attributes is created:

(a) Input variables. All arguments in the variable list become the input variables of this node.

(b) Predicate. A predicate is held according to the specified predicate name.

(c) Join method. In case a variable is input from multiple nodes to this node and a join is executed, any one of the alternatives of inner join, left outerjoin, right outerjoin, and full outerjoin is set in the final process of making an expression tree. If this node executes a selection, this attribute becomes unnecessary and is not set.

4. Conversion node. For each conversion description, a conversion node with the following attributes is created:

   (a) Input variables. The input arguments in the variable list become the input variables of this node.

   (b) Output variables. The output arguments in the variable list become the output variables of this node.

   (c) Function. The function with which the conversion is executed is held according to the specified function name.

In addition to the above mentioned attributes, all nodes have an indispensable node flag, and the source node and the conversion node have an initial indispensable flag.

4.3.2 Planning

When the mediator receives a query from the application program, it executes the query processing described in this section.

Creation of Query Objects

Syntax analysis is performed on the received query to create a query object reflecting the structure of the OEM object expressed by the query. The query object is composed of a nested structure of subobjects and each subobject has a form of [s] <qlabel [, qvalue]>, where [s] expresses any of "+," "−," or "?" optionally; qlabel expresses a label; and qvalue expresses an optional value.

Collating of the Query Object and the Head of the RMSL

The topmost subobject of the created query object is compared with the topmost subobject of the view template of each rule, and matched rules are selected. The following are the conditions under which the subobject [s]<qlabel [, qvalue]> of the query object matches the subobject [s]<vlabel, vvalue> of the view template:

[1] qlabel is "∗" or matches vlabel, and furthermore,

[2] qvalue is not specified; otherwise, if qvalue is a set of subobjects, for each elements e ∈ qvalue, an element w ∈ vvalue matched by e exists; if qvalue is an atomic value, vvalue is a variable.

For each rule whose view template matches the query object, the following processing is executed.

Creation of a Node Connection Table

All nodes created at the time of the syntax analysis of the RMSL are duplicated. The input variables to a duplicated node, excluding the view node and the output variables from there, are examined and a *node connection table* is created. An example is shown in Table 4.2. Columns corresponding to all variables in the RMSL are created in the node connection table. In each column, a pointer to

Table 4.2. A node connection table.

	$N	$R	$GL	$P	$L	$Rest1	$Rest2	$FN	$LN	$C	$GU
Input variable node	P1		P2					P1	P1	C1	P2
Output variable node	S2	S1	S2	S2	S3	S1	S2	S1	S1	S3	C1
Query condition		"faculty"		7720							

the duplicated node where the variable is input is written in the second row as an input variable node. In the table, the source node, the conversion node, and the predicate node are expressed by node names beginning with S, C and P, respectively. In the third row, pointers to the duplicated nodes where the variables are output are written as an output variable nodes.

Example 4.3

In Table 4.2, when making a duplicate **C1** of the conversion node corresponding to the function group_name($C; $GU) on the (7)th line of **RMS1**, a pointer to **C1** is written as an input variable node in the column of the variable $C and as an output variable node in the column of the variable $GU. □

Setting of Query Conditions to the Node Connection Table

The query object is compared with the view template again and the query conditions specified in the query are written to the node connection table. At the time of the comparison, the same conditions as for the comparison of the query object and the view template mentioned in Section 4.3.2 are used. Whenever the query conditions of the query object corresponding to the variable of the view template are found, the query conditions are written onto the fourth row of the column corresponding to the variable. In this process, the flag s of the subobject of the

query object is copied to the corresponding subobject of the view template.

Example 4.4

If Query (Q1) in Section 4.2.4 is compared with View template (V1) in Section 4.3.1, "faculty" corresponds to variable $R and 7720 to variable $P. Here, "faculty" and 7720 are each written to the node connection table, Table 4.2, as query conditions in the columns of $R and $P respectively. □

Creation of the Initial Expression Graph

According to the following rules, each column of the node connection table is examined in order, and the duplicated nodes are connected to each other:

1. If an input variable node and an output variable node are in the column, a directed edge is created from the former to the latter. If a query condition is set in this column, the initial indispensable node flag of the output variable node is set.

2. If the output variable node is a conversion node, and if a query condition is set as well, a new predicate node which has a predicate selecting tuples satisfying the query condition is created. A directed edge is created from this node to the output variable node.

3. If the input variable node is a predicate node or a conversion node, and if the output variable node is not specified, and if, in addition, a constant value is specified as a query condition, the constant value is held in the input variable node as the input.

Example 4.5

When we look at the column of the variable $N in the node connection table,

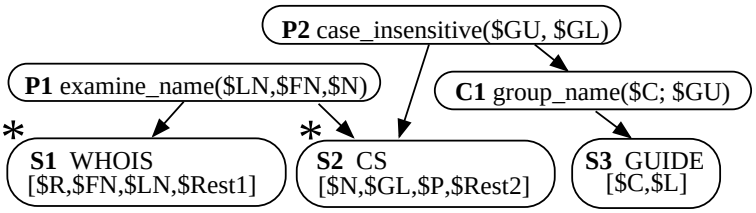


Figure 4.5. The initial expression graph.

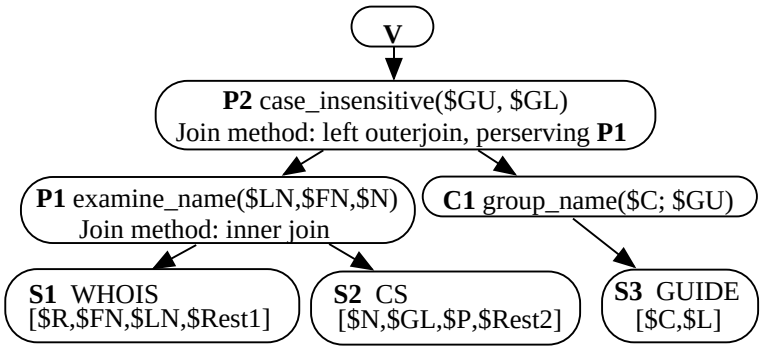


Figure 4.6. The final expression tree.

Table 4.2, we find that $N is output from node S2 and input to node P1, so a directed edge is created from P1 to S2. Thus as a result of processing of every column, the initial expression graph is created as shown in Figure 4.5. The mark "$*$" at nodes S1 and S2 indicates that the initial indispensable flags of these nodes are set. □

Conversion into an Expression Tree

The initial expression graph shown in Figure 4.5 is converted into an expression tree shown in Figure 4.6. Here, V represents a view node. This conversion method is described in Chapter 3, and it is not described here. A union node is finally

created and directed edges are connected to the root node of each expression tree[1] . In the above mentioned processing, an appropriate join method is selected from the alternatives of inner join and left/right/full outerjoin at each join node according to the algorithm shown in Chapter 3 and is specified as an attribute of the node.

4.4 Execution of Query Processing

Query processing starts from the root in the expression tree, and a depth first search is executed. Then, beginning with the leaf node of a tree structure finally reached, the processing described below, corresponding to the kind of node, is executed at each node in reverse order of arrival in the depth first search. In the source node, the variable table with information obtained from the information source is passed to the predecessor node of the directed edge; the conversion and predicate nodes receive variable tables one by one from the successor nodes of the directed edges, and the variable table with new information added to it is passed to the predecessor node of the directed edge. The query processing when the mediator defined in **RMS1** receives Query (Q1) is shown in Figure 4.7.

The query processing for each type of the nodes is as follows:

1. Source node. This node executes the query processing in the following order.

   [1] Creation of a subquery. A subquery in OEM-QL to issue to the wrapper of the information source is created based on the extraction template. Starting from the topmost subobject of the extraction template,

[1] If a query matches multiple rules of the RMSL, one expression tree is created per rule.

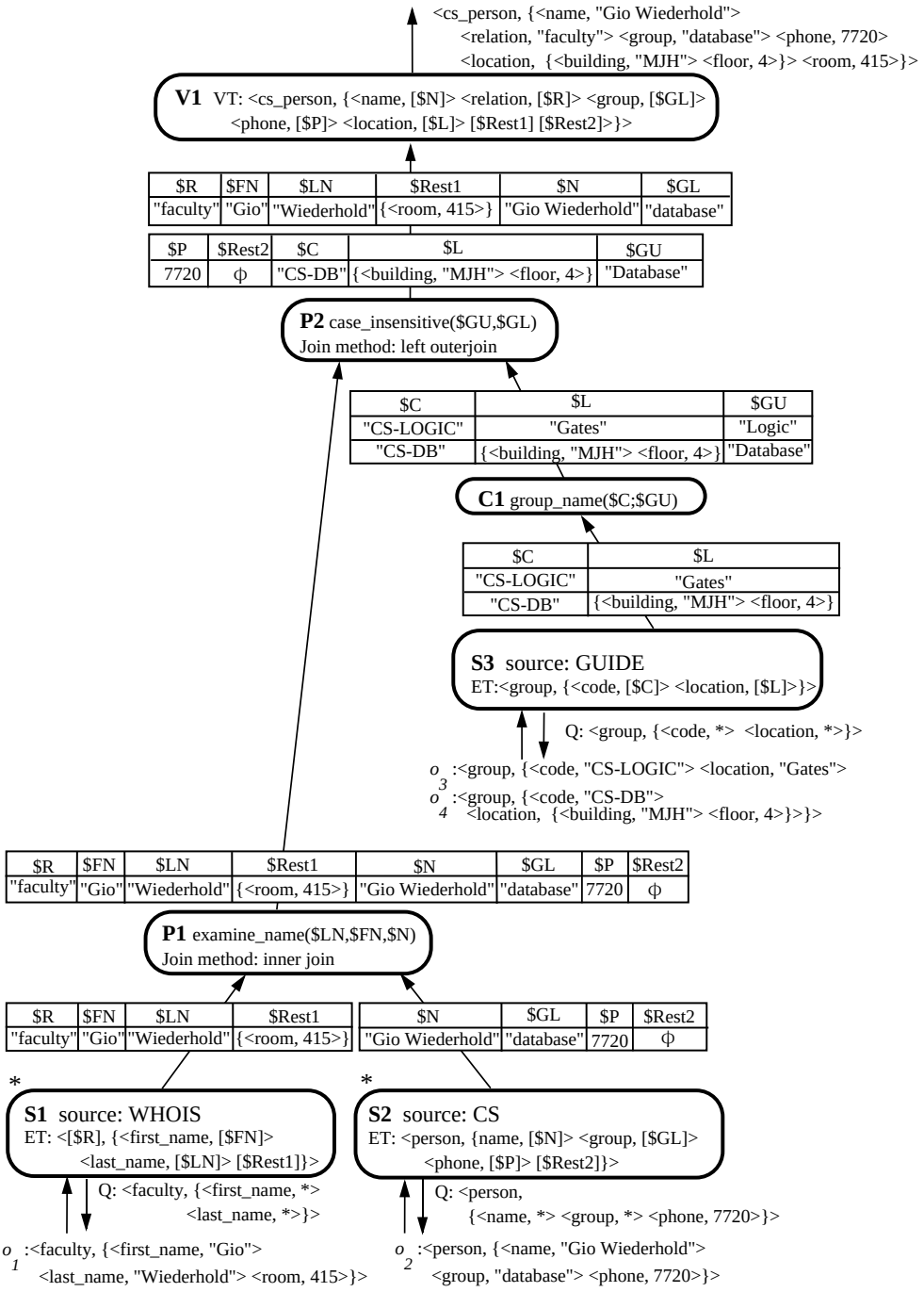


Figure 4.7. A trace of the query processing by the mediator. In the figure, ET stands for extraction template, VT for view template, and Q for subquery.

a depth first search is executed, then a label, a value, and various symbols are output in order in every newly reached subobject, forming an OEM-QL subquery. If an index indicating the column of the variable table (except the Rest variable) is set as the label or the value of the extraction template, the corresponding column in the node connection table is looked up. If any query condition is set in the "query condition" tuple, the condition is written as the label or the value of the subquery; if not, "$*$" is written.

Example 4.6

The following subquery is created for Extraction template (E1) in Section 4.3.1.

<faculty, {<first_name, *> <last_name, *> }>

□

[2] Issuance of subquery. A subquery is issued to the wrapper of the information source and the resulting OEM object is obtained[2] .

[3] Extraction of object elements from the OEM objects. An empty variable table is created. The topmost subobject of each OEM object from a retrieval result and the topmost subobject of the extraction template are collated. The conditions under which the subobject of the OEM object <olabel, ovalue> and the subobject of the extraction template <elabel, evalue> match are as follows:

[a] elabel matches olabel or is a variable, and furthermore,

[2] The set of the OEM objects obtained as a result are returned as the value of the subobject whose label is "Answer."

[b] evalue is an index to a variable table, or, if evalue is a set of subobjects, the element w $\in$ ovalue matched by e exists for each element e $\in$ evalue.

In this collating process, if any index to a variable table is set as a label or a value of a subobject of the extraction template, the object element corresponding to it is extracted from the OEM object and written to a variable table. One tuple of a variable table is created for each OEM object obtained as a result. The variable table thus forms the output of this node.

In source node **S3** in Figure 4.7, "Gates" of an atomic value is bound to the variable $L for the object with identity o_3, while the set of subobjects {<building, "MJH"> <floor, 4>} is bound to $L for the object with identity o_4. Like this, it is possible to bind any kind of object element to the variable, so that it is possible to process flexibly data whose structure differ from instance to instance.

2. Predicate node. A predicate node may have one or two successor nodes. In the case of one node, this node executes a selection; in the case of two nodes, a join. In the case of the selection, the object elements in the columns corresponding to the input variables are obtained per tuple from the variable table entering this node, and processing of the predicate is executed. If the processing result of the predicate is "false," the tuple is deleted from the variable table. The variable table obtained as a result is passed to the predecessor node. If this node is a join, a total of two variable tables enter one by one from each successor node. Thereupon, in all combinations of each tuple of the two variable tables, the predicate is applied to the object elements in the columns corresponding to the input variables, and the tuples

are joined if the predicate returns "true." The join method complies with the attribute (inner join, left/right/full outerjoin) specified in this predicate node. If any object element to be input to the predicate is missing, the predicate returns "false." The variable table thus joined is passed to the predecessor node.

3. Conversion node. The object elements in the columns corresponding to the input variables are obtained per tuple from the variable table entering this node, and the conversion processing is executed by the function. The conversion outputs are added to the same tuple in the columns which correspond to the output variables. As a result, the variable table is passed to the predecessor node.

4. View node. The OEM object is constructed according to the view template, per tuple of the entering variable table. That is to say, a depth first search is executed from the topmost subobject of the view template and a subobject of the OEM object is formed every time a subobject of the view template passes through. If a constant value or label is set for the view template, the label or the value is set to the subobject that is formed. If an index to a variable table column is specified for the view template, the object element pointed to by the index is obtained and set to the subobject.

   Unnecessary subobjects are excluded from the OEM object in accordance with the flag [s] set in the subobject of the view template when compared with the query object. In other words, an OEM object destined to become a view is created at the instant when the specified subobject with "?" passes. If the value of the view template is a set of subobjects, a subobject specified with "−" is excluded from the retrieval object of the depth first search. If any element of the set of subobjects is specified with "+", subobjects not

specified with "+" among the subobjects with the same level are excluded from the depth first search.

5. Union node. The OEM object of the retrieval result is passed to this node from each view node. At the time of execution, an OEM subobject with label "Answer" is created and the set of the OEM objects passed from each view node is set as the value.

The following retrieval result is produced by the above processing:

o_5:<Answer, { <cs_person, {<name, "Gio Wiederhold">
<relation, "faculty"> <group, "database"> <phone, 7720>
<location, {<building, "MJH"> <floor, 4>}>
<room, 415>}> (A1)

4.5 Implementation

The mediator and the wrapper that execute the query processing proposed in this chapter were implemented with programming language C++. As will be discussed in Section 4.6, the mediator and the wrapper were actually implemented as a single module. The source program size was approximately 19,500 lines, excluding the comments. In addition, the source program for specific information source is required for the module to work as a wrapper. For example, for a subset of ODBC (Open DataBase Connectivity) interface, an additional source program was implemented, requiring approximately 700 lines of the source code.

Using the implemented module, the execution time of the mediator was measured. The RMSL used for the measurement consisted of 5 source descriptions and one conversion description. The source descriptions corresponded to 5 information sources consisting of 3 Microsoft Jet databases (accessed via ODBC

interface), one flat file, and one on-memory associative array. The operating system of the computer was Windows 95, and the computer was equipped with a CPU of Pentium MMX of 233 MHz clock speed and a memory of 80 MB (Mega Bytes). The total CPU time for a query processing that collected a single OEM object from each wrapper in order and joined them to yield a single resultant OEM object was 523 milliseconds. During this period of time, 502 milliseconds were spent for the query processing of the Jet databases. The mediator specific part took the execution time of 19.1 milliseconds.

Thus the execution time at the mediator is significantly short compared with the data retrieval time at the information sources, provided a small number of objects are selected out of a large number of objects at the information sources.

4.6 Discussion

Query processing to offer a view of semistructured data requires a mechanism different from the ones for relational databases and object oriented databases. This chapter proposed the following items to realize the above:

1. A query processing method was proposed for integrating information sources by a mediator based on a self-descriptive object model. We specify with RMSL how to create the view at the mediator. In RMSL, a normal variable is bound to the label of an OEM object, an atomic value, or a set of subobjects. The Rest variable is bound to a set of subobjects. This mechanism enables us to integrate flexibly semistructured data obtained from heterogeneous, distributed information sources, to make a view freely and to issue queries to it.

2. The following mechanisms were proposed as a concrete implementation

method for dealing with OEM objects.

   (a) Extract object elements of any of the label of an OEM object, an atomic value, or a set of subobjects, obtained from each information source (wrappers or other mediators) by means of the extraction template and make a variable table.

   (b) Freely compose an OEM object to be used as a view according to the view template, by using the object elements obtained from the information sources and stored in the tuples of the variable table.

   (c) Make a subquery in OEM-QL, according to the extraction template.

3. The query language OEM-QL was proposed which can specify the subobjects to be obtained in detail. This language, with its high consistency with the RMSL, enables the system to return only necessary subobjects as a retrieval result by simple processing.

4. For forming a query processing plan, 5 kinds of nodes were defined as basic units: a source node, a predicate node, a conversion node, a view node, and a union node. For the present RMSL, it is possible to process all queries by using only these nodes.

In this chapter, RMSL was proposed as a revision to MSL in [PGMU96]. The reasons for the revision are as follows:

1. In MSL no distinction is made in the rule section between predicate returning "true" or "false" and function for the conversion of attributes, and they are described collectively as external predicates. Separately from the rule section, in the function declaring section the distinction between predicate and function is made by specifying 'bound' or 'free' for each variable.

Therefore, it is possible to assign a predicate and multiple functions to the same external predicate in the rule section. One of the predicate or the functions is then selected and used for the query processing, but the algorithm to determine which is to be used is not clarified. The function is used not only for join processing, but also for converting the attributes obtained from information sources and presenting them in another form in the view. In the latter case, it is impossible to implement the external predicate as the predicate. Thus it is appropriate to distinguish predicates and functions explicitly in the rule section.

2. In MSL, the descriptions of the tail in the rule section are all combined with "AND's," indicating implicitly the inner join as the join operator. This is, however, inappropriate for the case where tables are joined with the outerjoin, as adopted in this dissertation. Thus "AND's" are omitted in RMSL.

The descriptions for making a view with RMSL enables users of the mediator to retrieve desired data from multiple information sources by knowing only the view specified in the head. In the universal relation interface [Ull89] using a relational database, users can cast queries to an integrated universal relation, without any knowledge of the underlying, multiple relations that constitute the universal relation. In the universal relation, a total projection is generated in advance at a binding stage and the query to this is processed at an evaluation stage. This, however, is different from our approach in the following respects. First, in the universal relation it is intended that the representative instance window function is found automatically in the binding stage, for which the soundness of the processing order in the outerjoin becomes a problem. In this context, only when the hypergraph composed of nodes corresponding to information sources is

connected and is γ–acyclic, a sound outerjoin ordering exists [RU96] as discussed in Chapter 3. In this chapter it is premised that when a database administrator specifies RMSL for creating the view, the hypergraph is prevented from having γ–cycles or pure cycles. Second, with regard to the universal relation, all data are expressed as attributes of the single universal relation. The flat structure of the universal relation tends to cause collision of attribute names. In contrast, it is easy to avoid the problem of collision of attribute names in this proposed method, since they are expressed as the structured OEM objects. Third, in this method, unlike the universal relation interface, it is possible to define freely the functions for converting the object elements.

As described in Section 4.2.1, the mediators and wrappers have different roles: the mediators are responsible for integration of multiple information sources and for extraction of knowledge; the wrappers are responsible for concealing the difference of interface to native information sources from the mediators. In reality, however, the mediator and the wrapper share a considerable amount of the same functionalities: both of them accept the same query language from their clients, parse the query, construct OEM objects as a query result, and send the query result through the network back to the clients. Moreover, it is convenient for a mediator to have the ability to obtain data from the native, local information source that resides at the same site as the mediator. Hence in the implementation described above, a single software module plays the role of both the mediator and the wrapper. Also RMSL is extended to specify a native, local information source in the tail of the rule instead of a remote wrapper or a mediator. The drawback of such implementation is that it produces a larger executable module and requires a larger memory size, but this drawback is insignificant since we can increase the physical memory size with small cost due to the decrease of memory price. By combining the mediator and the wrapper, we can improve maintainability of the

common source codes for the mediator and the wrapper. We can also achieve more efficient query processing when the mediator and the native information source reside at the same site, by eliminating the communication between the mediator and the wrapper.

The developed mediator has been deployed for the integration of the map information and the ledger data in the facility ledger management system in the waterworks and working in a number of local autonomies in Japan for years. In order to integrate information sources by using the mediator, it is necessary to make a wrapper for every information source and to describe the RMSL. For a simple wrapper, the additional source code to create will be less than 1000 lines. Although some skill is needed to describe the RMSL, the designations of an RMSL to integrate about 3 information sources will require only several lines due to its declarative nature. It is thus unnecessary to develop a program for data integration processing in a conventional programming language, which greatly shortens the development of applications.

4.7 Related Work

The object of the schema integration consists in resolving the heterogeneity of the multiple information sources, integrating them, and offering a view required by users. Factors of heterogeneity include the difference of data models, the difference of schemas, the difference of vocabularies, the difference of types/accuracy/units of the data, evolution of a schema or data structure, and the difference of data definition/manipulation languages. It is known that the relational data model and its view definition language are insufficient for resolving such heterogeneity, even if all the information sources are based on the relational data model [KLK91]. The object oriented data model is used as a framework to

integrate the information sources in many projects [ASD+91, KCGS93, HM93, AB91, KFM96, BNS96, HB96, Ber91, LW94]. By using the object oriented data model, it is possible to define the method to manipulate data in order to conceal the difference of the data structure and the schema. On the other hand, a proposal with regard to the description of data integration using a predicate logic is made in order to describe facts and query processing succinctly and formally [CWN94, LRO96].

As mentioned above, many studies of schema integration exist and various systems have been proposed, but in comparison there are few publications describing the query processing methods for these systems. The query processing method for the integration by the relational data model has been studied as an extension of the query processing of a centralized relational database [Ull89, BOT86]. In addition, concerning the object oriented data model, it is common to use the relational data model as an intermediate form [DS96, LW94]. Regarding the query processing for integration by the predicate logic, algorithms for logic programming have been proposed which are different from the general query processing method of the relational database [CWN94, LRO96]. However, there is no literature that can deal with self-descriptive object model, which can flexibly integrate semistructured data, as proposed in this chapter.

4.8 Summary

In this chapter, with the object of integrating distributed, heterogeneous information sources with semistructured data, the query processing method was proposed based on the self-descriptive OEM object model. In this method the mediator formulates a query processing plan based on the RMSL and executes it. It was shown to be possible to process flexibly semistructured data whose structure

varies from instance to instance and to form OEM objects as a view.

In this method, strong constraints are imposed on the execution order of the query processing units, due to the conversion and the outerjoin operators used in the integration. Optimization after satisfying this constraint is a future research issue.

Chapter 5

Mediating Temporal Heterogeneity of Periodically Generated Data Sequences

5.1 Introduction

There is a wide variety of the real time systems and data broadcasting systems that deal with *periodically generated data sequences* (hereafter, called **pgds**'s for short). In this chapter, the problem of integrating **pgds**'s with different periodic intervals is addressed [MYU00, MYU99]. This is a novel type of heterogeneity among information sources and did not appear in the literature in the past. The systems that make use of **pgds**'s include power generation plants, steel and chemical plants, sewage systems, plane, rocket, and satellite control systems, and Intelligent Transport Systems (ITS).

Let us consider the case of ITS. ITS is a broad range of concepts which aim at achieving various traffic systems by integrating information sources includ-

ing sensors. The data to integrate include the traffic congestion levels on roads, road surface conditions, conditions of a driving vehicle and its neighboring vehicles, obstacles on the road, and the weather. Among such systems is a collision avoidance system, aiming at detecting obstacles and accidents on the road and stopping the incoming vehicles. As sensors to realize such a system, the following with respective data generation periodic intervals can be listed:

1. A camera with an image processing unit that detects obstacles and vehicles on the road. (Periodic intervals of 33 milliseconds.)

2. A scan laser radar that detects obstacles on the road. (Periodic intervals of 100 milliseconds.)

3. A laser radar that distinguishes the road condition among "dry", "wet", "frozen", and "snow-covered." (Periodic intervals of 120 milliseconds.)

As in the above case, when **pgds**'s with different periodic intervals are given, it is often required to obtain a data combination which approximates a *snapshot*, or a set of the data generated at the same point of time.

There are alternatives to the strategy of choosing the data combinations that are the approximations of the snapshots, but the alternatives do not always work. We consider two alternatives. The first alternative is to synchronize all the **pgds**'s: the periodic intervals of all the **pgds**'s are fixed at multiples of a fundamental period. This would be a good policy for a relatively small system if the whole system is designed and implemented at a time. This, however, becomes increasingly difficult as the system becomes more complex: some information sources are designed to generate data at specific intervals, making it impossible to synchronize all the **pgds**'s to the uniform periodic intervals; also if we are to integrate sub-systems designed on the basis of different periodic intervals, it is

inevitable to use the data generated at different periodic intervals. In addition, suppose that sensors with shorter periodic intervals are developed due to the technology advancement. If we are to incorporate such **pgds**, we also have to use the data generated at different periodic intervals. Thus synchronizing all the **pgds**'s does not necessarily work, especially for complex systems. The second alternative is to predict the data values at a specific point of time by interpolation or extrapolation. Such prediction does not provide accurate values, however, if the data values change rapidly compared with the periodic intervals of the data generation. There are also cases where discrete integer values (or bit sequences) are assigned to the states, for example, of devices and alarms. In such cases interpolation or extrapolation does not make sense. Thus prediction does not work unless the data values change continuously and gradually compared with the data generation periodic intervals.

For determining the optimum data combination of the data each obtained from different **pgds**, novel criteria are presented in this chapter: freshness and synchronousness of the data combinations. In integrating **pgds**'s, we have to choose which data combinations to use. First of all, it is important that the data are *fresh*, i.e., up-to-date. Second, the data combinations to use should be *synchronous*, i.e., the difference of the data generation times should be small. In reality, we need to find the optimum combination of freshness and synchronousness, since freshness and synchronousness are, in general, in the relation of a trade-off: on one hand, two sets of data may be generated almost simultaneously, but they can be too old; on the other hand, two sets of data may be generated relatively recently, but at too distant time points away.

The nature of **pgds** is that the data are generated continually. Thus it is a natural requirement to monitor and process the data repeatedly. In the above example of the collision avoidance system, suppose the case where the graphic

panel in the traffic control center displays the data obtained from the sensors. Here, a question arises: when should the system acquire the data and process them? One of the policies for the system is to report the data if they are generated at the same time, i.e., at the least common multiple (L.C.M.) of all the data intervals. This policy, however, is not always a good idea. First, there are cases where the data generation time points do not meet forever. For example, no data of two **pgds**'s with periodic intervals 10 and 20 are generated at the same time if they start at time $t = 0$ and $t = 5$. Second, L.C.M. may be too long, especially when there are many, different intervals. Another policy is to report to the user whenever new data are generated. This policy also has problems. First, some data may be generated at time points too far away from the other data. Second, there may be too many points of time when new data are generated, so that it may be beyond the capability of the CPU. This is likely to happen especially when there are many **pgds**'s with different intervals. In order to address such problems, a framework for *consecutive queries* is proposed based on an evaluation function which takes into account both freshness and synchronousness of the data. Also another framework for *interactive queries* is proposed as an extension to the conventional query: in this framework, the user issues a query with two parameters; one for restricting synchronousness of the data, and the other for restricting the wait time before the query result is returned. The latter parameter enables *forced wait*, which increases the probability of obtaining the query result based on fresher and more synchronous data.

We can also consider a different setting that requires to select optimum data combinations continually. Suppose that mobile clients acquire data via an "expensive," wireless network from servers that broadcast data at periodic intervals different from one another. If data communication charge is imposed based on the number of network connection times, it is appropriate for the mobile client

to acquire all the data of a data combination in a single connection, provided each server keeps its latest data until the next data is generated. If data communication charge is imposed based on the number of packets sent through the network or on the length of the network connection time, it would be appropriate to establish a mediator, for instance, at an office directly connected to the servers via an inexpensive network. The mediator receives the necessary data from the servers, and when all the data in the necessary data combination are acquired, then, the mediator extracts only the necessary part of the data and compresses them. The data, thus compressed, can either be delivered to the mobile clients by "server push" or "client pull".

Our focus is on acquisition of data from data combinations, and selection of data based on their values is beyond the scope of this study.

The rest of the chapter is organized as follows: In Section 5.2, examples are shown to give the motivation for choosing the appropriate data combinations of **pgds**'s. In Section 5.3, new criteria for choosing appropriate data combinations are proposed: freshness and synchronousness of the data. In Section 5.4, some properties of two **pgds**'s are revealed, with a focus on synchronousness. The properties give basic guidelines for designing data acquisition systems. In Section 5.5, upperbounds of synchronousness for two different conditions are determined. In Section 5.6, frameworks for two kinds of queries are proposed: consecutive queries and interactive queries. These queries take into account both freshness and synchronousness of the data. In Section 5.7, the effectiveness of the proposed framework is evaluated by applying it to the collision avoidance system. In Section 5.8, extensions to our framework are proposed for the cases where the generation time of certain data should be fixed time points prior to the other data. Also the applicability of the consecutive and interactive query frameworks to different evaluation functions is discussed. Section 5.9 describes related work. Finally 5.10

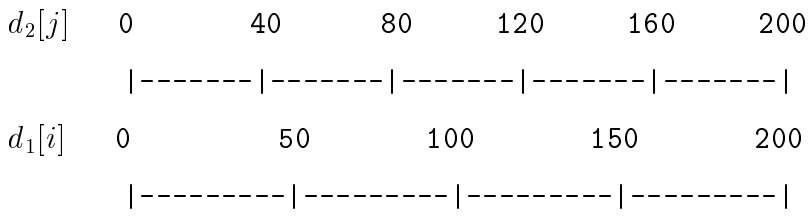


Figure 5.1. **pgds**'s d_1 and d_2 within the L.C.M., where $\tau_1 = 50$ and $\tau_2 = 40$.

summarizes the chapter.

5.2 Motivating Examples

Figure 5.1 represents data generation time points of two **pgds**'s. In **pgds** d_1, data are generated every 50 time points and in d_2, every 40 time points. This diagram shows the data generation time points within their L.C.M., which is 200. Also data sequences d_1 and d_2 start *in phase*, i.e., at the same time. Assume that a user issues a query which requires the data combination of d_1 and d_2. In this chapter, the processing time of the query is ignored based on the assumption that the time units are relatively longer than the processing time. Thus the user acquires the query result immediately.

First of all, we consider interactive queries. Suppose that a user issues a query at time $t = 0$ and receives the query result using the data combination of d_1 and d_2 generated at $t = 0$. In this case, both of the data are fresh and synchronous. Next, suppose that the user issues another query at time $t = 155$. At this time, we can consider using two combinations of data: $[t_1, t_2] = [100, 120]$ and $[t_1, t_2] = [150, 120]$, where t_1 and t_2 indicate the generation times of d_1 and d_2. If we use data combination $[100, 120]$, the two sets of data are generated at the distance of 20 time points and 55 time points have passed since the generation

of the older data. If we use data combination $[150, 120]$, the two sets of data are generated at the distance of 30 time points and 35 time points have passed since the generation of the older data. If we put priority on synchronousness of the data combination, it is appropriate to use $[100, 120]$, but if we put priority on freshness, $[150, 120]$ is more appropriate. In some cases, however, the difference between the time points over 25 is intolerable: in such cases, we have no choice but to use $[100, 120]$. Another policy is to wait for 5 time points. Then, data combination $[150, 160]$ is available at time 160. In this case, the two sets of data are generated at the distance of 10 time points, and 10 time points will have passed since the generation of the older data. This data combination is better in terms of both freshness and synchronousness.

Second, we consider consecutive queries. Remember the case of displaying the sensor data on the graphic panel at the traffic control center. If the periodic interval of displaying the query results should be shorter than the L.C.M. of 200 time points, we should use the data combinations even if the data generation time points do not meet exactly. If the difference of the data generation time points up to 10 is allowable, we can use the data combinations $[50, 40]$ and $[150, 160]$, in addition to $[0, 0]$. In this case, the query result based on data combination $[50, 40]$ is displayed until $t = 160$, when time passage since the generation of the older data at $t = 40$ is 120 time points. In some cases, the passage of time points 120 is not tolerable, and it is appropriate to use the data combination of $[100, 80]$ and $[100, 120]$, even though the difference of time points is over 10.

In order to incorporate the criteria of both freshness and synchronousness of the data, we define an evaluation function. The evaluation function can be used for both the consecutive and interactive queries.

5.3 Freshness and Synchronousness

We consider in this section dealing with two or more **pgds**'s. When such data sequences are given, we need criteria in order to determine which data combination to use. The criteria can be given in the form of an evaluation function, and we claim that the evaluation function be dependent on both freshness and synchronousness of the data combination. In this chapter we assume that the value of the evaluation function is equal to or greater than zero and that the smaller the value of the evaluation function is, the better it is. Under such assumptions, the value should be smaller when the data are *fresher* and more *synchronous*. It makes sense to restrict the domain of the evaluation function to the data combination whose data were acquired at most L.C.M. time points prior to the current time. The form of such evaluation function, of course, depends on each application, and we can think of various kinds of the form. In this chapter, we propose an example of the evaluation function that is applicable to most applications.

First, we define formally freshness and synchronousness of the data combination. Then, we give an example of the evaluation function that depends on the freshness and synchronousness.

5.3.1 Basic Definitions

In this chapter except Section 5.5, for simplicity we regard time as a discrete sequence of points with order and of the same interval, and assign to each point a non-negative integer in order, starting at 0.

Definition 5.1 (Periodically generated data sequence)
Periodically generated data sequence (**pgds**) d_i is defined as:

$$d_i = \langle d_i[0], d_i[1], \ldots \rangle.$$

Data generation time point $t_i[k]$ for each data $d_i[k]$ for $k = 0, 1, \ldots$ is denoted by

$$t_i[k] = t_i[0] + k \times \tau_i,$$

where τ_i is the integer that denotes the periodic, constant interval of d_i. k is called a sequence index to $t_i[k]$. Periodic time sequence t_i is defined as:

$$t_i = \langle t_i[0], t_i[1], \ldots \rangle.$$

□

Definition 5.2 (Sequence index function)
$I_i(t)$ at time point t is defined as

$$I_i(t) = \lfloor (t - t_i[0])/\tau_i \rfloor,$$

where $\lfloor real \rfloor$ is the greatest integer less than or equal to *real* value. □

Note that $I_i(t_i[k]) = k$. The following notations are also used for succinctness:

- $\tau_{i,j}^{LCM}$ is the least common multiple (L.C.M.) of τ_i and τ_j;
- $\tau_{i,j}^{GCD}$ is the greatest common divisor (G.C.D.) of τ_i and τ_j.

Definition 5.3 (Data combination)
A data combination is represented by C, which is an n-dimension array whose i-th element $C[i]$ is sequence index j to the data $d_i[j]$ in **pgds** d_i for $i = 1, 2, \ldots, n$. That is,

$$C = [C[1], C[2], \ldots, C[n]].$$

□

Strictly speaking, C is an array, and its elements identify a data combination, but the symbol C is also used to indicate the data combination that it identifies, as long as there is no ambiguity.

5.3.2 Definitions of Freshness and Synchronousness

The definitions of freshness and synchronousness are described in this section.

Definition 5.4 (Freshness)

Freshness, $E^{fresh}(C,t)$, of data combination C available at time t is defined as

$$E^{fresh}(C,t) = t - \min_{i=1,2,\ldots,n}(t_i[C[i]]). \tag{5.1}$$

□

Depending on applications, it makes sense to set up a limit to the freshness. This can be achieved by modifying $E^{fresh}(C,t)$, so that it returns a huge value when the time passage from its generation is beyond the limit.

Definition 5.5 (Synchronousness)

Synchronousness, $E^{sync}(C)$, of data combination C is defined as

$$E^{sync}(C) = \max_{i=1,2,\ldots,n}(t_i[C[i]]) - \min_{j=1,2,\ldots,n}(t_j[C[j]]). \tag{5.2}$$

□

It also makes sense to set up a limit to the synchronousness. This can be achieved by modifying $E^{sync}(C)$, so that it return a huge value when the time difference between the newest and oldest data is beyond the limit.

5.3.3 Evaluation Function

In this section, a typical example of the evaluation functions is described. Even though we can consider a variety of evaluation functions as criteria for selecting a data combination, the evaluation function described here is simple and can be used by most applications as demonstrated in Section 5.7.

Definition 5.6 (Evaluation function)

Evaluation function, $E(C,t)$, of data combination C at time t is defined as

$$E(C,t) = E^{fresh}(C,t) + w \times E^{sync}(C),$$

where w is a non-negative real number, representing a weight for $E^{sync}(C)$. □

Definition 5.7 (Optimum data combination)

Optimum data combination, $C^{opt}(t)$, at time t is defined as the data combination that is available at time t and yields the minimum value of $E(C,t)$.

□

Definition 5.8 (Optimum evaluation value)

Optimum evaluation value, $E^{opt}(t)$, at time t is given by

$$E^{opt}(t) = E(C^{opt}(t), t).$$

□

5.4 Basic Properties

In this section, basic properties on two **pdgs**'s are described.

Let τ_1 and τ_2 be periodic intervals of two **pgds**'s d_1 and d_2. τ_1 and τ_2 can be expressed by

$$\tau_1 = \tau_{1,2}^{GCD} \times \tau_1',$$

$$\tau_2 = \tau_{1,2}^{GCD} \times \tau_2',$$

where τ_1' and τ_2' are the integers that are relatively prime. Observe that

$$\tau_{1,2}^{LCM} \times \tau_{1,2}^{GCD} = \tau_1 \times \tau_2, \tag{5.3}$$

$$\begin{aligned}\tau_{1,2}^{LCM} &= \tau_{1,2}^{GCD} \times \tau_1' \times \tau_2' \\ &= \tau_1 \times \tau_2' \\ &= \tau_2 \times \tau_1'.\end{aligned}$$

Definition 5.9 (Residue)

Residue $r_1[k]$ for $t_1[k]$ in terms of τ_2 is defined to be

$$r_1[k] = t_1[k] - \lfloor t_1[k]/\tau_2 \rfloor \times \tau_2.$$

□

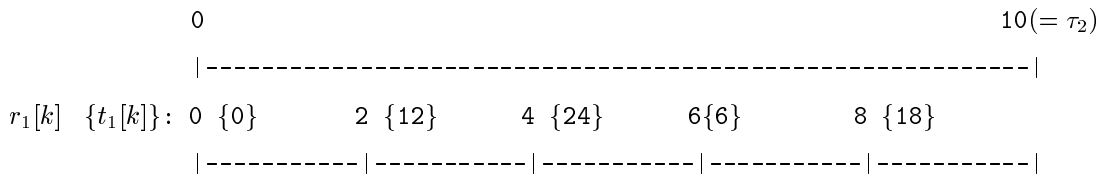
Observe that, for $t_1[0] = t_2[0] = 0$,

$$\min_i |t_2[i] - t_1[j]| = \min(r_1[j],\ \tau_2 - r_1[j]).$$

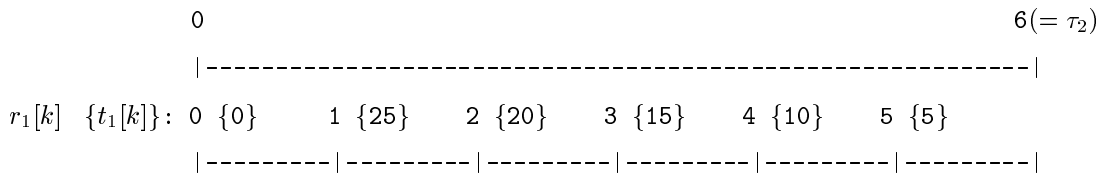
The meaning of this equation is as follows: we pick up arbitrary data $d_1[j]$ in **pgds** d_1. This data is generated at $t = t_1[j]$. Then, from **pgds** d_2, we pick up data $d_2[i]$ generated at such time $t = t_2[i]$ which is closest to $t_1[j]$. The distance between $t_2[i]$ and $t_1[j]$ is equal to the smaller of $r_1[j]$ or $(\tau_2 - r_1[j])$.

Figure 5.2 (a) shows residue $r_1[k]$ for **pgds** d_1 within 30, the L.C.M. of $\tau_1 = 6$ and $\tau_2 = 10$. The distance between two neighboring residues $r_1[0] = 0$ corresponding to $t_1[0] = 0$ (represented by $0\{0\}$ in Figure 5.2) and $r_1[2] = 2$ corresponding to $t_1[2] = 12$ (represented by $2\{12\}$) is 2. Also the distance between two neighboring residues $r_1[2] = 2$ corresponding to $t_1[2] = 12$ and $r_1[4] = 4$ corresponding to $t_1[4] = 24$ is 2. Like this, the distance between two neighboring residues $r_1[i]$ and $r_1[j]$ between which no other residues exist[1] is a constant integer. This observation is formalized by the following theorem.

[1] It does not hold, in general, that $i = j \pm 1$.



(a) When $\tau_1 = 6$, $\tau_2 = 10$, and $\tau_2' = 5$.



(b) When $\tau_1 = 5$, $\tau_2 = 6$, and $\tau_2' = 6$.

Figure 5.2. The distribution of $r_1[k]$'s, each corresponding to $t_1[k]$ between zero and the L.C.M.

Theorem 5.1

Let $D_{i,j}$ be the distance of neighboring residues $r_1[i]$ and $r_1[j]$ ($i \neq j$ and $|t_1[i] - t_1[j]| < \tau_{1,2}^{LCM}$) such that no other residues exist in between. Then, it holds that

$$D_{i,j} = \tau_{1,2}^{GCD}.$$

Proof: Let p be the number of the occurrences of d_1 generated within $\tau_{1,2}^{LCM}$. Here, we omit the last data if the data of d_1 are generated both at the beginning and at the end of time interval $\tau_{1,2}^{LCM}$. Then, we have

$$p = \frac{\tau_{1,2}^{LCM}}{\tau_1}. \tag{5.4}$$

Distance D between arbitrary two residues $r_1[i]$ and $r_1[j]$ is,

$$\begin{aligned} D &= |r_1[i] - r_1[j]| \\ &= |(t_1[i] - \lfloor t_1[i]/\tau_2 \rfloor \times \tau_2) - (t_1[j] - \lfloor t_1[j]/\tau_2 \rfloor \times \tau_2)| \\ &= |t_1[0] + i \times \tau_1 - \lfloor t_1[i]/\tau_2 \rfloor \times \tau_2 - t_1[0] - j \times \tau_1 + \lfloor t_1[j]/\tau_2 \rfloor \times \tau_2| \\ &= |i \times \tau_1 - \lfloor t_1[i]/\tau_2 \rfloor \times \tau_2 - j \times \tau_1 + \lfloor t_1[j]/\tau_2 \rfloor \times \tau_2| \\ &= \tau_{1,2}^{GCD} \times |i \times \tau_1' - \lfloor t_1[i]/\tau_2 \rfloor \times \tau_2' - j \times \tau_1' + \lfloor t_1[j]/\tau_2 \rfloor \times \tau_2'|. \end{aligned} \tag{5.5}$$

Thus D is always a multiple of $\tau_{1,2}^{GCD}$. Note that D is not equal to zero since $i \neq j$ and $|t_1[i] - t_1[j]| < \tau_{1,2}^{LCM}$ from our assumption. Equations (5.4) and (5.5) indicate that the distance between any two neighboring points of residues $r_1[i]$ and $r_1[j]$ is always $\tau_{1,2}^{GCD}$. This is obtained by proving the hypothesis is false that distances of n $(2 \leq n \leq p)$ pairs of neighboring residues less than τ_2 are greater than or equal to $\tau_{1,2}^{GCD} \times 2$ and that the distances of the other $(p-n)$ pairs are $\tau_{1,2}^{GCD}$. Assuming that the hypothesis is true, we have

$$\begin{aligned} \tau_2 &\geq (p-n) \times \tau_{1,2}^{GCD} + n \times (2 \times \tau_{1,2}^{GCD}) \\ &= p \times \tau_{1,2}^{GCD} + n \times \tau_{1,2}^{GCD} \\ &= \frac{\tau_{1,2}^{LCM}}{\tau_1} \times \tau_{1,2}^{GCD} + n \times \tau_{1,2}^{GCD} \text{ (from Equation (5.4))} \\ &= \frac{\tau_{1,2}^{LCM} \times \tau_{1,2}^{GCD}}{\tau_1} + n \times \tau_{1,2}^{GCD} \\ &= \frac{\tau_1 \times \tau_2}{\tau_1} + n \times \tau_{1,2}^{GCD} \text{ (from Equation (5.3))} \\ &= \tau_2 + n \times \tau_{1,2}^{GCD}. \end{aligned}$$

Obviously this is incorrect since $\tau_2 + n \times \tau_{1,2}^{GCD}$ is actually greater than τ_2. It is now proved that the hypothesis is false, and that the distance of the neighboring points of residues $r_1[i]$ and $r_1[j]$ is no greater than $\tau_{1,2}^{GCD}$. Remember that D is a multiple of $\tau_{1,2}^{GCD}$. This indicates that the distance between the neighboring points of residues $r_1[i]$ and $r_1[j]$ is exactly $\tau_{1,2}^{GCD}$. □

Definition 5.10 (Synchronousness summation)

Synchronousness summation, $E_{sum}^{sync}(d_1, d_2)$, is defined as

$$E_{sum}^{sync}(d_1, d_2) = \sum_{j=m,m+1,\ldots,m+\tau_2'-1} \min_i |t_2[i] - t_1[j]|,$$

where m is an integer greater than zero and $t_2[0] - t_1[m] \leq \tau_2/2$. □

This indicates that for each $t_1[j]$ (corresponding to d_1 of the first argument) within the period of $\tau_{1,2}^{LCM}$, we pick up its nearest neighbor $t_2[i]$, and sum up the distances between each pair. The smaller $E_{sum}^{sync}(d_1, d_2)$ is, the more "synchronous" d_1 and d_2 are.

For $t_1[0] = t_2[0] = 0$, this definition is reduced to:

$$E_{sum}^{sync}(d_1, d_2) = \sum_{j=0,\ldots,\tau_2'-1} \min_i |t_2[i] - t_1[j]|.$$

Definition 5.11 (Synchronousness average)

Synchronousness average, $E_{ave}^{sync}(d_1, d_2)$, is defined as

$$E_{ave}^{sync}(d_1, d_2) = E_{sum}^{sync}(d_1, d_2)/\tau_2'. \tag{5.6}$$

□

This indicates that for each $t_1[j]$ (corresponding to d_1 of the first argument) within the period of $\tau_{1,2}^{LCM}$, we pick up its nearest neighbor $t_2[i]$, and take an average of the distances between each pair.

Note that for $t_1[0] = t_2[0] = 0$, the distribution of $r_1[k]$ between zero and τ_2 is symmetric to the center. Also there exists a center point at $\tau_2/2$ if τ_2' is an even number, as in Figure 5.2 (b). If τ_2' is an odd number, as in Figure 5.2 (a), there is no center point. The following observation is derived from this:

Observation 5.1

For $t_1[0] = t_2[0] = 0$, if τ_2' is an even number,

$$\begin{aligned} E_{sum}^{sync}(d_1, d_2) &= 2 \times \sum_{k=1,\ldots,\tau_2'/2-1} (k \times \tau_{1,2}^{GCD}) + \frac{\tau_2'}{2} \times \tau_{1,2}^{GCD} \\ &= 2 \times \frac{\{1 + \tau_2'/2 - 1\} \times (\tau_2'/2 - 1)}{2} \times \tau_{1,2}^{GCD} + \frac{\tau_1'}{2} \times \tau_{1,2}^{GCD} \\ &= \frac{\tau_2'(\tau_2' - 2)}{4} \times \tau_{1,2}^{GCD} + \frac{\tau_2'}{2} \times \tau_{1,2}^{GCD} \\ &= \frac{(\tau_2')^2}{4} \times \tau_{1,2}^{GCD}, \end{aligned} \tag{5.7}$$

and if τ_2' is an odd number,

$$\begin{aligned} E_{sum}^{sync}(d_1, d_2) &= 2 \times \sum_{k=1,\ldots,(\tau_2'-1)/2} (k \times \tau_{1,2}^{GCD}) \\ &= 2 \times \frac{\{1 + (\tau_2' - 1)/2\} \times (\tau_2' - 1)/2}{2} \times \tau_{1,2}^{GCD} \\ &= \frac{(\tau_2' + 1)(\tau_2' - 1)}{4} \times \tau_{1,2}^{GCD} \\ &= \frac{(\tau_2')^2 - 1}{4} \times \tau_{1,2}^{GCD}. \end{aligned} \tag{5.8}$$

□

Theorem 5.2

Let d_1 and d_1' be **pgds**'s with the same periodic interval, and let d_2 and d_2' be **pgds**'s with the same periodic interval. Each of **pgds**'s d_1, d_2, d_1', and d_2' may have different initial data generation times. The difference between two synchronousness summations $E_{sum}^{sync}(d_1, d_2)$ and $E_{sum}^{sync}(d_1', d_2')$ is bounded by half the G.C.D. of τ_1 and τ_2. That is,

$$|E_{sum}^{sync}(d_1, d_2) - E_{sum}^{sync}(d_1', d_2')| \leq \tau_{1,2}^{GCD}/2.$$

Proof: We can fix the starting points $t_1[0]$ and $t_2[0]$ at zero, then, move $t_1[0]$ without loss of generality. Since the distance between two neighboring residues

is always $\tau_{1,2}^{GCD}$ as proved in Theorem 5.1 and the distribution of $r_1[k]$ between zero and τ_2 is symmetric to the center, we have to consider moving the starting point $t_1[0]$ only between zero and $\tau_{1,2}^{GCD}/2$. We consider first the case in which τ_2' is an even number. Refer to Figure 5.2 (b). Suppose we move the starting point $t_1[0]$ from zero to n where $0 < n \leq \tau_{1,2}^{GCD}/2$. This causes any point $r_1[k]$ where $0 \leq r_1[k] < \tau_2/2$ to increase the distance from its nearest neighbor, i.e., zero, by n; but for each point $r_1[k]$, there exists corresponding point $r_1[k] + \tau_2/2$ whose distance from its nearest neighbor, i.e., τ_2, decreases by n. Thus the total summation $E_{sum}^{sync}(d_1, d_2)$ of the distances between any point $r_1[k]$ where $0 \leq k < \tau_2'$ and its nearest neighbor remains the same. Now we consider the case in which τ_2' is an odd number. Refer to Figure 5.2 (a). In this case, moving the starting point $t_1[0]$ from zero to n where $0 < n \leq \tau_{1,2}^{GCD}/2$ causes any point $r_1[k]$ where $0 \leq r_1[k] < \tau_{1,2}^{GCD} \times (\tau_2' - 1)/2$ to increase the distance from its nearest neighbor, i.e., zero, by n; but for each point $r_1[k]$, there exists corresponding point $r_1[k]+\tau_{1,2}^{GCD}\times(\tau_2'+1)/2$ whose distance from its nearest neighbor, i.e., τ_2, decreases by n. There exists, however, a residue whose value is $\tau_{1,2}^{GCD} \times (\tau_2' - 1)/2$ and has no corresponding point to complement the increase. Thus the total summation $E_{sum}^{sync}(d_1, d_2)$ increases by n. As mentioned earlier, the maximum number of n to take into account is $\tau_{1,2}^{GCD}/2$. As a consequence, the maximum difference between $E^{sync}(d_1, d_2)$ and $E^{sync}(d_1', d_2')$ is $\tau_{1,2}^{GCD}/2$. □

Interesting properties of two **pgds**'s are derived from the above theorems and observation:

1. Consider the case where τ_2' is an odd number.

   (a) When $t_1[0] = t_2[0] + i \times \tau_{1,2}^{GCD}$ for any integer i, both the minimum and maximum values of synchronousness $E^{sync}(d_1, d_2)$ are smallest: the minimum value is zero and the maximum value is $\tau_{1,2}^{GCD} \times (\tau_2' - 1)/2$.

When $t_1[0] = t_2[0] + \tau_{1,2}^{GCD}/2 + i \times \tau_{1,2}^{GCD}$ for any integer i, both the minimum and maximum values of synchronousness $E^{sync}(d_1, d_2)$ are largest: the minimum value is $\tau_{1,2}^{GCD}/2$ and the maximum value is $\tau_2/2$.

(b) When $t_1[0] = t_2[0] + i \times \tau_{1,2}^{GCD}$ for any integer i, $E_{sum}^{sync}(d_1, d_2)$ is smallest, and the value is given by Equation (5.8). When $t_1[0] = t_2[0] + \tau_{1,2}^{GCD}/2 + i \times \tau_{1,2}^{GCD}$ for any integer i, $E_{sum}^{sync}(d_1, d_2)$ is maximum and the value can be calculated from Equation (5.8) as:

$$\begin{aligned} E_{sum}^{sync}(d_1, d_2) &= \frac{(\tau_1')^2 - 1}{4} \times \tau_{1,2}^{GCD} + \tau_{1,2}^{GCD}/2 \\ &= \frac{(\tau_1')^2 + 1}{4} \times \tau_{1,2}^{GCD}. \end{aligned}$$

2. Consider the case where τ_2' is an even number.

(a) When $t_1[0] = t_2[0] + i \times \tau_{1,2}^{GCD}$ for any integer i, the minimum value of synchronousness $E^{sync}(d_1, d_2)$ is smallest and the maximum value is largest: the minimum value is zero and the maximum value is $\tau_2/2$. When $t_1[0] = t_2[0] + \tau_{1,2}^{GCD}/2 + i \times \tau_{1,2}^{GCD}/2$ for any integer i, the minimum value of synchronousness $E^{sync}(d_1, d_2)$ is largest and the maximum value is smallest: the minimum value is $\tau_{1,2}^{GCD}/2$ and the maximum value is $\tau_2/2 - \tau_{1,2}^{GCD}/2$.

(b) $E_{sum}^{sync}(d_1, d_2)$ is constant and does not depend on the initial data generation times. The value is given by Equation (5.7).

The above properties give guidelines in designing data acquisition systems that deal with **pgds**'s of different periodic intervals. For example, if it is possible to determine the initial data generation times for two **pgds**'s τ_1 and τ_2 where τ_2' is an odd number, then the maximum synchronousness can be minimized by setting $\tau_1[0] = \tau_2[0]$.

5.5 Upper Bounds of Synchronousness

For building practical, hence reliable, systems, it is important to guarantee the upper bound of the *synchronousness* of the data combinations that compose an approximation of the snapshot.

In the following section, examples are shown to illustrate that synchronousness of data combinations depends on the initial data generation times. Then, in Section 5.5.2, two theorems are given to determine the upper bounds for the cases where all the **pgds**'s have the same periodic interval and where **pgds**'s have different periodic intervals. The theorems are applicable not only to discrete time sequences but also to continuous time.

5.5.1 Examples

Figure 5.3 shows data generation times of **pgds**'s d_1, d_2, and d_3 with periodic intervals 150, 120, and 100, respectively. In this section, we denote the data generated in **pgds** d_i at time t as $d_i(t)$ for succinctness. In Figure 5.3, the data combination consisting of $d_1(150)$, $d_2(240)$, and $d_3(200)$ has the synchronous of 90, which is the generation time difference of the first data $d_1(150)$ and the last data $d_2(240)$. Given **pgds**'s, we focus on the problem of finding an upper bound of synchronousness such that we can always acquire periodically the data combinations whose synchronousness is below the upper bound. In this section, we choose the largest periodic interval of the **pgds**'s as the period for acquiring the data combinations. We assume that every element of the data combination to acquire should be generated after the previous data combination was returned to the user. In our running example, our goal is to find an upper bound of the synchronousness that we can achieve every 150 time units, which is the largest among the periodic intervals of d_1, d_2, and d_3.

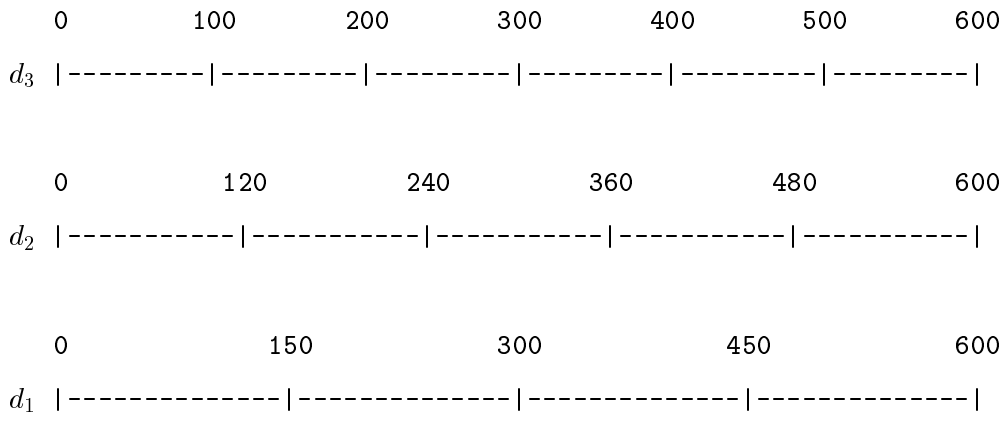


Figure 5.3. Data generation times of **pgds**'s d_1, d_2, and d_3 within the L.C.M., 600.

In the case of Figure 5.3, the synchronousness is upper-bounded by 60, and this can be achieved if we start acquiring a data combination at time $t = 50$. This fact can be found through a careful examination of the **pgds**'s. It is easily shown that the synchronousness is upper-bounded by 60 when we focus on the data $d_1(300)$. The data combination including this data should include the data either $d_2(240)$ or $d_2(360)$. In either case, the distance from the data $d_1(300)$ is 60 time units. Regarding d_3, there is no problem since data $d_3(300)$ is generated simultaneously with $d_1(300)$. If we start acquiring data combinations at time $t = 0$, a problem occurs at time $t = 450$: the data combination consisting of $d_1(300)$, $d_2(240)$, and $d_3(300)$ was acquired at time $t = 300$, and no new data combinations exist that produce synchronousness below 60 together with $d_1(450)$. Even if we wait until time $t = 480$, still there exist no data combinations that produce synchronousness below 60 together with the data $d_1(450)$. Only at time $t = 500$, the data combination is acquired that consists of $d_1(450)$, $d_2(480)$, and $d_3(500)$, producing the synchronousness value of 50. Hence, the initial time for

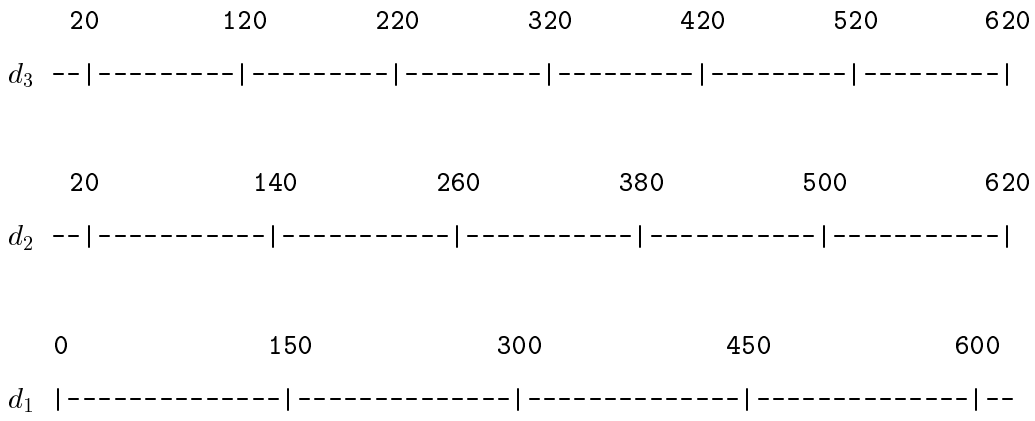


Figure 5.4. Data generation times of **pgds**'s d_1, d_2, and d_3 with different generation times of the initial data.

acquiring data combinations should be no earlier than time $t = 50$, which makes it possible to acquire the data combination at time $t = 500$ or later. It can be easily verified that the synchronousness value of 60 or less is always achieved if we start acquiring a data combination at time $t = 50$.

Now we examine another example shown in Figure 5.4. In this case, the synchronousness is upper-bounded by 70, and this can be achieved if we start acquiring a data combination at time $t = 20$. If we focus on $d_1(450)$, the data combination that produces the minimum synchronous consists of either $d_2(380)$ and $d_3(420)$ or $d_2(500)$ and $d_3(520)$, in addition to $d_1(450)$. Either the data combination gives the synchronousness value of 70. Also if we start acquiring data combinations earlier than time $t = 20$, no data are available for d_2 or d_3 at that time. It can be verified that the synchronousness value of 70 or less is always achieved if we start acquiring a data combination at time $t = 20$.

As the above two examples illustrate, the achievable synchronousness depends on the initial data generation times. Based on the assumption that the initial

data generation times of autonomous data sources are not controllable but that the periodic intervals of all the **pgds**'s are known, our goal is to give the synchronousness upper bound that does not depend on the initial data generation times.

In the next section, we present the least upper bound for the synchronousness of **pgds**'s with the same periodic interval. Then, we show not always the necessary, but sufficient value of the synchronousness upper bound for **pgds**'s with different periodic intervals.

5.5.2 Upper Bounds

We assume that a system has access to **pgds**'s d_1 through d_n, and receives a request for periodically acquiring synchronous data combinations from its user in the form:

$$Sync\,(d_1,\, d_2, \ldots,\, d_n).$$

In this section, time is regarded as continuous data.

We have the following assumptions:

1. For each **pgds** d_i $(i = 1, 2, \ldots, n)$, periodic interval τ_i is given, but the initial data generation time, i.e. $t_i[0]$, is not known to the system.

2. In response to the data acquisition request from the user, a new data combination should be returned to the user at the constant interval that is the same as the maximum periodic interval of the **pgds**'s.

3. The initial timing to return the data combination to the user is determined by the system arbitrarily.

4. Every element of the data combination to acquire should be generated after the previous data combination was returned to the user.

The problem we address in this section is to find an upper bound of the synchronousness of the data combinations consisting of **pgds**'s d_1 through d_n.

We describe in this section two theorems that give upper bounds of the synchronousness. Theorem 5.3 determines the least upper bound of the synchronousness for **pgds**'s with the same periodic interval. The meaning of the *least* upper bound is that the synchronousness never exceeds this upper bound regardless of the initial data generation times of the **pgds**'s, but that the synchronousness is equal to the upper bound in the worst case. Theorem 5.4 gives an upper bound of the synchronousness for **pgds**'s with different periodic intervals. Thus Theorem 5.4 deals with more general case than Theorem 5.3. In the case of **pgds**'s with the same periodic interval, however, the least upper bound provided by Theorem 5.3 is smaller, hence stricter, than the upper bound provided by Theorem 5.4.

Synchronousness of pgds's with the Same Periodic Interval

First of all, we consider the case where all the **pgds**'s have the same periodic interval.

Given n (≥ 2) **pgds**'s $d_1, d_2, \ldots, d_n$ with the same periodic interval τ and with different data generation times, the following theorem gives the least upper bound of the synchronousness of the data combinations for **pgds**'s $d_1, d_2, \ldots, d_n$.

Theorem 5.3

For any data $d_i[x]$ where $1 \leq x$, $\tau_i = \tau$, and $t_i[0] < \tau$ for $1 \leq i \leq n$, there exists a data combination C such that:

1. $d_i[x]$ is in C; and
2. $E^{sync}(C) \leq \tau(n-1)/n$.

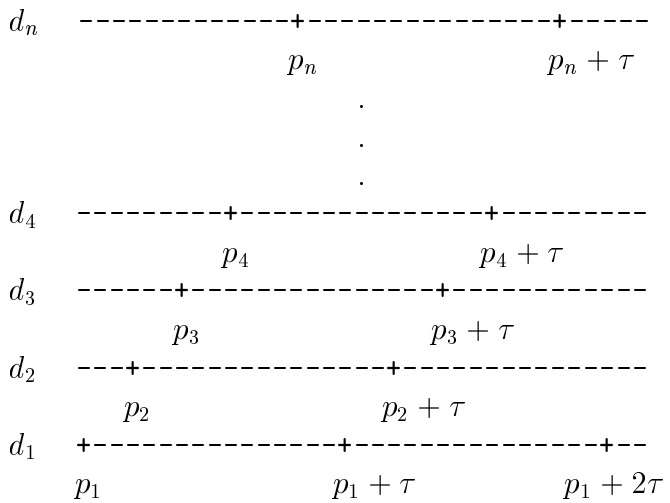


Figure 5.5. The assumption for the proof of Theorem 5.3.

Proof : When n **pgds**'s d_1 through d_n with the same periodic interval τ and with different data generation times are given, we can assume without loss of generality that there exists a set of data such that

$$p_1 < p_2 < p_3 < \cdots < p_n < p_1 + \tau, \tag{5.9}$$

where p_i for $i = 1, 2, 3, \ldots, n$ denotes generation time $t_i[x]$ $(x \geq 0)$ for data $d_i[x]$ in **pgds** d_i. This is illustrated in Figure 5.5. Now we examine the synchronousness of the data combinations that include data $d_1[x+1]$ generated at time $(p_1 + \tau)$. If we include, in addition to $d_1[x+1]$, $d_2[x]$ generated at p_2 as the first data in the data combination, the rest of the data generated at $p_3,\ p_4,\ \ldots,\ p_n$ are guaranteed from Equation (5.9) to be within the interval between p_2 and $p_1 + \tau$. Obviously synchronousness E_1 for this data combination is calculated as

$$E_1 = (p_1 + \tau) - p_2.$$

If we include, in addition to $d_1[x+1]$, $d_3[x]$ generated at p_3 as the first data in the data combination, the rest of the data generated at $p_4,\ p_5,\ \ldots,\ p_n$, and $p_1 + \tau$ are

guaranteed to be within the interval between p_3 and $p_2 + \tau$. For, from Equation (5.9), we have $p_1 + \tau < p_2 + \tau$ and

$$p_3 < p_4 < \cdots < p_n < p_1 + \tau < p_2 + \tau.$$

Obviously synchronousness E_2 for this data combination is calculated as

$$E_2 = (p_2 + \tau) - p_3.$$

In the same way, for the data combination whose first data $d_{i+1}[x]$ is generated at p_{i+1} for $i = 1, 2, 3, \ldots, n-1$, whose last data is generated at $p_i + \tau$, and that includes $d_1[x+1]$, the synchronousness E_i for this data combination is calculated as

$$E_i = (p_i + \tau) - p_{i+1}. \tag{5.10}$$

Finally for the data combination whose first data $d_1[x+1]$ is generated at $p_1 + \tau$, synchronousness E_n for this data combination is calculated as

$$\begin{aligned} E_n &= (p_n + \tau) - (p_1 + \tau) \\ &= p_n - p_1. \end{aligned} \tag{5.11}$$

Now, E_i's for $i = 1, 2, 3, \ldots, n$ cover all the synchronousness of the data combinations[2] that include the data generated at $p_1 + \tau$. From Equations (5.10) and (5.11), by taking the summation of all the E_i's, we have

$$\begin{aligned} \sum_{i=1}^{n} E_i &= \sum_{i=1}^{n-1} (p_i + \tau - p_{i+1}) + (p_n - p_1) \\ &= \tau(n-1). \end{aligned} \tag{5.12}$$

[2] We exclude the data combinations whose synchronousness is greater than the ones obtained by Equations (5.10) and (5.11).

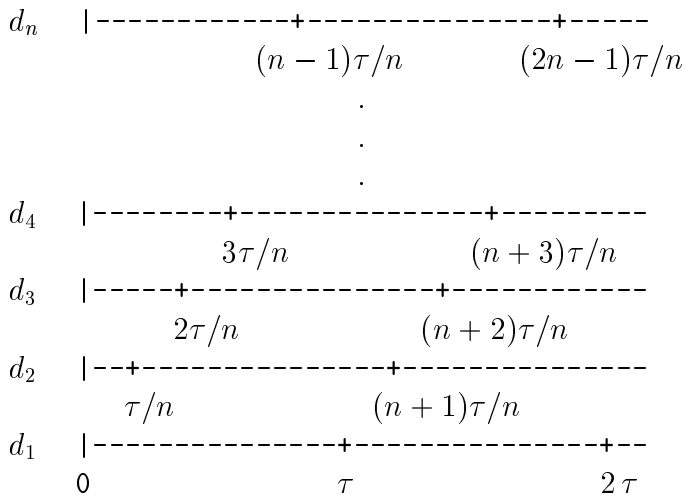


Figure 5.6. An example of the data combination that causes the maximum synchronousness of **pgds**'s with the same periodic interval.

Our goal is to find the least upper bound of the synchronousness E_j for all the data combinations that include the data generated at $p_1 + \tau$. From Equation (5.12), this least upper bound can be obtained when all the E_i's have the same value E_{ave}, the average of E_i's. For, if there exists E_j which is larger than E_{ave}, then, there exists some E_k which is smaller than E_{ave}, and we can choose E_k as the synchronousness of the data combination that includes $d_1[x+1]$. Thus the least upper bound of the synchronousness is E_{ave}, and calculated as

$$E_{ave} = \tau(n-1)/n. \tag{5.13}$$

□

In the proof of Theorem 5.1, for the least upper bound of synchronousness to occur, we can easily calculate from Equations (5.10), (5.11), and (5.13) that

$$
\begin{aligned}
p_n &= p_1 + \tau(n-1)/n \\
p_{n-1} &= p_1 + \tau(n-2)/n \\
&\vdots \\
p_2 &= p_1 + \tau/n.
\end{aligned}
$$

This is illustrated in Figure 5.6 for $p_1 = 0$.

In case some of the n **pgds**'s $d_1 \ldots d_n$ are generated at the same time, we can merge them and reduce accordingly the number n of the **pgds**'s. Then, we obtain **pgds**'s with different data generation times, and we can apply Theorem 5.1.

Synchronousness of pgds's with Different Periodic Intervals

Next, we consider the case where each **pgds** may have different intervals.

Theorem 5.4

Suppose n (≥ 2) **pgds**'s $d_1, d_2, \ldots, d_n$ are given where $\tau_1 \geq \tau_2 \geq \cdots \geq \tau_n$ and $t_i[0] < \tau_1$ for $0 < i \leq n$. For any data $d_1[x_1]$ ($1 \leq x_1$) of a **pgds** d_1, there exists a data combination C such that:

1. $d_1[x_1]$ is in C; and
2. $E^{sync}(C) \leq$

$$
\begin{cases}
\displaystyle\sum_{h=2}^{k-1}(\tau_h/2^{k-h}) & (n \geq 3, \text{ and there exists a minimum integer } k \\
& \quad \text{such that } 3 \leq k \leq n \text{ and } \tau_k \leq \displaystyle\sum_{h=2}^{k-1}(\tau_h/2^{k-h})) \quad (5.14) \\
\displaystyle\sum_{h=2}^{n}(\tau_h/2^{n-h+1}) & (\text{otherwise}). \quad (5.15)
\end{cases}
$$

Proof : First of all, we consider the case $n = 2$, i.e., there are only two **pgds**'s d_1 and d_2. As is easily observed in Figure 5.7(a), the upper bound of the minimum

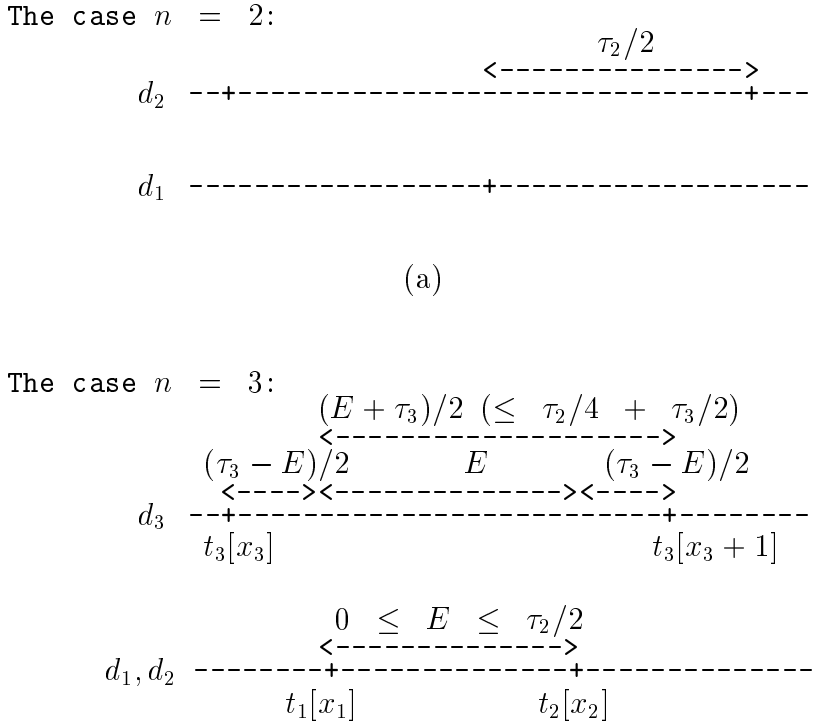


Figure 5.7. Maximum synchronousness of **pgds**'s with different intervals.

difference between the generation times of data $d_1[x_1]$ and data in d_2 is $\tau_2/2$. This upper bound occurs in the case where the generation time of $d_1[x_1]$ is exactly at the middle of the generation times of two consecutive data in d_2.

Next, let us consider the case $n = 3$. If it holds that

$$\tau_3 \leq \tau_2/2 \ (= \sum_{h=2}^{3-1} (\tau_h/2^{3-h})),$$

then, there is at least one data of d_3 in any time period of $\tau_2/2$. Hence, in this case, the maximum synchronousness is

$$\tau_2/2 = \sum_{h=2}^{3-1} (\tau_h/2^{3-h}).$$

Next we consider the opposite case, i.e., $\tau_3 > \tau_2/2$. Let $d_2[x_2]$ be the data in d_2 such that the time difference with $d_1[x_1]$ is minimum. Without loss of generality, we can assume $t_1[x_1] \leq t_2[x_2]$. Hence, $E^{sync}([x_1, x_2])$, the synchronousness of $d_1[x_1]$ and $d_2[x_2]$, is $t_2[x_2] - t_1[x_1]$. From the proof of the case $n = 2$,

$$0 \leq E^{sync}([x_1, x_2]) \leq \tau_2/2. \tag{5.16}$$

Let $d_3[x_3]$ and $d_3[x_3 + 1]$ be two consecutive data in d_3. Then, it follows that $t_3[x_3] < t_3[x_3 + 1]$. The synchronousness of the three data $d_1[x_1]$, $d_2[x_2]$, and $d_3[x_3]$ (or equivalently $d_3[x_3 + 1]$) becomes maximum when the following three conditions hold (see[3] Figure 5.7(b).):

$$\begin{aligned} t_3[x_3] &< t_1[x_1] \\ t_2[x_2] &< t_3[x_3 + 1] \\ t_1[x_1] - t_3[x_3] &= t_3[x_3 + 1] - t_2[x_2]. \end{aligned}$$

In this case, the synchronousness of the three data $d_1[x_1]$, $d_2[x_2]$ and $d_3[x_3]$ (or equivalently $d_3[x_3 + 1]$) is

$$\begin{aligned} & E^{sync}([x_1, x_2]) + (\tau_3 - E^{sync}([x_1, x_2]))/2 \\ &= (E^{sync}([x_1, x_2]) + \tau_3)/2 \\ &\leq (\tau_2/2 + \tau_3)/2 \qquad \text{(from Equation (5.16))} \\ &= \sum_{h=2}^{3} (\tau_h/2^{3-h+1}). \end{aligned}$$

The next step of the proof is the mathematical induction. Here, we assume that the theorem holds for $n = i$ $(3 \leq i)$. Our goal is to prove that the theorem also holds for $n = i + 1$.

[3] In Figure 5.7, we use E to represent $E^{sync}([x_1, x_2])$ for succinctness.

To begin with, we examine the case where Equation (5.14) holds for $n = i$. In this case, $\tau_{i+1} \leq \tau_k$ holds from our assumption, and it holds from the condition in Equation (5.14) that

$$\tau_{i+1} \leq \tau_k \leq \sum_{h=2}^{k-1} (\tau_h / 2^{k-h}).$$

Since τ_{i+1} is the periodic interval of d_{i+1}, there is at least one data of d_{i+1} in any time period starting at time t_s and ending at time t_e such that

$$t_e - t_s = \sum_{h=2}^{k-1} (\tau_h / 2^{k-h}).$$

From Equation (5.14), we can choose t_s and t_e so that every data in C is included between t_s and t_e. Thus the synchronousness of the data combination consisting of $d_1, \ldots, d_n, d_{n+1}$ is less than or equal to $t_e - t_s$. Consequently, Equation (5.14) holds for $n = i + 1$.

Next, we examine the case where Equation (5.15) holds for $n = i$. Analogously with the discussion in the case $n = 3$, let $d_{i+1}[x_{i+1}]$ and $d_{i+1}[x_{i+1} + 1]$ be two consecutive data in d_{i+1}. Then it follows that $t_{i+1}[x_{i+1}] < t_{i+1}[x_{i+1} + 1]$.

First, we consider the case where

$$t_{i+1}[x_{i+1} + 1] - t_{i+1}[x_{i+1}] \leq \sum_{h=2}^{i} (\tau_h / 2^{i-h+1}). \tag{5.17}$$

Since $t_{i+1}[x_{i+1} + 1] - t_{i+1}[x_{i+1}] = \tau_{i+1}$, if we let $k = i + 1$, Equation (5.17) is reduced to:

$$\tau_{i+1} = \tau_k \leq \sum_{h=2}^{k-1} (\tau_h / 2^{k-h}).$$

Thus there is at least one data of d_{i+1} in any time period starting at time t_s and ending at time t_e such that

$$t_e - t_s = \sum_{h=2}^{k-1} (\tau_h / 2^{k-h}).$$

Since Equation (5.15) holds for $n = i$,

$$\begin{aligned} E^{sync}(C) &\leq \sum_{h=2}^{i}(\tau_h/2^{i-h+1}) \\ &= \sum_{h=2}^{k-1}(\tau_h/2^{k-h}). \end{aligned}$$

Thus we can choose t_s and t_e so that every data in C is included between t_s and t_e, and the synchronousness of the data combination consisting of $d_1, \ldots, d_n, d_{n+1}$ is less than or equal to $t_e - t_s$. Consequently, Equation (5.14) holds for $n = i + 1$.

Second, we consider the opposite case where

$$t_{i+1}[x_{i+1}+1] - t_{i+1}[x_{i+1}] > \sum_{h=2}^{i}(\tau_h/2^{i-h+1}).$$

In this case, the synchronousness of the $i+1$ data $d_1[x_1]$, ... , $d_i[x_i]$ and $d_{i+1}[x_{i+1}]$ (or equivalently $d_{i+1}[x_{i+1}+1]$) becomes maximum when the following three conditions hold:

$$\begin{aligned} t_{i+1}[x_{i+1}] &< \min_{1\leq j\leq i} t_j[x_j], \\ \max_{1\leq j\leq i} t_j[x_j] &< t_{i+1}[x_{i+1}+1], \\ \min_{1\leq j\leq i} t_j[x_j] - t_{i+1}[x_{i+1}] &= t_{i+1}[x_{i+1}+1] - \max_{1\leq j\leq i} t_j[x_j]. \end{aligned}$$

Then, the synchronousness of these $i+1$ data is

$$\begin{aligned} & E^{sync}(C) + (\tau_{i+1} - E^{sync}(C))/2 \\ &= (E^{sync}(C) + \tau_{i+1})/2 \\ &\leq (\sum_{h=2}^{i}(\tau_h/2^{i-h+1}) + \tau_{i+1})/2 \quad \text{(from Equation (5.15))} \\ &= \sum_{h=2}^{i+1}(\tau_h/2^{(i+1)-h+1}). \end{aligned}$$

Thus Equation (5.15) holds for $n = i + 1$. □

5.6 Query Processing Frameworks

5.6.1 Architecture

We are now ready to present frameworks for querying information sources that include **pgds**'s. We assume the mediator architecture in [Wie92]. Here, the mediator mediates a new type of heterogeneity among the information sources: difference of the periods for data generation. We assume each information source generates a data sequence d_i at constant intervals of period τ_i, and the wrapper that is established for each information source processes the data and sends the data to the mediator. The mediator receives the data sent from the wrappers and processes the data. When the mediator knows old data are no longer necessary, it may discard them.

We can also consider another setting, as stated earlier. Mobile clients acquire data combinations from the mediator continually via an expensive, wireless network. In such a case, the mediator can extract only the necessary part of the data and compress them. For example, the mediator can extract only the temperature value of the local area from an HTML home page of a weather report, discarding all the image data and unnecessary text data, such as tags. The mediator can also compress large data, using some data compression algorithm. The data can either be delivered to the mobile clients by "server push", or "client pull". By "server push", the server initiates the communication with its clients, whereas by "client pull", the client initiates the communication with its servers.

5.6.2 Consecutive Query

Once a mediator receives a *consecutive query* from an application program, the mediator processes the query and returns the result whenever a new, optimum

Table 5.1. An example of the index table.

number	t_1	t_2
1	9	9
2	13	12
3	17	15
4	17	18

data combination is generated and received, until the mediator receives from the application program the command to stop the query processing. In order for the mediator to determine the time points that give the new, optimum data combination, two strategies are proposed: a static strategy and a dynamic strategy. Although both of the strategies result in choosing the same data combinations, they have merits and demerits as discussed at the end of this section. In what follows, the two strategies are described in detail.

Static Strategy

In the static strategy, the time points that give the new, optimum data combinations within the time period of the L.C.M. of all the periodic intervals are calculated and stored in an *index table* as shown in Figure 5.1. The index table identifies the generation times of each data in the data combination to use within the time period of L.C.M. of all the periodic intervals. Once the index table is created, the mediator looks up the index table to obtain the optimum data combinations repeatedly.

Before explaining the algorithm to create the index table, we introduce a new definition for succinctness:

Definition 5.12 (Newest data combination)
The newest data combination at time t, represented by $C^{new}(t)$, is defined to be the data combination that consists of the newest data of each **pgds** available at time t. □

Algorithm 5.1 describes how to create the index table. In this algorithm, first of all, the data combination of the least synchronousness S within the period of the L.C.M. is searched in Step 1, invoking the subroutine described in Algorithm 5.2. Starting from the last data generation time of the data combination thus obtained, an index table is created in Steps 2 and 3. In the index table, each column corresponds to **pgds** d_i, and each tuple identifies the generation times of the data in the optimum data combinations.

Algorithm 5.1 (Creation of index table)
[input]

- The initial data generation time $t_i[0]$ and the periodic interval τ_i for periodic time sequence t_i corresponding to **pgds** d_i, where $0 < i \leq n$.
- The value of weight w.

[output]

- An index table which determines the optimum data combination.

[method]

1. Invoke Algorithm 5.2 to obtain the output, t_b. Set $C^s := C^{new}(t_b)$, create a new tuple in the index table, and store the data generation time t_i of each data d_i in C^s, in the corresponding column of the newly created tuple. Execute $t := t_b + 1$.

2. If no data are newly generated at time t, go to Step 3. If $E(C^{new}(t), t) \leq E(C^s, t)$, then, set $C^s := C^{new}(t)$, create a new tuple in the index table immediately below the existing tuples, and store the data generation time t_i of each data d_i in C^s, in the corresponding column of the newly created tuple.

3. Increment t. If $t < t_b + LCM.$, go to Step 2. □

Algorithm 5.2 (The least synchronousness generation time)
[input]

- The initial data generation time $t_i[0]$ and the periodic interval τ_i for periodic time sequence t_i corresponding to **pgds** d_i, where $0 < i \leq n$.

[output]

- The time when the first data combination with the least synchronousness is available.

[method]

1. Let t_s be the earliest time when every initial data of all the **pgds**'s is available. That is, execute:

$$t_s := \max_i t_i[0].$$

Let LCM be the L.C.M. of all the τ_i's. Set $t := t_s$. Let C^s be $C^{new}(t_s)$ and let E^{sync}_{min} be the synchronousness $E^{sync}(C^s)$. If $E^{sync}_{min} = 0$, set $t_b := t_s$ and go to Step 4.

2. Increment t. If no data are newly generated at time t, go to Step 3. If $E^{sync}(C^{new}(t)) < E^{sync}_{min}$, then set $C^s := C^{new}(t)$, $t_b := t$, and $E^{sync}_{min} := E^{sync}(C^s(t))$. If $E^{sync}_{min} = 0$, set $t_b := t_s$ and go to Step 4.

3. If $t < t_s + LCM - 1$, go to Step 2.

4. Output t_b. □

The index table in Table 5.1 is for **pgds**'s d_1 and d_2 shown in Figure 5.8, where $t_1[0] = 5$, $t_2[0] = 3$, $\tau_1 = 4$, $\tau_2 = 3$, and $w = 2$. In this case, $LCM = 12$, $t_s = 5$, and $t_b = 9$.

In the index table, we number the tuples from the top down, starting from one, as shown in Table 5.1.

Once the index table is created, the mediator processes the consecutive query repeatedly according to the following algorithm.

Algorithm 5.3 (Consecutive query processing based on index table)
[input]

- A set of data sequence d_i, where $0 < i \leq n$.
- A consecutive query issued by the application program.
- The index table created by Algorithm 5.1.
- The values of t_b and LCM obtained at the end of Algorithm 5.1.

[output]

- The query results for a consecutive query issued at time t, where $t \geq t_b$.

[method]

1. Let t be the current time. Let k be a variable which identifies the tuple in the index table. Set $k := 1$.

2. Execute $r := t - \lfloor (t - t_b)/LCM \rfloor \times LCM$. Let k be the greatest integer such that the maximum value among t_i's in the k-th tuple in the index table is an integer less than or equal to r.

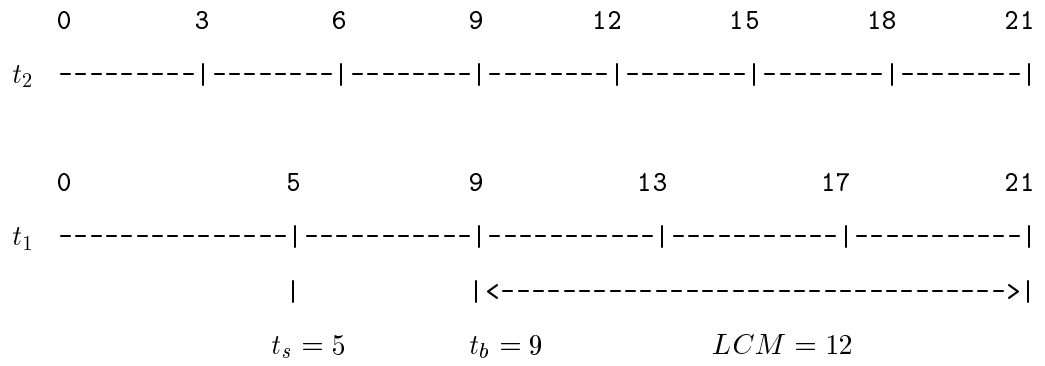


Figure 5.8. Data generation times of **pgds**'s d_1 and d_2, where $t_1[0] = 5$, $t_2[0] = 3$, $\tau_1 = 4$, and $\tau_2 = 3$.

3. For each t_i in the k-th tuple, obtain the data d_i generated at time $\lfloor (t - t_b)/LCM \rfloor \times LCM + t_i$. Process the query using the data combination thus obtained and return the query result to the application program.

4. If k identifies the bottom tuple of the index table, execute $k := 1$ and $r := r - LCM$; otherwise increment k.

5. Wait until the current time t is incremented as time passes by. Increment r. While waiting, if the mediator receives the command to stop the consecutive query processing, terminate this procedure.

6. Find the greatest value among t_i's in the k-th tuple. If r does not agree with the greatest value thus found, go to Step 5.

7. Go to Step 3. □

Dynamic Strategy

In the dynamic strategy, the mediator always keeps the *current* optimum data combination. Whenever a new data is generated at time t, the mediator calculates the evaluation value of the data combination consisting of the newest data of each **pgds**. The mediator also recomputes the evaluation value at time t of the previous optimum data combination. Then, the mediator selects the data combination that yields the smaller evaluation value out of the two data combinations. Thus the mediator determines the *current* optimum data combination at time t, processes the query, and returns the result to the application program. The rigorous algorithm is shown as follows.

Algorithm 5.4 (Dynamic consecutive query processing)
[input]

- A set of data sequence d_i, where $0 < i \leq n$.
- A consecutive query issued by the application program.
- The value of weight w.

[output]

- Consecutive query results based on the optimum data combinations.

[method]

1. Let C^{opt} be a variable that keeps the current optimum data combination. Let t be the time when the consecutive query is issued to the mediator. If $C^{new}(t)$ is not available at time t, wait until it is available. Set $C^{opt} := C^{new}(t)$. Then, process the query using C^{opt}, and return the query result to the application program.

2. Wait until the current time t is incremented as time passes by. While waiting, if the mediator receives the command to stop the consecutive query processing, terminate this procedure. Repeat this step if no data are newly generated.

3. If $E(C^{new}(t), t) \leq E(C^{opt}, t)$, set $C^{opt} := C^{new}(t)$, process the query using C^{opt}, and return the query result to the application program. Go to Step 2. □

In this section, the static and dynamic strategies for obtaining the optimum data combinations were proposed. In the static strategy, once the index table is created, the optimum data combinations are determined by a single table lookup and simple mathematical operations as described in Algorithm 5.3. This minimizes the overhead for determining the optimum data combinations while processing the consecutive query repeatedly. Thus the static strategy is suitable for the applications that need to save the CPU time for the query processing. In case the data generation times fluctuate as time passes by, it is possible to cope with the situation by recomputing the index table. In contrast, the dynamic strategy has the advantage that the index table is unnecessary and it requires less memory size. In addition, this strategy is more robust in that it does not depend on the fluctuation of data generation times. Thus depending on the nature of the applications, we have to choose the appropriate strategy.

5.6.3 Interactive Query

In this section, we extend the conventional queries to incorporate the notion of freshness and synchronousness of the data.

The extended query has parameters ε and δ, in addition to the conventional query, such as SQL. Parameter ε indicates the upper limit of synchronousness

E^{sync} of the data combination, whereas parameter δ indicates the upper limit of the time points that the application can wait until it obtains the query result. Thus a query from the application program to the mediator is given in the form:

$$Q(q, \varepsilon, \delta),$$

where q is a conventional query.

In the situation where the mediator can accept a query $Q(q, \varepsilon, \delta)$, the mediator creates an *index structure* in advance. When the query is actually issued, the mediator can quickly choose the optimum data combination for use in the query processing by referencing to the index structure.

The algorithm for creating the index structure is described first. An example of the index structure is shown in Figure 5.9. The index structure consists of the following tables: a time index table, synchronousness tables, and a data combination table. As Figure 5.9 indicates, the time index table has a pair of a "time"

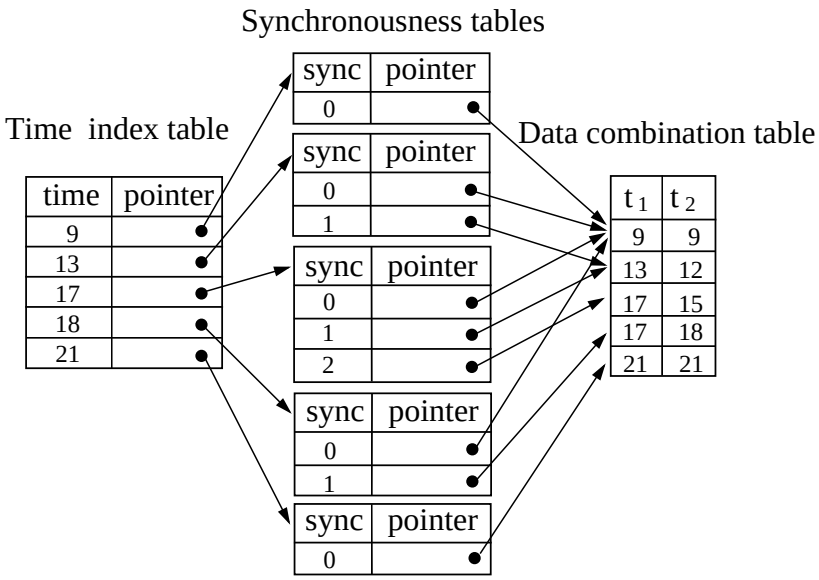


Figure 5.9. An example of the index structure.

column to identify the new data generation times and a "pointer" column to store the pointer to synchronousness tables. Each synchronousness table has a pair of a "sync" column to store the synchronousness of available data combinations and a pointer to a tuple in the data combination table. In each synchronousness table, the tuples are sorted in the order of synchronousness; moreover, the data combination pointed to by a tuple in the synchronous table always yields a smaller evaluation function value than the data combinations pointed to by the tuples above it. Each tuple in the data combination table contains the times that identify generation times of each data in a data combination. First, let us describe the algorithm to create the index structure.

Algorithm 5.5 (Creation of index structure)
[input]

- The initial data generation time $t_i[0]$ and the periodic interval τ_i for periodic time sequence t_i corresponding to **pgds** d_i, where $0 < i \leq n$.
- The value of weight w.

[output]

- An index structure which determines the optimum data combinations.

[method]

1. Let LCM be the L.C.M. of all the τ_i's. Invoke Algorithm 5.2 to obtain the output t_b, the time when the first data combination with the least synchronousness is available. Set $t := t_b$.
2. Create a new tuple in the data combination table and store the data generation time t_i $(1 \leq i \leq n)$ for $C^{new}(t_b)$. Create a synchronousness table and

store the pair of $E^{sync}(C^{new}(t_b))$ and a pointer to the newly created tuple in the data combination table. Create a new tuple in the time index table and store the pair of t_b and a pointer to the synchronousness table just created. Let C^{opt} be $C^{new}(t_b)$.

3. Increment t. If $t = t_b + LCM + 1$, terminate this procedure. If no data are newly generated at time t, repeat this step. If $E(C^{new}(t), t) > E(C^{opt}, t)$, repeat this step.

4. Create a new tuple in the data combination table and store $C^{new}(t)$. Create a new, empty synchronousness table. Copy the tuples of the lastly created synchronousness table to the newly created synchronousness table, except for the tuples whose synchronousness is greater than or equal to $E^{sync}(C^{new}(t))$. At the bottom of the newly created synchronousness table, store the pair of $E^{sync}(C^{new}(t))$ and a pointer to the newly created tuple of the data combination table. At the bottom of the time index table, store the pair of t and a pointer to the newly created synchronousness table. Let C^{opt} be $C^{new}(t)$. Go to Step 3. □

Once the index structure is created, a query $Q(q, \varepsilon, \delta)$ issued to the mediator is processed by the following algorithm.

Algorithm 5.6 (Interactive query processing)
[input]

- Query $Q(q, \varepsilon, \delta)$.
- The index structure.
- The values of LCM and t_b obtained at the end of Algorithm 5.5.

[output]

- The query processing result, or a null value.

[method]

1. Let t be the current time.

2. If $t > t_b$, go to step 3. If $t + \delta < t_b$, then return a null value and terminate this procedure; otherwise wait until time t_b, process the query using the data combination $C^{new}(t_b)$, return the query result, and terminate this procedure.

3. Execute $r := t - \lfloor (t - t_b)/LCM \rfloor \times LCM$. If $r + \delta < t_b + LCM$, then execute $r_{max} := r + \delta$; otherwise execute $r_{max} := t_b + LCM$.

4. In the "time" column of the time index table, find the greatest integer less than or equal to r. Let r_c be the integer value thus found. Follow the pointer in the same tuple of the integer thus found to obtain a synchronousness table. In "sync" column of the synchronousness table thus obtained, search for the greatest synchronousness less than or equal to ε. If such synchronousness is not found, return a null value and terminate this procedure; otherwise follow the pointer in the same tuple that contains the synchronousness thus found to obtain the tuple in the data combination table. Let the data combination identified by the tuple thus obtained be C^{past}.

5. Execute $s := \varepsilon + 1$ and let C^{future} be null. Set $r_f := r_c$.

6. In the "time" column of the index table, find the smallest integer greater than r_f. Reset r_f to the integer thus found. If $r_f > r_{max}$, go to Step 7. Follow the pointer to obtain a synchronousness table. If s is less than or

equal to the value in the "sync" column of the bottom tuple, repeat this step; otherwise set s to the value in the "sync" column of the bottom tuple and follow the pointer in the bottom tuple to obtain the tuple in the data combination table. Let the data combination thus obtained be C^{future}. If $s = 0$ or $r_f = r_{max}$, go to Step 7; otherwise repeat this step.

7. If C^{future} is null or $E(C^{past}, r) \leq E(C^{future}, r_f)$, let C^{opt} be C^{past}; otherwise let C^{opt} be C^{future}.

8. As soon as every data of data combination C^{opt} is obtained, calculate the actual generation time for each data d_i in the optimum data combination as $\lfloor (t - t_b)/LCM \rfloor \times LCM + t_i$, where t_i is the time stored in the data combination table's tuple that contains C^{opt}. Process the query, output the query result, and terminate this procedure. □

In order to determine the values of ε and δ, it is necessary to provide the administrator with some software tool, just as for determining the value of w. The upper bounds of ε are given by Theorems 5.3 and 5.4. Thus the user or the administrator can determine the appropriate value of ε less than or equal to the upper bound, depending on the purposes.

5.7 Application Example

In this section, we apply the consecutive query to a concrete example of ITS, and verify that the proposed framework brings about advantages.

A typical case where the data acquisition based on freshness and synchronousness of data combinations is useful is decision making that requires a wide variety of information. We examine here the case of collision avoidance system, mentioned in Section 5.1.

Various sensors are being developed for use in collision avoidance systems that monitor the road conditions, detect obstacles and accidents, and avoid successive accidents by stopping the incoming vehicles. Each of such sensors has merits and demerits in terms of robustness against the environments, processing time, and accuracy, as pointed out in [SKHS96]. Thus it is important to make use of multiple sensors, complementing the demerits of one another. Such technique of *sensor fusion* is meant to achieve high sensing capability that would not be realized with a single sensor.

As a sensor to detect obstacles and accidents on roads, a device with a movie camera and an image processing device in a single box is being developed. By processing the movie data obtained by the camera within the device, not the huge image data but the compact processing result can be transmitted via a network to the traffic center, reducing the network traffic drastically. Most commercial movie cameras in Japan and U.S.A. comply with the NTSC standard, and take 29.97 frames per second. This is equivalent to about 33 milliseconds per frame, hence the output from the device is fixed at intervals of 33 milliseconds or their multiples.

The problem of the device is that it sometimes takes the shadows of the sun or headlights as obstacles by mistake. This problem can be complemented by detecting the height of the would-be obstacles on the road with a laser radar: shadows are excluded since it has no height. At present, a radar laser whose data output intervals are fixed at 100 milliseconds is available [Aka99].

By fusing the sensing results of the above two sensors, accuracy is expected to improve. In such a case in order for both of the sensors to capture the object at the same time, synchronousness of the data combination to acquire is important. Also in order to notify the incoming vehicles of the obstacles or accidents as soon as possible, freshness of the data combination is important, too.

Table 5.2. Comparison of synchronousness, intervals, and data acquisition times within the L.C.M.

method	weight w	average synchronousness	maximum synchronousness	average interval	maximum interval	the number of acquisition
(a)	-	16	32	100	100	33
(b)	1	9.56	32	60.0	100	55
	5	8.57	24	76.7	100	43
	10	8.36	20	86.8	100	38
	50	8.28	17	97.1	100	34
	100	4.08	16	194	1716	17

Now we examine two methods for determining the data combinations obtained by two sensors:

(a) To choose the newest data every 100 milliseconds, which is the data acquisition intervals of the scan laser radar.

(b) To choose the optimum data combination based on the evaluation function proposed in Section 5.3.3.

First, let us examine method (a). In this method, data acquisition intervals are fixed at 100 milliseconds. The maximum synchronousness is calculated as the smaller periodic interval minus 1, which is 32 milliseconds. The average of the synchronousness is 16 milliseconds. The result is summarized in Table 5.2.

Now we examine method (b). To begin with, we calculate the values related to synchronousness, using the properties described in Section 5.4. Let the **pgds** obtained by the laser radar be d_1, and the one by movie camera be d_2. We have $\tau_1 = 100$ and $\tau_2 = 33$. It follows that $\tau_{1,2}^{GCD} = 1$, $\tau_{1,2}^{LCM} = 3300$, $\tau_1' = 100$, and

$\tau_2' = 33$. For $t_1[0] = t_2[0] = 0$, from Equation (5.8), we have

$$E_{sum}^{sync}(d_1, d_2) = 272.$$

From Equation (5.6), we have

$$E_{ave}^{sync}(d_1, d_2) \approx 8.24.$$

The maximum value of the synchronousness is $\tau_{1,2}^{GCD} \times (\tau_2' - 1)/2 = 16$, since τ_2' is an odd number. Next, based on the consecutive query described in Section 5.6.2, the following values within the L.C.M. are listed in Table 5.2 for different values of parameter w: average synchronousness, maximum synchronousness, average interval, maximum interval, and the number of acquisition.

Table 5.2 indicates that both the average and maximum values of the synchronousness obtained by method (a) are larger as compared with method (b) except for the maximum synchronousness for $w = 1$. In method (b), both of the values decrease as the value of w, the weight for synchronousness, increases. Note that, as the value of w increases, the maximum synchronousness approaches $\tau_{1,2}^{GCD} \times (\tau_2' - 1)/2 = 16$, the value obtained by the property described in Section 5.4. In contrast, the average intervals increase as the value of w increases. The maximum data acquisition intervals are fixed at 100 milliseconds, except for the case where $w = 100$. In practice, it is necessary to examine the table, such as Table 5.2, in determining the synchronousness weight w that is most suitable for the application.

Suppose an obstacle falls down from a vehicle driving at 80 km/h, which is equivalent to 22.2 m/s. The obstacle falls down from the vehicle at this initial speed. Let us choose the value of w as 50. In this case, we can see from Table 5.2 that the maximum synchronousness is 17 ms. During this period of time, the falling obstacle moves approximately 0.377 m. In other word, an obstacle whose

length is above this value can be captured by both the camera and the scan laser radar at the same time.

In contrast, method (a) yields the maximum synchronousness of 32 ms. During this period of time, the falling obstacle moves approximately 0.710 m, which is nearly twice the distance as compared with method (b). Thus by using the proposed method of (b), due to the reduction of the maximum synchronous, both the movie camera and the scan laser radar can capture obstacles which is nearly half the size, and the accuracy of detecting the obstacle improves.

In case an obstacle is captured only by the movie camera but not by the laser radar, the possibility remains that the obstacle is actually a shadow. In such a case, it would be appropriate to give a warning to the incoming vehicles. Even if it is an actual obstacle, the size is guaranteed to be small and it is unlikely to cause a serious accident. On the contrary, in case an obstacle is captured only by the laser radar but not by the movie camera, it is appropriate just to ignore it: the movie camera should capture the obstacle before or after the laser radar captures it.

5.8 Discussion

So far, we assumed that the fresher the data are, the better it is. In some cases, however, this assumption is inappropriate. Consider the case where we measure water flow along a pipe at two geographically distant points A and B. In normal conditions, suppose water that passes by point A reaches point B, say, in 60 time points. Thus we have $d_A[I_A[t-60]] = d_B[I_B[t]]$, where d_A and d_B denotes the water flow at points A and B. In order to check at point B that the water has not leaked since it passed by point A, we need data at point A that were generated 60 time points prior to the current time. We can cope with such cases with

minor modification of the definitions of freshness and synchronousness. Instead of using $t_i[C[i]]$ in Equations (5.1) and (5.2), we just should use $t_i[C[i]] + l_i$ in the case where it is appropriate to use the data generated l_i time points prior to the current time.

In Section 5.6.2, it was pointed out that the the dynamic strategy for the consecutive query is robust and does not depend on the fluctuation of the data generation times. This aspect is of particular importance since it enables us to deal with not only **pgds**'s but also random data sequences. That is, if we redefine d_i as a sequence of data generated randomly, and data combination C as an arbitrary combination of data each obtained from d_i for $1 \leq i \leq n$, then, we can use the freshness of Definition 5.4 and the synchronousness of Definition 5.5 in Section 5.3.2. Thus we can define evaluation functions for the optimum data combinations for random data sequences, just as Definition 5.6 in Section 5.3.3.

The algorithms for both static and dynamic strategies of the consecutive query and for the interactive query can be used not only for the evaluation function defined in Definition 5.6, but also for any evaluation function that has the following properties:

1. For data combinations C and C' and any period of time Δt, if $E(C, t) \leq E(C', t)$, then it holds that $E(C, t + \Delta t) \leq E(C', t + \Delta t)$.

2. Let $S(t)$ be the set of all the data combinations that are available at time t and include at least one data generated at t. Let $C^S(t)$ be any data combination in $S(t)$. Then, it holds that $E(C^{new}(t), t) \leq E(C^S(t), t)$.

5.9 Related Work

Acquiring and processing data sequences have been traditionally studied in the fields of real-time databases and temporal databases. Recently the field of "data broadcast" has been highlighted, and some of the research in this field also deals with periodic data sequences. There exists, however, little research in the literature that addresses the problems in handling freshness of **pgds**'s, and to our knowledge, no research deals with synchronousness of such data sequences.

In the field of real-time database systems, traditionally, main focus has been on transaction processing as in [LL73]; that is, how to keep the deadlines by appropriately assigning priorities on and scheduling tasks. In [AGM88], the authors discuss various task scheduling policies: "First come first serve", "Earliest deadline", "Earliest feasible deadline", "Least slack", and "Greatest value density" policies. In this approach, any data available at hand are used, and it is not considered which data combination to use, being different from our approach. In [XSS+97], the authors compare the performance of "Earliest deadline first" and "Least slack first" policies in combination with *data-deadlines*. According to the notion of data-deadlines, the data have expiration time, and the transaction that updates the data should commit before the data-deadlines. The paper also proposes the concept of *forced wait*: whenever a temporal data object is read by a transaction, the system checks if this transaction will commit before the validity of the data object expires. If the validity could expire before the commit then the transaction is made to wait until the data object is updated. The major criteria for task assignment policies are the transaction abort ratio and CPU utilization. The notion of data-deadlines is analogous to our notion of data freshness, but there are differences. First, data-deadlines are firm deadlines: if the transaction does not finish before the deadline, it aborts. The notion of data freshness, on

the other hand, deals with soft deadlines: the data age as time passes by, and are replaced by a newer data combination when the evaluation function of the newer data combination is smaller than the current data combination. Second, the notion of data-deadlines does not consider the data generation time, i.e., synchronousness among the data. In our approach, we consider the life time of each set of data. We even allow the data combinations whose life time do not agree at all: our framework covers the case where the generation time of one data set should be at certain time points prior to another data set, as discussed in Section 5.8. The work in [NL91] proposes a policy to assign a period for the transactions with data that are updated periodically. For critical data, higher priority is assigned so that the frequency of the transaction execution is closer to the data update frequency. Thus the goal of this work is optimum interval assignment to tasks, and different from our goal.

In the field of temporal databases, emphases have been put on modeling temporal data and querying them; there is little research that handles periodic data. The work in [KSW90] extends relational algebra to handle *a linear repeating point*, which is a set of points of integers of the form $c + kn$, where k and c are constant integers and n takes all values in integers. It extends the traditional relations to *generalized relations* with constraints on the attributes, in order to handle linear repeating points. It also proposes a temporal query language. This approach is different from ours in that the emphasis is on modeling the periodic data and defining the operations on the data. Our focus is on choosing the optimum data combinations. The research in [KO95] also deals with periodic temporal data. It models the data elements of *strictly-periodic* events, as well as *partially-periodic* events, in a uniform framework. It shows that the model is closed under the set theoretic operators. It also extends the object-oriented SQL and shows how the extended query language can express temporal queries. Its emphasis is again on

modeling the periodic data.

Recently various techniques for "data broadcast" have been proposed [FZ98]. In this context, strategies for efficient data broadcasting are studied [AAFZ95, AFZ97, BGH+92, HGLW87, IB94]. No research, however, deals with multiple servers broadcasting data at different periodic intervals, as in our approach.

5.10 Summary

The contributions in this chapter are summarized as follows:

1. New criteria were shown for choosing data combinations from **pgds**'s: freshness and synchronousness of data.

2. Basic property of two **pgds**'s were revealed in Theorem 5.1. Based on this theorem, the concepts of synchronousness summation and synchronousness average were proposed. They give guidelines for understanding the synchronousness of two data sequences. Theorem 5.2 shows that, in certain cases, the difference between arbitrary two synchronous summations can be reduced by up to half the G.C.D. by moving the starting point of the data sequences.

3. An evaluation function for choosing the data combinations was proposed taking into account both freshness and synchronousness of the data.

4. Frameworks were proposed for consecutive queries based on the evaluation function. Using the frameworks, the user can continuously monitor the query results based on the optimum data combinations. As implementation methods of the consecutive query, a static strategy and a dynamic strategy were proposed. The static strategy minimizes the overhead of the compu-

tation time while processing the queries repeatedly, whereas the dynamic strategy can deal with aperiodic data sequences as well as **pgds**'s.

5. A framework for interactive queries was proposed based on the evaluation function. This query has two parameters ε and δ, in addition to the conventional queries, such as SQL. Parameter ε indicates the upper limit of synchronousness and parameter δ indicates the upper limit of the time period that the user can wait until receiving the query result. The parameter δ expands the probability to acquire fresher and more synchronous data.

6. It was shown that the framework we proposed can be easily extended for use in the situation where the generation time of a particular **pgds** should be certain time points prior to the generation time of the other data.

7. The proposed framework of consecutive query was applied to a concrete example of the collision avoidance system in ITS. It was shown quantitatively that, in this particular case, both the average and maximum synchronousness can be reduced to nearly half compared with the naive method that acquires the newest data at constant intervals.

In the study, we did not consider the query processing time at the wrappers. The freshness and synchronousness of the data obviously depends on the processing time at the wrappers, and query optimization techniques involving the processing time are a future research issue. Also when we choose and join data from three or more **pgds**'s, the algorithms for choosing the data and determining the join order and other optimization techniques are research issues. In addition, in order to establish practical systems, we need to consider operations on the data, such as selection and projection. These are future research issues, too.

We presented two theorems for calculating upper bounds of the synchronousness of data combinations, given a set of **pgds**'s. Theorem 5.3 determines the least upper bound of the synchronousness for **pgds**'s with the same periodic interval. Theorem 5.4 gives an upper bound of the synchronousness for **pgds**'s that may have different periodic intervals. If we apply Theorem 5.4 to **pgds**'s with the same periodic interval, the upper bound obtained is larger than the least upper bound provided by Theorem 5.3. Thus there is room for improving Theorem 5.4. This improvement is a future research issue.

As a criterion for determining the optimum data combinations, an evaluation function, which is a weighted summation of freshness and synchronousness, was proposed. This evaluation function is straightforward and easy to understand its behavior. Depending on the nature of applications, however, we should consider different evaluation functions, possibly taking into account factors other than freshness and synchronousness. In particular, the proposed evaluation function does not consider the values of the data. Thus we may have to consider more sophisticated evaluation functions.

Chapter 6

Conclusions and Future Work

In the dissertation, issues involved in integration of heterogeneous information sources were studied.

In Chapter 2, the background of the research on integration of heterogeneous information sources was surveyed.

In Chapter 3, for a given set of extensional database predicates and a conjunctive query, the maximum information was defined as the tuples in the full disjunction that meet the query condition. For obtaining the query result, the outerjoin operator was shown to be essential. Next, algorithms were presented for creating the query processing plans that obtain the maximum information using the outerjoin operator. Then, an algorithm for creating a query processing plan that also includes predicate and foreign function operators for resolving semantic heterogeneities was presented.

In Chapter 4, a query processing method based on a self-descriptive object model was described. The self-descriptive object model was shown to be simple but flexible to absorb the structural discrepancies among various data models, even allowing the structure of each data instance to be different from one another. Several query processing techniques, such as extracting object elements using

the extraction template and creating the view using the view template, were proposed. The proposed query processing method was shown to resolve various aspects of both heterogeneity and autonomy described in Chapter 2.

Lastly, in Chapter 5, data acquisition strategies from periodically generated data sequences (**pgds**'s), each having a constant period of data generation, were proposed. The strategies were based on the novel criteria of freshness and synchronousness of the data combinations that consist of data each obtained from different **pgds**. Next, some properties on synchronousness, including its upper bounds, were shown. Then, query processing frameworks for consecutive queries and interactive queries were presented based on the evaluation function, which was a combination of freshness and synchronousness. Finally, as a concrete example, it was shown that the proposed framework can be applied to a road obstacle and accident detection system with multiple sensors, enabling the accurate detection of the objects nearly half the size as compared with a naive algorithm.

The future work includes the following:

1. In Chapter 3, the algorithm that incorporates outerjoin, predicate, and foreign function operators was presented. Though the algorithm works fine for the test suite, its correctness has not been proved. Thus the proof of the correctness of the algorithm is a future research issue.

2. The self-descriptive object model, OEM, described in Chapter 4 is different from the model used in XML, which is expected to be wide spread as a common data exchange model. In particular, the order of subobjects is preserved in the XML model while subobjects are treated as a set in the OEM object model. This difference is significant in joining information sources. In order to handle XML, the proposed query processing framework and the intermediate data structure should be revised.

3. In Chapter 5, the evaluation function was defined to be a linear combination of freshness and synchronousness. The most suitable form of the evaluation function, however, depends on the applications. For example in a mobile environment, the minimization of network connection times between the clients and the servers while keeping the clients's data up-to-date may be regarded as criteria. Thus provision of more general framework for determining the optimum combination of **pgds**'s is a future work.

References

[AAFZ95] S. Acharya, R. Alonso, M. Franklin, and S. Zdonik. Broadcast disks: Data management for asymmetric communication environment. In *Proceedings of the ACM SIGMOD*, pages 199–210, San Jose, USA, June 1995.

[AB91] S. Abiteboul and A. Bonner. Objects and views. In *Proc. ACM SIGMOD Conference*, pages 238–247, Denver, USA, 1991.

[Abi97] S. Abiteboul. Querying semi-structured data. In *Proc. of ICDT and LNCS*, volume 1186, pages 1–18. Springer-Verlag, 1997.

[AFZ97] S. Acharya, M. Franklin, and S. Zdonik. Balancing push and pull for data broadcast. In *Proceedings of the ACM SIGMOD*, pages 183–194, Tucson, USA, June 1997.

[AGM88] R. Abbott and H. Garcia-Molina. Scheduling real-time transactions. *SIGMOD Record*, 17(1):71–81, Mar. 1988.

[Aka99] M. Akasu. Compact scan laser radar for vehicles. *Mitsubishi Denki Giho*, 73(10):60–63, October (in Japanese) 1999.

[ASD+91] R. Armed, P. Smedt, W. Du, W. Kent, M. Ketabchi, W. Litwin,

A. Rafii, and M. Shan. The pegasus heterogeneous multidatabase system. *IEEE Computer*, 24(12):19–27, Dec. 1991.

[Ber91] E. Bertino. Integration of heterogeneous data repositories by using object-oriented views. In *Proc. of the International Workshop on Interoperability in Multidatabase Systems*, pages 22–29, Kyoto, Japan, 1991.

[BGH+92] Thomas F. Bowen, Gita Gopal, Gary E. Herman, Takako M. Hickey, K. C. Lee, William H. Mansfield, John Raitz, and Abel Weinrib. The datacycle architecture. *CACM*, 35(12):71–81, December 1992.

[BGI95] G. Bhargava, P. Goel, and B. Iyer. Hypergraph based reordering of outer join queries with complex predicates. In *Proc. of ACM-SIGMOD Int. Conf. on the Management of Data*, pages 304–215, San Jose, USA, 1995.

[BLN86] C. Batini, M. Lenzerini, and S. Navathe. A comparative analysis of methodologies for database schema integration. *ACM Computing Surveys*, 18(4):323–364, Dec. 1986.

[BNS96] E. Bertino, M. Negri, and L. Sbattella. An overview of the comandos integration system. In O. Bukhres and A. Elmagarmid, editors, *Object-Oriented Multidatabase Systems*, pages 379–422, New Jersey, USA, 1996. Prentice-Hall.

[BOT86] Y. Breitbart, P. Olson, and G. Thompson. Database integration in a distributed heterogeneous database system. In *Proc. 2nd International IEEE Conference on Data Engineering*, pages 301–310, Los Angels, USA, 1986.

[Bun97] P. Buneman. Semistructured data. *Proc. of ACM PODS*, pages 117–121, 1997.

[CGMH+94] S. Chawathe, H. Garcia-Molina, J. Hammer, K. Ireland, Y. Papakonstantinou, J. Ullman, and J. Widom. The tsimmis project: Integration of heterogeneous information sources. In *Proceedings of IPSJ Conference*, 1994.

[Che90] A.L.P. Chen. Outer join optimization in multidatabase systems. In *2nd Int. Symp. on Databases in Parallel and Distributed Systems*, pages .211–218, 1990.

[CS93] S. Chaudhuri and K. Shim. Query optimization in the presence of foreign functions. In *Proc. of the 19h Int. Conf. on Very Large Databases*, pages 529–542, Dublin, Ireland, 1993.

[CWN94] S. Chakravarthy, W. Whang, and S. Navathe. A logic-based approach to query processing in federated databases. *Information Sciences*, 79:1–28, 1994.

[Dat83] C.J. Date. The outer join. In *Proc. of the 2nd Int. Conf. on Databases*, Cambridge, England, Sep. 1983.

[Dav92] M.M. David. Advanced capabilities of the outer join. *ACM SIGMOD RECORD*, 21(1):65–70, Mar. 1992.

[DS96] W. Du and M. Shan. Query processing in pegasus. In *in Object-Oriented Multidatabase Systems*, pages 449–471, New Jersey, 1996. Prentice-Hall.

[Fag83] R. Fagin. Degrees of acyclicity for hypergraphs and relational database schemes. *Journal of ACM*, 30(3):514–550, July. 1983.

[FZ98] M. Franklin and S. Zdonik. Data in your face: Push technology in perspective. In *Proceedings of the ACM SIGMOD*, pages 516–519, Seattle, USA, June. 1998.

[GGMT94] L. Gravano, H. Garcia-Molina, and A. Tomasic. The effectiveness of gloss for the text-database discovery problem. *Proceedings of the ACM SIGMOD Conference*, 23(2):126–137, 1994.

[GL94] C. Galindo-Legaria. Outerjoins as disjunctions. In *Proc. of ACM-SIGMOD Int. Conf. on the Management of Data*, pages 348–358, Minneapolis, USA, 1994.

[GLR92] C. Galindo-Legaria and Arnon Rosenthal. How to extend a conventional optimizer to handle one- and two-sided outerjoin. In *Proc. of the 8th Int. Conf. on Data Engineering*, 1992.

[Gru93] T. R. Gruber. A translation approach to portable ontology specifications. *Knowledge Acquisition*, 5(2):199–220, 1993.

[GTW92] T. R. Gruber, J. M. Tenenbaum, and J. C. Weber. Toward a knowledge medium for collaborative product development. In *Proceedings of the Second International Conference on Artificial Intelligence in Design*, pages 413–432. Kluwer Academic, 1992.

[HB96] S. Heiler and B. Blaustein. Eis: a framework for federated software tools and databases. In O. Bukhres and A. Elmagarmid, editors, *Object-Oriented Multidatabase Systems*, pages 423–448, New Jersey, USA, 1996. Prentice-Hall.

[HGLW87] Gary E. Herman, Gita Gopal, K. C. Lee, and Abel Weinrib. The datacycle architecture for very high throughput database systems.

In *Proceedings of the ACM SIGMOD*, pages 97–103, San Francisco, USA, May 1987.

[HM93] J. Hammer and D. McLeod. An approach to resolving semantic heterogeneity in a federation of autonomous, heterogeneous database systems. *International Journal of Intelligent and Cooperative Information Systems*, 2(1):51–83, 1993.

[IB94] T. Imielinski and B. Badrinath. Mobile wireless computing: Challenges in data management. *CACM*, 37(10):18–28, October 1994.

[Kamse] Y. Kambayashi. Recent research and development of multi-database systems. *Journal of Information Processing Society of Japan*, 35(2):105–119, 1994 (in Japanese).

[KCGS93] W. Kim, I. Choi, S. Gala, and M. Scheevel. On resolving schematic heterogeneity in multidatabase systems. *Distributed and Parallel Databases*, 1:251–279, 1993.

[KFM96] W. Klas, P. Frankhauser, and P. Muth. *Database integration using the open object-oriented database system VODAK*, pages 472–532. Prentice-Hall, New Jersey, USA, o. bukhres and a. elmagarmid edition, 1996.

[KLK91] R. Krishnamurthy, W. Litwin, and W. Kent. Language features for interoperability of databases with schematic discrepancies. In *Proc. ACM SIGMOD Conference*, pages 40–49, Denver, USA, 1991.

[KO95] A. Kurt and M. Ozsoyaglu. Modeling and querying periodic temporal databases. In *DEXA Workshop*, pages 124–133, 1995.

[KSW90] F. Kabanza, J. M. Stevennne, and P. Wolper. Handling infinite temporal data. In *Proceedings of the Ninth ACM SIGACT-SIGMOD-SIGART Symposium on Principles of Database Systems*, pages 392–403, Nashville, Tennessee, Apr. 1990. ACM press.

[LL73] C. L. Liu and J. W. Layland. Scheduling algorithms for multiprogramming in a hard-real-time environment. *Journal of ACM*, 20(1):46–61, Jan. 1973.

[LMR90] W. Litwin, L. Mark, and N. Roussopoulos. Interoperability of multiple autonomous databases. *ACM Computing Surveys*, 22(3):267–293, 1990.

[LRO96] A. Levy, A. Rajaraman, and J. Ordille. Querying heterogeneous information sources using source descriptions. In *Proc. 22nd VLDB Conference*, pages 251–262, Mumbai, India, 1996.

[LW94] B. S. Lee and G. Wiederhold. Outer joins and filters for instantiating objects from relational databases through views. *IEEE Transactions on Knowledge and Data Engineering*, 6(1):108–119, Feb. 1994.

[Makse] A. Makinouchi. The next generation database systems. *The Trans. of the IEICE*, 78(1):65–75, 1995 (in Japanese).

[MRa86] D. Maier, D. Rozenshtein, and D.S. Warren and. *Window functions*, pages 213–246. JAI Press, London, p. kanellakis edition, 1986.

[Mun96a] K. Munakata. Integration of heterogeneous information sources (expository article). *Systems, Control and Information*, 40(12):514–521, December 1996.

[Mun96b] K. Munakata. Query processing for integration of heterogeneous information sources using outer joins. In *Proc. of Int. Symp. on Cooperative Database Systems for Advanced Applications*, volume 1, pages 129–136, Dec. 1996.

[Mun99] K. Munakata. Integration of maximum information using outerjoins, predicates and foreign functions. *IEICE Transactions on Information and Systems*, E82-D(1):64–75, January 1999.

[Munse] K. Munakata. Query processing for integration of information sources based on a self-descriptive object model. *Trans. IEICE*, J81-D-I(6):1–13, Jun. 1998, in Japanese. Also in *Systems and Computer in Japan* by John Wiley & Sons. 30(3):839–850, 1999.

[MUV84] D. Maier, J.D. Ullman, and M.Y. Vardi. On the foundations of the universal relation model,. *ACM Trans. on Database Systems*, 9(2):283–308, 1984.

[MYU99] K. Munakata, M. Yoshikawa, and S. Uemura. On synchronousness of periodically generated data sequences. In *International Symposium on Database Applications in Non-Traditional Environments (DANTE) '99*, pages 298–305, Kyoto, Japan, November 1999.

[MYU00] K. Munakata, M. Yoshikawa, and S. Uemura. Integrating periodic data sequences based on freshness and synchronousness. In *IEEE 10th International Workshop on Research Issues in Data Engineering*, San Diego, USA, February 2000.

[Nis94] T. Nishida. Very large knowledge base systems. *Journal of Information Processing Society of Japan*, 35(2):130–139, 1994.

[NL91] H. Nakazato and K. J. Lin. Interval assignment for periodic transactions in real-time database systems. In *Proceedings of the Seventh International Conference on Data Engineering*, pages 378–385, Kobe, Japan, Apr. 1991.

[OOse] Special issue on object oriented database system. *Journal of Information Processing Society of Japan*, 32(5):489–613, 1991 (in Japanese).

[PGGMU95] Y. Papakonstantinou, A. Gupta, H. Garcia-Molina, and J. Ullman. A query translation scheme for rapid implementation of wrappers. In *Int. Conf. on Deductive and Object-Oriented Databases*, pages 161–186, Singapore, Dec. 1995.

[PGM96] Y. Papakonstantinou and H. Garcia-Molina. Object fusion in mediator systems. In *Proceedings of 22nd International Conference on Very Large Data Bases*, pages 413–424, 1996.

[PGMU96] Y. Papakonstantinou, H. Garcia-Molina, and J. Ullman. Medmaker: A mediation system based on declarative specifications. In *International Conference on Data Engineering*, pages 132–141, 1996.

[PGMW95] Y. Papakonstantinou, H. Garcia-Molina, and J. Widom. Object exchange across heterogeneous information sources. In *International Conference on Data Engineering*, pages 251–260, 1995.

[RGL90] A. Rosenthal and C. Galindo-Legaria. Query graphs, implementing trees and freely-reorderable outerjoins. In *Proc. of ACM-SIGMOD Int. Conf. on the Management of Data*, pages 39–48, Atlantic City, USA, 1990.

[RR84] A. Rosenthal and David Reiner. Extending the algebraic framework of query processing to handle outerjoins. In *Proc. of the 10th Int. Conf. on Very Large Databases*, pages 291–299, Singapore, 1984.

[RU96] A. Rajaraman and J.D. Ullman. Integrating information by outerjoins and full disjunctions. In *PODS*, pages 238–248, 1996.

[SKHS96] K. Shinkawa, Y. Kajiwara, Y. Honda, and H. Suzuki. An automated highway system. *Mitsubishi Denki Giho*, 70(12):16–21, December (in Japanese) 1996.

[SL90] A. P. Sheth and J. A. Larson. Federated database systems for managing distributed, heterogeneous, and autonomous databases. *ACM Computing Surveys*, 22(3):183–236, 1990.

[TK99] Hideki Tai and Kazuya Kosaka. The aglets project. *CACM*, 42(3):100–101, 1999.

[TTC+90] G. Thomas, G. R. Thompson, C. W. Chung, E. Barkmeyer, F. Carter, M. Templeton, S. Fox, and B. Hartman. Heterogeneous distributed database systems for production use. *ACM Computing Surveys*, 22(3):237–266, 1990.

[Ull88] J. D. Ullman. *Principles of Database and Knowledge-base Systems Volume I.* Computer Science Press, 1988.

[Ull89] J. D. Ullman. *Principles of Database and Knowledge-base Systems Volume II.* Computer Science Press, 1989.

[VP97] V. Vassalos and Y. Papakonstantinou. Describing and using query capabilities of heterogeneous sources. In *Proc. of VLDB*, pages 256–265, Athens, Greece, August 1997.

[Wie92] G. Wiederhold. Mediators in the architecture of future information systems. *IEEE Computer*, 25(3):38–49, Feb. 1992.

[Wie96a] G. Wiederhold. *Intelligent Integration of Information*. Kluwer Academic Publishers, 1996.

[Wie96b] G. Wiederhold. A security mediator for health care information. In *AMIA (formerly SCAMC)*, 1996.

[WPW+97] D. Wong, N. Paciorek, T. Walsh, J. DiCelie, M. Young, and B. Peet. Concordia: An infrastructure for collaborating mobile agents. In *First International Workshop on Mobile Agents 97 (MA'97)*, Berlin, Germany, April 1997.

[XSS+97] M. Xiong, R. Sivasankaran, J. A. Stankovic, K. Ramamritham, and D. Towsley. Scheduling access to temporal data in real-time databases. In A. Bestavros, K. J. Lin, and S. H. Son, editors, *Real-Time Database Systems Issues and Applications*, pages 167–191. Kluwer Academic Publishers, 1997.

List of Publications

Journal Paper

1. Koichi Munakata: "Query Processing for Integration of Information Sources Based on a Self-Descriptive Object Model," *IEICE Transactions on Information and Systems*, Vol.J81-D-I, No.6, pp.1-13, June 1998 (in Japanese), also in *Systems and Computers in Japan*, Vol.30, No.3, pp.10–22, March, 1999.

2. Koichi Munakata, "Integration of Maximum Information Using Outerjoins, Predicates and Foreign Functions," *IEICE Transactions on Information and Systems*, Vol.E82-D, No.1, pp.64–75, January, 1999.

3. Koichi Munakata, Masatoshi Yoshikawa, and Shunsuke Uemura: "Acquiring Data Combinations from Periodically Generated Data Sequences Based on Freshness and Synchronousness," *Transactions on Information Processing Society of Japan,*, TOD5, February, 2000 (in Japanese).

International Conference

4. Koichi Munakata: "Query Processing for Integration of Heterogeneous Information Sources Using Outerjoins," *Proceedings of International Sympo-*

sium on Cooperative Database Systems for Advanced Applications, Vol.1, pp.129–136, Kyoto, Japan, December 1996, also in Yahiko Kambayashi and Kazumasa Yokota ed."Cooperative Databases and Applications," pp.92–99, World Scientific Publishing Co. Pte. Ltd., Singapore, 1997.

5. Koichi Munakata: "Integration of Semistructured Data Using Outer Joins," *Workshop on Management of Semistructured Data*, Arizona, USA, available at "http://www.research.att.com/ suciu/workshop-papers.html", May, 1997.

6. Koichi Munakata, Masatoshi Yoshikawa, and Shunsuke Uemura: "On Synchronousness of Periodically Generated Data Sequences," *International Symposium on Database Applications in Non-Traditional Environments (DANTE) '99*, pp.298-305, Kyoto, Japan, November, 1999.

7. Koichi Munakata, Masatoshi Yoshikawa, and Shunsuke Uemura: "Integrating Periodic Data Sequences Based on Freshness and Synchronousness," *IEEE 10th International Workshop on Research Issues in Data Engineering*, San Diego, USA, February, 2000.

Domestic Conference

8. Koichi Munakata, Masatoshi Yoshikawa, and Shunsuke Uemura: "Acquiring Data Combinations from Periodically Generated Data Sequences Based on Freshness and Synchronousness," *Advanced Database Symposium '99*, pp.141–150, Tokyo, Japan, December, 1999 (in Japanese).

Other Publications

Journal Paper

9. Hideo Taguchi, Koichi Munakata, Koichi Kiriyama, and Katsuhiko Fujii: "Brain Control in Voluntary Muscular Activities," *IEICE Transactions on Information and Systems*, Vol.J70-D, No.12, pp.2762–2772, December, 1987 (in Japanese).

10. Tatsuya Masuda, Koichi Munakata, and Katsuhiko Fujii: "Parametric Structural Analysis for Strongly Connected Data Based on Paired Comparisons," *Transactions of SICE (short paper)*, Vol.22, No.10, pp.1115–1117, October, 1986 (in Japanese).

11. Kenji Shima, Koichi Munakata, Shoichi Washino, Shinji Komori, Yasuya Kajiwara, and Setsuhiro Shimomura: "The Application of a Data-Driven Processor to Automotive Engine Control," *IEICE Transactions on Electronics*, Vol.E76-C, No.12, pp.1794–1803, December, 1993.

International Conference

12. Tetsuo Yamasaki, Koichi Munakata, Kenji Shima, Shin'ichi Yoshida, and Hiroaki Terada: "An Implementation of a High Level Language for a Data-Driven Processor," *25th Asilomar Conference on Signals, Systems, and Computers*", MP2-9, Pacific Grove, USA, November, 1991.

Others

13. Koichi Munakata: "Integration of Heterogeneous Information Sources," *Systems, Control and Information* (Expository article),Vol.40, No.12, pp.514-521, December, 1996 (in Japanese).