UTILIZING INTEGRITY CONSTRAINT KNOWLEDGE IN HETEROGENEOUS DATABASES: A METHODOLOGY FOR SCHEMA INTEGRATION AND SEMANTIC QUERY PROCESSING

by

Ramesh Venkataraman

A Dissertation Submitted to the Faculty of the COMMITTEE ON BUSINESS ADMINISTRATION In Partial Fulfillment of the Requirements For the Degree of DOCTOR OF PHILOSOPHY In the Graduate College THE UNIVERSITY OF ARIZONA

1995
INFORMATION TO USERS

This manuscript has been reproduced from the microfilm master. UMI films the text directly from the original or copy submitted. Thus, some thesis and dissertation copies are in typewriter face, while others may be from any type of computer printer.

The quality of this reproduction is dependent upon the quality of the copy submitted. Broken or indistinct print, colored or poor quality illustrations and photographs, print bleedthrough, substandard margins, and improper alignment can adversely affect reproduction.

In the unlikely event that the author did not send UMI a complete manuscript and there are missing pages, these will be noted. Also, if unauthorized copyright material had to be removed, a note will indicate the deletion.

Oversize materials (e.g., maps, drawings, charts) are reproduced by sectioning the original, beginning at the upper left-hand corner and continuing from left to right in equal sections with small overlaps. Each original is also photographed in one exposure and is included in reduced form at the back of the book.

Photographs included in the original manuscript have been reproduced xerographically in this copy. Higher quality 6" x 9" black and white photographic prints are available for any photographs or illustrations appearing in this copy for an additional charge. Contact UMI directly to order.
As members of the Final Examination Committee, we certify that we have read the dissertation prepared by Ramesh Venkataraman entitled *Utilizing Integrity Constraint Knowledge in Heterogeneous Databases: A Methodology For Schema Integration And Semantic Query Processing*.

and recommend that it be accepted as fulfilling the dissertation requirement for the Degree of Doctor of Philosophy.

Dr. Sudha Ram

Dr. J. F. Nunamaker, Jr.

Dr. David E. Pingry

Final approval and acceptance of this dissertation is contingent upon the candidate's submission of the final copy of the dissertation to the Graduate College.

I hereby certify that I have read this dissertation prepared under my direction and recommend that it be accepted as fulfilling the dissertation requirement.

Dissertation Director

Dr. Sudha Ram

Date

Date
STATEMENT BY AUTHOR

This dissertation has been submitted in partial fulfillment of requirements for an advanced degree at The University of Arizona and is deposited in the University Library to be made available to borrowers under rules of the library.

Brief quotations from this dissertation are allowable without special permission, provided that accurate acknowledgment of source is made. Requests for permission for extended quotation from or reproduction of this manuscript in whole or in part may be granted by the head of the major department or the Dean of the Graduate College when in his or her judgment the proposed use of the material is in the interests of scholarship. In all other instances, however, permission must be obtained from the author.

SIGNED: [Signature]
ACKNOWLEDGMENTS

I would like to express deep gratitude to my advisor, Sudha Ram, for her mentoring and support during my tenure as a doctoral student. Without her encouragement and advice during the past three years, I would not have been able to complete this dissertation. She has been a model advisor and has had a significant influence on my approach to research and teaching.

I would also like to thank Jay Nunamaker for providing me with the opportunity to earn my doctorate at one of the finest MIS doctoral programs in the country. The resources and support provided to the graduate students at the University of Arizona are exceptional. I would also like to thank David Pingry for his efforts in ensuring that the same quality of support was available to me during the last two years of my doctoral study.

I would like to acknowledge the role that the following people have played in making me who I am today: my colleagues in the Information Systems department at the University of Maryland Baltimore County, the faculty members and students of the MIS and Electrical Engineering departments at The University of Arizona, the Computer Science department at The University of Iowa and Birla Institute of Technology, and my teachers in school.

Being an only child, the support and company of my friends has always been an important component of my life. I would like to thank all my friends for always being there for me.

My parents, Neela and P. Venkataraman, have made a lot of sacrifices to ensure that I always received a good education. I would like to thank them for their love, affection and support at every stage of my life. I would also like to thank my (late) grandfather, Padmanabhan, my aunt, Jaya, my uncle, Kalyanamaran and my cousin, Gowri, for providing me a home away from home for five formative years of my life. I also thank my wife’s parents, Gomathi and S.B. Mani. I wish they were alive today to share this accomplishment with me.

Finally, I would like to thank my wife, Gayathri, for putting up with my endless days and nights away from home at a time when she needed my support. I dedicate this dissertation to her.
# TABLE OF CONTENTS

**LIST OF FIGURES** ........................................................................................................... 7  
**LIST OF TABLES** ............................................................................................................... 9  
**ABSTRACT** ....................................................................................................................... 11  
**CHAPTER 1: INTRODUCTION** ............................................................................................... 13  
  1.1 Research Methodology .................................................................................................... 19  
**CHAPTER 2: LITERATURE REVIEW** .................................................................................... 22  
  2.1 Heterogeneous Databases ............................................................................................... 22  
  2.2 Schema Integration .......................................................................................................... 28  
    2.2.1 Classification of Schema Integration Strategies ......................................................... 39  
    2.2.2 Interschema Relationship Identification (IRI) ............................................................ 54  
    2.2.3 Semantic Heterogeneity ............................................................................................ 62  
    2.2.4 Integrated Schema Generation .................................................................................. 64  
    2.2.5 Schema Mapping Generation .................................................................................... 72  
    2.2.6 Schema Integration Toolkits ..................................................................................... 74  
  2.3 Semantic Query Optimization .......................................................................................... 83  
    2.3.1 Integrity Constraints ................................................................................................ 84  
    2.3.2 Major approaches to Semantic Query Optimization .................................................. 85  
  2.4 Query Processing in Heterogeneous Databases ............................................................... 106  
**CHAPTER 3: A METHODOLOGY FOR HETEROGENEOUS DATABASE MANAGEMENT** .............................................................................................................................. 111  
  3.1 Schema Translation ......................................................................................................... 116  
  3.2 The Unifying Semantic Model (USM) ............................................................................ 118  
  3.3 Schematic Interschema Relationship Identification ....................................................... 123  
    3.3.1 Entity Class Definition Similarity ............................................................................. 124  
    3.3.2 Attribute Similarity ................................................................................................ 128  
    3.3.3 Relationship Similarity ............................................................................................. 131  
    3.3.4 Heterogeneous Relationships ................................................................................. 139
3.3.5 Evaluation of Schematic Interschema Relationships .................... 141
3.4 Constraint-based relationship generation .......................................... 142
  3.4.1 Evaluation of constraint-based relationships ............................. 149
3.5 "Real World" Interschema Relationship Generation ............................. 154
3.6 Integrated Schema Generation ...................................................... 156
3.7 Integrity Constraint Integration ..................................................... 157

CHAPTER 4: SYSTEM ARCHITECTURE AND DESCRIPTION ................................................................. 179
  4.1 Traditional Schema Integration Components ..................................... 179
    4.1.1 A Blackboard Architecture for Schema Integration ................. 183
  4.2 Components of the Enhanced Methodology ...................................... 211
    4.2.1 Constraint-based (CB) Relationship Generator ....................... 211
    4.2.2 "Real World" Interschema Relationship Generator .................... 212
    4.2.3 Integrity Constraint Integrator ........................................... 214

CHAPTER 5: SEMANTIC QUERY PROCESSING ................................................................. 219
  5.1 Simulation Study ...................................................................... 227
    5.1.1 Case 1: Equivalence Class ................................................. 231
    5.1.2 Case 2: Superclass of a subsumption relationship .................... 232
    5.1.3 Case 3: Subclass of a subsumption relationship ....................... 232
    5.1.4 Case 4: Union class generated as a result of overlap relationship... 233
    5.1.5 Case 5: Original classes of an overlap relationship ................ 235
    5.1.6 Case 6: Disjoint Classes .................................................... 236
  5.2 Parameters and Procedures .......................................................... 239
  5.3 Results .................................................................................. 244

CHAPTER 6: CONCLUSIONS AND FUTURE RESEARCH ................................................................. 258
  6.1 Future Research .................................................................... 260

REFERENCES .................................................................................. 268
LIST OF FIGURES

Fig. 2.1 Global Schema Approach ............................................................. 25
Fig. 2.2 Federated Schema Approach .......................................................... 27
Fig. 2.3 Traditional Schema Integration Methodology ............................... 32
Fig. 2.4 Example Local Schemas .............................................................. 34
Fig. 2.5 Translated Schemas .................................................................... 35
Fig. 2.6 Integrated Schema ...................................................................... 38
Fig. 2.7 View Generation using a Query .................................................. 43
Fig. 2.8 New View Generation using a Query ......................................... 45
Fig. 2.9 Instance-level Integration Problems ........................................... 47
Fig. 3.1 Enhanced Methodology for Schema Integration .......................... 112
Fig. 3.2 Example USM Schema .............................................................. 120
Fig. 3.3 Example Schemas ...................................................................... 127
Fig. 3.4 Integrity Constraint Evaluation .................................................. 152
Fig. 3.5 Example Schema 2 .................................................................... 172
Fig. 4.1 Dependencies among Schema Integration Tasks ....................... 180
Fig. 4.2 “Classical” Expert System ........................................................... 184
Fig. 4.3 Architecture of a Typical Blackboard System ............................ 185
Fig. 4.4 Blackboard Architecture for Schema Integration ...................... 189
Fig. 4.5 Four Level Blackboard Architecture ........................................... 190
Fig. 4.6 Example Translated Schemas ....................................................... 192
Fig. 4.7 An Example of a Populated Blackboard ................................. 198
Fig. 4.8 Example Integrated Schema .................................................... 200
Fig. 4.9 Sample Screen showing an USM Schema ................................ 205
Fig. 4.10 Sample Screen showing the Communications Facility .......... 206
Fig. 4.11 Relationship Assertion between Entity Classes ...................... 209
Fig. 5.1a SQP without our methodology ................................................. 220
Fig. 5.1b SQP Using our Methodology ..................................................... 220
Fig. 5.2a Example Schema 1 ................................................................. 222
Fig. 5.2b Integrated Schema and Constraints ......................................... 222
Fig. 5.3a Example Schema 2 ................................................................. 225
Fig. 5.3b Integrated Schema ................................................................. 225
Fig. 5.4 Logical Simulation Model .......................................................... 228
Fig. 5.5 Condition A ........................................................................... 246
Fig. 5.6 Condition B ........................................................................... 252
Fig. 5.7 Condition C ........................................................................... 253
Fig. 5.8 Condition D ........................................................................... 254
Fig. 5.9 Condition E ........................................................................... 255
Fig. 5.10 Condition F .......................................................................... 256
Fig. 5.11 Condition G ......................................................................... 257
Fig. 6.1 An Agent Based Architecture for Heterogeneous Databases ....... 264
**LIST OF TABLES**

Table 2.1 Two-way Classification of Schema Integration Methodologies..............52
Table 2.2 Methodologies for Schema Integration.............................................53
Table 2.3 Summary of Interschema Relationship Generation Techniques...............59
Table 2.3 (Contd.) Summary of Interschema Relationship Generation Techniques.....60
Table 2.4 Summary of Integrated Schema Generation Techniques......................73
Table 2.5 Summary of Schema Integration Toolkits.........................................82
Table 2.6 Summary of Different Forms of Semantic Query Optimization...............89
Table 2.7 Summary of Semantic Query Optimization Techniques.......................100
Table 2.7 (Contd.) Summary of Semantic Query Optimization Techniques.............101
Table 2.8 Classification of Semantic Query Processing Methodologies Based on Database Type...................................................................................103
Table 2.9 Prototype Heterogeneous Database Systems......................................110
Table 3.1 Entity Class Meta-Data Similarity....................................................126
Table 3.2 Attribute Similarity............................................................................130
Table 3.3a Interaction Relationship Similarity...............................................135
Table 3.3b Generalization/Specialization Relationship Similarity........................136
Table 3.3b (Cont.) Generalization/Specialization Relationship Similarity.............137
Table 3.3c Composites/Grouping Relationship Similarity..................................138
Information sharing among databases requires the development of techniques for accessing data from multiple heterogeneous databases. One approach to providing interoperability among these databases, is to define one or more schemas representing a coherent view of the underlying databases. A review of existing research on schema integration, the process of generating integrated schemas, points to the need for development of techniques for identifying objects in multiple databases that may be related. The development of efficient mechanisms for accessing heterogeneous databases is another issue that has received very little attention in the literature.

This dissertation describes a seven step methodology for utilizing integrity constraint knowledge from multiple heterogeneous databases. The methodology extends traditional approaches to schema integration by proposing additional steps that describe how integrity constraints can be used, in a heterogeneous database environment, to improve the interschema relationship identification process and generate additional semantics, in the form of integrity constraints, at the integrated schema level. The dissertation introduces the concept of constraint-based relationships among objects in heterogeneous databases and describes how these relationships can be used to integrate integrity constraints specified on heterogeneous databases. The dissertation also elaborates on how these integrated integrity constraints can be used to facilitate semantic query processing in a heterogeneous database environment. The description of a system that implements the various phases of the methodology is also presented. A unique feature of the system is that it uses blackboard architectures to facilitate the human-computer interaction needed during schema integration. A simulation study that
shows the potential benefits of performing semantic query processing, in a heterogeneous database environment, using the integrated integrity constraints generated by our methodology is also presented.
CHAPTER 1

INTRODUCTION

Modern organizations utilize a number of diverse databases to accomplish their day-to-day data management functions. Typically, these databases are heterogeneous in that they store different types of data, represent data differently, use different software to manage the data and run on different computer hardware. Over time each organization can have many of these databases, each accomplishing a portion of the data management needs of the organization. Organizations are increasingly finding the need to access data from multiple such databases to support day-to-day functions such as decision making and corporate planning (Kamel and Kamel, 1992). Research on heterogeneous database management has emphasized the development of mechanisms that provide users access to needed data from multiple databases through an integrated interface, while preserving the local autonomy of the databases, i.e., without making changes to the existing databases, database management systems or application programs (Appleton, 1985). Techniques from several areas of information technology need to be integrated to achieve this objective. Appleton (1985) identifies four specific areas: 1) User interface technology, 2) data management technology,
3) network transaction management technology and 4) interprocessor communication technology. In this dissertation, we approach this problem from a data management perspective.

From a data management perspective, issues in heterogeneous database management can be divided into two categories: integration issues and access issues. Two popular approaches to heterogeneous database integration are: the global schema approach and the federated schema approach (Bright and Hurson, 1992; Sheth and Larson, 1990). In the global schema approach, local schemas corresponding to each local database are integrated into a single schema. This schema shields the user from the heterogeneity of the underlying databases by providing them with the illusion of accessing data from a single integrated database. In the federated approach, each local schema provides an export schema, i.e., a portion of its schema that it is willing to share with other databases. Each database then uses these export schemas to define an import schema, a partial global schema representing information from remote databases that is accessible locally. Thus, each node has access to the information defined in its import schema in addition to the information defined in its local schema (Sheth and Larson, 1990).
The development of mechanisms for integrating individual local schemas are crucial to providing heterogeneous database interoperability using either approach. The process of generating one or more integrated schemas from existing schemas is referred to as schema integration. Issues in schema integration have been studied by database researchers for over a decade. Batini et. al. (1986) summarize the characteristics of early schema integration methodologies and identify four stages in the schema integration process: schema translation, pre-integration, schema comparison and schema conformance. We refer to the latter two stages as interschema relationship identification (IRI) and integrated schema generation (ISG). The objective of the IRI step is to analyze the underlying schemas and identify instances where the same real-world concept has been represented differently in different schemas or instances where different real-world concepts have similar representations in different schemas. The objects representing the real-world concepts described above are said to be in conflict and hence the IRI process is also referred to as conflict identification. The objective of the integrated schema generation phase is to generate an integrated representation of the underlying schemas. This requires that all conflicts identified in the IRI stage be resolved. Hence, the ISG process is also referred to as conflict resolution. Much of the research on schema integration has focused on the development of conflict resolution mechanisms (Gotthard et. al. 1992; Shoval and Zohn 1991; Spaccapietra et. al. 1992; Larson et. al. 1989; Navathe et. al. 1986). However, most of these
methodologies place the burden of conflict identification on users/designers providing little or no support for them. Some methodologies do provide toolkits that allow users/designers to browse schemas and record assertions about relationships among objects in the schemas (Sheth and Marcus, 1992). Support for automatic conflict identification when available is typically not comprehensive and is based on comparing schematic characteristics of certain objects, such as names of entity classes or attributes, to detect possible conflicts. It has been accepted in the literature that there is much room for improvement in the conflict identification phase. The problem with existing approaches to automated conflict identification is that they use schematic information as their sole source of knowledge. It is our contention that using other types of knowledge about the semantics of the database can improve the conflict identification process. One such type of knowledge is the semantics conveyed by integrity constraints defined on the databases being integrated.

Hence, the first research question addressed by this dissertation is "How can one utilize knowledge available through integrity constraints to improve conflict identification during schema integration?"

The second major issue in heterogeneous database management is the development of efficient mechanisms for accessing data from the databases once one or more integrated
schemas have been derived. The development of access mechanisms in a heterogeneous environment, utilizing either the federated or global schema approaches, needs to meet two criteria: i) access should be "transparent", i.e., users should be able to access data without being aware of the existence of multiple underlying databases and ii) access should be efficient. Meeting the first criteria requires the development of SQL like languages that will allow users to make queries against the integrated schema(s). An example of such a language is USM-SQL (Paschalidaes, 1993). Efficient database access in a heterogeneous environment entails that we use/develop mechanisms that do not rely on the syntactic structure of the underlying databases to improve their efficiency since such information may not always be accessible. Semantic query processing (SQP)\(^1\) techniques have been shown to utilize the semantics of a database to make data access more efficient in relational and deductive databases (Seigel et al., 1992; Shenoy and Ozsoyoglu, 1989). These techniques utilize the semantic knowledge specified in the integrity constraints to transform user queries into equivalent yet more efficient database queries. The fact that SQP techniques utilize semantic rather than syntactic knowledge makes them an ideal candidate for use in a heterogeneous environment. However, to be able to perform SQP on queries formulated

\(^1\) Semantic Query Processing techniques attempt to transform queries issued on a database using the constraints specified on it. This may result in multiple such queries being generated. Semantic Query Optimization techniques additionally attempt to generate a rank-ordering of the transformed queries using appropriate heuristics. We use the terms Semantic Query Processing and Semantic Query Optimization interchangeably to refer to the process of transforming queries.
against an integrated schema it is necessary to generate integrity constraints that are applicable to the integrated schema.

**Hence, the second research question addressed by this dissertation is, "How can one utilize integrity constraints specified on multiple databases to perform semantic query processing in a heterogeneous database environment?"**

This dissertation presents a methodology for utilizing integrity constraint knowledge to facilitate schema integration and semantic query processing in a heterogeneous database environment. The specific contributions of this dissertation are:

1) it presents an enhanced technique for IRI using multiple knowledge sources. This enhanced technique
   a) presents a comprehensive set of heuristics for schematic IRI
   b) introduces the concept of constraint-based interschema relationships and presents techniques for generating these relationships
   c) describes how schematic and constraint-based interschema relationships can be used to generate "real world" interschema relationships, thus reducing the amount of user/designer interaction required during IRI.
2) it presents a technique for integrating integrity constraints belonging to multiple heterogeneous databases and describes how these integrated constraints can be used to facilitate SQP in a heterogeneous database environment.

1.1 Research Methodology

We use formal modeling, systems development and systems validation using simulation as the primary methodologies in this dissertation. This multi-methodological approach is based on the guidelines suggested in Nunamaker (1992) and consists of four phases:

1) **Conceptual framework development:** In this stage we define the research question(s) that will be addressed in the research. For the work presented in this dissertation, we present two specific research questions.

2) **Theory Development:** This phase includes the development of new ideas, concepts, algorithms, models or methodologies that can be used to solve the problems identified in the research question. In this dissertation, we propose enhancements to the traditional schema integration methodology by adding several new steps.
3) Systems Development: Once new ideas have been defined and solutions have been proposed, one needs to develop a system that implements the ideas and algorithms for different phases of the methodologies developed in the previous stage. Such a system can then be used as a vehicle to validate the proposed solution. The systems development process can be subdivided into two phases:

a) Developing a system architecture: The system architecture provides a roadmap for the system building process. Researchers must "identify the constraints of the environment, state the focus of the work and define the functionality of the resulting system".

b) Analyze, design and build the system: In this phase one applies standard software engineering principles to develop a system that implements the principles of the solution(s) proposed in the theory development phase. Such a system proves the feasibility of the proposed solution.

In this dissertation, we describe a novel architecture for schema integration and present the description of a system based on the architecture.
4) System Validation: In this phase one uses other research methodologies, most notably behavioral research methodologies such as experimentation and case studies/field studies, to validate the solutions proposed in the initial phase of this methodology. We use simulations to validate the techniques proposed in this dissertation.

The next chapter presents a summary of related research and the specific research hypotheses being studied in this dissertation. Chapter 3 introduces an enhanced methodology for schema integration and presents the details of the various steps proposed in the enhanced methodology. Chapter 4 presents the description of a system that implements the various phases of the enhanced methodology. Chapter 5 presents the details of a simulation experiment used to illustrate the benefits of using the enhanced methodology. Finally, Chapter 6 summarizes the contributions of this dissertation and presents directions for future research.
CHAPTER 2

LITERATURE REVIEW

2.1 Heterogeneous Databases

The objective of heterogeneous database research is to develop methods that overcome the limitations posed due to the existence of different forms of heterogeneity that may exist in diverse databases and provide access to the underlying data. There are several issues that need to be addressed in a heterogeneous database environment, including: transparency, access language, autonomy, transaction processing and performance. Different approaches to heterogeneous database management address each of these issues to varying degrees. Heiler et. al. (1991) describe five different levels of transparency that can be achieved when integrating heterogeneous databases: data model transparency, schema transparency, source transparency, dependency/constraint transparency and time/context transparency. Data model transparency refers to whether the existence of different underlying data models is known to the user or not. Schema transparency hides schematic differences in representation, e.g., differences in the structure or semantics of data elements from the user. Source transparency hides the details of where the data resides and what techniques
are needed to access the data from the user. For example, complete source transparency would mean that users need not be aware of the existence of different data access languages or the fact that the underlying databases reside over a network. Dependency/constraint transparency hides the details of update operations from the user. Finally, time/context transparency hides the existence of different versions of the same data from the user. On the autonomy axis the underlying databases can be completely autonomous or have no autonomy at all. The choice of access language can range from having a single unified language to express queries to having the capability to express queries in one of the languages of the underlying databases and having it translated to other database access languages as needed.

There are three popular approaches to heterogeneous database interoperability: the global schema approach, the federated schema approach and the multidatabase approach (Bright and Hurson, 1992; Sheth and Larson, 1990; Kamel and Kamel, 1992). The choice of approach is primarily dictated by the level of transparency and autonomy desired. The first two approaches attempt to provide users with transparent access to the underlying databases (component databases or CDBs). Two of the key objectives of heterogeneous database integration include: 1) maintaining site autonomy, i.e., each site should still have control
over the data and who accesses the data and 2) resolving semantic heterogeneity, i.e., the differences in the semantics of the data stored in the underlying databases.

**Global Schema Approach:** In the global schema approach, the local schemas are integrated into a single schema. This schema shields the user from the heterogeneity of the underlying databases by giving them the illusion that they are accessing data from a single integrated database. Hence, the global schema approach provides complete data model, schema and source transparency. However, most implementations of the global schema approach require that data updates be handled through the global system resulting in loss of autonomy of the underlying databases. Most implementations of the global schema approach use a single data access language. Fig. 2.1 shows the general architecture of a heterogeneous database management system that uses the global schema approach.
Fig. 2.1 Global Schema Approach
Federated Databases: The global schema approach assumes that all the underlying databases want to share all their data with the other databases. This approach, however, may not be suitable for scenarios where the underlying databases want to share only certain portions of their data. The solution in such cases is to use federated databases (Sheth and Larson, 1990; Bright and Hurson, 1992). In a federation, each local database has an export schema and an import schema. The export schema is the portion its data that the local database is willing to share with the rest of the databases. The import schema contains information about remote data that can be accessed from the local databases. Thus, each import schema is an integrated representation of the data available in remote databases. The federated database approach provides data model, schema and source transparency for the portion of data that the underlying databases are willing to share. Federated approaches allow local databases to maintain autonomy and usually provide multi-lingual access. Users using a particular database can issue queries in the underlying database's query language. If the query requires remote data access then the query is appropriately transformed and issued to the remote database(s). Fig. 2.2 shows the general architecture of a heterogeneous database management system using the federated schema approach.
Fig. 2.2 Federated Schema Approach
**Multidatabases:** The multidatabase approach to heterogeneous database management is intended to eliminate the need for generating or maintaining any (partially) integrated schema (Bright and Hurson, 1992). The objective in the multidatabase approach is not to provide location or data transparency to the user. Rather, these approaches concentrate on providing users with tools, such as, advanced query languages that allow users to specify and access data from a local or remote site. These systems, however, do try to provide some level of access transparency in that user queries specified using a query language are translated into appropriate queries on local and remote databases by the system. Thus, multidatabase systems can be thought of as providing partial source transparency by hiding the heterogeneity in the underlying data access languages from the user. Limited support for data model and schema transparency is also provided and is usually embedded in the characteristics of the multidatabase language.

### 2.2 Schema Integration

Schema integration is at the core of methodologies that use the first two approaches to providing heterogeneous database interoperability. Schema integration is the process of generating one or more integrated schemas from (portions of) existing local schemas. These schemas represent the semantics of the databases being integrated and are used as inputs to
the integration process. The output of the process is one or more integrated schemas representing the semantics of the underlying databases. The output schema(s) are represented using a common data model and hide any heterogeneity due to schematic differences in the underlying databases or differences in data models. These schemas are then used to formulate queries, some of which may possibly need to span multiple databases.

The term schema integration has been loosely used in the literature to refer to methodologies that facilitate integration of schemas as defined above, as well as methodologies for view integration, i.e., integration of user views defined on a database schema. This is because many of the techniques applicable in a schema integration context can be used in view integration and vice versa.

View integration is the process of generating a single integrated schema from multiple user views and is typically used in the design of a new database schema. Hence, view integration is used in top-down database design. We start with multiple user views, generate the integrated schema corresponding to these views and then design the database corresponding to that schema. Schema integration on the other hand is a bottom-up process because it attempts to integrate existing databases.
However, the two processes differ in important ways (Spaccapietra et al., 1992; Sheth and Larson, 1990):

1) In view integration, users define views using a single data model. In schema integration, since the underlying databases can be heterogeneous, the schemas to be integrated may be represented using multiple data models.

2) At the time that view integration is performed, user views do not reflect existing data in a database. However, in schema integration we integrate schemas that represent existing databases. This is an important distinction, because the schema generated by the schema integration process cannot violate the semantics of the existing databases. However, in view integration since the views represent abstract objects, there is more flexibility in the interpretation of their semantics.

Schema integration is a complex and time consuming problem. The primary reason for this complexity arises from the fact that most schematic representations do not capture the intended semantics of the underlying databases completely. Hence, the process of integration requires extensive interaction with database designers and administrators to
understand the semantics of the databases and ensure that the semantics of the integrated schema do not violate the semantics of the underlying databases. This also means that the process of schema integration cannot be completely automated (Ram, 1991). However, tools that can reduce the amount of human interaction can be developed and are elaborated upon later in this chapter. It should also be noted that schema integration is not a one-time process. Since the integrated schema represents the underlying databases, changes to it may be needed due to:

1. Changes in the database structure which result in changes to the underlying schemas;
2. Changes in the constraints specified on the underlying databases; and,
3. Changes to data values due to additions, modifications or deletes in the underlying databases.

As a result, a desirable property of any schema integration approach is that it should be able to dynamically handle changes to the underlying databases. Figure 2.3 shows the steps involved in a typical schema integration methodology.
Fig. 2.3 Traditional Schema Integration Methodology
1. Schema Translation: This phase requires that schemas corresponding to the individual databases being integrated be translated into schemas using a common model. Traditionally, a semantic model, such as, the Entity-Relationship model (Chen, 1976) has been used for this purpose. Figure 2.4 shows an example of a relational and a network database. The schemas are examples of databases that may exist in a bank such as Bank One or your local credit union. The first database contains information about customers and their accounts in the bank. The second database keeps track of loans issued to borrowers. The translation of these schemas may be performed manually or with the aid of a translation tool. However, even with the use of a translation tool it is more than likely that some form of manual interaction with the tool will be needed. Any schema translation technique should possess the following characteristics: a) the translated schema should represent the semantics of the underlying database completely and b) it should be possible to translate a query issued on the translated schema into queries on the local databases. Figure 2.5 shows the translated ER representations of the schemas in Figure 2.4.
CUSTOMER(SSNO, FName, LName, Address, PhNo, NYears, CR)

ACCOUNTS(AcctNo, Balance, Type, CustNo)

DB1
Relational Schema

<table>
<thead>
<tr>
<th>IDNo</th>
<th>First</th>
<th>Last</th>
<th>Address</th>
<th>PhNo</th>
<th>CR</th>
<th>NYears</th>
<th>NLoans</th>
</tr>
</thead>
</table>

Borrows

Loans

<table>
<thead>
<tr>
<th>LoanNo</th>
<th>Amount</th>
<th>Type</th>
</tr>
</thead>
</table>

DB2
CODASYL Schema

Fig. 2.4 Example Local Schemas
Fig. 2.5 Translated Schemas
2. Schematic Interschema Relationship Generation: The objective of this phase is to identify objects, i.e., entities, attributes and relationships, in the underlying schemas that may be related and to categorize the relationships among them. This is done by examining the semantics of the objects in the different databases and identifying relationships based on their semantics. The semantics of an object can be ascertained by analyzing schematic properties of entity classes, attributes and relationships in the schema as well as by interacting with designers and exploiting their knowledge and understanding of the application domain. For example, integrity constraints, cardinality and domains are properties of attributes that convey their semantics. The ultimate objective of this step is the generation of a reliable set of relationships among database objects. It is important that these relationships be accurate because they are used as inputs to the integrated schema generation phase. For example, in Figure 2.5, we would identify that the two entities named customer in schema 1 and borrower in schema 2 are related to each other. In addition, we can classify the relationship as being a subsumption relationship, i.e., the set of borrowers in the second database is a subset of the set of customers identified in the first database. Finally, we would identify attributes in the two entity classes that may be related.

3. Integrated Schema Generation: In this phase, the interschema relationships generated previously are used to generate an integrated representation of the underlying schemas.
generating such a representation involves resolving various forms of heterogeneity that may exist between related objects in the underlying databases. the different forms of heterogeneity can be classified into two broad categories: a) differing representations of similar objects in different databases and b) differences in data values of similar objects in different databases.

the integrated schema generation process resolves these different forms of heterogeneity and generates an integrated schema that hides the heterogeneity from the user. in our example, in the integrated schema (figure 2.6) a subsumption relationship between customer and borrower is generated to reflect the nature of the relationship among these entity classes. note that the attributes ssno & idno have been integrated into a single attribute (custno) in the superclass. this type of integration assumes that these attributes have been identified as being equivalent to each other in the interschema relationship generation step.

4. schema mapping generation: this step accompanies the integrated schema generation step, and involves storing information about mappings between objects in the transformed (integrated) schemas and objects in the local schemas. such mappings are important for
Fig. 2.6 Integrated Schema
query transformation. For example, we would need to note that the attribute CUSTNO in the integrated schema (Figure 2.6) maps back to SSNO in database 1 and IDNO in database 2. It should be noted that these steps may need to be performed iteratively to resolve the heterogeneity and arrive at an integrated representation(s) of the underlying schemas.

2.2.1 Classification of Schema Integration Strategies

Two primary properties distinguish integration strategies in the literature: a) the abstraction level at which integration is attempted, which in turn dictates the types of heterogeneity that need to be considered by a methodology and b) the semantics conveyed by the input schemas. The semantic richness of the input schemas is dependent on the data model used. Hence, we classify schema integration strategies based on the abstraction level at which they operate and on the data model used to represent input schemas. A third classification based on the degree to which the integration methodologies can deal with changes to the underlying databases can also be generated. However, as will be discussed later, this classification parallels the classification based on abstraction level.

*Classification based on Abstraction Level*
Integration methodologies presented in the literature can be classified as operating at one of the following levels:

1. User Views: Most view integration methodologies fall in this category. The objective of view integration methodologies is to integrate several user schemas (representing users' views of a database) into a single integrated schema. Hence, view integration is part of the top-down database design process. It is typical for most users' views to be represented using a common data model. As a result, it is unlikely that these methodologies require the schema translation step. Moreover, since the views do not represent an existing database, much of the intended semantics is conveyed by the schema itself. If we assume that the schemas shown in Figure 2.5 represent user views of the banking database, then the integrated schema generated in Figure 2.6 would represent the result of applying view integration strategies on those views. The integrated schema could then be used as the starting point for designing a new database. Many of the integration methodologies reported in Batini et. al. (1986) are view integration methodologies. Batini and Lenzerini (1984), Navathe and Gadgil (1982), Biskup and Convent (1986), Navathe et. al. (1986), Shoval and Zohn (1991) and Gotthard et. al. (1992) are examples of view integration methodologies.
2. Conceptual Schema: The methodologies that operate at this level generate one or more integrated schemas from schemas of the local databases being integrated. To achieve this objective, it is necessary for the methodologies to cope with both structural and semantic heterogeneity in the underlying databases. Methodologies at this level can be divided into two classes: a) those that generate integrated schemas by applying schema restructuring operators to the underlying databases, i.e., schema restructuring methodologies and b) those that generate an integrated representation by developing views or defining queries on the local databases of interest, i.e., view generation methodologies. The difference between these strategies is explained in the next few paragraphs.

Prominent examples of schema restructuring methodologies include, El-Masri et. al. (1986), Larson et. al. (1989) and Spaccapietra et. al. (1992). The application of any of these methodologies to the schemas in Figure 2.4 (representing local schemas after translation) would result in an integrated schema similar to that shown in Figure 2.5. Thus, the primary difference between schema restructuring strategies and view integration strategies lies in the fact that in schema restructuring methodologies the schemas being integrated are derived from heterogeneous data models and represent an underlying database.
Examples of approaches using the view generation methodology can be found in Kim and Seo (1991), Ahmed et. al. (1991), Bertino (1991), Kaul et. al. (1990). In our example, assume that we relax the restriction that all borrowers have an account with the bank. To generate an entity class representing the set of all customers who are associated with the bank, we would define a query on the customer and borrower entity classes from the underlying databases and create a supertype entity class called ALL_CUSTOMERS. Figure 2.7 shows the view and the query that can result in such a view being generated. It should be noted that if the SSNO and IDNO fields in the customer and borrower entity classes are different, we may have to define a new attribute called CUSTNO and define a function that maps CUSTNO to IDNO and SSNO. The same argument holds for other attributes that may differ. It is interesting to note that although early methodologies such as Mannino and Effelsberg (1984), Templeton et. al. (1987), Motro and Buneman (1981) and Casanova and Vidal (1983) seem to adopt the view generation paradigm, the process of developing the integrated view in these methodologies is much closer to the schema restructuring paradigm.
CREATE TYPE ALL_CUSTOMERS
SUPERTYPE OF CUSTOMER, BORROWER

Fig. 2.7 View Generation using a Query
The primary difference between the schema restructuring and view generation approaches lies in the static and dynamic nature of these approaches respectively. An integrated schema generated using schema restructuring is a representation that reflects schema definitions at the time integration was performed. Any changes to the underlying databases that affect the schemas, will require that the process be repeated. The view generation approach is more dynamic, since the integrated representation is generated by defining a view on the local schemas. As a result, if the schemas change, only a new view needs to be defined and that too only if the change can have a possible effect on the existing view. For example, if the bank wanted to merge information about its money market account customers (maintained separately) with the rest of the databases, we would define a new view which would include customers with regular, loan and money market accounts as shown in Figure 2.8.

3. Data Level: Methodologies at this level rely upon actual data values to accomplish integration. Much of the work at this level has focused on integrating relational databases. Instance-level integration strategies presented in DeMichiel (1989), Prabhakar et. al. (1993), and Chatterjee and Segev (1991) fall into this category. Methodologies at this level address two main problems:
CREATE TYPE ALL_CUSTOMERS
SUPERTYPE OF CUSTOMER, BORROWER, MM_CUSTOMER

Fig. 2.8 New View Generation using a Query
a) Entity Identification: How does one identify representations of the same real world entity in different databases, and b) Attribute-value conflicts: How does one deal with differences in data values among attributes that represent the same real-world entity?

Such differences can arise due to differing attribute domains as well as differences in the actual data values stored in the databases. For example, let us assume that we are trying to generate a relation that represents the list of customers with outstanding loans. This could be generated by defining an intersection of the two databases shown in Figure 2.9. The process of intersection is relatively easy, if the two relations share a common key. Generating an integrated relation requires that we identify, for instance, that two tuples represent the same person. However, consider the tuples shown in Figure 2.9. It is clear that tuple 1 in the customer relation and tuple 2 in the borrower relation refer to the same entity. However, because the SSNO and IDNO do not match they cannot be used as the sole source of identification for matching tuples. The combination of last and first name also cannot be used since, there may exist more than one customer with the same combination. Thus, differences in data and the lack of a key can make identification of related tuples and performing a join a difficult process. Instance-level integration deals with resolving such incompatibilities. It should be noted that changes to the data values may void any integration performed previously. Thus instance-level integration strategies are
<table>
<thead>
<tr>
<th>SSN</th>
<th>FName</th>
<th>Lname</th>
<th>Address</th>
<th>PhNo</th>
</tr>
</thead>
<tbody>
<tr>
<td>12345600</td>
<td>John</td>
<td>Doe</td>
<td>111, E. Mocking Bird Lane</td>
<td>555-333</td>
</tr>
<tr>
<td>12346500</td>
<td>Bob</td>
<td>Smit</td>
<td>2, W. Broadway</td>
<td>555-234</td>
</tr>
<tr>
<td>13269500</td>
<td>John</td>
<td>Doe</td>
<td>1, E. Speedway</td>
<td>555-989</td>
</tr>
</tbody>
</table>

CUSTOMER Relation

<table>
<thead>
<tr>
<th>IDNO</th>
<th>First</th>
<th>Last</th>
<th>Address</th>
<th>PhNo</th>
</tr>
</thead>
<tbody>
<tr>
<td>500</td>
<td>John</td>
<td>Doe</td>
<td>1, E. Speedway</td>
<td>989</td>
</tr>
<tr>
<td>600</td>
<td>John</td>
<td>Doe</td>
<td>111, E. Mocking Bird Lane</td>
<td>333</td>
</tr>
<tr>
<td>101</td>
<td>Jane</td>
<td>Smit</td>
<td>5000, Melrose Avenue</td>
<td>353</td>
</tr>
<tr>
<td>500</td>
<td>Bob</td>
<td>Smit</td>
<td>2, W. Broadway</td>
<td>234</td>
</tr>
</tbody>
</table>

BORROWER Relation

Fig. 2.9 Instance-level Integration Problems
inherently dynamic in nature.

**Classification based on data model of input schemas**

Strategies for schema integration are highly dependent on the semantics conveyed by the local schemas. Since this is directly related to the type of model used, one can classify methodologies based on the data model used to represent the local schemas.

1. **Relational model based approaches**: The earliest schema integration methodologies used relational models to represent the local schemas (Al-Fedaghi and Schuermann, 1981; Casanova and Vidal, 1983). The drawback of using a relational model is the limited expressive power of the model which results in inadequate semantics being captured by the schemas. Hence, more recent conceptual/view level integration mechanisms have adopted models with more expressive semantics such as semantic and object-oriented models. However, the widespread existence of relational databases, the simplicity of the relational model as well as the existence of a powerful query formulation language seem to make the relational model and relational databases ideal starting points for new research prototypes. As a result, the relational model has been the choice of researchers developing prototypes of heterogeneous database systems. For example, the prototypes and methodologies described
in Templeton et. al. (1987), Deen et. al. (1987), Chung (1990), Kim and Seo (1991) all use relational models. Relational models and databases are also the choice of researchers attempting to solve heterogeneity problems at the data level (DeMichiel, 1989; Chatterjee and Segev, 1991; Prabhakar et. al., 1993).

2. Semantic model based approaches: These approaches use variants of the Entity-Relationship model to represent local schemas as well as the integrated schemas. Larson et. al. (1989), Spaccapietra et. al. (1992), and Shoval and Zohn (1991) are examples of methodologies that belong to this category. The primary reason for the use of semantic models is that these models are capable of expressing richer semantics than the relational model, which can then be exploited during schema integration. Since semantic models are most commonly used to represent views and conceptual schemas, most of these methodologies fall into the user view/conceptual schema level categories of the previous classification.

3. Object oriented model based approaches: These approaches use an object oriented model to represent the local schemas. The reason we classify these approaches separately is that unlike semantic model based approaches, some of the methodologies in this category attempt to integrate methods along with schemas. They also deal with integration of
complex attributes and object hierarchies, issues which are not dealt with in the semantic model based approaches. Most of these methodologies also fall into the view integration/conceptual schema integration category presented above. Examples of research belonging to this category include, Gotthard et. al. (1992), Bertino (1991), Thieme and Siebes (1993), Czejdo and Taylor (1991) and Kaul et. al. (1990).

4. Logic-based approaches: Logic based approaches to schema integration have recently begun to appear in the literature. This represents a natural step in the development of schema integration methodologies, because first-order logic has been shown to be capable of representing the semantics of relational databases in a formal manner. Using a logic-based approach also provides the capability to capture more semantics than is possible using semantic models. For example, logic based models allow us to express semantic integrity constraints. Semantic integrity constraints are explicit user-defined integrity constraints that have been found useful in query transformations (Shekhar et. al., 1993). Whang et. al. (1991) also note that it is easier to translate relational schemas to logic based schemas than to semantic models. Whang et. al. (1991) and Johannesson (1993) present logic-based approaches to schema integration.
Tables 2.1 and 2.2 present a classification of the methodologies in tabular format. Table 2.1 explains how to identify the particular abstraction level or data model category in which a particular methodology fits. Each row provides a list of methodologies that use a particular class of data models, and each column lists the methodologies that belong to a particular abstraction level. Table 2.2 provides the same information in a slightly different manner. Each row in the table identifies a methodology and provides details about the methodology. The final column in table 2.2 indicates the type of model used to represent the output of the integration process and whether this output is an integrated schema or a view.

In the next section, we summarize the key research efforts that have contributed to the interschema relationship identification (IRI), integrated schema generation (ISG) and schema mapping generations steps of the schema integration process. We pay particular attention to methodologies for IRI and identify shortcomings in existing approaches to IRI. It is appropriate to classify IRI techniques based on the abstraction level at which they operate. This is because IRI techniques generate relationships among objects based on the semantic knowledge available, and the abstraction level defines the nature of semantic knowledge (about the databases) available. Hence, our discussion of IRI techniques focuses on methodologies belonging to these two categories. On the other hand,
<table>
<thead>
<tr>
<th>Abstraction</th>
<th>Data</th>
<th>Conceptual</th>
<th>Data</th>
</tr>
</thead>
<tbody>
<tr>
<td>Logic</td>
<td></td>
<td>Whang et. al. (1991), Johannesson (1993)</td>
<td></td>
</tr>
</tbody>
</table>

Table 2.1 Two-way Classification of Schema Integration Methodologies
<table>
<thead>
<tr>
<th>Methodology</th>
<th>Abstraction Level</th>
<th>Data Model</th>
<th>Output Data Model/Type of Representation</th>
</tr>
</thead>
<tbody>
<tr>
<td>Templeton et. al. (1987)</td>
<td>Conceptual</td>
<td>Relational</td>
<td>Relational/View</td>
</tr>
<tr>
<td>Whang et. al. (1992)</td>
<td>Conceptual</td>
<td>Logic-based</td>
<td>Logic/View</td>
</tr>
<tr>
<td>Gotthard et. al. (1992)</td>
<td>View</td>
<td>Object-Oriented</td>
<td>Object-Oriented/View</td>
</tr>
<tr>
<td>Shoval and Zohn (1991)</td>
<td>View</td>
<td>Semantic</td>
<td>Semantic/View</td>
</tr>
<tr>
<td>Prabhakar et. al. (1993)</td>
<td>Data</td>
<td>Relational</td>
<td></td>
</tr>
<tr>
<td>Chatterjee and Segev (1991)</td>
<td>Data</td>
<td>Relational</td>
<td></td>
</tr>
<tr>
<td>DeMichiel (1989)</td>
<td>Data</td>
<td>Relational</td>
<td></td>
</tr>
<tr>
<td>Ahmed et. al. (1991)</td>
<td>Conceptual</td>
<td>Object-Oriented</td>
<td>Object-Oriented/View</td>
</tr>
<tr>
<td>Kaul et. al. (1990)</td>
<td>Conceptual</td>
<td>Object-Oriented</td>
<td>Object-Oriented/View</td>
</tr>
<tr>
<td>Bertino (1991)</td>
<td>Conceptual</td>
<td>Object-Oriented</td>
<td>Object-Oriented/View</td>
</tr>
<tr>
<td>Czejdo and Taylor (1991)</td>
<td>Conceptual</td>
<td>Object-Oriented</td>
<td>Object-Oriented/View</td>
</tr>
<tr>
<td>Kim et. al. (1991)</td>
<td>Conceptual</td>
<td>Object-Oriented/Relational</td>
<td>Object-Oriented/View</td>
</tr>
<tr>
<td>Johannesson</td>
<td>Conceptual</td>
<td>Logic-based</td>
<td>Logic/Schema</td>
</tr>
</tbody>
</table>

Table 2.2 Methodologies for Schema Integration
ISG techniques are dependent on the model level classification. This is logical because the characteristics of the data model used, its semantics and the heterogeneity in the representation of the underlying databases that use these models are the primary factors affecting the ISG process. Hence, the discussion on integrated schema generation is centered around the data model used. Finally, schema mapping generation is relevant only in the context of conceptual model based approaches. Thus, the discussion on schema mapping generation focuses on the differences between the schema restructuring and view generation approaches.

2.2.2 Interschema Relationship Identification (IRI)

The objective of this phase is to identify objects in the underlying schemas that may be related and to classify the relationships among them.

*Conceptual Schema based approaches*

Conceptual schema based IRI techniques utilize a two phase process consisting of: a) identifying objects that are related and b) classifying the relationships among these objects.
The first phase requires that the intended semantics of objects in databases be extracted and objects that are semantically related be identified. Once a potential set of related objects has been identified, the second phase involves classifying these relationships into various categories. The inability of existing data models to convey the true semantics of the databases causes this phase to require extensive interaction with a designer or expert who has an understanding of the applications and domains served by the database.

Conceptual schema based approaches use the knowledge conveyed by the various schematic constructs to deduce relationships among objects. Entity classes, attributes, and relationships represent the primary schematic constructs that can be analyzed to arrive at these relationships. Larson et. al. (1989) describe various characteristics of an attribute that can be used to establish relationships among attributes of an entity. Properties include: uniqueness property, lower and upper cardinality constraints, the domain of the attribute, static and dynamic integrity constraints, security constraints, the set of allowable operations on the attribute and the scale (interpretation) of the attribute. The authors suggest that attributes be compared on these properties and provide definitions for assessing their degree of equivalence based on these properties.
The result of analyzing the schemas is that objects that are semantically related are identified. However, it is necessary not only to identify but also to classify the relationships among these objects. The classification generated is dependent on the methodology used.

1. Larson et. al. (1989) generate four types of equivalences between attributes. These are: a \textit{equal} b, a \textit{contains} b, a \textit{contained-in} b and a \textit{overlap} b. They go on to define five types of relationships among entities and relationships, each of which can be derived based on attribute equivalences of key attributes. These relationships include: A \textit{equal} B, A \textit{contains} B, A \textit{contained-in} B, A \textit{overlap} B and A \textit{disjoint} B. Users are asked to specify one of these types of relationships for every entity/relationship whose attributes have equivalence relationships specified on them.

2. DeSouza (1986) and Hayne and Ram (1990) describe relationships among objects in terms of degrees of similarity and dissimilarity. Such a classification is amenable to automating the interschema relationship determination process. It is assumed that the objects and their degrees of similarity will be presented to the schema integrator for generation of relationships described in step 1 above.
3. Song et. al. (1992) suggest that semantic relationships among database objects be classified into four types: weak semantic relation, compatible semantic relation, equivalent semantic relation and mergeable semantic relations. These relations are defined in terms of property sets and key property sets (i.e., attributes and key attributes). A weak semantic relation implies overlap in property sets, compatible semantic relation implies an overlap in key property sets, equivalent relation implies identical key property sets and a mergeable relation implies identical property sets.

**Data based approaches**

The objective of most IRI techniques using the data-based approach is to determine instances of entity classes in different databases that refer to the same real-world entity. The simplest approach assumes that relations from different databases possess a common key. Hence, tuples that have a common key value (DeMichiel, 1989) identify the same real-world entity. However, as noted in Prabhakar et. al. (1993) a common key may not always be available. This is referred to as the key equivalence problem. Pu (1991) present a probabilistic key equivalence technique that in essence evaluates the probability that two tuples refer to the same real world entity. Chatterjee and Segev (1991) suggest that one compare not only keys but all attribute values in tuples to compute the probability that two
tuples refer to the same real world entity. Finally, Prabhakar et. al. (1993) present an extended definition of key equivalence based on the concept of instance-level functional dependencies to identify tuples that may be equivalent. All of the techniques presented above are intended to solve the problem of referencing tuples from heterogeneous database relations.

Table 2.3 presents a summary of the IRI techniques described above. Each row in the table identifies the type of data model used to represent (translated) local schemas, the inputs to the system, the schematic properties used for interschema relationship identification and the outputs produced by this process. Each row also identifies the unique feature of each of these techniques and the degree to which these techniques are automated. All the techniques use knowledge from a single abstraction level (conceptual schema or data level) to identify relationships among objects.

The use of multiple sources of knowledge to aid in interschema relationships identification is also beginning to receive some attention in the literature. Li and Clifton (1994) present an automated technique for determining attribute equivalence that combines schematic and data-level knowledge. Their method uses discriminators from the schema level, such as, data type of the attributes, their length and the existence of constraints, and data-level, such
<table>
<thead>
<tr>
<th>System/Paper</th>
<th>Data Model</th>
<th>Schematic Objects Used</th>
<th>Input Format</th>
</tr>
</thead>
<tbody>
<tr>
<td>SIS (deSouza, 1986)</td>
<td>Abstract Conceptual Schema</td>
<td>Entities, Attributes and Relationships</td>
<td>Schemas</td>
</tr>
<tr>
<td>Sheth et. al. (1988)</td>
<td>Entity Category Relationship Model (ECR)</td>
<td>Attributes</td>
<td>Attribute Equivalence Assertions</td>
</tr>
<tr>
<td>BERDI (Sheth and Marcus, 1992)</td>
<td>Entity Category Relationship Model (ECR)</td>
<td>Attributes</td>
<td>Attribute Hierarchy</td>
</tr>
<tr>
<td>MUVIS (Hayne and Ram, 1990)</td>
<td>Semantic Data Model (SDM)</td>
<td>Entities and Attributes</td>
<td>Schemas</td>
</tr>
<tr>
<td>Gotthard et. al. (1992)</td>
<td>Complex Entity-Relationship Model (CERM)</td>
<td>Object Classes, Attributes and Roles</td>
<td>Schemas</td>
</tr>
<tr>
<td>Shoval and Zohn (1991)</td>
<td>Binary Relationship Model</td>
<td>Entity</td>
<td>Schemas</td>
</tr>
</tbody>
</table>

Table 2.3 Summary of Interschema Relationship Generation Techniques
<table>
<thead>
<tr>
<th>System/Paper</th>
<th>Outputs</th>
<th>Extent of Automation</th>
<th>Key Feature</th>
</tr>
</thead>
<tbody>
<tr>
<td>SIS (deSouza, 1986)</td>
<td>Similarity Values</td>
<td>Completely Automated</td>
<td>Use of multiple properties with varying weights for each schema object</td>
</tr>
<tr>
<td>Sheth et. al. (1988)</td>
<td>Relationship Assertions - Entities, Attributes and Relationships</td>
<td>Manual w/system assistance</td>
<td>Use of attribute equivalence as basis for generating relationships among other objects</td>
</tr>
<tr>
<td>BERDI (Sheth and Marcus, 1992)</td>
<td>Relationship Assertions - Entities, Attributes and Relationships</td>
<td>Partially Automated with extensive system assistance</td>
<td>Use of attribute hierarchy as basis for generating relationships among other objects</td>
</tr>
<tr>
<td>MUVIS (Hayne and Ram, 1990)</td>
<td>Similarity Values - Only Equivalences</td>
<td>Automated</td>
<td>Use of multiple properties for comparing each entities and attributes</td>
</tr>
<tr>
<td>Gotthard et. al. (1992)</td>
<td>Predicates asserting equivalences among object classes, attributes and roles</td>
<td>Automated w/Manual Input at end</td>
<td>Interschema Relationship Identification in an Object Oriented Model</td>
</tr>
</tbody>
</table>

Table 2.3 (Contd.) Summary of Interschema Relationship Generation Techniques
as, patterns in numeric and character fields, to determine equivalence among the attributes. These discriminators are detected automatically by their system and used as input to a neural network that categorizes the attributes using a self-organizing map algorithm. Castellanos (1993) presents a technique that is based on schematic enrichment of local schemas. An enrichment technique is used to enhance the semantics of the database before the conversion of local schemas into the common data model. The types of knowledge that their methodology tries to extract from the local schemas are keys, functional dependencies, inclusion dependencies and exclusion dependencies. This knowledge is gained by examining the schema as well as the extension of the database.

However, the use of integrity constraints as a source of knowledge for determining interschema relationships has received but a passing reference in the literature. Larson et al. (1989) suggest that integrity constraints are one of the many sources of knowledge about an attribute that contributes towards the definition of the real-world states of an attribute. However, despite the fact that semantic integrity constraints convey significant information about the semantics of the data in the underlying database (King, 1981; Seigel et al., 1988), their use in interschema relationship identification has not been studied in the heterogeneous database literature. This dissertation is to our knowledge the first attempt
to study the development of relationships among database objects based on the knowledge conveyed by the integrity constraints specified on the databases. We describe how both schematic and integrity constraint knowledge can be used for interschema relationship identification. We believe that by using multiple source of knowledge one can arrive at interschema relationships among database objects that can be comparable in accuracy to interschema relationships asserted by the user for a majority of objects in the databases being integrated, thus reducing designers' burden during this phase of schema integration.

2.2.3 Semantic Heterogeneity

Once the underlying databases have been analyzed to identify interschema relationships the next problem that needs to be addressed is that of semantic heterogeneity. Semantic heterogeneity refers to the differences in semantics of the same real world "concept" as represented in different databases. This heterogeneity is caused because of the loss of information resulting from representing the real world concepts using the abstractions available in the database model being used. One of the problems posed by heterogeneous databases is to "determine a posteriori from incomplete and heterogeneous representations if
symbols from multiple databases map to the same underlying reality" (Gangopadhyay and Barsalou, 1991). This was the focus of the IRI techniques presented above.

Chatterjee and Segev (1991a) divide semantic heterogeneity among attributes into two classes: structural incompatibility and semantic incompatibility. Structural incompatibility results from differences in type, formats, units and granularity. Sources of semantic incompatibility are, synonyms, homonyms, incomplete information and surrogates.

Kim and Seo (1991) divide semantic heterogeneity (in the context of relational databases) into two classes: schema conflicts and data conflicts. Schema conflicts are further classified into table-versus-table conflicts, attribute-vs-attribute conflicts and table-versus-attribute conflicts. Data conflicts can arise because of wrong data and different representations of the same data.

Gangopadhyay and Barasalou (1991) provide a broader classification of semantic heterogeneity and provide the following categories: discrepancies in data definition, discrepancies in data structures when the underlying data models are the same and discrepancies in data structures when the underlying data models are different.
Urban and Wu (1991) describe several forms of heterogeneity that can be found in heterogeneous databases. The first is due to the structural differences in representation caused by the different data models used by the underlying databases. The second form is caused due to representational differences. For example, an object represented as a relation in one database may be represented as an attribute in another database. The third form of heterogeneity mentioned in the paper arises when different units are used to represent data. Yet another form arises due to differences in interpretation of the data from one database to the next.

The process of resolving semantic heterogeneity in the component schemas and generating an integrated schema is the focus of integrated schema integration techniques. The next section presents details of several integrated schema generation methodologies.

2.2.4 Integrated Schema Generation

The objective of this phase of schema integration is to develop an integrated representation that reflects the semantics of the underlying databases.

*Semantic Model approaches*
The discussion presented below focuses on techniques used in schema restructuring methodologies. This is because, in view generation methodologies, the techniques are driven by the capabilities of the data language used. The primary issue addressed by the methodologies is that of generating an integrated representation that reflects the semantics of the underlying databases. The primary technique adopted is the creation of generalization/specialization relationships in the integrated schema. The schema in Figure 2.6 is an example. Larson et. al. (1989) present an approach to schema integration that is based on the premise that any pair of objects whose identifying attributes can be integrated, can themselves be integrated. The work presented in this paper extends Mannino and Effelsberg (1984)'s work. As discussed in the previous section the authors define four types of relationships between attributes: equivalent, contains, contained-in and overlaps. Database designers are responsible for identifying the type of relationship between different pairs of attributes. Entity class and relationship equivalence are then defined in terms of relationships between identifier attributes and classified into four categories: equal, contains/contained-in, overlap and disjoint. Rules for integrating entity classes and relationships belonging to each category are presented along with rules for integrating attributes. Categorizing the relationships among attributes, entity classes and relationships is
the primary contribution of this research, and is used to a certain extent by most methodologies that have appeared since.

Larson et. al. (1989) presented general guidelines for transforming related objects into objects in the integrated schema. However, since an object is typically linked with other objects in the schema, such a transformation may require that changes be made to the links between objects in order to generate a correct integrated schema. Elmasri and Navathe (1984) and Navathe et. al. (1986) present techniques for integrating entity class and relationship pairs that may have one of the five possible relationship pairs identified above. They present their work in the context of an Entity-Category Relationship model. It should be noted that these rules implicitly require that naming conflicts be handled by the methodologies. Naming conflicts manifest themselves in two forms: 1) the first occurs when two unrelated objects share the same name. In this case, one of the objects needs to renamed and 2) two equivalent objects, have the same name. In this case, a decision has to be made as to which name should be used in the integrated schema. The work presented in these papers can only deal with relationships among objects represented using the same schematic construct.
Structural conflicts arise when two related objects are defined using different data model constructs or using the same construct with different properties. For example, in Figure 2.6, IDNO and SSNO represent related attributes, but they may have different properties. Most methodologies for schema integration address the structural conflict problem. Spaccapietra and Parent (1991) and Spaccapietra et. al. (1992) present a methodology for integration of any two types of objects. They view a schema as a graph with edges and nodes. Relationship between objects in the schema are specified using correspondence assertions. The correspondence assertions presented in this paper can be regarded as extensions of the concepts of object and relationships equivalence presented in Larson et. al. (1989). The ability to specify relationships between objects of different types is a unique contribution of Spaccapietra et. al. (1992). The authors provide a comprehensive set of rules for integrating objects, specifying in detail the actions to be taken to integrate objects related through assertions. For example, one of the issues dealt with is the resolution of differences in domains and cardinalities of attributes that need to be integrated. Another contribution of their methodology is the specification of rules to integrate paths. A path between two objects can be visualized as a set of links in a schema. Hence, such a path may include one or more entity classes, their attributes and relationships. The rules for integration of paths in two schemas thus involves the simultaneous integration of multiple schema objects (objects in the path). This is different from most other methodologies which
typically specify integration rules for a pair of objects. A methodology for resolving structural conflicts is also presented in Bouzeghoub and Comyn-Wattiau (1990). Comyn-Wattiau and Bouzeghoub (1993) deal with another problem, that of integrating differing constraints such as, cardinality constraints and key and functional dependencies, during schema integration. Many of the early methodologies summarized in Batini et. al. (1986) also presented guidelines for resolving structural conflicts in a variety of models. For example, El-Masri and Wiederhold (1979) address the problem of integration for structural models; Yao et. al. (1982), Dayal and Hwang (1984) and Motro and Buneman (1981) present methodologies for functional models and Batini and Lenzirini (1984) focus on the Entity-Relationship model.

Object-Oriented approaches

Object-Oriented approaches deal with all the issues relevant to semantic-model based approaches as well as some additional ones. Gotthard et. al. (1992) present a description of a methodology that utilizes object-oriented schemas as the input schemas to a view integration algorithm. The details of their methodology and toolkit are presented in section 2.2.4.
Additional issues that are dealt with by object-oriented techniques are two fold:

1. They develop mechanisms for integrating class hierarchies. A class hierarchy represents a set of classes participating in generalization/specialization relationships, such as \textit{Borrower ISA Customer}. Such a class hierarchy can be complicated by the fact that an attribute in one of the classes may have another entity class as its domain, leading to recursive hierarchies. Thieme and Siebes (1993) present techniques for integrating class hierarchies on the basis of semantic and structural equivalence of classes in the hierarchy. They define structural equivalence of classes based on type equivalence. Semantic equivalence is defined in terms of functional equivalence as determined by the methods specified in the classes being compared. The key contribution of this work, is in recognizing the additional complexity introduced by object-oriented constructs (in the form of potentially recursive class hierarchies) and in presenting a technique for integrating them. Sull and Kashyap (1992) describe a schema integration methodology that also integrates object-oriented schemas. The aim of the paper however is to present a methodology that is self-organizing, i.e., updates to local schemas can be propagated unambiguously to the integrated schema. The authors present strategies to map from relational schemas and rulebases to object-oriented schemas. They then present strategies to integrate these object-oriented schemas. As was the case with
the other object-oriented schema integration methodologies the primary emphasis is on integrating class hierarchies.

2. The methodologies have to deal with integration of methods. Two cases arise here a) new methods may need to be defined for the integrated view, and b) preexisting methods in the entity classes being integrated may need to be integrated. Since, existing methods may differ in name and parameters, techniques for resolving these differences need to be developed. The methodology presented in Bertino (1991) presents some techniques for incorporating method integration during structural integration. Kaul et al. (1990) present techniques for inheriting constraints (in the integrated class) from the classes being integrated without making changes to the existing method definitions.

Logic-based approaches

Work on using logic-based approaches for schema integration is still in its early stages. Thus, unlike the previous two categories, no specific thread of research can be identified. However, as mentioned previously, the desirability of the logic-based approaches stems from the powerful semantics conveyed by a logic based representation as well as the formal nature of such a representation.
Whang et. al. (1991) describe a rule-based approach to schema integration. Each of the local schemas being integrated is represented as a schema using first order logic. These databases constitute the extensional databases (EDB). The integrated schema is then defined by a set of first order logic rules applicable to the EDBs. In other words, the integrated schema is a set of intensional database (IDB) relations. The power of this approach lies in its use of a logic-based representation which is capable of capturing all the semantics of a semantic model. At the same time, the existence of a logic-based integrated schema makes it conducive to query processing because most SQL-based languages can be translated easily into logic-based queries. Also, since the integrated schema is generated using horn clauses, a query issued against an integrated schema relation can be broken down using resolution until the query is expressed in terms of the base relations. The authors also mention that the semantics inherent in a logic-based schema can be exploited to facilitate semantic query optimization. Johannesson (1993) describes the importance of schema transformations in view integration. The basic problem addressed here is that of merging semantically equivalent yet structurally different concepts. They suggest that the schemas to be integrated be standardized by applying schema transformations prior to integration. They present their work in a logic-based modeling context. The paper presents transformations algorithms for partial attributes, m-m attributes, lexical attributes and attributes with fixed ranges. In
addition, transformation algorithms for lattice structures and stable subtypes are presented. Again, it is interesting to note that a logic-based model is used to define these transformations, although the underlying schemas being transformed maybe relational or object-oriented in nature. The primary reason seems to be that a logic-based representation allows for formal definitions of the transformations mentioned above.

Table 2.4 presents a summary of the integrated schema generation techniques discussed above.

2.2.5 Schema Mapping Generation

This step is performed concurrent to both the schema translation and integrated schema generation steps of a schema integration methodology. Developing mappings during schema translation is necessary for issuing correct queries to the local databases. This mapping may be stored as a dictionary at each local database. The mapping generated during the integrated schema generation process, maps an object in the integrated schema to objects in the local schemas being integrated. If the schema restructuring approach is used, then this mapping is generated as the integrated schema is being generated and stored in a global directory/dictionary (Templeton et. al., 1987). If the view generation approach is used, this
mapping is usually defined as part of the query/statement used to create the new view. Once
again the mapping information is usually stored in a global catalog (Ahmed et. al., 1991).

<table>
<thead>
<tr>
<th>Paper</th>
<th>Data Model</th>
<th>Inputs</th>
</tr>
</thead>
<tbody>
<tr>
<td>Larson et. al. (1989)</td>
<td>Entity Category Relationship Model</td>
<td>Equivalence Assertions defined by users</td>
</tr>
<tr>
<td>Spaccapietra et. al. (1991)</td>
<td>Multiple-Models</td>
<td>User-defined Correspondence Assertions</td>
</tr>
<tr>
<td>Shoval and Zohn (1991)</td>
<td>Binary-Relationship</td>
<td>Homonym and Synonym Assertions</td>
</tr>
<tr>
<td>Gotthard et. al. (1992)</td>
<td>Complex-Entity Relationship</td>
<td>Factual Predicates</td>
</tr>
<tr>
<td>Geller et. al. (1991)</td>
<td>Dual</td>
<td>Not explicitly mentioned</td>
</tr>
<tr>
<td>Whang et. al. (1991)</td>
<td>Logic-based</td>
<td>Not explicitly mentioned</td>
</tr>
<tr>
<td>Johannesson (1991)</td>
<td>Logic-based</td>
<td>Transformed schemas</td>
</tr>
<tr>
<td>Thieme and Siebes (1992)</td>
<td>Object-Oriented</td>
<td>Object-Oriented Schemas</td>
</tr>
<tr>
<td>Comyn-Wattiau and Bouzeghoub (1992)</td>
<td>Semantic Model</td>
<td>Domain and Cardinality Constraints</td>
</tr>
</tbody>
</table>

Table 2.4 Summary of Integrated Schema Generation Techniques
2.2.6 Schema Integration Toolkits

The methodologies presented in the previous sections describe the general principles that can be used to achieve schema integration. It is clear from this discussion that schema integration is a complex and time consuming process. The automation of this process is thus desirable. However, automation of the process presents a number of challenges. Sheth and Gala (1991) note that the schema integration process cannot be completely automated. In fact, substantial interaction with designers is required during all phases of the schema integration process. This is primarily because of the fact that schema integration attempts to understand the semantics of existing databases using representations that cannot completely capture the intended semantics of the data. However, it is possible to automate schema integration to the extent that such tools can be used to relieve the user of mundane tasks, thus reducing the amount of user interaction. This section presents a description of tools that automate portions of the schema integration process.

One of the first efforts to automate any phase of the schema integration process was DeSouza (1986). This work focused on interschema relationship identification. The author presents an expert system designed to integrate conceptual schemas defined using the Abstract Conceptual Schema (ACS) (Stocker and Cantie, 1983). A set of functions (called
resemblance functions) that can be used to compare objects in the schemas are defined. These functions use both names and structure to estimate the resemblance between constructs. Each resemblance function also has a weight associated with it. This weight indicates the relative importance that the user would like to place on the resemblance function. For example, if having similar attributes is the most important criteria, then the weight associated with that function would be high. Objects whose computed values of similarity fall above a certain threshold are presented to the user as being possibly similar.

The significant contributions of this paper to interschema relationship identification are:
a) it uses multiple properties of a database object in analyzing schema objects for similarity; and b) it associates variable weights with each of these properties. The drawback is that their methodology is specific to ACS schemas. Also, this paper does not deal with the integrated schema generation step.

Sheth et. al. (1988) present a tool that leads users/designers through the schema integration process. Users are led through a five-step process: Schema Information Collection, Equivalence Class Creation and Deletion (Entities and Categories), Equivalence Class Creation and Deletion (Relationships), User Assertions (entities and categories) and User Assertions (Relationships). In the schema information collection step the schemas to be integrated are input to the tool in the form of Entity-Category Relationship schemas. Users
are asked to specify relations among attributes for entities and relationships that the user thinks may be related. Once these equivalences have been specified, they are used to generate an ordered list of object (entity and relationship) pairs. The ordering indicates the likelihood that an object pair may need to be integrated. Users are then required to analyze this ordered list and specify one of five types of relationships between the objects. These relationships are again based on Larson et. al. (1989) and include: equal, contained_in, contains, disjoint but integratable, disjoint and non-integratable. The toolkit presented in this paper requires a large amount of interaction with the users/designers. Users can only specify equivalence assertions among attributes, limiting the amount of semantic information that can be captured. This deficiency is addressed in the next generation toolkit BERDI (Sheth and Marcus, 1992). BERDI allows users to define relationships among objects that belong to a potentially related set of entities, called entity clusters. The authors also allow users to assert three types of relationships among attributes: equivalence, inclusion and disjoint. The system also provides mechanisms for generating attribute hierarchies based on these relationship assertions among attribute pairs. However, even with all these modifications, the burden of identifying related entity clusters is still placed on the user.
Hayne and Ram (1990) present some techniques for automated interschema relationship identification during schema integration. In their methodology schemas are represented using a variant of the Semantic Data Model (SDM) (Hammer and McLeod, 1981). They compute similarities of entity classes based on names and the properties of the associated attributes.

Shoval and Zohn (1991) present a methodology and toolkit for integrating schemas represented using the Binary-Relationship model. A binary integration strategy is used. The methodology and toolkit reported in Shoval and Zohn (1991) provides the bare minimum of support for automatic interschema relationship identification in the form of being able to detect homonym conflicts (based on name matches) automatically. They present techniques for resolving naming and structural conflicts. Naming conflicts dealt with in the methodology include homonym or synonym conflicts. Structural conflicts dealt with in the methodology include: a) type conflicts, e.g., object type-relationship conflicts, b) dependency conflicts which arise when two relationships have different cardinality constraints, c) key conflicts arising from two object types having the same identifier, d) constraint conflicts arising from differing inter-relationship constraints and e) hierarchy conflicts which occur when similar object types in different schemas have different hierarchy links. Although, the methodology presented is comprehensive for the Binary-
Relationship model, this is primarily because the number of possible cases in the binary-relationship model is limited. Hence, there isn’t much generalizability of the methodology to more common models that have been derived from the E-R model.

Gotthard et. al. (1992) present a view integration methodology for schemas specified using an object-oriented model. Their methodology presents some techniques for automated interschema relationship identification. The schemas in this methodology are represented using CERM (complex entity-relationship model), an object-oriented model. They introduce the concept of assumption predicates which specify possible similarities between structures in the schemas to be integrated. Their functions for computing assumption predicates use names and intensions as the primary means of arriving at these assumption predicates. For example, attributes are compared on their names and domains. These assumption predicates are then presented to the user for confirmation. The input to the integrated schema generation step is a set of factual predicates. Each factual predicate specifies relationships among similar object types and relationships. For example, to specify relationships between two object types a and b, designers can use one of the following four predicates: equal\(_{\text{obj}}(a,b)\), subset\(_{\text{obj}}(a,b)\), arbitrary\(_{\text{obj}}(a,b)\) and disjoint\(_{\text{obj}}(a,b)\), which specify equal, subset, overlap and disjoint relationships respectively. A similar set of predicates is available for specifying relationships among relationship types. For each of these predicates
the authors define integration primitives that when applied will generate an integrated object of the appropriate type. The authors deal with structural conflicts by defining a set of transformation primitives that will make the objects in question integration compatible. The primary contribution of this work is that it defines a methodology applicable to integrating object-oriented schemas. The methodology suffers from two drawbacks. First, their methodology is specific to the (CERM) object-oriented model. Second, the assumption predicates generated specify only equivalence relationships among the various types of objects.

All of the techniques described above fall into the category of schema restructuring mechanisms operating at the conceptual schema level (Table 2.1).

Ahmed et. al. (1991) describe the Pegasus system, a system that utilizes a view generation mechanism and operates at the conceptual schema level. Integration is achieved by defining new objects using a Heterogeneous Object Structured Query Language (HOSQL) query. The language is used for generating object-oriented classes from existing database relations. Users can also use the language to create supertypes of classes defined on the underlying databases. Differences in domains and schematic representations of similar objects are resolved through the definition of mapping functions using HOSQL.
Kaul et. al. (1990) describe the architecture of ViewSystem, an object oriented environment that facilitates integration of heterogeneous information bases using a view generation technique. The intermediate schemas in ViewSystem are represented using VODAK, an object oriented language. The system provides class constructors, e.g., specialization and generalization constructors which can be used to derive new classes from existing classes. Techniques for inheriting methods (in the new class) from existing classes are also described. An object-oriented and set-oriented query language is also provided as part of the environment.

Table 2.5 presents a summary of the tools discussed above. The table shows the phases of the schema integration process addressed by each of the toolkits, the degree to which each of these phases is automated, the data model used by each toolkit, and the key feature of each toolkit. It should be noted that a majority of the toolkits described above are prototype systems which have been developed as a proof of concept for techniques presented in the papers. The system presented in Ahmed et. al. (1991) is the closest to being a commercial toolkit. The toolkits cover a broad spectrum of the classification presented in section 2.2.2 including view integration approaches, schema restructuring, and view generation approaches to schema integration.
All of the methodologies and toolkits presented above discuss principles involved in merging semantically equivalent structures. The ultimate objective of this step is the resolution of conflicts among the local databases and the generation of a unique representation of the underlying databases. However, such a representation does not convey all the semantics of the underlying databases. In particular, it does not reflect the semantics of the actual data values stored in the underlying databases. In this dissertation, we present a methodology that integrates integrity constraints along with individual schemas in a heterogeneous database environment. Such a process will generate a more comprehensive representation of the underlying databases. In addition, the additional semantics conveyed by these integrity constraints can be used to optimize queries. This process is referred to as semantic query optimization. The next section presents a description of literature related to semantic query optimization.
<table>
<thead>
<tr>
<th>Methodology</th>
<th>Abstraction Level</th>
<th>Data Model</th>
<th>Interschema Relationship Identification</th>
<th>Integrated Schema Generation</th>
</tr>
</thead>
<tbody>
<tr>
<td>SIS (DeSouza, 1986)</td>
<td>Conceptual</td>
<td>Semantic/ Abstract Conceptual Schema</td>
<td>Automated</td>
<td>N/A</td>
</tr>
<tr>
<td>MUVIS (Hayne and Ram, 1990)</td>
<td>View</td>
<td>Semantic/ Semantic Data Model</td>
<td>Automated</td>
<td>Automated/ based on Navathe et. al. (1986)</td>
</tr>
<tr>
<td>Shoval and Zohn (1991)</td>
<td>View</td>
<td>Semantic/ Binary Relationship Model</td>
<td>Partially Automated w/ extensive Manual Input</td>
<td>Automated / focus on resolution conflicts in schemas</td>
</tr>
<tr>
<td>Gotthard et. al. (1992)</td>
<td>View</td>
<td>Object-Oriented / Complex Entity Relationship Model</td>
<td>Partially Automated w/ user input</td>
<td>Automated / uses integration primitives</td>
</tr>
<tr>
<td>Kaul et. al. (1990)</td>
<td>Conceptual</td>
<td>Object-Oriented/ VODAK</td>
<td>Manual</td>
<td>Integration achieved by defining views using specialized</td>
</tr>
<tr>
<td>Ahmed et. al. (1991)</td>
<td>Conceptual</td>
<td>Object-Oriented / HOSQL</td>
<td>Manual</td>
<td>Integration achieved by defining views in HOSQL</td>
</tr>
</tbody>
</table>

Table 2.5 Summary of Schema Integration Toolkits
2.3 Semantic Query Optimization

Query optimization is the process of transforming a user query into an equivalent database query that can be processed more efficiently. Research on query optimization can be classified into two categories: syntactic and semantic query optimization. Syntactic query optimization attempts to restructure a user query such that the various operations that the query need to be performed on a database can be performed more efficiently. However, in transforming the query care must be taken to ensure that the reordering of operations does not change the nature of the query, i.e., the answers to the new query should be the same as the original query.

The concept of semantic query optimization was introduced separately by Hammer and Zdonik (1980) and King (1981). The basic idea in semantic query optimization is to utilize the knowledge about the semantics of the data to transform user queries into one or more equivalent database queries that are more efficient than the original user query. Integrity constraints (Nicolas, 1982; Ullman, 1988) are the primary source of semantic knowledge utilized by semantic query optimization mechanisms.
2.3.1 Integrity Constraints

Integrity constraints are an important source of knowledge about the semantics of the database they are defined on. The semantics conveyed by the integrity constraints depend on domain elements of the attributes and/or the actual values stored in the database. Integrity constraints in databases can be classified into various categories. Date (1986) classifies constraints into state and transition constraints. State constraints apply to the individual states of the databases. Transition constraints specify how one state of the database can be transformed into another state of the database. Date (1986) also classifies constraints into domain and table constraints. In a domain constraint, the free variables in the constraint range over domains, while in table constraints the variables range over tables. Table constraints can be further classified into single row and multiple row constraints. Single row constraints define constraints that can be evaluated by testing the values in a single tuple. The constraint

\[
\text{PX.City = 'Chicago' } \leftarrow \text{ PX(qty, color, chicago), color='red'}
\]

is an example of a single row constraint. Multiple row constraints involve rows from one or more tables.
One can also classify constraints as being implicit or explicit constraints (Shekhar, et. al., 1993). Functional dependencies and referential integrity constraints are examples of implicit constraints since they are implicit in the definition of the database. On the other hand, explicit constraints specify restrictions on the values that attributes can take or other properties and relationships among attributes.

Researchers in semantic query optimization (SQO) are interested in exploiting these explicit constraints as they provide semantic information that is not available from examining the data model alone. Such explicit constraints are also referred to in the literature by the term semantic integrity constraints (SIC). The major types of semantic integrity constraints that are dealt with in the semantic query optimization literature are: subset constraints, implication constraints and aggregate constraints (Shenoy and Ozsoyoglu, 1989; Ishakbeyouglu and Ozsoyoglu, 1991).

2.3.2 Major approaches to Semantic Query Optimization

The primary issues that distinguish semantic query optimization techniques in the literature are: 1) the query transformation mechanism used by the methodologies and 2) the types of
databases dealt with by the methodologies. Hence, we classify semantic query processing methodologies based on these characteristics.

**Classification based on transformation mechanisms**

Semantic query processing mechanisms can be classified as being static or dynamic in nature. Static or compiled approaches preprocess the set of constraints defined on a database and reduce them into a form that can be readily applied to an incoming query. Chakravarthy et. al. (1990) and An and Henschen (1992) are examples of methodologies that use this approach. Dynamic approaches do not perform preprocessing of any kind and evaluate each query against the set of constraints specified on the database. King (1981), Shenoy and Ozsoyoglu (1989), Bertino and Musto (1992) and Seigel et. al. (1988) are examples of approaches that belong to this category.

1. **Compiled Approaches:** Chakravarthy et. al. (1990) is an example of a semantic query processing mechanism that precompiles the constraints specified on a database to improve query processing efficiency. Their approach to SQO is based on the concept of partial subsumption which in turn is derived from the concept of subsumption in first-order logic (Grant and Minker, 1992). A clause C is said to subsume a clause D if we can construct a
resolution refutation tree that results in a null clause using C as its root and elements of \(-D\) at each step. Partial subsumption is the process of applying the subsumption principle using an integrity constraint instead of C and a database rule in place of D. The objective of partial subsumption (as the name suggests) is not to generate a tree that ends in a null clause. Instead, we generate a resolution refutation tree whose end result is a clause that is simpler than the original constraint C. Such a clause is referred to as a residue.

Semantic compilation is the process of generating such residues by applying each constraint to the underlying database relations. Chakravarthy et. al. (1990) describe a two-phased approach to semantic query optimization that is based on the concept of semantic compilation. First, the integrity constraints specified on the database are compiled with the database definition. The residues resulting from this process are then attached to the existing rules in the database (in our case the relation definitions). A rule/axiom generated using this process is termed a semantically constraint axiom/rule. These axioms can then be used to semantically transform queries.

The functional components of the semantic query transformer identified by Chakravarthy et. al. (1990) are as follows: the Query/Residue modifier that uses all of the semantically constrained axioms to transform the incoming query into a set of queries; the Reducer
which evaluates residues that may have been sufficiently instantiated as a result of the query/residue modification step; the Filter which eliminates residues that will not be useful in transforming the modified query; the Strategiser which is used to prioritize the residues; and the Generator which is used to generate alternative queries based on the priorities established by the strategist.

Table 2.6 presents the primary types of semantic query transformation discussed in the literature. The methodology described in Chakravarthy et. al. (1990) is capable of transforming a query using each of the techniques described in table 2.6. Residues (generated during the semantic compilation phase) are classified into three categories: 1) a goal clause, 2) a unit clause and 3) a horn clause with a nonempty body and a nonempty head. Below, we describe how the various transformations listed in Table 2.6 can be effected using residues from each category:

*Goal clause:* A goal clause is a clause of the form \( \leftarrow x > 200 \). If a residue is a goal clause, it can help query transformation in two ways:

1) the residue can subsume the query. In such a case the query has no answers and hence has need to access the database (access elimination). For example, consider the query
\[ \text{Ships}(x_1, x_2, x_3, x_4, x_5, x_6, x_7), \text{ (} x_5 > 250 \text{) and the residue} \]

\[ \leftarrow x_5 > 200 \]

<table>
<thead>
<tr>
<th>Query Transf. Type</th>
<th>Description of Transformation</th>
</tr>
</thead>
<tbody>
<tr>
<td>Access Elimination</td>
<td>Application of a constraint to a query indicates that a component of a query violates the constraint and can hence never be satisfied</td>
</tr>
<tr>
<td>Index Introduction</td>
<td>Use a constraint to replace an existing restriction on an attribute with a restriction on a indexed (clustered) attribute</td>
</tr>
<tr>
<td>Restriction Introduction</td>
<td>Use a constraint to introduce further restrictions on an already constrained query</td>
</tr>
<tr>
<td>Restriction Elimination</td>
<td>Use a constraint to eliminate a restriction specified (on an unindexed attribute) in the query</td>
</tr>
<tr>
<td>Scan Reduction</td>
<td>Introduce a restriction into a join query which reduces the number of qualifying tuples from the outer relation</td>
</tr>
<tr>
<td>Join Elimination</td>
<td>Use integrity constraints to eliminate a join from the query</td>
</tr>
<tr>
<td>Join Introduction</td>
<td>Introduce a join into a query to make the overall query processing strategy more efficient</td>
</tr>
</tbody>
</table>

Table 2.6 Summary of Different Forms of Semantic Query Optimization
The query asks us to list all tuples in the ships relation in which the value of attribute $x5$ is $> 250$. However, the residue indicates that $x5$ is never greater than 200 in the Ships relation. Thus, the application of the residue introduces a contradiction in the query, because any answer to the query with the condition specified in it will violate the integrity constraint that led to the residue ($\leftarrow x5 > 200$). Hence, the fact that the query cannot have any answers can be determined without accessing the database.

2) the clause can be used to introduce a new constraint on the query especially if the attribute in the clause is a clustered attribute. This is done by converting the goal clause into a unit clause and applying the techniques shown in the next subsection. For example, the goal clause $\leftarrow x < 200$ can be converted into the unit clause $x \geq 200$.

*Unit Clause:* A unit clause is a clause with a null body and single atom in its head. Such a residue can be used to eliminate joins, introduce joins or introduce/eliminate restrictions from a query. For example, given the residue, $x5 > 350$ and the query

$$\leftarrow \text{Ships}(x1, x2, \text{supertanker}, x4, x5, x6, x7), (x5 > 300)$$

the restriction on $x5$ in the query can be eliminated as being superfluous.
Unit clauses can also be used for restriction introduction. For example, given the residue

\[ x_5 > 150 \]

which can be represented as the unit clause \( x_5 \leq 150 \)

and the query

\[ \text{Ships}(x_1,x_2,\text{bulk}_\text{cargo},x_4,x_5,x_6,x_7) \]

we can generate a new query with a restriction on attribute \( x_5 \)

\[ \text{Ships}(x_1,x_2,\text{bulk}_\text{cargo},x_4,x_5,x_6,x_7), x_5 \leq 150 \]

If the \text{Ships} relation happens to be clustered on this attribute, the introduction of such a restriction can reduce the search space in the relation.

\textit{Horn Clause with nonempty body and a nonempty head:} Such a clause can be used for join elimination when the body of the residue subsumes the query. It can also be used to limit searches in query evaluation. For example, given the residue,
\[ x_2 > 100 \leftarrow \text{Cargoes}(y_1, y_2, y_3, y_4, y_5), y_5 > 300 \]

and the query

\[ \leftarrow \text{Ships}(x_1, x_2, x_3, x_4, x_5, x_6, x_7), \text{Cargoes}(y_1, y_2, y_3, y_4, y_5), (y_5 > 300) \]

we can apply the residue to the query to arrive at the improved query

\[ \leftarrow \text{Ships}(x_1, x_2, x_3, x_4, x_5, x_6, x_7), \text{Cargoes}(y_1, y_2, y_3, y_4, y_5), (y_5 > 300), (x_2 > 100) \]

The transformed query is more efficient because it reduces the number of Ship tuples qualifying for the join. This technique is referred to as Scan Reduction.

An and Henschen (1992) present an approach to semantic query optimization that is also based on the notion of precompiling integrity constraints. They classify integrity constraints into three categories for use in query optimization: 1) integrity constraints that will be used for constraint (restriction) introduction, 2) constraints that will be used for restriction elimination and 3) constraints that will be used for restriction introduction in a join
operation. The constraints available in the knowledge base are used to construct an optimized constraint graph (OCG). Each atom in a constraint is represented as a vertex and an edge exists between each vertex. The paper deals with three forms of transformations, index introduction and scan reduction which are grouped into the category of constraint introduction and restriction elimination.

2. Dynamic Approaches: King (1981) and Shenoy and Ozsoyoglu (1989) present examples of semantic query optimization mechanisms that do not precompile integrity constraints. We present a detailed description of these mechanisms below.

King (1981) describes the architecture of QUIST, one of the first systems to use semantic knowledge for query optimization. QUIST performs semantic query optimization in three steps: the planning step, the generation step and the testing step.

The Planning Step: In this step, the system identifies relations specified in the query on which additional constraints should be sought. QUIST uses a set of constraint generation heuristics to determine if it is worthwhile to try and transform the incoming query into semantically equivalent queries. The heuristics can be divided into two categories: a) scanning a relation and b) joining a relation. In the first category, the basic heuristics
include: a) Try to exploit a clustered index, b) push a constraint up a hierarchy and c) don't introduce unlinked joins. In the second category, heuristics include: a) Move a constraint across a join boundary, b) Don't push a constraint down a hierarchy, c) use a strongly restricted clustered index and d) Try to eliminate a dangling relation. If the input query cannot be transformed using one of the heuristics described above then the query is not transformed semantically. Note, that in this step the decision whether to transform or not is based solely on the characteristics of the underlying database.

The Generation Step: This step is performed only if the system finds it worthwhile to perform semantic query optimization on the query based on results obtained in the planning step. The system iteratively generates a number of semantically equivalent queries by applying semantic integrity constraints to the incoming query. The system performs this step in two phases: 1) Rule selection and 2) Semantic Query transformation.

Rule Selection: In this step, the QUIST system identifies constraints in the knowledge-base that would be useful in semantically transforming the query. Constraints are identified as being relevant, applicable and effective. A constraint is relevant if: 1) the attribute constrained in the consequent of the constraint is involved in the query and there is only one left-hand side constraint or 2) every attribute constrained on the left-hand side is involved in
the query. A constraint is said to be applicable if and only if each of its relevant attributes is at least as strongly constrained by the query as the constraint itself. Finally, a rule is considered to be effective if the result of asserting it, in conjunction with the query, results in a stronger constraint than the constraint specified in the original query.

Semantic Query Transformation: The semantic query transformation process generates semantically queries, given the original query and the set of constraints identified as being effective in the generation step. The system can either introduce new constraints, introduce a new joins or eliminate constraints. For example, if there are N effective constraints, then the system will generate $2^N - 1$ queries as follows. If $Q$ is the original query and $C_1$, $C_2$ ..., $C_n$ are the additional constraints generated then the system generates new constraints as $Q_i = Q \land C_i$, $i = 1..n$. This generates a new constraint on the relevant attribute.

The types of semantic query transformation that QUIST can perform include: Access Elimination or detection of unsatisfiable conditions, Index Introduction, Scan Reduction, Join Elimination and Join Introduction.

*The Testing Step:* In this step, each of the semantically equivalent queries generated in the previous step are analyzed using conventional query optimization methods so that the
semantically equivalent query with the lowest estimated cost can be determined. This is done by estimating the cost of executing each of the semantically optimized queries. A cost estimation model based on System R (Selinger, 1979) is used to arrive at this estimate.

Shenoy and Ozsoyoglu (1989) present an approach to semantic query optimization that uses clausal forms for representing integrity constraints and uses a graph notation for representing a query. A query is represented as a graph in which, the attributes as well as constants specified in the query are the vertices of the graph and the edges are the joins and restriction specifications in the query. Their methodology deals with two types of constraints: subset and implication. Semantic query transformation using their methodology is performed in two stages: semantic expansion and semantic reduction.

*Semantic Expansion:* The semantic expansion step consists of two stages. The first stage of the semantic expansion step involves generating a canonical graph representation of the query. The second stage involves adding new restriction or join edges to the canonical graph. This is done by identifying the implication constraints whose antecedent atoms are satisfied by the graph and adding a restriction or join edge corresponding to their consequent atoms. The result is a semantically expanded query graph $G_m$. 
Semantic Reduction: Semantic reduction also consists of two phases: relation elimination and edge elimination. The relation elimination process examines the graph $G_m$ to identify and eliminate any redundant relations in the graph. If the system finds such relations, then the relation as well as all edges involving the relation are eliminated. The authors provide five criterion that qualifies a relation for elimination: 1) None of the target attributes should be from the relation, 2) all restrictions on non-join attributes should be redundant, 3) there is at most one join attribute, 4) all joins are equijoins and 5) there is a subset constraint involving the join attribute $R.A$ such that $R.A$ is a superset of $S.B$ is true where $S$ is a relation that can be joined with $R$. The edge elimination process eliminates redundant edges from the query. An edge is redundant if it is satisfied by the consequent atoms of an implication constraint of which all the antecedent atoms are satisfied by the query. This step is similar to the semantic query transformation step in King (1981). The primary semantic query transformation techniques presented in the paper include, Restriction Elimination, Index Introduction, Scan Reduction and Join Elimination. In addition to the methodology for query transformation, the authors also describe the implementation of a module that maintains semantic integrity constraints in a database.

Other approaches to semantic query optimization that use dynamic transformation mechanisms are Seigel et. al. (1992), Sun and Yu (1994) and Bertino and Musto (1992).
Seigel et. al. (1992) describe a heuristics based technique for semantic query optimization. that can be used to apply index introduction, scan reduction, restriction introduction and access elimination techniques. Sun and Yu (1994) extend the types of queries that can be semantically optimized by presenting techniques that are applicable for tree and chain queries. By specifying these algorithms, they eliminate restrictions placed by most of the SQO techniques on the types of allowable queries. Bertino and Musto (1992) present formal proofs for the correctness of transformations produced as a result of semantic query transformations. The primary transformation techniques dealt with in the paper are 1) modification of predicate in selection, i.e., index introduction, 2) join elimination and 3) access elimination. The authors present their work in the context of a compiled transaction approach. In a compiled transaction approach the query execution strategy is determined before the query is executed. Since a query may need to be executed as several transactions, it is possible that during an intermediate state, an integrity constraint that was used for semantic query transformation may have been violated. The authors show how defining restrictions on the use of semantic query transformation rules in the context of compiled transactions can prevent incorrect results from being generated.

Table 2.7 summarizes the key characteristics of the methods discussed above. All of the papers presented above, use heuristics to determine which types of transformations to apply
to an incoming query. However, there are times when the cost of determining the alternative queries can exceed the cost of an non-transformed query. Shekhar et. al. (1992) address this important component of the semantic query optimization problem, namely evaluating the relative costs of query execution and query optimization. This problem becomes increasingly relevant as the size of the knowledge base (that stores the integrity constraint) increases. They present two termination criterion that can be used to abort the semantic query optimization process: 1) the first is based on trying to balance the execution and optimization costs, 2) the second is based on the concept of diminished marginal returns. They utilize the techniques described in Mackert and Lohman (1986) to estimate the cost of query execution for the original and intermediate queries generated.
<table>
<thead>
<tr>
<th>System/Paper</th>
<th>Database Type</th>
<th>Class of Constraints</th>
<th>Query Transf. Types</th>
</tr>
</thead>
<tbody>
<tr>
<td>Chakravarthy et. al. (1990)</td>
<td>Deductive and Relational</td>
<td>Horn Clauses</td>
<td>Access Elimination, Index Introduction, Restriction Introduction, Restriction Elimination, Join Introduction, Join Elimination and Scan Reduction</td>
</tr>
<tr>
<td>King (1981)</td>
<td>Relational</td>
<td>First-order Well Formed Formulas</td>
<td>Access Elimination, Index Introduction, Join Introduction, Join Elimination and Scan Reduction</td>
</tr>
<tr>
<td>Shenoy and Ozsoyoglu (1989)</td>
<td>Relational</td>
<td>Subset and Implication Constraints (First-Order Logic)</td>
<td>Restriction Elimination, Index Introduction, Scan Reduction, Join Elimination</td>
</tr>
<tr>
<td>An and Henschven (1992)</td>
<td>Relational</td>
<td>Horn Clauses</td>
<td>Index Introduction, Scan Reduction, Join Elimination</td>
</tr>
<tr>
<td>Sun and Yu (1994)</td>
<td>Relational</td>
<td>Subset and Implication Constraints (First-Order Logic)</td>
<td>Restriction Introduction, Restriction Elimination, Index Introduction and Join Elimination</td>
</tr>
<tr>
<td>Bertino and Musto (1992)</td>
<td>Relational</td>
<td>Well Formed Formulas</td>
<td>Restriction Introduction, Index Introduction, Restriction Elimination and Join Elimination</td>
</tr>
<tr>
<td>Seigel et. al. (1992)</td>
<td>Relational</td>
<td>Simple Rules: Single attribute in antecedent and consequent</td>
<td>Access Elimination, Index Introduction, Restriction Elimination, Restriction Introduction and Scan Reduction</td>
</tr>
</tbody>
</table>

Table 2.7 Summary of Semantic Query Optimization Techniques
<table>
<thead>
<tr>
<th>System/Paper</th>
<th>Class of Queries</th>
<th>Constraint Processing Strategy</th>
<th>Key Feature</th>
</tr>
</thead>
<tbody>
<tr>
<td>Chakravarthy et. al. (1990)</td>
<td>Prolog Unit Clauses</td>
<td>Precompilation of Constraints</td>
<td>Use of partial subsumption to generate residues and using them for SQO</td>
</tr>
<tr>
<td>King (1981)</td>
<td>Restrict-Join-Project Relational Queries</td>
<td>Constraints processed for each query</td>
<td>Generation of constraint targets; Transformation by merging targets with queries</td>
</tr>
<tr>
<td>Shenoy and Ozsoyoglu (1989)</td>
<td>Select and Join Queries</td>
<td>Constraints processed for each query</td>
<td>Construct Query Graph; SQO by operations on graph such as edge elimination; Algorithm for maintaining integrity constraints</td>
</tr>
<tr>
<td>An and Henschchen (1992)</td>
<td>Select and Join Queries</td>
<td>Precompilation of constraints</td>
<td>Construct Optimized Constraint Graph</td>
</tr>
<tr>
<td>Sun and Yu (1994)</td>
<td>Select, Join, Tree and Chain Queries</td>
<td>Constraints processed for each query</td>
<td>Construct Query Graph; SQO by modifying graph;</td>
</tr>
<tr>
<td>Bertino and Musto (1992)</td>
<td>Not explicitly mentioned</td>
<td>Constraints processed for each query</td>
<td>SQO by applying Semantic Transformation Rules; SQO in the presence of compiled transactions</td>
</tr>
<tr>
<td>Seigel et. al. (1992)</td>
<td>Select and Join Queries</td>
<td>Constraints processed for each query</td>
<td>SQO by using heuristics for each type of transformation; Technique for automatic rule derivation based on query characteristics</td>
</tr>
</tbody>
</table>

Table 2.7 (Contd.) Summary of Semantic Query Optimization Techniques
Classification based on type of database

The techniques presented in King (1981), Shenoy and Ozsoyoglu (1989), Chakravarthy et. al. (1990), Sun and Yu (1994) as well as Bertino and Musto (1992) present SQO techniques applicable to relational databases. Semantic query optimization has also been described in the literature in the context of other types of databases. The majority of these are in the context of deductive databases. This is because as noted in Chakravarthy et. al. (1990) the intensional component of a deductive database are just rules on the extensional database that define new predicates. In fact, the discussion in Chakravarthy et. al. (1990) is presented in the more generic context of a deductive database. Another paper that discusses semantic query optimization in a deductive database context is Lakshmanan (1990). Lueng and Muntz (1990) discuss the role of semantic query optimization in a temporal database context. The authors claim that temporal integrity constraints can be used more effectively than traditional constraints since constraints involving time occur more naturally in a database, which results in richer semantics being available for semantic query optimization. Topaloglou et. al. (1992) discuss semantic query optimization in the context of a knowledge base management system (KBMS). In addition to using integrity constraints, they also describe the use of temporal knowledge in semantic query transformation. Sun et. al. (1991a) and Sun et. al. (1991b) discuss semantic query optimization in an object oriented
database context. Table 2.8 presents a classification of SQO methodologies based on the type of database used in the methodologies.

<table>
<thead>
<tr>
<th>Paper</th>
<th>Database Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>King et. al. (1981)</td>
<td>Relational</td>
</tr>
<tr>
<td>Shenoy and Ozsoyoglu (1989)</td>
<td>Relational</td>
</tr>
<tr>
<td>Bertino and Musto (1992)</td>
<td>Relational</td>
</tr>
<tr>
<td>Sun and Yu (1994)</td>
<td>Relational</td>
</tr>
<tr>
<td>Chakravarthy et. al. (1990)</td>
<td>Deductive and Relational</td>
</tr>
<tr>
<td>Lakshmanan (1990)</td>
<td>Deductive</td>
</tr>
<tr>
<td>Topaloglou et. al. (1992)</td>
<td>Temporal</td>
</tr>
<tr>
<td>Leung and Muntz (1990)</td>
<td>Temporal</td>
</tr>
<tr>
<td>Sun et. al. (1991a) &amp; Sun et. al. (1991b)</td>
<td>Object-Oriented</td>
</tr>
</tbody>
</table>

Table 2.8
Classification of Semantic Query Processing Methodologies Based on Database Type
All of the papers described above assume that a set of semantic integrity constraints for a given database have been specified. However, as noted by Yu and Sun (1992) and Seigel (1988), most methodologies assume that this information is specified by DBA's or other experts. However, experts may not always be able to provide the best rules that characterize the semantics of the database for query optimization purposes. The next paragraph describes three papers that address the problem of automatic constraint generation.

Seigel et. al. (1992) present a methodology for automatically deriving semantic integrity constraints for use in semantic query optimization. They use AI based rule discovery techniques to arrive at rules based on feedback on database usage patterns and past database performance. They define a four step rule derivation process. In the first step, characteristics of proposed rules, rules generated during semantic query optimization of queries that have not been matched, are evaluated in the context of their estimated cost savings to see if these proposed rules would generate worthwhile rules. In the second step, the most promising of these proposed rules are selected. In the third step, a query based on the selected proposed rules is issued to the database to check the validity of the proposed rules. In the final step, results of this query are evaluated and new rules are derived. The authors also present techniques for maintaining derived rules. The issue of maintaining semantic integrity constraints in the presence of database updates is also addressed by...
Ishakbeyoglu and Ozsoyoglu (1991). Yu and Sun (1989) present a technique for deriving and maintaining dynamic integrity constraints, constraints that are valid for the current state of the database. Results of queries specified after the last update are compared. If the set of tuples retrieved is identical then rules relating the restrictions in the queries are added to the database. Shekhar et. al. (1993) present an approach to rule discovery that is based on analyzing the data in the database. The idea is to generate query transformation rules based on data-distributions. A data-distribution is a function mapping the tuples of a database onto a grid whose coordinates are themselves functions of attributes in the database.

All of the papers described above have dealt with semantic query processing (SQP) in the context of a single database system. van Kuijk (1990) and Pan et. al. (1992) appear to be the only discussions of SQP in a multiple database context. van Kuijk (1990) presents SQP in the context of a distributed database while Pan et. al. (1992) present their discussion in a multidatabase context. However, the discussion presented in each of these papers is rudimentary at best and does not address any new issues that may arise from using semantic query optimization in a multiple database context. Reddy et. al. (1993) refer to the possibility of performing semantic query processing using an integrated set of integrity constraints. However, they do not describe any mechanisms for deriving the integrated set of integrity constraints. To the best of our knowledge, this dissertation is the first to
describe a methodology for generating an integrated set of integrity constraints and use these constraints to facilitate SQP in a heterogeneous database context.

2.4 Query Processing in Heterogeneous Databases

Query processing issues have received limited attention in the heterogeneous database literature. This is probably due to the fact that researchers are still focusing on solving problems related to the integration of heterogeneous databases. However, prototype systems such as Multibase (Smith et. al., 1981), PRECI* (Deen et. al., 1987), ADDS (Breitbart et. al., 1986), IMDAS (Krishnamurthy et. al., 1987), Mermaid (Templeton et. al., 1987), and CALIDA (Rajinikanth et. al., 1990) each have a query processing module. We present a brief summary of the query processing features of four of these systems. All of these prototypes use a global schema approach to resolve database heterogeneity. Hence, the query language provided to the user is dependent on the data model used to specify the global schema.

In Smith et. al. (1981) the functional data model is used to represent the global schema. Queries are specified against the global schema using DAPLEX. The query processing module consists of three components: a query translator, a query processor and a local
database interface (LDI). The query translator translates an incoming query (which reference global entities and attributes) into a set of queries that reference local entities and attributes. The query processor takes this set of queries and generates a strategy for executing these queries. The queries generated by the query processor are then sent to the LDI in an internal format. The LDI translates these queries into queries or programs on the local database. The results of the queries from each database are sent back to the query processor which consolidates the results and presents them in an appropriate format for the user.

PRECl* (Prototype of a Relational Canonical Interface) (Deen et. al. (1987) uses an extended relational algebra called Preci Algebraic Language (PAL) as its query language. The architecture of PRECl* is a hybrid between a federated and global schema approaches. Databases managed by the system are classified into two types: inner nodes, which contribute towards the Global Database Schema (GDS) and outer nodes which do not contribute to the GDS. Query processing consists of three steps: Global Query processing, Global Subquery processing and Nodal Query processing which are performed by modules corresponding to these operations. A user query is processed by the global query processor which transforms the query into a tree consisting of subtrees, where each subtree represents a global subquery on an individual node. These queries are then decomposed into local
queries by the global subquery processor and finally executed by the nodal query processors.

IMDAS (Krishnamurthy et. al., 1987) is a system intended to integrate CAD/CAM databases. Hence, it uses a language suited for representing such databases, OSAM* (Su et. al., 1988). User queries in IMDAS are posed in a common global data manipulation language (GDML) which is based on SQL with additional features such as support for complex and abstract data types, strong data type checking for complex and abstract data types, attribute inheritance etc., provided to support the nature of the underlying databases. The query processing component consists of five parts: Data Manipulation Language Service (DMLS), Query Mapping Service (QMS), Transaction Manager (TM), Data Assembler (DASM) and Data Dictionary Service (DDS). The DMLS takes a user query issued against external views and generates a query tree on the global conceptual view. This query on the global conceptual view is mapped and decomposed into queries on the local databases using the information available in the DDS. The TM manages distributed query execution on the databases. Functions performed by the TM include distributed consistency control and recovery. Finally, the DASM is responsible for assembling data from multiple sources into a final result.
Mermaid (Templeton et. al., 1987) allows users to specify queries using SQL or ARIEL. The Mermaid system provides explicit support for global query optimization in a heterogeneous database environment. The query optimization algorithms available for use include the semijoin algorithm and the replication algorithm. These algorithms are commonly used for query optimization in a homogeneous distributed database environment. Templeton et. al. (1987) describe how these algorithms can be extended to a heterogeneous environment.

Table 2.9 summarizes the characteristics of these prototype systems. From the description of each of these prototype systems it is clear that the ability to process queries is an important aspect of heterogeneous database management. It is also important to (wherever possible) optimize queries (globally) before translating them into local subqueries. However, syntactic optimization of global queries is typically not possible because of the heterogeneous nature of the underlying databases. Optimization based on the semantics of the underlying data is, however, possible if the semantics can be captured at the global schema level. In this dissertation, we describe how one can perform semantic query optimization in a heterogeneous database environment. As discussed in the previous section, the semantics of the underlying databases (for query processing purposes) are best represented by the semantic integrity constraints describing the databases. Hence, to
facilitate semantic query processing on queries issued against global schemas in a heterogeneous database environment it is necessary to generate a set of constraints applicable at the global level. The generation of these global constraints and their use in facilitating semantic query processing is one of the primary contributions of this dissertation and is described in the next chapter.
In this section, we present a brief description of the various steps in a methodology that utilizes integrity constraint knowledge in a heterogeneous database environment. We compare the new methodology to the traditional global schema approach shown in Fig. 2.3.

The two major drawbacks of the traditional methodology that we attempt to address in our new methodology are:

1) The use of a single source of knowledge for interschema relationship identification. This results in these methodologies requiring extensive user input, because not too much confidence can be placed on decisions generated by such an automated tool.

2) An integrated schema generated using traditional schema generation methodologies is only a partial representation of the semantics of the underlying database.
Fig. 3.1 Enhanced Methodology for Schema Integration
Figure 3.1 presents an enhanced methodology that utilizes integrity constraints to address the drawbacks presented above. The new methodology consists of seven steps:

1) **Schema Translation**: This phase is common to the traditional and new methodology and involves the translation of local schemas into schemas in a common model.

2) **Schematic Interschema Relationship Generation**: This phase is also common to both methodologies and involves identifying interschema relationships based on their schematic properties. However, unlike the traditional methodology, user confirmation of interschema relationships is not sought after this phase. Instead, confirmation of these relationships is sought by analyzing the databases using another source of knowledge, namely integrity constraints.

3) **Constraint-based Interschema Relationship Generation**: This phase is unique to the new methodology and involves analysis of the integrity constraints specified on the underlying databases to generate interschema relationships among objects. These relationships are generated based on the degree to which integrity constraints involving an object a in database $D_1$ are valid (after appropriate transformation) in database $D_2$ and constraints involving an object b in database $D_2$ are valid in database $D_1$, where a and b are
schematically related objects. Such relationships provide additional information about interschema relationships by generating relationships that are dynamic in nature. Constraint-based relationships are dynamic in nature because they are based on evaluating constraints specified on a database $D_1$ against the current data in database $D_2$ and vice versa. Changes to data in a database may change the nature of relationships since constraints that were previously valid may no longer be valid. Generating constraint-based relationships on a periodic basis will ensure that the relationships accurately reflect the current state of the databases. It should also be noted that the constraint-based relationship generation process is not restricted to generating relationships among pairs of database objects.

4) "Real World" Interschema Relationship Generation: This phase is unique to the new methodology. In this phase, the schematic and constraint-based relationships generated in steps 2) and 3) are utilized to arrive at "real world" relationships among objects. Ideally, one would like to arrive at "real world" relationships that reflect reality as closely as possible. It is hoped that the use of schematic and constraint-based relationships (which can both be generated in an automated fashion) will enable us to arrive at a better set of interschema relationships among database objects compared to using schematic knowledge alone. User interaction in the form of confirmation/rejection of relationships generated by the system is expected in this phase of the methodology. It is hoped that using two sources
of knowledge will enable us to reduce the burden of interschema relationship identification
put on the user by most methodologies.

5) Integrated Schema Generation: This phase is common to the traditional and the new
methodology. The primary difference is in the inputs to the integrated schema generation
process. In the new methodology, the "real world" relationships generated in the previous
step, rather than schematic relationships, are used as inputs to the schema integration
algorithms.

6) Integrity Constraint Integration: The next two phases of the methodology are unique to
the new methodology. The objective of this phase of the methodology is the generation of
integrity constraints that are applicable at the integrated schema level. To achieve this
objective, constraint-based and "real world" relationships generated in steps 3 and 4, the
integrated schema generated in step 5 as well as the knowledge of the integration strategies
used in step 5, are used to generate a set of integrated constraints applicable to the integrated
schema. Such a set of integrated integrity constraints augments the integrated schema and
results in a more comprehensive representation of the underlying databases. As noted
before, constraint-based relationships are dynamic in nature which means that the set of
integrity constraints that are applicable at the integrated level may also change with time.
Hence, the constraint-based relationship generation and integrity constraint integration steps in the new methodology are expected to be performed periodically to reflect the current state of the underlying databases.

7) Semantic Query Processing: In addition to generating a more comprehensive representation, the presence of integrated integrity constraints provides us with an opportunity to use the semantic query processing (SQP) techniques elaborated on in King (1981) and Chakravarthy et. al. (1990) in a heterogeneous database context. The use of these techniques would allow us to transform queries formulated against the integrated schema into equivalent more efficient local database queries.

In the rest of this chapter we present detailed descriptions of the following steps of the new methodology: schematic interschema relationship generation, constraint-based relationship generation and integrity constraint integration. These steps constitute the core contributions of this dissertation. In chapter 5 we describe how the integrated integrity constraints can be used to facilitate semantic query processing in a heterogeneous database environment.

3.1 Schema Translation
The objective of the schema translation step is to translate the local schemas (specified using the local database models) into schemas in a common model, usually a semantic model such as the E-R model (Chen, 1976). Such a translation achieves two objectives: 1) it removes data model heterogeneity and 2) allows users to utilize the rich set of constructs available in the semantic models to generate a more accurate representation of the underlying databases than is possible using the more traditional models such as the relational (Codd, 1970), hierarchical or network models.

The process of translating a schema represented in a traditional data model to a semantic model cannot be completely automated. However, semi-automated techniques can be used to help designers in this process. Navathe and Awang (1987) present heuristics for translating schemas from a relational model to an E-R model, given a set of functional dependencies. A tool that adopts this technique and translates from the relational to the Unifying Semantic Model (USM) model (Liu, 1993) is available as part of the system described in the next chapter. Translating from a hierarchical schema to an E-R model is a lot more complicated and requires more input from the users/designers. A tool that assists designers in the translation of schemas from hierarchical to the USM model (Fu, 1993) is also available as part of the system described in the Chapter 4.
3.2 The Unifying Semantic Model (USM)

The Unifying Semantic Model (Ram, 1991) is a semantic model that synthesizes and extends the constructs found in semantic models such as the E-R model (Chen, 1976), SDM (Hammer and McLeod, 1981), NIAM (Nijssen and Halpin, 1989) and IDEFIX (Loomis, 1986). Schemas in the USM are defined in terms of entity classes. An entity class defines an abstract representation of a set of similar real-world entities. Every entity class in an USM schema is characterized by a name, a general description and a description of its function in the schema. In addition, entity classes can have a set of associated attributes. Every attribute in the USM has a domain associated with it. A domain defines the set of legal that an attribute can take. Each domain also has a type associated with it. In addition to its domain, each attribute has other properties, such as cardinality, whether it is mandatory or optional, whether the attribute or single-valued etc. associated with it.

Entity classes can be related to other entity classes through different types of relationships. USM allows us to define several types of relationships among entity classes. These include:

1) Interaction or Association Relationships - An interaction relationship relates members of participating entity classes to members of other participating entity classes through
attributes. In figure 3.2, the relationship *Inspection-Inspected_Machine* between the entity classes *Machine* and *Inspection* is an example.

2) Generalization/Specialization Relationships - A generalization/specialization relationship defines a supertype/subtype relationship between entity classes. In figure 3.2, *Robot* is a specialization of the entity class *Equipment*.

3) Composites, Groupings and Aggregates - A composite relationship defines a new entity class which has other entity classes as its members. A grouping or aggregate relationship defines an entity class whose members are physically or logically made up of members or sets of members from some other entity classes (Ram and Storey, 1993). For example, in figure 3.2, *Computer Types* is a composite class defined on the base class *Computer*.

The rich set of constructs available in the USM allows designers to embed more semantics in schema definitions than is possible using traditional models. The properties associated with these constructs can then be used to analyze schemas and identify relationships among pairs of objects in the schemas. The heuristics for identifying these relationships are defined in the next section.
Fig. 3.2 Example USM Schema
A Logic-based Representation of the Unifying Semantic Model

Integrity constraints are best represented using some form of first-order logic. Hence, to generate a representation of integrity constraints associated with a database, it is necessary to generate a logic-based representation of the schema. The following logic-based representation of the USM is based on Reiter (1984)'s description of representing semantic models using first-order logic.

An entity class is represented as a predicate, with attributes as its arguments. A schematic representation of the entity class assumes that all attributes are universally quantified. For example,

\[
\text{SHIPS(shipname, owner, Shiptype, draft, deadweight, capacity, captain)}
\]

represents the entity class SHIPS, with attributes shipname, owner etc.

A reference to a specific entity class would require that all arguments in the predicate be instantiated. For example,

\[
\text{SHIPS("sailor's paradise", "Onassis", "Supertanker", 1000, 1500, 2000, "John Maddux")}
\]
would be an instance of SHIPS.

An interaction relationship is also represented as a variable argument predicate. A typical relationship is represented as, $A(x_1, y_1) \rightarrow B(x_1, x_2, x_3) \wedge C(y_1, y_2)$ where $B$ and $C$ are predicates representing entity classes. For example, the assignment relationship between SHIPS and OFFICERS would be represented as

\[
\text{ASSIGNMENT}(\text{Captain}, \text{Ship\_Assigned}) \rightarrow \\
\text{SHIPS}(\ldots, \text{Captain}) \wedge \text{OFFICERS}(\text{Ship\_Assigned}, \ldots)
\]

where Captain and Ship\_Assigned are the attributes in the respective entity classes participating in the interaction relationship.

Generalization/Specialization relationships can be expressed using a multiple implications. Consider the following example,

\[
\text{Ruritanian\_Ships}(x) \rightarrow \text{Ships}(x) \\
\text{Oil\_Tankers}(x) \rightarrow \text{Ships}(x)
\]
The first formula indicates that all Ruritanian_Ships are Ships and the second indicates that all Oil_Tankers are Ships. Thus, this representation captures the semantics of Ships being a superclass of both Ruritanian_Ships and Oil_Tankers.

3.3 Schematic Interschema Relationship Identification

Interschema Relationship Identification (IRI) is the process of analyzing the underlying schemas and identifying objects in the schemas that may be related to each other. These objects may have been represented differently in the two schemas. Hence, depending on the representations, interschema relationships can be classified into two categories: homogeneous and heterogeneous relationships. Homogeneous relationships occur when concepts are defined using the same construct in both schemas. Accordingly, such relationships may exist between two entity classes, two attributes or two relationships. Heterogeneous relationships occur when the same concept is defined using different constructs in the underlying schemas. Examples of such relationships are entity class-attribute, entity class-relationship and attribute-relationship.

Our approach to identifying interschema relationships involves measuring the similarity or dissimilarity between entity classes, attributes and relationships in the schemas to be
integrated. We use an index of similarity (IS) for this purpose. The IS can take on values between 0 and 1. A high IS between objects indicates a high probability of a relationship between the objects. A high ID suggests that the objects in question may not be related.

Since the primary construct used to define a schema is an entity class, we begin the IRI process by comparing entity classes. The index of similarity between two entity classes is computed as:

$$IS = f(ED_s, A_s, R_s)$$

where

- $ED_s$ - is the index of similarity of the entity class definition
- $A_s$ - is the index of similarity of the attributes associated with the entity classes and
- $R_s$ - is the index of similarity of the relationships in which the entity classes participate

### 3.3.1 Entity Class Definition Similarity

Entity class definition similarity is computed as:

$$ED_s = f(N_s, GD_s, FD_s)$$
where

\( N_s \) - is the similarity of the names of the entity classes

\( GD_s \) - is the similarity of the general descriptions of the entity classes and

\( FD_s \) - is the similarity of the functional descriptions associated with the entity classes

Table 3.1 summarizes the properties of the entity class definition that we use to compute EDs along with their respective weights. The table indicates that there are three properties that contribute towards the calculation of entity class definition similarity and that each component is weighted equally in the calculation of ED_s.

The first component that is evaluated is the similarity of entity classes' names. We check to see if the entity class names match. If they do not, we lookup the general and/or domain specific synonym/abbreviation dictionary to see if they are synonymous. The general synonym dictionary, contains synonyms that we would expect to find in a standard thesaurus. On the other hand, the domain specific dictionary contains synonyms and abbreviations that are unique to the environment in which the databases exist. Both dictionaries associate a weight with each pair of synonym. The higher the weight the stronger the match. Exact synonyms such as, TOPIC and SUBJECT, would be considered
strong matches. Words that are not exact synonyms such as, TOPIC and THEME would be considered weak matches. In comparing then general and functional descriptions, stop words such as prepositions and other common words are removed from the sentence. The abbreviated sentence is then compared based on keywords and a index of similarity is generated.

---

2 The tables indicate the current value of the weights defined in the system. These weights are expected to evolve over time.
Fig. 3.3 Example Schemas
Example 1: In the schema shown in figure 3.3, let us assume that a lookup of the synonym dictionary reveals that the weight associated with the synonym pair TOPIC-SUBJECT is 0.9. Let us also assume that a comparison of their functional and general descriptions yields similarity values of 0.9 and 0.8 respectively. Using the weights given in table 3.1, the entity class definition similarity for the pair TOPIC-SUBJECT would be computed as

\[
ED_s = 0.33 \times 0.9 + 0.33 \times 0.9 + 0.33 \times 0.8 = 0.86
\]

It should be noted that in computing the similarity between the other entity class pairs (TOPIC,PAPER), (PROFESSIONAL SOCIETY, SUBJECT), (PROFESSIONAL SOCIETY, PAPER), (AUTHOR, SUBJECT) and (AUTHOR, PAPER) it is likely that we would find them dissimilar and would hence arrive at a high degree of dissimilarity for each pair of entity classes.

3.3.2 Attribute Similarity

Attribute similarity for a pair of attributes is computed as:

\[
IAS = f(PA_{a_1}, SA_{a_2})
\]
where

\[ PA_{as} \] - is the similarity of the pair of attributes on primary aspects, and

\[ SA_{as} \] - is the similarity of the pair of attributes on secondary aspects

The various properties associated with an attribute are divided into two categories: primary and secondary. Primary aspects, such as attribute names and domains, represent properties that are representative of the attributes and should act as good discriminators in comparisons. Accordingly, we associate higher weights with these properties. Secondary aspects by themselves are not sufficient for determining similarity between attributes. However, in conjunction with primary aspects they enable us to provide users with a more accurate estimate of similarity. Table 3.2 summarizes these properties and the weights associated with them. It shows that the attribute similarity computation function associates a weight of 0.7 for the primary aspects and 0.3 for the secondary aspects. Within the primary aspects, attribute names and domain names are assigned equal weight. Similarly, all the properties that contribute towards the secondary aspects are also assigned equal weight.
### Primary Aspects (Wt. - 0.7)

<table>
<thead>
<tr>
<th>Weight</th>
<th>Property</th>
</tr>
</thead>
<tbody>
<tr>
<td>0.5</td>
<td>Similarity of Attribute Names</td>
</tr>
<tr>
<td>0.5</td>
<td>Similarity of Domains</td>
</tr>
</tbody>
</table>

### Secondary Aspects (Wt. - 0.3)

<table>
<thead>
<tr>
<th>Weight</th>
<th>Property</th>
</tr>
</thead>
<tbody>
<tr>
<td>0.2</td>
<td>Mandatory/Optional</td>
</tr>
<tr>
<td>0.2</td>
<td>Changeable/Not Changeable</td>
</tr>
<tr>
<td>0.2</td>
<td>Cardinality</td>
</tr>
<tr>
<td>0.2</td>
<td>Ordering/Duplicates</td>
</tr>
<tr>
<td>0.2</td>
<td>Other Constraints</td>
</tr>
</tbody>
</table>

**Table 3.2 Attribute Similarity**

For each pair of attributes, the weighted average of the primary aspects is determined by comparing the attribute names, domains (which includes a domain name comparison and a domain type comparison). If the IS as a result of this computation is above a threshold, the remaining properties are compared and a secondary aspect similarity is computed. The IS
for this pair of attributes is computed as a weighted average of the primary and secondary aspects.

The overall attribute similarity \((A_s)\) for the pair of entity classes is computed as a function of the \(IS_{as}\) (of attribute pairs) whose values are above the specified threshold.

Example 1 (Contd.): In comparing the attributes of entity classes TOPIC and SUBJECT, comparing NAME and TOPIC_ID would result in no matches on primary aspects. On the other hand, comparing the attributes NAME and NAME we find a perfect match on name. Let us also assume that we find a perfect match on domain name. Further, let us assume that \(SA_{as}\) for this pair of attributes is 0.85. Hence, using the weights given in table 3.2,

\[
IAS = 0.7 \times (0.5 \times 1 + 0.5 \times 1) + 0.3 \times (0.85) = 0.7 + 0.255 = 0.955
\]

If the threshold for individual attribute similarity is set at 0.75, then the overall attribute similarity of the entity classes is 0.955 since only one attribute pair has an individual similarity value above the specified threshold.

3.3.3 Relationship Similarity
The final component that needs to be computed in order to arrive at the IS for a pair of entity classes is the similarity of relationships in which the entity classes participate. Entity classes can participate in three types of relationships. Hence, the similarity of the relationships is computed as:

\[ R_s = f(\text{IR}_{is}, \text{GS}_{is}, \text{CG}_{is}) \]

where

\( \text{IR}_{is} \) - is the similarity of the \( i \)th interaction relationship in which the entity class participates

\( \text{GS}_{is} \) - is the similarity of the \( i \)th generalization/specialization relationship in which the entity class participates, and

\( \text{CG}_{is} \) - is the similarity of the \( i \)th composite/grouping relationship in which the entity class participates

The individual ISs for each pair of relationships is computed as:

\[ \text{IR}_{is}/\text{GS}_{is}/\text{CG}_{is} = f(\text{PA}_s, \text{SA}_s) \]
where

$P_{Ass}$ - is the similarity of the pair of relationships on primary aspects and

$S_{Ass}$ - is the similarity of the pair of relationships on secondary aspects.

Tables 3.3a, 3.3b and 3.3c list the primary and secondary aspects of each type of relationship and the associated weights used in deriving similarities. For example, table 3.3a shows that in the case of interaction relationships, primary aspects are assigned a combined weight of 0.7. The table also shows that each primary aspect property contributes equally towards the calculation of the primary aspect value. The properties that contribute towards the secondary aspect value for interaction relationships are, however, not weighted equally. Table 3.3a shows that participation cardinality is assigned a weight of 0.7 compared to a weight of 0.3 assigned to weakness annotation. Tables 3.3b and 3.3c convey similar information about the weights assigned to primary and secondary aspects of generalization/specialization and composite/grouping relationships.

The process of computing relationship similarities is similar to the attribute similarity computation discussed above. For each pair of relationships, we begin by computing the primary aspect similarity. As with attribute similarity, secondary aspect similarities are
computed contingent on primary aspect similarity crossing a certain threshold. The overall similarity of each pair of relationship is computed as a weighted average of primary and secondary aspect similarities for that pair. This process is repeated for each pair of relationships. The function for computing the overall relationship similarity for a pair of entity classes associates variable weights with each relationship category. The actual weights associated are based on 1) the number of categories of relationships in which the entity class participates and 2) the number of relationships in each category.
Table 3.3a Interaction Relationship Similarity^3

<table>
<thead>
<tr>
<th>Weight</th>
<th>Property</th>
</tr>
</thead>
<tbody>
<tr>
<td>0.25</td>
<td>Similarity of Relationship Names</td>
</tr>
<tr>
<td>0.25</td>
<td>Similarity of Attributes</td>
</tr>
<tr>
<td></td>
<td>Participating in the Relationship</td>
</tr>
<tr>
<td>0.25</td>
<td>Similarity of other participating entity classes</td>
</tr>
<tr>
<td>0.25</td>
<td>Degree of the relationship</td>
</tr>
<tr>
<td>0.7</td>
<td>Participation Cardinality</td>
</tr>
<tr>
<td>0.3</td>
<td>Weakness Annotation (if any)</td>
</tr>
</tbody>
</table>

^3 The tables indicate the current value of the weights defined in the system. These weights are expected to evolve over time.
Table 3.3b Generalization/Specialization Relationship Similarity

Attribute-Defined Relationships

<table>
<thead>
<tr>
<th>Weight</th>
<th>Property</th>
</tr>
</thead>
<tbody>
<tr>
<td>0.33</td>
<td>Similarity of attributes that define the relationship</td>
</tr>
<tr>
<td>0.33</td>
<td>Similarity of constraints that define the sub-classes</td>
</tr>
<tr>
<td>0.34</td>
<td>Similarity of other participating entity classes</td>
</tr>
<tr>
<td>1.00</td>
<td>Sub-class constraints</td>
</tr>
</tbody>
</table>

Set-Operation-Defined Relationships

<table>
<thead>
<tr>
<th>Weight</th>
<th>Property</th>
</tr>
</thead>
<tbody>
<tr>
<td>0.4</td>
<td>Set-Operator Match</td>
</tr>
<tr>
<td>0.6</td>
<td>Similarity of other participating entity classes</td>
</tr>
</tbody>
</table>

4 The tables indicate the current value of the weights defined in the system. These weights are expected to evolve over time.
Table 3.3b (Cont.) Generalization/Specialization Relationship Similarity

Roster-Defined Relationships

<table>
<thead>
<tr>
<th>Weight</th>
<th>Property</th>
</tr>
</thead>
<tbody>
<tr>
<td>0.8</td>
<td>Similarity of other participating entity classes</td>
</tr>
</tbody>
</table>

Primary Matches (Wt. - 0.8)

Secondary Matches (Wt. - 0.2)

1.00   Sub-class constraints

---

5 The tables indicate the current value of the weights defined in the system. These weights are expected to evolve over time.
### Table 3.3c Composites/Grouping Relationship Similarity

#### Composite Relationships

<table>
<thead>
<tr>
<th>Weight</th>
<th>Property</th>
</tr>
</thead>
<tbody>
<tr>
<td>0.33</td>
<td>Similarity of Composite Class Name</td>
</tr>
<tr>
<td>0.33</td>
<td>Similarity of Entity Classes on which the Composite Class is defined</td>
</tr>
<tr>
<td>0.33</td>
<td>Similarity of Member Classes</td>
</tr>
</tbody>
</table>

#### Grouping/Aggregate Relationships

<table>
<thead>
<tr>
<th>Weight</th>
<th>Property</th>
</tr>
</thead>
<tbody>
<tr>
<td>0.4</td>
<td>Similarity of Class Name</td>
</tr>
<tr>
<td>0.4</td>
<td>Similarity of member entities</td>
</tr>
<tr>
<td>0.2</td>
<td>Similarity of Constraints placed on member entities</td>
</tr>
</tbody>
</table>
Example 1 (Contd.): In the example schema, the entity classes TOPIC and SUBJECT participate in interaction relationships only. A comparison of the two relationships on their primary aspects results in a similarity value that is below the relationship similarity threshold (0.75). Hence, the contribution of the relationship similarity towards the determination of overall similarity of the entity classes TOPIC and SUBJECT is zero.

The overall similarity for the entity class pair TOPIC-SUBJECT is computed as,

\[
IS = 0.4 \times ED_a + 0.3 \times A_s + 0.3 \times R_s
\]

\[
IS = 0.4 \times 0.86 + 0.3 \times 0.955 + 0.3 \times 0
\]

\[
= 0.645
\]

3.3.4 Heterogeneous Relationships

Compared to homogeneous interschema relationships, heterogeneous relationships are harder to detect automatically. This is because, objects involved in heterogeneous interschema relationships have few common properties on which they can be compared. Heterogeneous interschema relationships can exist between

- entity class - attribute
- entity class - interaction relationship
Name matching is used as the primary means of determining IS for all types of heterogeneous relationships. Other properties used to determine specific types of heterogeneous interschema relationships include:

- entity names and domain names of attributes (entity class - attribute)
- cardinality of relationship and cardinality of attributes (attribute - relationship)
- similarity of other participating entity classes (entity class - relationship, entity class - attribute)

Computationally, the IS for heterogeneous interschema relationships is determined in conjunction with the computation of the IS for homogeneous conflicts.

Example 1 (Contd.): In our example schema, while comparing the two entity classes AUTHOR and PAPER we would also compute similarity values for all the entity class - attribute and entity class - relationship pairs. In our example, we would find a perfect name match between the entity class AUTHOR and attribute AUTHOR. Hence, the system will store this as a potential heterogeneous interschema relationship.
3.3.5 Evaluation of Schematic Interschema Relationships

Larson et. al. (1989) point out that entity classes in schemas may be related to each other using either equivalence, overlapping or subsumption relationships. In order to achieve a comprehensive integrated schema, it is important for the system to determine the existence of such relationships in the schemas.

The relationship between two similar entity classes is computed as:
\[ \text{ECR} = f(\text{EDS}, \text{AS}) \]

where

- \( \text{EDS} \) - is the index_of_similarity of the entity classes and
- \( \text{AS} \) - is the overall index_of_attribute_similarity.

The heuristics used to determine the type of relationship are as follows:

- Entity classes with a high value (\( > 0.75 \)) for both EDS and AS are likely to be equivalent.
- Entity classes that have a low value (close to the threshold) of EDS are likely to be overlapping.
- If the entity class similarity is neither too high nor too close to the threshold, attribute similarity is used to discriminate between overlapping and subsumption relationships.
particular, if entity class A is subsumed by entity class B, then the attribute similarity for this pair should be high, since almost all attributes in A should also be attributes in B.

Example 1 (contd.): In our example, upon completion of the conflict identification phase the system will notify the user(s) that the entity class pair TOPIC-SUBJECT has a similarity value of 0.65 and possibly classify the entity classes to be possibly overlapping. It will also inform the user that an entity class - attribute conflict was detected. Since, the rest of the entity class pairs were deemed to be dissimilar (since their similarity values fell below the threshold) the system will assume that each of these entity classes is disjoint.

In a methodology that utilizes schematic information alone we would present these recommendations to the user. The user would then accept or reject the system's recommendations or choose to add new pairs of objects to the list of related objects. However, in our methodology, recommendations about relationships among objects is not presented to the user until after relationships based on integrity constraints have been generated.

3.4 Constraint-based relationship generation
The objective of this phase is to generate interschema relationships among database objects based on the characteristics of constraints specified on the databases.

An integrity constraint of the form,

\[ S_I \leftarrow R_I \]

where, \( S_I \) utilizes an attribute \( x_I \) and \( R_I \) utilizes an attribute \( y_I \), describes restrictions placed by certain values of \( y_I \) on the values that \( x_I \) takes. Hence, the integrity constraint can be thought of as describing a property of the attribute \( y_I \) as well as describing characteristics of the entity class \( E \) to which \( y_I \) belongs. If the same constraint is valid in another database for an attribute \( y'_I \) belonging to an entity class \( E' \), then it is likely that \( y_I \) and \( y'_I \) have some relationship as do \( E \) and \( E' \).

This is the premise underlying constraint-based relationships among objects (entities and attributes) in heterogeneous databases. Every constraint specified against a database is considered to involve one or more database objects. Accordingly, we associate a constraint with the object it involves. Thus, each object can be thought of as having a set of constraints associated with it. Accordingly, the set of constraints involving two objects can have four
types of relationships between them: equivalence, overlap, subsumption and disjoint. We present definitions for these relationships below.

Let $D_1$ and $D_2$ represent the underlying databases. $A$ and $B$ are objects (entities or attributes) between which we are trying to determine constraint-based relationships.

We define the word *involving* an object as follows.

We restrict our discussion to integrity constraints and databases that can be represented as Horn clauses. Formally, an integrity constraint is a Horn clause of the form,

$$ S \leftarrow R_1, R_2, R_3, \ldots, R_n $$

where each $S$ defines a restriction on an attribute in $R_{i=1,m}$ and $R_{i=1,m}$ are relations involved in the constraint. For each $R_i$, there may be one or more attributes $A_i$ with value restrictions. Every relation $R_i$ with an attribute $A_i$ is said to be *involved* in the constraint. If the attributes $A_i$ *involved* in the constraint belong to different relations, then the constraint becomes a part of two sets, one belonging to each object $R_i$. 
Consider the following constraints:

Schema A: Ships(x1,OWNER,SHIPTYE,ORIGIN,x5,x6,x7)

IC-A:

OWNER='Onassis' ← Ships(x1,OWNER,supertanker,ORIGIN,x5,x6,x7) — (a)
(if Shiptype = 'Supertanker' then Owner='Onassis')

Schema B: Boats(x1,COUNTRY,OWNER,TYPE)

IC-B:

← Boats(x1, ICELAND,OWNER,TYPE) — (b)
(there are no Boats with Country = 'ICELAND')

In IC-A, the constraint is said to involve the predicate (entity class) Ships and attribute SHIPTYE. In IC-B, the constraint is said to involve the predicate (entity class) Boats and attribute Country.

Definition 0: We define the operator \( \omega \) as the operator that checks the validity of a constraint on a database. If \( x \) is a constraint and \( D \) is a database, \( x \omega D \) is true if the query corresponding to the constraint is valid in database \( D \). For every object \( A \) and \( B \) in the databases \( D_1 \) and \( D_2 \) the sets IC-A and IC-B represent the set of integrity constraints involving the object, i.e, constraints involving the predicate.
Definition 1: A CBequiv B if the constraints involving the objects can be placed into two sets IC-A and IC-B, such that

\[ \forall x, x \in IC-A, x \omega D_2 \land \]
\[ \forall y, y \in IC-B, y \omega D_1 \]

The definition states that A CBequiv B, if the constraints involving object A in database D_1 are valid in database D_2 and every constraint involving object B in database D_2 is valid in database D_1.

Definition 2: A CBsubsumes B if the constraints involving the objects can be placed into three sets, IC-A, IC'-B and IC''-B such that

\[ \forall x, x \in IC-A, x \omega D_2 \land \]
\[ \forall y, y \in IC'-B, y \omega D_1 \land \]
\[ \forall z, z \in IC''-B, z \neg \omega D_1 \]

where
i) \( IC\cdot B = IC'\cdot B \cup IC''\cdot B \),

ii) \( IC'\cdot B \cap IC''\cdot B = \emptyset \)

iii) \( IC'\cdot B \neq \emptyset \), \( IC''\cdot B \neq \emptyset \)

In other words, every constraint involving object A in database \( D_1 \) is valid in database \( D_2 \) but there is at least one constraint involving object B in database \( D_2 \) is not valid in database \( D_1 \). Hence, the set of constraints involving object B can be divided into two non-empty subsets, \( IC'\cdot B \) and \( IC''\cdot B \), where \( IC'\cdot B \) contains the constraints that are valid in database \( D_1 \) and \( IC''\cdot B \) contains the constraints that are not valid in database \( D_1 \).

**Definition 3:** A C overlaps B if the constraints involving the objects can be placed into four sets, \( IC'\cdot A \), \( IC''\cdot A \), \( IC'\cdot B \) and \( IC''\cdot B \), such that,

\[
\forall w, w \in IC'\cdot A, \ w \not\in D_2 \wedge \\
\forall x, x \in IC''\cdot A, \ x \not\in D_2 \wedge \\
\forall y, y \in IC'\cdot B, \ y \in D_1 \wedge \\
\forall z, z \in IC''\cdot B, \ z \not\in D_1
\]
where

i) $IC^\prime - A = IC'^\prime - A \cup IC''^\prime - A$, $IC^\prime - B = IC'^\prime - B \cup IC''^\prime - B$

ii) $IC'^\prime - A \cap IC''^\prime - A = \phi$, $IC'^\prime - B \cap IC''^\prime - B = \phi$

iii) $IC'^\prime - A \neq \phi$, $IC''^\prime - A \neq \phi$, $IC'^\prime - B \neq \phi$, $IC''^\prime - B \neq \phi$

In other words, there are some constraints involving object A in database $D_1$ that are not valid in database $D_2$ and there are some constraints involving object B in database $D_2$ that are not valid in database $D_1$. Thus, we can divide the set of constraints involving object A in database $D_1$ and object B in database $D_2$ into four non-overlapping, non-empty sets, $IC'^\prime - A$, $IC'^\prime - B$, $IC''^\prime - A$ and $IC''^\prime - B$. $IC'^\prime - A$ and $IC'^\prime - B$ are the sets that consist of constraints involving an object that are valid in another database, and $IC''^\prime - A$ and $IC''^\prime - B$ are the sets that consist of constraints involving an object in a database that are not valid in the other database.

Definition 4: A CBdisjoint B if the constraints involving the objects can be placed into two sets, $IC^\prime - A$ and $IC^\prime - B$ such that

$$\forall x, x \in IC^\prime - A, x \not\in D_2 \land$$
∀ y, y ∈ IC-B, y → D₁

In other words, no constraint involving object A in database D₁ is valid in database D₂ and no constraint involving object B in database D₂ is valid in database D₁.

The definitions presented above are mutually exclusive, i.e., only one type of relationship can exist between any pair of objects. The above relationships are also exhaustive, i.e, no other type of constraint-based relationship can exist between two objects.

3.4.1 Evaluation of constraint-based relationships

Generating constraint-based relationships means that we need to implement the operator \( \circ \) that evaluates the validity of a constraint on a database. Evaluating the validity of a constraint specified on a database D₁ in a database D₂ requires:

1) the transformation of the constraint (on D₁) into a valid query on database D₂.
2) evaluating the results to test the validity of the constraint.
Performing the first step in the context of a heterogeneous database presents a problem because the predicate and attribute names (in \( D_2 \)) that correspond to the ones specified in the integrity constraint (in \( D_1 \)) are not known and vice-versa. In addition, trying to determine the validity of every single integrity constraint will take a significant amount of time. Hence, certain heuristics are needed to limit the object pairs that will be subject to the constraint-based relationship generation process.

Fortunately, both these problems can be solved by using knowledge about schematic relationships (prefixed by \( SCH \)) generated using the heuristics described in section 3.3. We use the schematic knowledge generated above as the starting point for constraint-based relationship (prefixed by \( CB \)) generation. Objects that are schematically disjoint are not evaluated for constraint-based relationships. For each pair of objects that is schematically related, the integrity constraints involving object \( A \) in database \( D_1 \) are transformed into a legal query on database \( D_2 \) and vice versa. The results of these queries are then evaluated and \( CB \) relationships between the objects are generated based on the definitions presented above.
The process of transforming a constraint (represented as a prolog clause) into SQL queries is described in Draxler (1993). However, before we can transform a constraint on database $D_1$ into a query on database $D_2$ the following preconditions must be met:

1) for every predicate (entity) in the body (antecedent) of the constraint there must exist a corresponding predicate (entity) in database $D_2$ that is schematically related to it.

2) for every attribute involved in the constraint, there must exist a corresponding attribute belonging to a predicate in database $D_2$ that is schematically related to it.

3) the attribute in the head (consequent) of the constraint must have a schematically related attribute belonging to a predicate in database $D_2$.

Consider the example shown in fig. 3.4,
Schema A: Ships(x1,OWNER,SHIPTYPE,ORIGIN,x5,x6,x7)

IC-A:  OWNER = 'Onassis' ← Ships(x1,OWNER,supertanker,ORIGIN,x5,x6,x7) --- (a)

(if Shiptype = 'Supertanker' then Owner = 'Onassis')

Schema B: Boats(x1,COUNTRY,OWNER,TYPE)

IC-B:  ← Boats(x1, ICELAND, OWNER, TYPE) --- (b)

(there are no Boats with Country = 'ICELAND')

Predicates Ships and Boats are SCHequivalent.

Ships.OWNER and Boats.OWNER are SCHequivalent.

Ships.ORIGIN and Boats.COUNTRY are SCHequivalent.

Ships.SHIPTYPE and Boats.TYPE are SCHequivalent.

Fig. 3.4 Integrity Constraint Evaluation

To determine the constraint-based relationships between the predicates Ships and Boats we would need to evaluate the validity of each of the constraints against the other database. Using the schematic interschema relationships available to us we would transform the integrity constraint in (a) into the query:

\[
\text{retrieve } * \text{ from Boats where OWNER = 'Onassis'}
\]
We could then say that (a) is valid in the Boats database if all the tuples resulting from this query have TYPE = 'supertanker'.

Similarly, we would transform the constraint (b) into the query

\[
\text{retrieve Ships from Ships where ORIGIN = 'ICELAND'}
\]

and say that (a) is valid in the Ships database if no tuples are returned as a result of this query. If both constraints are valid then from definition 1 we would able to assert that Ships \(\equiv\) Boats. It should be noted that, we can also assert that Ships.SHIPTYPE \(\equiv\) Boats.TYPE and Boats.COUNTRY \(\equiv\) Ships.ORIGIN.

Constraint-based relationships among database objects serve two main purposes:

1) They provide a set of interschema relationships, among database objects, generated by analyzing a different knowledge source (integrity constraints).

2) They provide a starting point for the generation of a integrated set of integrity constraints.
The next two sections describe how constraint-based relationships can be used for these purposes.

3.5 "Real World" Interschema Relationship Generation

The schematic relationships and constraint-based relationships generated by the processes described in sections 3.3 and 3.4 respectively can be used to arrive at a set of interschema relationships among database objects that are closer to reality than either individual set of relationships taken alone. We refer to these relationships as "real world" relationships (RWR).

Table 3.4 shows the different "real-world" relationships that can be generated from constraint-based and schematic relationships among entity classes. For example, the table shows that, if it is true for two objects A and B that, A SCHequiv B and A CBequiv B then we would generate A RWequiv B. Table 3.4 has been generated with the goal of ensuring that the integrated schema generated can represent a complete set of integrity constraints,
without violating the semantics of the underlying database. We make extensive use of these rules during the integrity constraint integration process (described in section 3.7).

<table>
<thead>
<tr>
<th>Constraint-based / Schematic Relationships</th>
<th>A CBequiv B</th>
<th>A CBsubsume B</th>
<th>a CBOverlap B</th>
</tr>
</thead>
<tbody>
<tr>
<td>A SCHequiv B</td>
<td>A RWequiv B</td>
<td>A RWsubsume B</td>
<td>A RWoverlap B</td>
</tr>
<tr>
<td>A SCHsubsume B</td>
<td>A RWsubsume B</td>
<td>A RWsubsume B</td>
<td>A RWoverlap B</td>
</tr>
<tr>
<td>A SCHoverlap B</td>
<td>A RWoverlap B</td>
<td>A RWoverlap B</td>
<td>A RWoverlap B</td>
</tr>
</tbody>
</table>

Table 3.4 Rules for Generating "Real World" Interschema Relationships from Schematic and Constraint-based Relationships

In addition to automatically generating "real-world" relationships among database objects based on this table, the following heuristics are used to generate a prioritized set of items to assist the user/designer in IRI:

1) If there are constraint-based and schematic relationships between a pair of objects then that pair is likely to be related in some fashion. Hence, such pairs should be given high priority.
2) If a pair of objects is related schematically but no constraint-based relationship exists, then such a pair should be given lower priority.

3) If a pair objects is schematically disjoint, then they should be given lowest priority.

By generating such a prioritized list of pairs, the task of generating interschema relationships among objects can be achieved more efficiently, since the number of pairs that users have to deal with is reduced. More importantly, the users/designers can treat the items in the prioritized list with confidence, since they are generated using information from multiple knowledge sources.

3.6 Integrated Schema Generation

The strategy for integrated schema generation used in this paper is based on that described in Larson et. al. (1989). Table 3.5, summarizes the basic rules for transformation of entity classes, given a set of relationships among them. For example, if two entity classes A and B have been determined to be schematically equivalent, then based on the transformation rule in the first row of table 3.5 we would generate a single entity class AB’ in the integrated schema.
<table>
<thead>
<tr>
<th>Schematic Relationships</th>
<th>Transformation Rules</th>
</tr>
</thead>
<tbody>
<tr>
<td>A SCHequiv B</td>
<td>Generate a single entity class AB'</td>
</tr>
<tr>
<td>A SCHsubsumes B</td>
<td>Generate two entity classes A' and B', such that A' is a superclass of B'</td>
</tr>
<tr>
<td>A SCHoverlap B</td>
<td>Generate three entity classes, AB', A' and B', where AB' = RWS(A) U RWS (B) and A' and B' are subclasses of AB'.</td>
</tr>
<tr>
<td>A SCHdisjoint B</td>
<td>Generate A' and B'</td>
</tr>
</tbody>
</table>

Table 3.5 Summary of Larson et. al. (1989)'s Integrated Schema Generation Rules

3.7 Integrity Constraint Integration

The objective of this phase is to generate a set of integrity constraints at the integrated schema level. However, generating the unified set of integrity constraints is not a trivial process. In particular, one cannot simply transform the integrity constraints (involving a particular predicate in a local schema) into its equivalent form in the integrated schema. This is because an integrity constraint that involves an entity at the integrated level implies that the constraint is valid in all databases which have entity classes that can be mapped to
the integrity entity. Hence, such a transformation may result in erroneous information being supplied to the user.

Constraint-based relationships provide us with information about the validity of constraints specified in one database in other databases. Hence, once the CB relationships among objects have been determined, we can utilize them to generate an integrated set of integrity constraints that are applicable at the global level. In this section, we present the rules for generating such a set of integrated constraints.

Since every integrity constraint applicable to the integrated schemas will be associated with entity classes at the global level, generation of an integrated set of constraints is dependent on the schema integration strategy used to generate entities at the global schema level. We follow the guidelines presented in Larson et. al (1989) for schema integration (Table 3.5).

We present the heuristics for integration of integrity constraints for the four categories of "real world" relationships that can exist between two entity classes in the integrated schema. For each category of "real world" relationship we describe how the integrated integrity constraints for different categories of constraint-based relationships can be generated. Table 3.4 shows how the different categories of "real world" relationships in the integrated
schema can be generated based on the constraint-based and schematic relationships among local database objects. It is possible for two sets of objects to have the same real-world relationships but have differing schematic representations in the integrated schema. This is because the schematic representations of the "real world" relationships illustrated in table 3.4 are dependent on the degree to which the two (local database) objects are schematically related. To facilitate the integrity constraint integration process we further classify schematic equivalence and subsumption relationships (among local database objects) into strong and weak equivalence/subsumption relationships. The basis for this classification is presented below.

1) A SSE (Strong Schematic Equivalent) B if all attributes in A and B have been determined to be equivalent.

2) A WSE (Weak Schematic Equivalent) B if a majority of A and B's attributes are equivalent. However, both A and B have additional attributes that are not related using an equivalence relationship.
3) A SSUP (Strong Schematic Superclass) B if all of A’s attributes are either equivalent to or have a subsumption relationship with B’s attributes. However, B has some additional unrelated attributes.

4) A WSUP (Weak Schematic Superclass) B if a majority of A’s attributes are equivalent to or have a subsumption relationship with B’s attributes. However, both A and B have additional attributes that are not related to any attributes in the other entity class.

Definition 5: We define the operator $\theta$, the transformation operator, as the operator which transforms a constraint on a local database to a constraint at the integrated level. The $\theta$ operator ensures that the new constraint utilizes the appropriate predicate and argument names.

Case 1: A RWequiv B

Table 3.4 suggests that a RWequiv relationship can exist among objects with two types of schematic relationships (A SSE B and A WSE B). Such a relationship is represented, schematically, by a single entity class AB’ in the integrated schema, which represents the semantics of both the underlying entity classes. Table 3.4 also indicates that the only type of
constraint-based relationship that can result in the generation of the above real world relationship is a CBequiv relationship.

**Rule 1:** If there is a CBequiv relationship, then the individual integrity constraints can be transformed as follows:

\[
\forall x, x \in IC-A, x \theta AB' \\
\forall y, y \in IC-B, y \theta AB'
\]

where the \( \theta \) operator generates an integrity constraint that can be expressed in terms of the integrated schema's predicate \( AB' \).

Informally, rule 1 requires that all integrity constraints involving the predicate \( A \) and \( B \) into equivalent integrity constraints using the predicate \( AB' \). This makes sense because by definition of \( A \) CBequiv \( B \), we know that all constraints involving \( A \) are valid in database \( D_2 \) and all constraints involving \( B \) are valid in database \( D_1 \). Since \( AB' \) represents an integrated representation of the underlying entity classes, integrity constraints specified on \( AB' \) need to be applicable to both underlying databases, which we know is the case because
A CBequiv B. Hence, rule 1 does not violate any of the semantics of the underlying databases.

Consider the databases in Fig. 3.4. Given these conditions schema integration is likely to result in a predicate $Ship_Boats(x1, OWNER, SHIPTYPE, ORIGIN, x5, x6, x7, x8)$ where $x8$ is an attribute corresponding to $x1$ in the predicate Boats. Based on rule 1, we would generate the following integrity constraints at the integrated level:

$$\text{OWNER} = 'Onassis' \leftarrow Ship_Boats(x1, OWNER, supertanker, ORIGIN, x5, x6, x7, x8)$$

$$\leftarrow Ship_Boats(x1, OWNER, SHIPTYPE, ICELAND, x5, x6, x7, x8)$$

Note that without the existence of the CBequiv relationship we would not be able to generate the transformed integrity constraints since such an integrity constraint assumes that the constraints apply to both the underlying databases which may or may not be the case.

Case 2: A RWsubsumes B

Table 3.4 indicates that such a relationship can exist among objects that are schematically equivalent or subsumed. The schematic representation of such a relationship in the
integrated schema is dependent on whether the objects in question are strongly or weakly equivalent/subsumed. Table 3.4 also shows that a RWsubsume relationship can exist when the underlying objects have a CBequiv or CBsubsume relationship. We present rules for integrity constraint integration for these two types of CB relationships.

\[ A \text{ CBequiv } B \]

Two cases of schematic relationships are possible: A SSUP B and A WSUP B.

**Case 1: A SSUP B**

A RWsubsumes relationship among two objects that have a SSUP relationship is represented in the integrated schema by two entity classes A' and B', such that B' is a subclass of A'. Integrity constraints belonging to the original entity classes A and B are integrated as follows:

**Rule 2:** If A CBequiv B, then the individual integrity constraints involving A and B are transformed as follows:
i) \( \forall w, w \in IC-A, w \theta A' \),

ii) \( \forall x, x \in IC-B, x \theta B' \),

iii) \( \forall y, y \in IC-B, y \theta A' \),

iv) \( \forall z, z \in IC-A, z \theta B' \),

Informally, we transform all integrity constraints involving the predicates \( A \) and \( B \) into equivalent integrity constraints using the predicates \( A' \) and \( B' \) respectively. In addition, \( B' \) inherits all of \( A' \) constraints.

The first two items in rule 2 transform all integrity constraints involving the predicate \( A \) and \( B \) into equivalent integrity constraints using the predicate \( A' \) and \( B' \) respectively. This transformation is justified because \( A' \) and \( B' \) map to \( A \) and \( B \) in their respective databases and hence integrity constraints involving \( A' \) and \( B' \) respectively refer to the original databases, where by definition the integrity constraints are valid.

The third item in rule 2 suggests that all of \( B' \) constraints be transformed into constraints involving \( A' \). This transformation is also justified because \( A \ CBequiv B \). The final item in rule 2 suggest that \( B' \) inherits all of \( A' \) constraints. Such an inheritance is possible and is correct because, we already know (from the definition of \( A \ CBequiv B \)) that \( \forall w, w \in IC- \).
A, w \in D_2, i.e., all constraints involving A' have been shown to be applicable to database D_2. Hence, z \in B' does not violate the semantics of the database.

**Case 2: A WSUP B**

A RWsubsumes relationship among objects with a WSUP relationship is represented in the integrated schema by an entity class AB' which has A' and B' as its subclasses. AB' contains all the attributes of A that have an equivalence or subsumption relationship with attributes of B. The additional attributes in A, however, are attributes of A' alone. All attributes of AB' are inherited by A' and B' respectively.

**Rule 2a:** If \( A \equiv B \), then the individual integrity constraints involving A and B are transformed as follows:

i) \( \forall u, u \in IC-A, u \in AB' \),

ii) \( \forall v, v \in IC-B, v \in AB' \),

iii) \( \forall w, w \in IC-A, w \in A' \),

iv) \( \forall x, x \in IC-B, x \in B' \),

v) \( \forall y, y \in IC-B, y \in A' \),
vi) \( \forall z, z \in IC-A, z \theta B' \)

These rules are similar to rule 2. Since A CBequiv B, the first two items suggest that all constraints involving A and B be transformed into constraints using AB' (Note: these do not include constraints specified on unrelated attributes, since they do not play a role in determining CB equivalence). The next two items suggest that constraints involving A and B be transformed into constraints involving A' and B' respectively. The final two items represent the inheritance of additional constraints, generated due to CB equivalence, from AB'.

**A CBsubsume B**

From table 3.4, we can see that when A CBsubsumes B, a A RWsubsumes B relationship can be a result of a A SCHequiv B or A SCHsubsume B relationship. Since, schematic equivalences/subsumption can be either weak or strong, we need to deal with four subcases of schematic relationships, A SSE B, A WSE B, A SSUP B and A WSUP B.

**Case 1: A SSE B**
A RW subsumes relationship among two objects that are SSE (yet differ based on their constraints) is represented by two entity classes A' and B', where A' is a superclass of B'. All the attributes belonging to A and B are transformed into attributes in A' and are subsequently inherited by B'. Integrity constraint integration is performed according to the following rule:

**Rule 3:** If A CB subsume B, then the individual integrity constraints are transformed as follows:

i) \( \forall w, w \in IC-A, w \theta A' \),

ii) \( \forall x, x \in IC'-B, x \theta B' \),

iii) \( \forall y, y \in IC-B, y \theta A' \),

iv) \( \forall z, z \in IC-A, z \theta B' \),

Informally, the first item transforms all constraints involving A into constraints involving A'. The second item transforms the subset of constraints involving that are applicable to A into constraints on A'. The third item transforms all of B' constraints into constraints on B'. The final item shows that all of A's constraints are inherited by B'
Case 2: A WSE B

A RWsubsumes relationship among two objects that are WSE is represented (in the integrated schema) by creating an entity class AB' and two subclasses A' and B'. AB' is a representation of the commonalties between the entity classes A' and B'. Thus, the attributes in A' and B' that are related to each other are transformed into attributes in AB'. However, the unrelated attributes are transformed into attributes in A' and B' alone.

Rule 3a: If A CBsubsume B, then the individual integrity constraints are transformed as follows:

i) \( \forall u, u \in IC-A, u \theta AB' \),

ii) \( \forall v, v \in IC'-B, v \theta AB' \),

iii) \( \forall w, w \in IC-A, w \theta A' \),

iv) \( \forall x, x \in IC-B, x \theta B' \),

v) \( \forall y, y \in IC'-B, y \theta A' \),

vi) \( \forall z, z \in IC-A, z \theta B' \)
Since A CBsubsume B, all of A's constraints are applicable to both A and B and thus can be transformed into constraints on AB'. However, only those constraints applicable to A (belonging to the set IC'-B) can be transformed into constraints on AB'. This is represented in the first two items of the rule. The next two items indicate that constraints involving A and B should be transformed into constraints involving A' and B' respectively. The last two items ensure that additional constraints in AB' (not specified on A' and B' respectively) are inherited. Thus, A' inherits all the (transformed) constraints of B that are applicable to A and B' inherits all of A's (transformed) constraints.

**Case 3: A SSUP B**

This case is represented (schematically) in the integrated schema by two entity classes A' and B', such that, A' is a superclass of B'. Thus, this case is similar to A SSE B. However, the difference is that B' will contain attributes that are not inherited from A'. The rules for integrity constraint integration, however, are the same as in case 1.

**Rule 3b:** If A CBsubsume B, then the individual integrity constraints are transformed as follows:

i) $\forall w, w \in IC-A, w \theta A'$,
Case 4: A WSUP B

This case is represented schematically using a representation similar to A WSE B case. We generate an entity class AB' that is representative of the commonalities among A and B, and generate A' and B' as subclasses of AB'. A' and B' contain representations of the unrelated attributes in A and B (in addition to the related attributes which are inherited from AB'). The rules for integrity constraint integration in this case are similar to case 2.

Rule 3c: If A CBsubsume B, then the individual integrity constraints are transformed as follows:

i) \( \forall u, u \in IC-A, u \theta AB' \),

ii) \( \forall v, v \in IC'-B, v \theta AB' \),

iii) \( \forall w, w \in IC-A, w \theta A' \),

iv) \( \forall x, x \in IC-B, x \theta B' \),
v) \( \forall y, y \in IC'-B, y \theta A' \),

vi) \( \forall z, z \in IC-A, z \theta B' \)
\[
\text{Ships}(x_1, \text{OWNER}, \text{SHIPTYPE}, \text{ORIGIN}, \text{WT}, x_6, x_7)
\]

\[
\text{Oil\_Tankers}(x_1, \text{COUNTRY}, \text{OWNER}, \text{TYPE}, \text{DEADWT})
\]

with the equivalences:

\[
\text{Ships} \text{RW}_{\text{subsumes}} \text{Oil\_Tankers}
\]

\[
\text{Ships}.\text{SHIPTYPE} \text{RW}_{\text{equiv}} \text{Oil\_Tankers}.\text{TYPE}.
\]

\[
\text{Ships}.\text{OWNER} \text{RW}_{\text{equiv}} \text{Oil\_Tankers}.\text{OWNER}.
\]

\[
\text{Ships}.\text{ORIGIN} \text{RW}_{\text{equiv}} \text{Oil\_Tankers}.\text{COUNTRY}.
\]

\[
\text{Ships}.\text{DEADWT} \text{RW}_{\text{equiv}} \text{Oil\_Tankers}.\text{DEADWT}
\]

\textbf{IC-A:}

\[
\text{OWNER} = '\text{Onassis}' \leftarrow \text{Ships}(x_1, \text{OWNER}, \text{supertanker}, \text{ORIGIN}, \text{WT}, x_6, x_7)
\]

\[
\text{SHIPTYPE} = '\text{supertanker}' \leftarrow \text{Ships}(x_1, \text{OWNER}, \text{SHIPTYPE}, \text{ORIGIN}, \text{WT}, x_6, x_7), \text{WT} > 200
\]

\textbf{IC-B:}

\[
\leftarrow \text{Oil\_Tankers}(x_1, \text{ICELAND}, \text{OWNER}, \text{TYPE}, \text{DEADWT})
\]

\[
\text{TYPE} = '\text{pressurized tanker}' \leftarrow \text{Oil\_Tankers}(x_1, \text{uae}, \text{OWNER}, \text{TYPE}, \text{DEADWT})
\]

\textbf{Integrated Schema:}

\[
\text{Int\_Ships}(x_1, \text{OWNER}, \text{SHIPTYPE}, \text{COUNTRY}, \text{WT}, x_6, x_7)
\]

\[
\text{Int\_Oil\_Tankers}(x_1, \text{COUNTRY}, \text{OWNER}, \text{TYPE}, x_5)
\]

\textbf{Fig. 3.5 Example Schema 2}
If Ships and Oil_Tankers had a CBequiv or CBsubsume relationship, the set of integrated constraints, applying rule 2, would be

**IC-Int_Ships:**

\[
\text{OWNER} = \text{'Onassis'} & \leftarrow \text{Int}_{-} \text{Ships}(x1, \text{OWNER}, \text{supertanker}, \text{ORIGIN}, \text{WT}, x6, x7) \\
\text{SHIPTYPE} = \text{'supertanker'} & \leftarrow \text{Int}_{-} \text{Ships}(x1, \text{OWNER}, \text{SHIPTYPE}, \text{ORIGIN}, \text{WT}, x6, x7), \text{WT} > 200 \\
& \leftarrow \text{Int}_{-} \text{Ships}(x1, \text{OWNER}, \text{SHIPTYPE}, \text{iceland}, \text{WT}, x6, x7) \\
\text{SHIPTYPE} = \text{'pressurized tanker'} & \leftarrow \text{Int}_{-} \text{Ships}(x1, \text{OWNER}, \text{SHIPTYPE}, \text{uae}, \text{WT}, x6, x7)
\]

**IC-Int_Oil_Tankers:**

\[
\text{OWNER} = \text{'Onassis'} & \leftarrow \text{Int}_{-} \text{Oil}_{-} \text{Tankers}(x1, \text{COUNTRY}, \text{OWNER}, \text{supertanker}, \text{DEADWT}) \\
\text{SHIPTYPE} = \text{'supertanker'} & \leftarrow \text{Int}_{-} \text{Oil}_{-} \text{Tankers}(x1, \text{COUNTRY}, \text{OWNER}, \text{supertanker}, \text{DEADWT}), \\
\text{DEADWT} > 200 \\
& \leftarrow \text{Int}_{-} \text{Oil}_{-} \text{Tankers}(x1, \text{iceland}, \text{OWNER}, \text{TYPE}, \text{DEADWT}) \\
\text{TYPE} = \text{'pressurized tanker'} & \leftarrow \text{Int}_{-} \text{Oil}_{-} \text{Tankers}(x1, \text{uae}, \text{OWNER}, \text{TYPE}, \text{DEADWT})
\]

It should be noted that the process of integrity constraint integration has resulted in additional knowledge (in the form of constraints) being generated at the integrated level.
The ability to generate such additional constraints results in a more complete definition of the integrated schema. It also means that there are more constraints associated with some of the entity classes at the integrated level. The use of these additional constraints for semantic query optimization can potentially result in substantial savings. An important thing to note that it is only possible to inherit constraints that have been specified on attributes that have a RW relationship among them.

Case 3: A RWoverlap B

Table 3.4 indicates that this case can occur for varying combinations of schematic and constraint-based relationships. However, in all cases the schematic representation of such relationships is by a new entity class AB' which has A' and B' as its two subclasses, where AB' can be thought of as representing the commonalities between A and B. Thus, in the case where A SSE B, AB' would contain only the equivalent attributes. Since, a common schematic representation is used to represent all the RWoverlap cases (unlike the RWS case), the rules for integration of integrity constraints can be described in the context of the CB relationship between A and B. As can be seen from table 3.4, three possible CB relationships can exist among these classes.
Rule 4: If A CBequiv B, then the individual integrity constraints are transformed as follows:

i) \( \forall x, x \in IC-A, x \theta A' \),

ii) \( \forall y, y \in IC-A, y \theta AB' \),

iii) \( \forall z, z \in IC-A, z \theta B' \)

ia) \( \forall u, u \in IC-B, u \theta B' \),

iia) \( \forall v, v \in IC-B, v \theta AB' \),

iiia) \( \forall w, w \in IC-B, w \theta A' \)

The transformation rules for the overlap can be interpreted as follows: rules i) and ia) are obvious, since A' and B' are the integrated representation of A and B respectively. Rules ii) and iia) transform constraints involving A and B that are valid in the other databases. Since, we have a CBequiv relationship, all constraints are valid. Hence, we transform all constraints involving A and those involving B into constraints on AB'. An implicit assumption made here is that, if all constraints involving A and B are valid, then the attributes involved in the constraints would have been determined to be part of the overlap.
between A and B and would exist in AB'. We will elaborate on this point further in the section on attribute-level constraint integration. Rules iii) and iiiia) then represent the inheritance of constraints by the other subclass of AB'.

**Rule 5:** If A CBsubsume B, then the individual integrity constraints are transformed as follows:

\[
\forall x, x \in IC-A, x \theta A', \\
\forall y, y \in IC-A, y \theta AB', \\
\forall z, z \in IC-A, z \theta B' \\
\forall u, u \in IC-B, u \theta B', \\
\forall v, v \in IC'-B, v \theta AB', \\
\forall w, w \in IC'-B, w \theta A',
\]

Rule 5 is similar to rule 4 with one difference. Instead of transforming all constraints involving B into constraints involving AB', we only transform those constraints that are valid in database D_1. Thus, only these constraints get inherited and transformed into constraints involving A'.
Rule 6: If A C overlap B, then the individual integrity constraints are transformed as follows:

∀ x, x ∈ IC-A, x \( \theta \) A',
∀ y, y ∈ IC'-A, y \( \theta \) B',
∀ z, z ∈ IC'-A, z \( \theta \) AB'

∀ u, u ∈ IC-B, u \( \theta \) B',
∀ v, v ∈ IC'-B, v \( \theta \) A',
∀ w, w ∈ IC'-B, w \( \theta \) AB'

Rule 6 is also similar to rules 4 and 5 with the difference being that only those constraints involving A and B, that are valid in the other database are transformed into constraints involving AB' and inherited by B' and A' respectively.

Case 4: A RW disjoint B
In this case A and B are transformed into completely unrelated entities A' and B' in the integrated schema. According to the heuristics utilized in generating constraint-based relationships, relationships between RWdisjoint entity classes are not generated. Hence, the only possible CB relationship is CBdisjoint. The transformation rules are hence simple, and involve transformation of the constraints involving A and B into constraints involving A' and B' respectively. Formally,

\[ \forall x, x \in IC-A, x \theta A' \]
\[ \forall y, y \in IC-B, y \theta B' \]
This chapter describes the design and implementation of a system that implements the techniques presented in the Chapter 3. We divide the description of the system into two sections: a) the traditional schema integration components and b) the integrity constraint integration components.

4.1 Traditional Schema Integration Components

Figure 4.1 shows the various phases in a typical schema integration methodology, emphasizing the data dependencies among the various tasks. Automation of each task requires the development of a knowledge (computational) engine to solve that task. Hence, most traditional integration methodologies would need a schema translation engine, an interschema relationship identification engine, an integrated schema generation engine and a schema mapping generation engine.
Fig. 4.1 Dependencies among Schema Integration Tasks
Although software tools can be developed to facilitate many of the tasks described above, human interaction is required to accomplish the following activities during the schema integration process:

1. in the schema translation phase user interaction is needed to translate local schemas, or interact with software translation tools, or modify schemas translated using the translation tools. Such interaction is needed to ensure that the individual translated schemas reflect the underlying databases accurately.

2. confirmation, rejection or modification of relationships generated by the automated conflict identification component.

3. identification and assertion of conflicts that may have been overlooked by the automated conflict identification component.

4. naming/renaming of attributes, relationships and entity classes in the integrated schema.
5. enhancing the semantics of objects in the integrated schema by adding properties to objects in the schema or generating constraints among entity classes.

6. resolving conflicts among objects that could not be resolved by the conflict resolution component.

It is clear that in addition to any computational tools, interaction with users/designers plays a very important role during every step of the schema integration process. Researchers in the schema integration literature agree that the inherently semantic nature of the problem implies that complete automation of these tasks is very difficult, if not impossible (Sheth and Gala, 1989), and point to the need for explicit support for human interaction during the schema integration process. However, to our knowledge none of the existing methodologies have been able to provide an elegant solution to the problem of human interaction during schema integration. Below, we describe an architecture of a schema integration system that explicitly accommodates for the human interaction needed during schema integration. A unique characteristic of the system is that it utilizes blackboard architectures to facilitate the interaction identified above.
4.1.1 A Blackboard Architecture for Schema Integration

The blackboard model can be viewed as a natural evolution of AI systems which seeks to eliminate the inherent weaknesses in the classical expert systems architecture (Engelmore and Morgan, 1989). Fig 4.2 illustrates the structure of a typical expert system. In this paradigm of problem solving the inference engine uses the knowledge base and any information available in the working memory to produce new results until a termination condition is detected. There are two major drawbacks of this approach (Engelmore and Morgan, 1989; Parsaye et. al., 1989):

1) The use of the knowledge is implicit in the structure of the knowledge base. This becomes a problem for large systems, since the knowledge base becomes difficult to understand and hence difficult to maintain.

2) The knowledge representation scheme is dictated by the type of inference engine.
Fig. 4.2 "Classical" Expert System
Fig. 4.3 Architecture of a Typical Blackboard System
Fig. 4.3 shows the structure of a typical blackboard system. It consists of three basic components (Rich and Knight, 1989; Parsaye et. al., 1989):

**Knowledge Sources**: A set of independent knowledge sources (engines) which need to share information for solving a particular problem.

**Blackboard**: A shared global database (or data structure) used as the sole means of communication and interaction among the knowledge sources.

**Scheduler or control system**: A unit that monitors changes to the blackboard and decides on what actions to take next.

Blackboard architectures provide an elegant mechanism for cooperation among multiple knowledge sources. Since, the users/designers can be regarded as a type of knowledge source, blackboard architectures can provide an elegant solution to the problem of human interaction during the schema integration process. Several characteristics of blackboard architectures lend themselves for use in schema integration:

a. The blackboard provides a mechanism for independent knowledge sources to elegantly and effectively share information with each other. Since the schema
integration process can be divided into several independent tasks, such as, schema translation, conflict identification and conflict resolution, a blackboard can be used to share information among these tasks.

b. Each knowledge source encompasses both the knowledge and the inference mechanism needed to use that knowledge. By requiring that information be shared solely through the blackboard it is possible to have these knowledge sources, each (possibly) using a different knowledge representation scheme and inference mechanism, interacting with each other to solve a problem. Since each task in the schema integration process has a well defined set of inputs and outputs it is possible to use the blackboard to coordinate interaction among the various agents (computational and human).

c. The nature of schema integration tasks also means that some of the tasks can be performed concurrently provided the data required for computation is available. For example, the integrated schema generation component can make decisions on integrating objects concurrently with the interschema relationship generation process, as long as some objects have been identified as being similar previously by the interschema relationship component or by human agents.
The Blackboard

The overall architecture of the traditional schema integration components of our system is shown in fig. 4.4. The figure shows that the system consists of three semi-automated components: a Schema Translation Engine, an Interschema Identification (IRI) or Conflict Identification (CI) engine and an Integrated Schema Generation (ISG) or Conflict Resolution (CR) engine.

The structure of the blackboard is a key component of the schema integration system. As shown in fig. 4.5, information on our blackboard resides in one of four levels. The higher the level, the closer the information at the level is to a goal state. Each schema integration task is performed by one or more knowledge sources. All knowledge sources (except the human integrators) in the schema integration system utilize a forward reasoning mechanism. Forward reasoning engines utilize information available in the initial state to make progress towards their goal. Thus, they utilize data from lower levels and output information to higher levels. A brief description of the information stored at each level of the blackboard is given below:
Fig. 4.4 Blackboard Architecture for Schema Integration
<table>
<thead>
<tr>
<th>Level</th>
<th>Content</th>
</tr>
</thead>
<tbody>
<tr>
<td>Goal Level</td>
<td>Integrated Schema</td>
</tr>
<tr>
<td>Fact Level</td>
<td>User-Confirmed Interschema Relationships</td>
</tr>
<tr>
<td>Assertion Level</td>
<td>System Generated Interschema Relationships</td>
</tr>
<tr>
<td>Data Level</td>
<td>Individual Schemas to be integrated</td>
</tr>
</tbody>
</table>

Fig. 4.5 Four Level Blackboard Architecture
Data Level

A USM representation of each schema that needs to be integrated is stored at the lowest level of the blackboard. This information is the output of the schema modeling and translation process and represents the raw data that is going to be utilized by the conflict identification engine. Information is stored in the form of pairs of entity classes, relationships and attributes that need to be processed.

Using our example schemas (fig 4.6), the data level of the blackboard would contain the four entity class pairs, Customers & Borrowers, Customers & loan, Accounts & Borrower and Accounts & Loan. In addition, the list of the entity classes' attributes and relationships in which each of these entity classes participates will also be available.

Assertion Level

This level stores the output of the interschema relationship identification (conflict identification) process. Relationships (assertions) generated by the conflict identification process are best classified as being transient in nature because they may not always be accurate and may need confirmation from the user. Assertions about both
Fig. 4.6 Example Translated Schemas
heterogeneous and homogeneous conflicts are stored at this level. Each section, except the object relationships section, at this level contains assertions indicating the similarity values of a pair of objects. *Customers - Borrowers, 0.90* and *Accounts - Loans, 0.70* (the numeric value indicating the index_of_similarity) would be examples of *transient* assertions stored in the entity class definition section at the assertion level. Similarly, similarity indices for attributes and relationships would be stored in the other sections (of the blackboard) at this level. Once the entity class definition, attribute and relationship similarities for a pair of entity classes have been determined, their weighted average is computed and asserted in the entity class similarity section of the blackboard.

The section labeled *object relationship* stores assertions about relationships among objects. These relationships can be one of four types, *equivalent, em contains/contained-in, overlap and disjoint*. These relationships are generated by the system based on heuristics (described in section 3.3.5) that cumulatively evaluate the various similarity indices (entity class definition, attribute and relationship) available from the blackboard.

*Fact Level*
The information at this level is generated as a result of human interaction with the system. Transient assertions about relationships between various database objects (from the assertion level) that have been confirmed by the human integrators are stored at the fact level. Some of the data at this level may also be generated by designers' asserting facts about relationships between objects that were not generated by the conflict identification engine. The fact level stores facts corresponding to relationships between entity classes, attributes and relationships, as well as facts about heterogeneous conflicts. These assertions take the form of one of the four categories of relationships listed previously. Customers contains Borrowers and Accounts overlap Loans are examples of facts that could be stored in the entity class section of this level of the blackboard. Similarly, relationships among attributes and relationships would be stored in the other two sections of this level of the blackboard.

**Goal Level**

The information at this level is generated by the conflict resolution engine. This engine, in conjunction with human designers, where needed, utilizes the information at the fact level to generate integrated representation(s) of the underlying schemas. The rules
presented in Larson et. al. (1989) and summarized in table 3.5 are used by the integrated schema generation engine. For example, if two entity classes A and B are determined to be equivalent, the rules for integration suggest that a single integrated entity class be generated in the integrated schema. The name of this entity class could either be generated by the system or be specified by the user.

The Knowledge Sources

The schema integration system consists of four types of knowledge sources: schema translation, conflict identification, conflict resolution and the human integrators (Fig. 4.4). Each type of knowledge source may consist of one or more components.

*Schema Translation*

The schema translation engine translates the database schemas being integrated from their current data model to the Unifying Semantic Model (USM) and posts the schemas onto the data level of the blackboard. In our toolkit, we provide tools that translate from the hierarchical and relational models to the USM. Liu (1993) and Fu (1993) present the description of a toolkit that translates from the E-R model and hierarchical
models to the USM respectively. Both these toolkits use heuristics to evaluate the local schema objects' properties and generate equivalent USM constructs. However, since the data model being used to represent the translated schema has more semantic capabilities than the original model, significant interaction with database designers/users is needed to arrive at an accurate representation of the underlying databases.

*Conflict Identification (CI)*

The CI engine analyzes the schemas from the data level of the blackboard and generates *transient* assertions about the similarity or dissimilarity between entity classes, attributes and relationships in the schemas being integrated. The assertions generated by the engine are posted on the *assertion level* of the blackboard.

The CI engine consists of three distinct components: entity class definition similarity, attribute similarity and relationship similarity computation engines. These engines utilize the heuristics presented in sections 3.3.1-3.3.3 to generate these assertions. The different components of the CI engine are activated by data becoming available on the blackboard. The CI scheduler is responsible for monitoring the blackboard and
triggering the appropriate components as necessary. Each CI engine component uses information about the pair of objects that it needs (from the data level), performs similarity computations and posts *transient* assertions on the assertion level of the blackboard. In addition, each engine also posts *transient* assertions about any heterogeneous conflicts on this level of the blackboard.

In our example schema, the entity class definition similarity computation engine would evaluate the four pairs of entities, *Customers & Borrower*, *Customers & Loan*, *Accounts & Borrower and Accounts & Loan*, and generate the result shown in fig. 4.7. Similarly, similarity indices for attribute and relationship pairs would be generated by the attribute similarity and relationship similarity computation engines respectively. However, the heuristics described in sections 3.3.2 & 3.3.3 are used to avoid attribute and relationship similarity computations for dissimilar entity classes.
Fig. 4.7 An Example of a Populated Blackboard
Conflict Resolution (CR)

The CR engine consists of three components: the CR scheduler, the preprocessor and the integrator.

The preprocessor operates on the similarity indices available at the assertion level of the blackboard. It uses heuristics to cumulatively evaluate the various similarity indices and arrive at relationships among pairs of entity classes. As pointed out earlier, these relationships may be equivalence, subsumption, overlapping or disjoint (Larson et. al., 1989). The results of the computation are posted to the object relationships section of the assertion level of the blackboard for confirmation by human integrators.

The integrator utilizes information from the various areas of the fact level of the blackboard to integrate the individual schemas. Output from the integrator is posted to the goal level of the blackboard. Finally, the CR scheduler monitors the blackboard and triggers the preprocessor and integrator when appropriate. In our example, the preprocessor would evaluate the similarity indices and generate the transient assertions *Customers equivalent to Borrowers* and *Accounts overlap Loans* as shown in fig. 4.7. These would then be confirmed/modified by the user to generate the set of facts shown
Fig. 4.8 Example Integrated Schema
in the fact level (fig. 4.6). The integrator would then operate on these facts and generate the integrated schema shown in fig. 4.8.

**Human Integrators**

Human integrators are the last type of knowledge source. They interact with the computational engines through the blackboard. This interaction is the focus of the next section.

**Human-Computer Cooperation using the Blackboard**

This section elaborates on how the use of blackboard architectures facilitates interaction between human agents and the various computational knowledge sources.

**Human Interaction**

The primary interaction between human agents and computational knowledge sources is through the blackboard. Specifically:
1. User interaction with the system is needed during schema translation to ensure that an accurate and comprehensive representation of the underlying databases is generated. Such interaction is elicited in two ways: a) During the translation itself when user input is sought to confirm heuristics-based decisions made by schema translation tools and b) after the translation of the local schema is complete, by providing the opportunity for a group of users to modify the translated schemas. Each schema generated by this process appears on the data level of the blackboard. This causes the CI scheduler to trigger the appropriate components of the CI engine.

2. Information on the assertion level of the blackboard becomes available to human agents as and when they are posted by the various components of the CI engine as well as the preprocessing component of the CR engine. The role of human agents is to transform these transient assertions into facts by modifying or confirming them. Human agents can view the similarity values available at the assertion level to assist them in the decision making process. They can also view other objects on the blackboard, such as, individual schemas, the (partially) integrated schema, facts and other assertions. Human agents can also assert facts about relationships among objects independent of the computational engines. This is particularly relevant for heterogeneous conflicts, where the system may not be able to detect all possible
conflicts. In such cases, human agents can operate directly on information at the data level and bypass the assertion level by posting facts onto the fact level of the blackboard.

3. The CR scheduler monitors the fact level of the blackboard. When facts about entity class, attribute or relationship similarity are posted at this level, the CR scheduler detects them and triggers the integration component of the conflict resolution engine. User interact with the CR engine to name/rename entity classes, relationships and attributes in the integrated schema. For example, while integrating the equivalent attributes SSNO and IDNO into a single attribute, user input is needed to define a new name, such as CUSTNO, for the integrated attribute. The results of the integration process are posted to the goal level of the blackboard. Users can view/modify the properties of various objects in the integrated schema as they appear on the fact level of the blackboard. Users can also interact with the blackboard to resolve conflicts that the CR engine is unable to resolve. Since both the CI and CR engines operate in the background and the human agents operate solely on information on the blackboard, it is possible for human agents to work concurrently with the computational engines (as long as there is information available on the blackboard), thus achieving significant savings in time. This concurrency is a key advantage of using blackboard architectures.
Use of the system

The toolkit has been developed using C under Microsoft Windows. The majority of the tools that comprise the schema integration system are designed to operate in a multi-user environment. We have adapted the concept of Electronic Meeting Systems technology (Nunamaker et. al., 1991) to support the multi-user aspects of our tools. A network of IBM PS/2 workstations, connected together by a token ring LAN is currently being used as a test platform.

Users begin interacting with the schema integration system during the schema translation phase. Users interact with the translation tools via system prompted dialogs to ensure that the translated schemas reflect the semantics of the underlying databases accurately. Users can also view/modify the translated schemas or model new schemas using a collaborative database modeling tool (Ram and Ramesh, 1994). This tool provides a graphical environment for multiple users to work on modifying/viewing USM schemas. An example of an USM schema modeled using the tool is shown in Fig. 4.9. The tool provides for appropriate consistency and concurrency control mechanisms to ensure that schemas are consistent across the network. The tool also
Fig. 4.9 Sample Screen showing an USM Schema
Fig. 4.10 Sample Screen showing the Communications Facility
provides users with (computer) communication channels for resolving differing viewpoints among users during any phase of the modeling process. Fig. 4.10 shows the communication paradigm that users can use to communicate with each other.

Once the schemas to be integrated have been translated, the conflict identification engine (CI) (operating in the background) begins the conflict identification phase by analyzing entity class pairs, as well as attribute and relationship pairs associated with each entity class pair. As it posts the information associated with these pairs on the assertion level of the blackboard, it also triggers the appropriate CI component. Each CI component, i.e., entity class definition, attribute or relationship similarity, operates independently and in parallel as long as the requisite data is available on the blackboard. Using the example shown in Fig. 4.7, the CI scheduler marks the entity pair Customers and Borrowers in the entity class pair area of the fact level of the blackboard and triggers the entity class definition similarity (EDS) computation engine. It then monitors the entity class definition area at the assertion level of the blackboard and awaits the output of the engine. Once the EDS engine asserts a similarity value, the CI scheduler, depending on whether the similarity value asserted crossed a threshold, a) marks the relationship and attribute pairs (for the current set of entity classes being compared) at the data level for processing and posts the next entity class pair or b)
asserts that the current entity class pair is disjoint and posts the next entity pair, thus eliminating the need for attribute and relationship pair similarity computation for disjoint entities.

The CI scheduler also monitors the assertions posted by the individual CI engines, and generates the cumulative entity class similarity index using the information generated by the other conflict identification components. Heterogeneous conflicts (if any) are detected by the individual computation engines and posted directly onto the assertion level. Finally, the CI scheduler triggers the preprocessor component of the CR engine whenever a complete set of similarity values for a pair of entity classes is available on the blackboard. The preprocessor analyzes the various indices of similarity and generates assertions about relationships among schema objects. This information is posted onto the assertion level of the blackboard.

Users can interact with the schema integration as long as appropriate information is available on the blackboard. Users generate facts from transient assertions by either confirming or modifying assertions generated by the various CI components. Fig.4.11 shows an example of a dialog box used to assert relationships between
Fig. 4.11 Relationship Assertion between Entity Classes
two entity classes. Any information posted on the blackboard is available to assist the user in the generation of these facts.

As facts are generated by the user, the CR scheduler detects their appearance on the fact level of the blackboard and triggers the integration component of the CR engine. The CR engine (also operating in the background) analyzes the facts to generate integrated schema objects at the goal level of the blackboard. Once again users interact with the system through forms. The system prompts users with dialog boxes for activities such as, naming/renaming of objects in the integrated schema and resolution of conflicts that cannot be resolved by the system. Modification of properties of objects in the integrated schemas to generate a semantically accurate integrated schema is also achieved through dialog boxes. For example, users could enhance the definition of the subclass relationship between Customers and Borrowers by specifying that Borrowers is an attribute-defined subclass of Customers, i.e., there exists an attribute in the Customers entity class that can be used to identify the set of entities that belong to the entity class Borrowers.
The prototype of the traditional components of the system using this was developed using C and Visual Basic and runs under the Microsoft Windows environment. The system is capable of running on any network running Novell version 3.11 and above.

4.2 Components of the Enhanced Methodology

The components of the system that implement the steps unique to the enhanced methodology are: constraint-based relationship generator, "real world" relationship generator and the integrity constraint integrator. Currently, these components have been implemented as independent modules and have not been integrated with the blackboard architecture described in the previous section. Below, we present a description of the current functionality of the various components and how we anticipate merging these components with the blackboard based system.

4.2.1 Constraint-based (CB) Relationship Generator

This component uses the schematic interschema relationships, generated during the conflict identification phase of schema integration, and the integrity constraints specified on the databases to generate CB relationships among database objects. The
evaluator reads in the schematic relationships and integrity constraints into appropriate data structures. For every pair of objects that are schematically related it transforms integrity constraints involving that object into a form that can be used by the Prolog to SQL compiler (Draxler, 1993) to generate SQL queries. Results of the queries for each set of integrity constraints can then be used to generate constraint based interschema relationships using the techniques described in section 3.1. It is assumed that results of queries to the database can be formatted to suit the needs of the constraint-based relationship generator. The output of the constraint-based relationship generator is a set of CB relationships among entity classes and attributes. These relationships are appended to the schematic relationships generated during the conflict identification phase and used as input to the "real world" relationship generator described below. When this module is integrated with the blackboard architecture described in the previous section, it can be expected that this module will be triggered by the appearance of transient assertions generated by the schematic conflict identification component. The output of this module would then be a set of transient assertions being generated to a separate constraint-based relationships section of the blackboard.

4.2.2 "Real World" Interschema Relationship Generator
This module utilizes the schematic and constraint-based interschema relationships generated by the conflict identification and constraint-based relationship generator components to generate "real world" interschema relationships based on the heuristics presented in table 3.4. Currently, this module reads in data from four text files: a) eq.txt: which describes the set of schematic and constraint-based relationships among entity classes in the schema b) atteg.txt: which describes the schematic and constraint-based relationships among attributes belonging to the entity classes c) ic1.txt: which contains the integrity constraints specified on the first database and d) ic2.txt: which contains the integrity constraints specified on the second database. Table 4.1 shows an example set of input data. The "real world" relationships generated by this component are not written to file. Instead, they are stored in memory and used by the Integrity Constraint Integrator described in the next section.

When this module is integrated with the blackboard architecture described in section 4.1, this module will read the schematic and constraint-based transient assertions from the assertion level of the blackboard and the constraints from the data level of the blackboard and generate "real world" relationships onto the object relationships section of the fact level of the blackboard. Users could then modify these assertions as appropriate and generate information on the fact level of the blackboard.
4.2.3 Integrity Constraint Integrator

This module utilizes the real-world interschema relationships, integrity constraints and the knowledge of the conflict resolution strategy used, to generate a set of integrated integrity constraints applicable at the integrated schema level. As mentioned previously, currently this module is integrated with the "real world" relationship generator module. Once the "real world" relationships among the objects have been generated (in memory), the set of entity classes and attributes in the integrated schema are also generated by the module. The integrity constraint integrator then generates the integrated set of integrity constraints applicable to the objects at the integrated level using the IC integration rules described in section 3.7. Table 4.2 shows the output generated when the schemas and constraints specified in table 4.1 are integrated. Currently, the integrated schema and the names at the integrated schema level are generated automatically by the system. However, when this module is integrated with the blackboard based system described previously, the schema at the integrated level would be generated by the conflict resolution engine in consultation with the user. Once the integrated schema has been generated (and posted on the goal level), the integrity constraint generator would use the schema from the goal level of the blackboard as well as the integrity constraints from the data level of the blackboard to generate the set of
integrated integrity constraints, which would then be asserted onto the goal level of the blackboard.
Table 4.1 Sample Input Data

eq.txt:

Ships SCHEQUIV Dinghy
Boats SCHSUBSUMES Oil_Tankers
Ships CBEQUIV Dinghy
Boats CBSUBSUMES Oil_Tankers

atteq.txt:

Ships.SHIPTYPE SCHEQUIV Dinghy.SHIPTYPE
Boats.SHIPTYPE SCHSUBSUMES Oil_Tankers.SHIPTYPE
Boats.DEADWT SCHEQUIV Oil_Tankers.x5

ic1.txt:

OWNER = 'onassis' < _ Ships(x1,x2,OWNER,SHIPTYPE,x4,x5,x6,x7),
SHIPTYPE = 'supertanker'.
1
SHIPTYPE = 'tanker' < _ Ships(x1,x2,OWNER,SHIPTYPE,x4,x5,x6,x7),
x7 = 'testing'.
DEADWT > 200 < _ Boats(x1, x2, SHIPTYPE, DEADWT), SHIPTYPE = 'tanker'.

ic2.txt:

OWNER = 'onassis' < _ Oil_Tankers(x1, x2, OWNER, SHIPTYPE, x4, x5, x6, x7),
SHIPTYPE = 'supertanker'.

DEADWT > 200 < _ Dinghy(x1, x2, SHIPTYPE, DEADWT), SHIPTYPE = 'tanker'.

1
Table 4.2 Sample Output Data

OWNER = 'onassis' < _

Ships_Dinghy(x1, x2, OWNER, SHIPTYPE, x4, x5, x6, x7, x1, x2, DEADWT), SHIPTYPE = 'supertanker'.

SHIPTYPE = 'tanker' < _

Ships_Dinghy(x1, x2, OWNER, SHIPTYPE, x4, x5, x6, x7, x1, x2, DEADWT), x7 = 'testing'.

DEADWT > 200 < _

Ships_Dinghy(x1, x2, OWNER, SHIPTYPE, x4, x5, x6, x7, x1, x2, DEADWT), SHIPTYPE = 'tanker'.

DEADWT > 200 < _ Int_Boats(x1, x2, SHIPTYPE, DEADWT), SHIPTYPE = 'tanker'.

x5 > 200 < -

Int_Oil_Tankers(x1, x2, OWNER, SHIPTYPE, x4, x5, x6, x7), SHIPTYPE = 'tanker'.

OWNER = 'onassis' < _

Int_Oil_Tankers(x1, x2, OWNER, SHIPTYPE, x4, x5, x6, x7), SHIPTYPE = 'supertanker'.

CHAPTER 5

SEMANTIC QUERY PROCESSING

Semantic query processing (SQP) techniques (King, 1981; Chakrabarty et al., 1990) utilize integrity constraint knowledge to transform queries into more efficient semantically equivalent queries. In a heterogeneous database environment, users formulate queries on the integrated schema. These queries are then translated into sub-queries (using the global to local mapping information) in the languages of the databases that need to be accessed (Landers and Rosenberg, 1982). By generating an integrated set of integrity constraints, using the techniques described in section 3.7, we can perform semantic query processing at the integrated schema level by treating the integrated schema and the integrated constraints as being representative of a single database. Once we have performed SQP and transformed the queries appropriately, the semantically transformed queries can be translated into sub-queries on the underlying databases. An alternative approach to performing SQP in a heterogeneous database environment, that would not require the use of our integrity constraint integration methodology, would be to first transform the queries into local queries and then perform SQP based on the individual set of integrity constraints defined on the databases. Fig. 5.1a & 5.1b illustrate these two scenarios diagrammatically.
Fig. 5.1a SQP without our methodology

Fig. 5.1b SQP Using our Methodology
As noted earlier, the process of integrity constraint integration can sometimes result in the generation of integrity constraints that are applicable at the global schema level, but are not explicitly specified at the individual database level. The presence of these additional constraints provides the semantic query processor with more opportunities for transforming a query specified on the global schema before issuing it to the local schema. The ability to perform SQP against heterogeneous databases can result in substantial savings because of two primary reasons: 1) it can eliminate access to one of the underlying databases and 2) it can result in an improved query being generated to all of the underlying databases compared to improved queries being generated to a single database.

We illustrate the applicability of semantic query transformation techniques in a heterogeneous environment using the following simple examples. The examples illustrate the savings that can result by performing SQP using an integrated set of integrity constraints (which requires the use of our methodology) when compared to using the individual (local) integrity constraints. In the discussion below, we ignore any cost overhead introduced by the databases being (possibly) distributed. All estimations are based on database access costs estimated using the techniques discussed in King (1981) and Mackert and Lohman (1986).
Schema A:

Ships (x1, OWNER, SHIPTYPE, ORIGIN, Deadwt, x6, x7)

SHIPTYPE = 'supertanker' ← Ships(x1, OWNER, SHIPTYPE, ORIGIN, Deadwt, x6, x7),

Deadwt > 150

Schema B:

Boats (x1, COUNTRY, OWNER, TYPE, Weight)

← BOATS(x1, iceland, OWNER, TYPE, Weight)

Fig. 5.2a Example Schema 1

Integrated Schema:

Ship_Boats(x1, OWNER, SHIPTYPE, COUNTRY, Wt, x6, x7, x8)

SHIPTYPE = 'supertanker' ← Ship_Boats(x1, OWNER, SHIPTYPE, COUNTRY, Wt, x6, x7, x8), Wt > 150

← Ship_Boats(x1, OWNER, SHIPTYPE, iceland, Wt, x6, x7, x8)

Fig. 5.2b Integrated Schema and Constraints
Example 1

Consider the schema shown in fig. 5.2.

Let us assume the following parameters for the underlying databases: Ships = 20,000 tuples; Boats = 25,000 tuples and Oil_Tankers = 10,000 tuples. No. of tuples per page = 20, Time per page fetch = .01 seconds.

Assume that the user now issues the following query

**Select Owner from Ship_Boats where Shiptype = 'Bulk Cargo' and Deadwt > 250**

Without our methodology, the sequence of steps followed to transform the query would be as follows:

1) Transform the query into sub-queries. This would result in queries on Ships and Boats being generated.

2) Semantically process the sub-queries using the integrity constraints specified against each relation at each local database. Thus, the query on Boats would not be semantically
transformed since there is no applicable constraint. The query on Ships would be transformed and the need to access the SHIPS relation would be eliminated.

Hence, the estimated cost of database access in this case would be

\[
\text{time to access Ships} + \text{time to access Boats} = 0 + 12.5 = 12.5 \text{ units}
\]

However, if we use our methodology the sequence of steps followed in query transformation would be as follows:

1) Semantically transform query. Since there is a constraint on SHIP_BOATS that can transform this query, the transformation is applied, resulting in access to both underlying databases being eliminated. Hence, the estimated time for database access in this case is 0 units.
Ships (x1, OWNER, SHIPTYPE, ORIGIN, WT, x6, x7)
Oil_Tankers(x1, COUNTRY, OWNER, TYPE, DEADWT)

IC-A:
OWNER = 'Onassis' ← Ships(x1, OWNER, supertanker, ORIGIN, WT, x6, x7)
SHIPTYPE = 'supertanker' ← Ships(x1, OWNER, SHIPTYPE, ORIGIN, WT, x6, x7), WT > 200

IC-B:
← Oil_Tankers(x1, iceland, OWNER, TYPE, DEADWT)
TYPE = 'pressurized tanker' ← Oil_Tankers(x1, uae, OWNER, TYPE, DEADWT)

Fig. 5.3a Example Schema 2

Integrated Schema:

Int_Ships (x1, OWNER, SHIPTYPE, COUNTRY, WT, x6, x7)
Int_Oil_Tankers (x1, COUNTRY, OWNER, TYPE, DEADWT)

IC_Int_Ships:
OWNER = 'Onassis' ← Int_Ships (x1, OWNER, supertanker, ORIGIN, WT, x6, x7)
SHIPTYPE = 'supertanker' ← Int_Ships (x1, OWNER, SHIPTYPE, ORIGIN, WT, x6, x7), WT > 200
← Int_Ships (x1, OWNER, SHIPTYPE, iceland, WT, x6, x7)
SHIPTYPE = 'pressurized tanker' ← Int_Ships (x1, OWNER, SHIPTYPE, uae, WT, x6, x7)

IC_Int_Oil_Tankers:
OWNER = "Onassis" ← Int_Oil_Tankers (x1, COUNTRY, OWNER, supertanker, DEADWT)
TYPE = 'supertanker' ← Int_Oil_Tankers (x1, COUNTRY, OWNER, TYPE, DEADWT),
DEADWT > 200 -------- (a)
← Int_Oil_Tankers (x1, iceland, OWNER, TYPE, DEADWT)
TYPE = 'pressurized tanker' ← Int_Oil_Tankers(x1, uae, OWNER, TYPE, DEADWT)

Fig. 5.3b Integrated Schema
Example 2

Consider the example schemas shown in fig. 5.3. Assume that the user issues the following query against the database schemas:

Select Owners from Int_Oil_Tankers where Wt > 200

Let us assume that the attributes SHIPTYPE in the Ships relation and TYPE in the Oil_Tankers relation are indexed attributes.

Without the use of our methodology, we would transform the query into a sub-query on the Oil_Tankers relation. This sub-query would then be subjected to SQP based on the individual constraints specified against the Oil_Tankers relation in the local database. However, the sub-query on the Oil_Tankers relation would undergo no semantic transformation since no applicable constraints exist in the relation.

Using our methodology, due to the presence of the additional constraint (a) on the entity class Int_Oil_Tankers (generated during the integrity constraint integration process), the user's query would be transformed into a query using the TYPE attribute in the Int_Oil_Tankers entity class. This transformed query would then be transformed into a sub-query which uses the indexed attribute TYPE in the Boats relation. This in turn can lead to substantial savings in query processing time.
5.1 Simulation Study

It is clear from the above examples, that under the appropriate circumstances the use of our methodology can result in considerable savings in query processing time. To illustrate the benefits of using our methodology we simulated the execution of a 1000 queries on a sample database. Two simulation models were constructed, one representing a database that does not utilize the IC integration methodology proposed in this dissertation (control group) and one that uses the methodology proposed in this dissertation (the experimental group). The models were constructed using the SIMAN language. The simulations themselves were run on a VAX 4000/300 machine running the VMS operating system.

Fig. 5.4 shows a logical representation of how the simulation models calculate the delay associated with a query. The figure shows that a query's delay is determined based on four factors:

1) The first factor that affects the delay associated with a query is the type of entity class in the global schema on which the query is formulated. Since the entity classes in the global schema are generated as a result of the integrated schema generation process, we can divide the set of entity classes into six distinct categories (Larson et. al., 1989): equivalence class,
Fig. 5.4 Logical Simulation Model
superclass of a subsumption relationship, subclass of a subsumption relationship, union class generated as a result of an overlap relationship, original classes of an overlap relationship and disjoint relationship. For example, an entity belonging to the equivalence class in the integrated schema would have been generated as a result of an equivalence relationship being specified between two entity classes in the local schemas. Once the entity class category associated with the query has been determined, the actual entity class belonging to the appropriate category is selected at random (from our example databases).

2) The second characteristic that is associated with a query is whether the query is going to be semantically transformed. Since one of the purposes of our study was to evaluate the effect of semantic query transformation, we varied the percentage of queries that would be semantically transformed in our experiments.

3) The third characteristic that is associated with a query is the type of semantic transformation that it is going to undergo. As noted in Table 2.7, there are several different types of semantic transformations. In our study, we simulated the different types of semantic transformations by associating different delays for the various transformations.
4) Once the characteristics described above have been associated with a query, the delay associated with the query is determined by the simulation model. This delay represents an estimate of the time it would take to process the (transformed or non-transformed) queries on the underlying databases. As noted in step 1, a query can be formulated on six different categories of entity classes. Section 3.7.1 shows that the rules for generating integrated integrity constraints are dependent on the category of relationship between the underlying entity classes. Hence, the delay associated with a particular query is calculated as a function of the entity class category on which the query is formulated.

The following paragraphs elaborate on how the delays associated with a query are calculated by the simulation models for the control and experimental groups. The following terminology is used in the discussion:

Gsqo - Time taken to generate optimized queries at the global level. This represents a measure of the amount of time it takes to semantically process a query and generate an alternate query.

Lsqo - Time taken to generate optimized queries at the local database level. This represents the amount of time it takes to semantically process a query at the individual database level.
\( \text{Dt} \) - Total anticipated delay for the query. This estimate includes the amount of time needed to optimize the queries as well as execute the queries on the appropriate database.

\( \text{DSQi} \) - Total delay for executing an optimized query \( i \).

\( \text{DQi} \) - Total delay for executing a non-optimized query \( i \).

5.1.1 Case 1: Equivalence Class

An equivalence entity class at the integrated level is generated as a result of an equivalence relationship between two entity classes. If we make the assumption that the same constraint is not explicitly specified in both the underlying databases, a query issued against such a class will get transformed as follows:

a) In the control group, a query on the global entity is transformed into two queries on the local database, one of which is optimized and the other is not. Hence,

\[
\text{Dt} = Lsq + \text{DSQ} + \text{DQ2}
\]

b) in the experimental group, any query that is optimized at the global level will be optimized on both databases. Hence,
\[ Dt = Gs_qo + DSQ_1 + DSQ_2 \]

5.1.1 Case 2: Superclass of a subsumption relationship

As shown in the section on integrity constraint integration, the generation of a superclass of a subsumption relationship does not result in the generation of any additional constraints at this level. As a result, any query that is optimized at the global level will also be optimized at the local level. Hence, the delays associated with the queries are as follows:

a) for the control group: \[ Dt = Ls_qo + DSQ_1 \]

b) for the experimental group: \[ Dt = Gs_qo + DSQ_1 \]

5.1.3 Case 3: Subclass of a subsumption relationship

This case was illustrated in example 2 of the preceding subsection. We need to model two subcases here:

1) the query is transformed by one of the subclasses' original constraints. Since such a constraint exists at both the integrated and local levels, the query will get optimized at both levels. Hence, the delays associated with a transformed query are as shown in Case 2 above.
2) the query is transformed (at the global level) by a constraint inherited from the superclass. In this case,

a) in the control group no transformation is possible. This is because the constraint being used to transform the query (at the global level) has not been explicitly specified on the local database. Hence,

\[ D_t = L_{sqo} + D_{Q1} \]

b) in the experimental group, the original query is transformed using the inherited constraint. Hence,

\[ D_t = G_{sqo} + D_{SQ1} \]

5.1.4 Case 4: Union class generated as a result of overlap relationship

This case is similar to the equivalence case with the exception that only a certain percentage of the underlying classes' constraints exist at the global level. Once again two cases arise:
1) the query is transformed at the global level. In this case, assuming that the same constraint has not been explicitly specified on the local databases

a) in the control group, a query on the global entity is transformed into two sub-queries on the local databases, one of which gets transformed at local level. Hence,

\[ D_t = L_{sq_0} + D_{SQ1} + D_{Q2} \]

b) in the experimental group, the query gets transformed at the global level resulting in a semantically transformed query being issued to both the local databases. Hence,

\[ D_t = G_{sq_0} + D_{SQ1} + D_{SQ2} \]

2) the query is not transformed at the global level, but is transformed at one of the local levels. Hence,

a) in the control group, one of the local queries gets transformed at the local level while the other does not. Hence,
\[ Dt = Ls_{q0} + DSQ1 + DQ2 \]

b) for the experimental group, no transformation takes place and hence,

\[ Dt = DQ1 + DQ2 \]

5.1.5 Case 5: Original classes of an overlap relationship

This case is similar to Case 3 above. As indicated in section 3.7, constraints associated with these classes can either be inherited from the superclass or be the original constraints specified on the local databases. Hence, two subcases exist:

1) the query is transformed by one of the classes’ original constraints. In this case, any query that is optimized globally is also going to be optimized locally. Hence,

a) in the control group

\[ Dt = Ls_{q0} + DSQ1 \]

b) in the experimental group,
Dt = Gsqo + DSQ1

2) the query is transformed (at the global level) by a constraint inherited from the superclass. In this case,

a) in the control group no transformation takes place since the constraint has not been specified on the local database. Hence,

Dt = Lsqo + DQ1

b) in the experimental group, the query is transformed using the inherited constraint. Hence,

Dt = Gsqo + DSQ1

5.1.6 Case 6: Disjoint Classes
In this case the delays are similar to Case 2, since all constraints applicable at the global level are also applicable at the local level. Hence,

a) in the control group

\[ Dt = Lsqo + DSQ1 \]

b) in the experimental group

\[ Dt = Gsqo + DSQ1 \]

The formulae for calculating the delays for the various cases in our simulation study are summarized in table 5.1.
<table>
<thead>
<tr>
<th>Entity Class Category</th>
<th>Experimental Group</th>
<th>Control Group</th>
</tr>
</thead>
<tbody>
<tr>
<td>Equivalence Class</td>
<td>$Dt = Gsqo + DSQ_1 + DSQ_2$</td>
<td>$Dt = Lsqo + DSQ_1 + DQ_2$</td>
</tr>
<tr>
<td>Superclass of Subsumption</td>
<td>$Dt = Gsqo + DSQ_1$</td>
<td>$Dt = Lsqo + DSQ_1$</td>
</tr>
<tr>
<td>Subclass of Subsumption</td>
<td>Case 1: $Dt = Gsqo + DSQ_1$</td>
<td>Case 1: $Dt = Lsqo + DSQ_1$</td>
</tr>
<tr>
<td></td>
<td>Case 2: $Dt = Gsqo + DSQ_1$</td>
<td>Case 2: $Dt = Lsqo + DQ_1$</td>
</tr>
<tr>
<td>Union Class of Overlap</td>
<td>Case 1: $Dt = Gsqo + DSQ_1 + DSQ_2$</td>
<td>Case 1: $Dt = Lsqo + DSQ_1 + DQ_2$</td>
</tr>
<tr>
<td></td>
<td>Case 2: $Dt = SQ_1 + DQ_2$</td>
<td>Case 2: $Dt = Lsqo + DSQ_1 + DQ_2$</td>
</tr>
<tr>
<td>Individual Class of Overlap</td>
<td>Case 1: $Dt = Gsqo + DSQ_1$</td>
<td>Case 1: $Dt = Lsqo + DSQ_1$</td>
</tr>
<tr>
<td></td>
<td>Case 2: $Dt = Gsqo + DSQ_1$</td>
<td>Case 2: $Dt = Lsqo + DQ_1$</td>
</tr>
<tr>
<td>Disjoint Classes</td>
<td>$Dt = Gsqo + DSQ_1$</td>
<td>$Dt = Lsqo + DSQ_1$</td>
</tr>
</tbody>
</table>

**Table 5.1**

Delay Calculations for Various Entity Class Categories
5.2 Parameters and Procedures

The purpose of our simulation study was to illustrate the benefits of performing semantic query processing using our methodology. Based on the description of the simulation model presented in the previous section it is clear that the two key factors that play a role in determining the delay for a query are: 1) $P_e$ - the entity class category (in the global schema) on which the query is issued and 2) $P_t$ - the probability that a query will be transformed. Hence, these were the two main factors varied in our simulation study.

To study the effect of the entity class category on a query's delay, we created seven different scenarios. Each scenario was designed to study the effect of delaying a majority of queries according to one of the six cases summarized in table 5.1. Thus, in six of the seven scenarios (B-F) a majority of the queries were associated with a particular category of entity class. In scenario A, the probabilities were set up to ensure an even distribution of queries across entity class categories. The different scenarios and the probabilities associated with the different entity class categories in each scenario are shown in table 5.2a. We were also interested in evaluating the effect of probability of semantic query transformation on a query's delay. Hence, we also used eight different transformation probability levels in our study. The various probabilities of transformations used in the study are listed in table 5.2b.
<table>
<thead>
<tr>
<th>Scenario</th>
<th>Equivalence</th>
<th>Superclass</th>
<th>Subclass</th>
<th>Unionclass</th>
<th>Individual Overlap</th>
<th>Disjoint</th>
</tr>
</thead>
<tbody>
<tr>
<td>A (Even)</td>
<td>12.5</td>
<td>20</td>
<td>30</td>
<td>12.5</td>
<td>12.5</td>
<td>12.5</td>
</tr>
<tr>
<td>B (Heavy Superclass)</td>
<td>12.5</td>
<td>45</td>
<td>7.5</td>
<td>12.5</td>
<td>12.5</td>
<td>12.5</td>
</tr>
<tr>
<td>C (Heavy Disjoint)</td>
<td>15</td>
<td>15</td>
<td>15</td>
<td>15</td>
<td>15</td>
<td>25</td>
</tr>
<tr>
<td>D (Heavy Unionclass)</td>
<td>12.5</td>
<td>12.5</td>
<td>12.5</td>
<td>40</td>
<td>15</td>
<td>7.5</td>
</tr>
<tr>
<td>E (Heavy Subclass)</td>
<td>12.5</td>
<td>12.5</td>
<td>40</td>
<td>12.5</td>
<td>12.5</td>
<td>10</td>
</tr>
<tr>
<td>F (Heavy Individual Overlap)</td>
<td>12.5</td>
<td>12.5</td>
<td>12.5</td>
<td>12.5</td>
<td>40</td>
<td>10</td>
</tr>
<tr>
<td>G (Heavy Equivalence)</td>
<td>40</td>
<td>12.5</td>
<td>12.5</td>
<td>12.5</td>
<td>12.5</td>
<td>10</td>
</tr>
</tbody>
</table>

Table 5.2a

Probabilities Associated with Various Entity Class Categories in Different Scenarios

<table>
<thead>
<tr>
<th>Condition</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
</tr>
</thead>
<tbody>
<tr>
<td>% of Queries Transformed Semantically</td>
<td>0.00</td>
<td>5.00</td>
<td>15.00</td>
<td>25.00</td>
<td>50.00</td>
<td>62.50</td>
<td>75.00</td>
<td>87.50</td>
</tr>
</tbody>
</table>

Table 5.2b

Various Transformation Probabilities used in the Study
In our simulation study, for each of the seven scenarios, we ran 8 simulation (one for each level of transformation probability). Thus, data was collected for a total of 56 different simulation runs. Each run simulated the execution of 1000 queries through the control group or experimental group model and was repeated 20 times to ensure sufficiently tight confidence intervals. We used the two sample databases shown in table 5.3a in our study. The size of each relation (in tuples) in the databases is shown in parentheses. The first database was identical to the database used by King (1981) to study semantic query processing and consisted of relations ranging in size from 1,000 to 25,000 tuples. The second database consisted of relations that were related by an equivalence, subsumption, overlap or disjoint relationship to the relations in the first database. The size of the relations in this database ranged from 700 to 25,000 tuples. The integrated schema and the categories to which the entity classes in the schema belong to are shown in table 5.3b.

Since we used only SELECT queries in our study, the delay for non-optimized queries was set to the amount of time it would take to scan through a table (assuming no indexed attribute exists). All delays were calculated based on a page access time of 5 ms and a page size of 4K, assuming that 20 records can fit into one page. For example, for the SHIPS relation with 20,000 tuples, the non-optimized delay was calculated as:

\[
\text{Number of pages} = \frac{20000}{20} = 1000; \text{Delay} = 1000 \times 5 = 5000 \text{ ms}
\]
Delays for optimized queries were set at 0-50% of the non-optimized query delay. The differing degrees of delay in the optimized queries represented the different types of possible semantic query transformations. The delay for Gsqo was assumed to 5 ms for a database of 100 constraints (Shekhar et. al., 1992). A proportional amount based on the number of constraints was calculated and used in the study. The delay for Lsqo was set at half of the Gsqo value based on the assumption that the underlying databases contributed equally towards the global constraints.

<table>
<thead>
<tr>
<th>Database 1</th>
<th>Database 2</th>
</tr>
</thead>
<tbody>
<tr>
<td>SHIPS (20,000)</td>
<td>BOATS (25,000)</td>
</tr>
<tr>
<td>PORTS (1,000)</td>
<td>OIL-TANKERS (10,000)</td>
</tr>
<tr>
<td>CARGOES (25,000)</td>
<td>NON-OIL-TANKERS (15,000)</td>
</tr>
<tr>
<td>OWNERS (1,000)</td>
<td>MERCHANDISE (15,000)</td>
</tr>
<tr>
<td>POLICIES (25,000)</td>
<td>INSCOMP (700)</td>
</tr>
<tr>
<td>INSURERS (500)</td>
<td>SHORT-TERM-POL (10,000)</td>
</tr>
<tr>
<td></td>
<td>LONG-TERM-POL (13,000)</td>
</tr>
</tbody>
</table>

Table 5.3a

Individual Database Schemas
<table>
<thead>
<tr>
<th>Entity Class Name</th>
<th>Entity Class Category</th>
</tr>
</thead>
<tbody>
<tr>
<td>SHIP-BOATS</td>
<td>EQUIV, SUP</td>
</tr>
<tr>
<td>PORTS</td>
<td>DISJ</td>
</tr>
<tr>
<td>INSURERS</td>
<td>EQUIV</td>
</tr>
<tr>
<td>OIL-TANKERS</td>
<td>SUB</td>
</tr>
<tr>
<td>NON-OIL-TANKERS</td>
<td>SUB</td>
</tr>
<tr>
<td>CARGOES</td>
<td>INDOVERLAP</td>
</tr>
<tr>
<td>MERCHANDISE</td>
<td>INDOVERLAP</td>
</tr>
<tr>
<td>SHIPS-LOAD</td>
<td>UNIONOVERLAP</td>
</tr>
<tr>
<td>POLICIES</td>
<td>SUP</td>
</tr>
<tr>
<td>SHORT-TERM-POL</td>
<td>SUB</td>
</tr>
<tr>
<td>LONG-TERM-POL</td>
<td>SUB</td>
</tr>
<tr>
<td>OWNERS</td>
<td>DISJ</td>
</tr>
</tbody>
</table>

Table 5.3b

Integrated Schema
5.3 Results

The primary objective of this study was to illustrate the potential benefits of performing semantic query processing, facilitated by the use of our integrity constraint integration methodology, at the integrated schema level. We intended to show the benefits of our methodology by documenting the savings in query processing time for 1000 queries.

Table 5.4 shows the delay for 1000 queries (averaged across 20 simulation runs) in the control and experimental group in Scenario A. Recall from table 5.2a that scenario A distributed the 1000 queries evenly across entity classes (in the integrated schema) belonging to the six entity class categories. Each column in table 5.4 represents the delay for 1000 queries when a certain percentage of queries are subject to semantic query transformation. The last row indicates the level of statistical significance at which there is a difference between the average delays in the control and experimental groups. An examination of the first two columns of table 5.4 indicates that there isn’t a significant difference in the delays, at the $\alpha = .05$ level, when only 0% or 5% of the queries are subject to semantic query transformation. However, an examination of the p-values in the rest of the columns indicates that there is a significant difference between the means at the same level of significance (since the p-values are < .05). This indicates that when at least
Table 5.4 Scenario A

Even Distribution of Queries Across All Entity Class Categories

Time for 1000 queries (x 10^6)

<table>
<thead>
<tr>
<th>Proby/Group</th>
<th>0.00</th>
<th>5.00</th>
<th>15.00</th>
<th>25.00</th>
<th>50.00</th>
<th>62.50</th>
<th>75.00</th>
<th>87.50</th>
</tr>
</thead>
<tbody>
<tr>
<td>Control</td>
<td>3.24</td>
<td>3.19</td>
<td>3.11</td>
<td>3.03</td>
<td>2.81</td>
<td>2.68</td>
<td>2.58</td>
<td>2.48</td>
</tr>
<tr>
<td>Exp</td>
<td>3.24</td>
<td>3.17</td>
<td>3.00</td>
<td>2.80</td>
<td>2.40</td>
<td>2.17</td>
<td>1.96</td>
<td>1.74</td>
</tr>
<tr>
<td>p</td>
<td>1.00</td>
<td>0.32</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
</tr>
</tbody>
</table>
Fig. 5.5 Condition A
Even Distribution of Queries Across All Entity Class Categories
15% of the queries are transformed semantically, the SQP at the integrated schema level results in significant savings in query processing time. Fig. 5.5 shows a diagrammatic representation of the delays in the control and experimental conditions and the savings in time that can be achieved by the use of our methodology.

Tables 5.5 through 5.10 show the mean delays for experimental and control groups for scenarios B through G respectively. Each of these tables should be interpreted in a fashion similar to table A. An examination of these tables shows that in each of the scenarios (which represent the different ways in which a query issued against the global schema can be semantically transformed into more efficient database queries) there is a statistically significant difference in query processing time between the control and experimental group when at least 15% of the queries are semantically transformed. These savings (for scenarios B through G) are represented diagrammatically in Figs. 5.6 through 5.11.

The results show that using our methodology can result in significant savings as long as at least 15% of the queries issued to the integrated database can be transformed by the use of appropriate semantic integrity constraints. It is clear that the curves in Figs. 5.5 through 5.11 are similar, the only difference being in the starting and ending values of the average delay.
for 1000 queries. This demonstrates that the savings from the use of our methodology are not dependent on the category of entity class on which the majority of queries are issued.

These results suggest that the use of our methodology for integrity constraint integration can result in savings in query processing time as long as sufficient integrity constraints at the integrated schema level can be generated. The generation of these constraints is in turn dependent on the number of explicitly specified constraints at the local database level and the existence of related objects in the databases being integrated. Other factors that need to be considered in evaluating the potential use of our methodology are the types of queries formulated and the frequency of changes to the underlying databases' semantics. The former factor is important because if none of the queries being formulated can be transformed by the integrated integrity constraints then the use of our methodology would not result in any savings. The latter factors is important because, in weighing the potential use of our methodology for integrity constraint integration, we need to factor in the cost of generating the integrated set of integrity constraints. If the number of changes to the underlying databases is frequent, then the cost of generating integrated integrity constraints begins to accrue and may nullify any savings due to the use of our methodology. This is, however, expected to be less of an issue in legacy systems.
### Table 5.5 Scenario B

**Heavy Concentration of Queries on Superclass Entity Classes**

**Time for 1000 queries ($x 10^6$)**

<table>
<thead>
<tr>
<th>Proby/Group</th>
<th>0.00</th>
<th>5.00</th>
<th>15.00</th>
<th>25.00</th>
<th>50.00</th>
<th>62.50</th>
<th>75.00</th>
<th>87.50</th>
</tr>
</thead>
<tbody>
<tr>
<td>Control</td>
<td>3.49</td>
<td>3.40</td>
<td>3.27</td>
<td>3.15</td>
<td>2.79</td>
<td>2.62</td>
<td>2.41</td>
<td>2.07</td>
</tr>
<tr>
<td>Exp</td>
<td>3.49</td>
<td>3.40</td>
<td>3.21</td>
<td>3.02</td>
<td>2.56</td>
<td>2.31</td>
<td>2.09</td>
<td>1.59</td>
</tr>
<tr>
<td>p</td>
<td>0.82</td>
<td>0.75</td>
<td>0.01</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
</tr>
</tbody>
</table>

### Table 5.6 Scenario C

**Heavy Concentration of Queries on Disjoint Entity Classes**

**Time for 1000 queries ($x 10^6$)**

<table>
<thead>
<tr>
<th>Proby/Group</th>
<th>0.00</th>
<th>5.00</th>
<th>15.00</th>
<th>25.00</th>
<th>50.00</th>
<th>62.50</th>
<th>75.00</th>
<th>87.50</th>
</tr>
</thead>
<tbody>
<tr>
<td>Control</td>
<td>3.01</td>
<td>2.97</td>
<td>2.88</td>
<td>2.83</td>
<td>2.66</td>
<td>2.57</td>
<td>2.45</td>
<td>2.38</td>
</tr>
<tr>
<td>Exp</td>
<td>3.01</td>
<td>2.93</td>
<td>2.80</td>
<td>2.65</td>
<td>2.31</td>
<td>2.14</td>
<td>1.96</td>
<td>1.77</td>
</tr>
<tr>
<td>p</td>
<td>1.00</td>
<td>0.06</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
</tr>
</tbody>
</table>
Table 5.7 Condition D

Heavy Concentration of Queries on Union Entity Classes

Time for 1000 queries (x 10^6)

<table>
<thead>
<tr>
<th>Proby/Group</th>
<th>0.00</th>
<th>5.00</th>
<th>15.00</th>
<th>25.00</th>
<th>50.00</th>
<th>62.50</th>
<th>75.00</th>
<th>87.50</th>
</tr>
</thead>
<tbody>
<tr>
<td>Control</td>
<td>3.09</td>
<td>3.12</td>
<td>3.17</td>
<td>3.23</td>
<td>3.38</td>
<td>3.43</td>
<td>3.52</td>
<td>3.42</td>
</tr>
<tr>
<td>Exp</td>
<td>3.09</td>
<td>3.09</td>
<td>3.10</td>
<td>3.08</td>
<td>3.07</td>
<td>3.05</td>
<td>3.05</td>
<td>3.07</td>
</tr>
<tr>
<td>p</td>
<td>1.00</td>
<td>0.11</td>
<td>0.01</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
<td>0.04</td>
</tr>
</tbody>
</table>

Table 5.8 Condition E

Heavy Concentration of Queries on Subclass Entity Classes

Time for 1000 queries (x 10^6)

<table>
<thead>
<tr>
<th>Proby/Group</th>
<th>0.00</th>
<th>5.00</th>
<th>15.00</th>
<th>25.00</th>
<th>50.00</th>
<th>62.50</th>
<th>75.00</th>
<th>87.50</th>
</tr>
</thead>
<tbody>
<tr>
<td>Control</td>
<td>3.06</td>
<td>3.03</td>
<td>2.97</td>
<td>2.92</td>
<td>2.71</td>
<td>2.63</td>
<td>2.55</td>
<td>2.46</td>
</tr>
<tr>
<td>Exp</td>
<td>3.06</td>
<td>2.99</td>
<td>2.85</td>
<td>2.68</td>
<td>2.30</td>
<td>2.10</td>
<td>1.88</td>
<td>1.68</td>
</tr>
<tr>
<td>p</td>
<td>1.00</td>
<td>0.16</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
</tr>
</tbody>
</table>
Table 5.9 Condition F

Heavy Concentration of Queries on Individual Overlap Entity Classes

Time for 1000 queries (x 10\(^6\))

<table>
<thead>
<tr>
<th>Proby/Group</th>
<th>0.00</th>
<th>5.00</th>
<th>15.00</th>
<th>25.00</th>
<th>50.00</th>
<th>62.50</th>
<th>75.00</th>
<th>87.50</th>
</tr>
</thead>
<tbody>
<tr>
<td>Control</td>
<td>3.23</td>
<td>3.19</td>
<td>3.12</td>
<td>3.04</td>
<td>2.83</td>
<td>2.71</td>
<td>2.61</td>
<td>2.52</td>
</tr>
<tr>
<td>Exp</td>
<td>3.23</td>
<td>3.14</td>
<td>2.98</td>
<td>2.83</td>
<td>2.40</td>
<td>2.16</td>
<td>1.98</td>
<td>1.74</td>
</tr>
<tr>
<td>p</td>
<td>1.00</td>
<td>0.15</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
</tr>
</tbody>
</table>

Table 5.10 Condition G

Heavy Concentration of Queries on Equivalent Entity Classes

Time for 1000 queries (x 10\(^6\))

<table>
<thead>
<tr>
<th>Proby/Group</th>
<th>0.00</th>
<th>5.00</th>
<th>15.00</th>
<th>25.00</th>
<th>50.00</th>
<th>62.50</th>
<th>75.00</th>
<th>87.50</th>
</tr>
</thead>
<tbody>
<tr>
<td>Control</td>
<td>3.87</td>
<td>3.83</td>
<td>3.73</td>
<td>3.63</td>
<td>3.37</td>
<td>3.22</td>
<td>3.11</td>
<td>2.96</td>
</tr>
<tr>
<td>Exp</td>
<td>3.87</td>
<td>3.78</td>
<td>3.56</td>
<td>3.34</td>
<td>2.81</td>
<td>2.52</td>
<td>2.26</td>
<td>1.96</td>
</tr>
<tr>
<td>p</td>
<td>1.00</td>
<td>0.02</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
</tr>
</tbody>
</table>
Fig. 5.6 Condition B
Heavy Concentration of Queries on Superclass Entity Classes
Fig. 5.7 Condition C
Heavy Concentration of Queries on Disjoint Entity Classes
Fig. 5.8 Condition D
Heavy Concentration of Queries on Union Entity Classes
Fig. 5.9 Condition E
Heavy Concentration of Queries on Subclass Entity Classes
Fig. 5.10 Condition F
Heavy Concentration of Queries on Individual Overlap Entity Classes
Fig. 5.11 Condition G
Heavy Concentration of Queries on Equivalent Entity Classes
Efficient information sharing among databases, scientific or business, requires the development of techniques for accessing data from multiple heterogeneous databases. In this dissertation, we described the use of integrity constraint knowledge to improve the schema integration process as well as facilitate efficient access, through the use semantic query processing techniques, in a heterogeneous database environment. This dissertation makes multiple contributions to the database literature. Specifically:

1) We proposed an enhanced methodology for schema integration. This seven step methodology is unique in that 1) unlike existing methodologies it use multiple knowledge sources to arrive at interschema relationships among database objects and 2) the output of the methodology is a representation of the underlying databases that conveys more semantics about the databases than a schematic representation.
2) We presented a comprehensive set of heuristics for schematic IRI. A key characteristic of these heuristics is that they use the schematic information conveyed by all types of schema objects (objects, attributes and relationships) to arrive at schematic interschema relationships.

3) We introduced the concept of constraint-based relationships among objects and presented techniques to evaluate these relationships. These represent relationships among objects in heterogeneous databases that are based on the characteristics of the integrity constraints involving the objects. Since these relationships are generated by querying the underlying databases, they represent relationships based on the current state of the database as opposed to schematic relationships that represent a static "compile-time" view of the underlying database.

3) We presented heuristics for generating "real world" relationships among database objects based on multiple sources of knowledge, namely the constraint-based and schematic relationships. These "real world" relationships reflect the relationships among the database objects based on their real world semantics. We also presented heuristics that can be used to generate a prioritized list of object pairs for users/designers to evaluate and generate relationships.
4) We presented rules for generating an integrated set of integrity constraints applicable to the integrated schema. The existence of such a set of constraints at the integrated schema level conveys more semantics about the underlying databases than a schematic representation.

5) We also described a system that implements the various proposed phases of the enhanced methodology. A unique characteristic of the system is the use of blackboard architectures for supporting the dynamics of human interaction needed during schema integration. We described how the blackboard based system can be used to provide a cooperative environment in which multiple sources of computational and human agents can interact.

6) Finally, we described how an integrated set of integrity constraints can be used to facilitate semantic query processing in a heterogeneous database environment. We used simulations to quantify the potential savings that can be achieved through the use of semantic query processing techniques in a heterogeneous database environment.

6.1 Future Research
As part of our future research, we would like to conduct experiments to validate the heuristics for schematic interschema relationship identification presented in section 3.3. We would also like to incorporate concepts from linguistics and information retrieval theory to the IRI process. Bright et. al. (1994) describe a system that builds a hierarchy for terms used in the schemas based on existing classification schemes such as Roget's thesaurus and Webster's dictionary. This hierarchy is represented using the Summary Schemas Model (SSM) and represents a concise description of the data available in the local schemas. We believe that the use of such concepts from linguistics and information retrieval can be used to aid in the interschema relationship identification process.

Another area of future research is that of extending the number of sources of knowledge used in interschema relationship identification. This dissertation presented a technique that combines two sources of knowledge to aid in interschema relationship identification, namely schematic and integrity constraint knowledge. We would like to extend our approach and use more than two sources of knowledge. One possible source of knowledge is the actual data stored in the database. Li and Clifton (1994) present an automated technique for determining attribute equivalence that combines schematic and data-level knowledge in the context of relational databases. Their method uses
discriminators from the schema level, such as, data type of the attributes, their length and the existence of constraints, and data-level, such as, patterns in numeric and character fields, to determine equivalence among the attributes. These discriminators are detected automatically by their system and used as input to a neural network that categorizes the attributes into related groups. There has also been substantial work done in the data mining literature (Holsheimer and Siebes, 1994) that attempt to generate relationships among database objects by discovering patterns in the data. Combining data-level relationships with constraint-based and schematic relationships should enable us to automatically generate interschema relationships that closely reflect the real-world semantics of the underlying database objects and would require little or no interaction with the users.

The computational engines for interschema relationship identification described in this dissertation use a forward-reasoning mechanism. However, the use of blackboard architectures does allow interaction among engines using various reasoning paradigms. Hence, we would like to investigate the use of case-based and frame-based reasoning mechanisms in an IRI context.
One of the assumptions made in our methodology for integrity constraint integration is that the underlying databases have a well-defined set of integrity constraints explicitly defined on them. However, such constraints are typically embedded within the application programs that use the database. Hence, an area of future research is the discovery of integrity constraints in legacy systems. Shekhar et. al. (1993) present a technique for discovering integrity constraints in a database using data mining techniques. Their work can also be used as a starting for exploring this avenue of research.

A key limitation of the integrity constraint integration methodology presented in section 3.7 is that it assumes that the set of integrity constraints being integrated represent a complete set of constraints. However, such a complete specification of constraints may not always be available. Hence, we need to develop techniques that relax this restriction and extend the applicability of our approach to databases with an incomplete specification of integrity constraints.

From an implementation perspective, first, we would like to integrate the components that implement the phases of the enhanced methodology with the blackboard based architecture for traditional schema integration. We would then like to test the use of our
Fig. 6.1 An Agent Based Architecture for Heterogeneous Databases
system against a few sample databases and compare the results with the simulation results. We would also like to investigate the use of other novel architectures for providing heterogeneous database interoperability. Specifically, we would like to investigate the viability of using an intelligent agent based architecture for heterogeneous interoperability. Fig. 6.1 shows the proposed architecture for heterogeneous database integration using intelligent agents. Each agent in the diagram "serves" a single database and performs the following functions:

1. It serves as an intermediary between the users and the underlying databases. Hence, each agent contains the know-how required to translate a user query into a query on the local database that it serves.

2. Its knowledge base contains all the meta-information needed to facilitate heterogeneous database interoperability. Examples of knowledge that an agent would store include, terminological and domain specific knowledge (data dictionary), the export and import schema relevant to the database served by the agent, and semantic integrity constraints specified on the local database.
An agent-based architecture such as the one shown in Fig. 6.1 can be used to support the various tasks in schema integration.

1. Schema Integration: Each agent can be expected to implement one of the techniques described in chapter 3 for IRI and ISG. Communication among agents can then be achieved by the use of a standard agent communication language, such as, KQML (Labrou and Finin, 1994). However, we would still need to do research on defining the protocols needed to allow multiple agents to cooperate and complete the various phases of the schema integration process.

2. Semantic Query Processing: In this dissertation, we described the use of integrity constraints to facilitate semantic query processing context of a global schema approach. The use of agent-based architectures will require us to develop new techniques that can be used in a federated database environment. Integrity constraints associated with a database can be stored in the knowledge base of the agent serving that database. Each agent would then communicate with other agents to generate an integrated set of integrity constraints applicable to the import schema. These integrated constraints would also be stored in the agent’s knowledge base. When a user issues a query to the agent, the agents will first semantically transform the query (using the integrated
integrity constraints) before dispatching the queries to the relevant agents serving the databases that need to be accessed. Once again, communication among agents is expected to take place using a standard agent communication language, such as, KQML.

Finally, we would like to investigate the economic viability of applying our methodology to a real world setting. The study presented in section 5.1 illustrated that the use of our methodology can result in savings in query processing time. However, the savings achieved from query processing time could easily be outweighed by several other factors, such as, the cost of implementing the approach (integrating the databases) and the costs due to any changes to the organization's computing infrastructure that would be needed for successful implementation of our methodology. A small scale implementation of our methodology in a real world setting would enable us to get a better grip on these costs and would enable us to develop an economic cost model of implementing our approach. Such a model would then allow us to evaluate the actual benefits of using our methodology on a case-by-case basis.
REFERENCES


