CORDS Schema Integration Environment*

November 1995
External Technical Report
ISSN 0836-0227-95-391

Patrick Martin and Wendy Powley
Dept. of Computing and Information Science
Queen's University at Kingston
Kingston, Ontario
Canada K7L 3N6
({martin, wendy}@qucis.queensu.ca)

Abstract

The CORDS Multidatabase System (MDBS) provides applications with an integrated view of a collection of distributed heterogeneous data sources. Applications are presented with a relational view of the available data and are able to access the data using standard SQL operations. An application's view of the data is defined by a process called schema integration. This paper describes the schema integration method developed for the CORDS MDBS and outlines a set of tools to support the method.

1. Introduction

Recent progress in communication and database technologies has drastically changed user data processing capabilities. The current data processing situation is characterized by a growing number of applications that require access to data from various pre-existing databases distributed among sites in a network. These databases are pre-existing in the sense that they were created independently, in an uncoordinated way, without any consideration of future integration. They can also be heterogeneous, that is they may use different underlying data models, different data definition and manipulation facilities, and run under different operating systems and on different hardware [2].

There are three approaches to integrating these databases. The first is *physical integration* of all data needed by an application into one database. The drawbacks to this method are that it is expensive, it does not allow maintaining data independently, and it leads to unnecessary replication. The second approach is providing *interoperability*, that is integration at the access language level, which provides users with functions for manipulating data in visibly distinct databases [10]. Interoperability is convenient for

* This research is supported by IBM Canada Ltd. and the National Science and Engineering Council of Canada.

database administrators since it does not require data integration, but it places an extra burden on application developers by forcing them to explicitly deal with a variety of data sources. The third approach is *logical integration* of all data used by an application into one logical database which gives the application developers a single interface and hides the differences among the component databases (CDBSs). Popular names for this type of system are federated database systems and multidatabase systems (MDBSs).

One way to view the work in the area of MDBSs is to look at the systems and solutions in the three dimensional space defined by distribution, heterogeneity and autonomy. The prior research in distributed databases dealt with the problems of distribution. The current research is focused on the problems related to heterogeneity, and the problems related to autonomy have so far not been addressed. Most of the research into heterogeneity has studied system issues such as transaction management, concurrency control and recovery. Much less progress can be claimed regarding the semantic issues of heterogeneity [15].

Schema integration, within the context of MDBSs, is the process of combining related schema objects from multiple CDBSs into a single logical view of the combined data. Semantic heterogeneity appears during schema integration in the form of *schema conflicts* among the CDBSs' schemas which result from either the use of different structures for the same information or from the use of different specifications for the same structure, which includes different data models, names, data types and constraints for semantically equivalent objects.

In the paper we present a schema integration methodology and describe a set of tools to support the methodology. The remainder of the paper is structured as follows: Section 2. outlines the CORDS MDBS project, and briefly describes the system structure and the schema architecture of the MDBS. Section 3. presents the schema integration environment developed for the CORDS MDBS. It discusses a method for schema integration based on an analysis of the conflicts present in the CDBSs' schemas and then outlines a toolkit developed to support the method. Section 4. presents an integration scenario which uses the method and toolkit. Section 5. discusses related work and Section 6. summarizes the paper.

2. CORDS MDBS

CORDS (COnsortium for Research on Distributed Systems) was a collaborative research project involving the IBM Centre for Advanced Studies, IBM Research and a number of universities in Canada and the United States. The focus of CORDS was the development, operation, and management of distributed applications. It concentrated on issues within five core areas: application management services, data management services, visualization and user interfaces, development languages and tools, and midware and high speed networks [5].

The CORDS MDBS [2] as indicated by the system architecture diagram in Figure 1, is a full-function DBMS. The common data model used in the CORDS MDBS is the relational model, so schemas define a collection of data in terms of relational tables and their columns and any applicable constraints. Applications interact with a MDBS Server

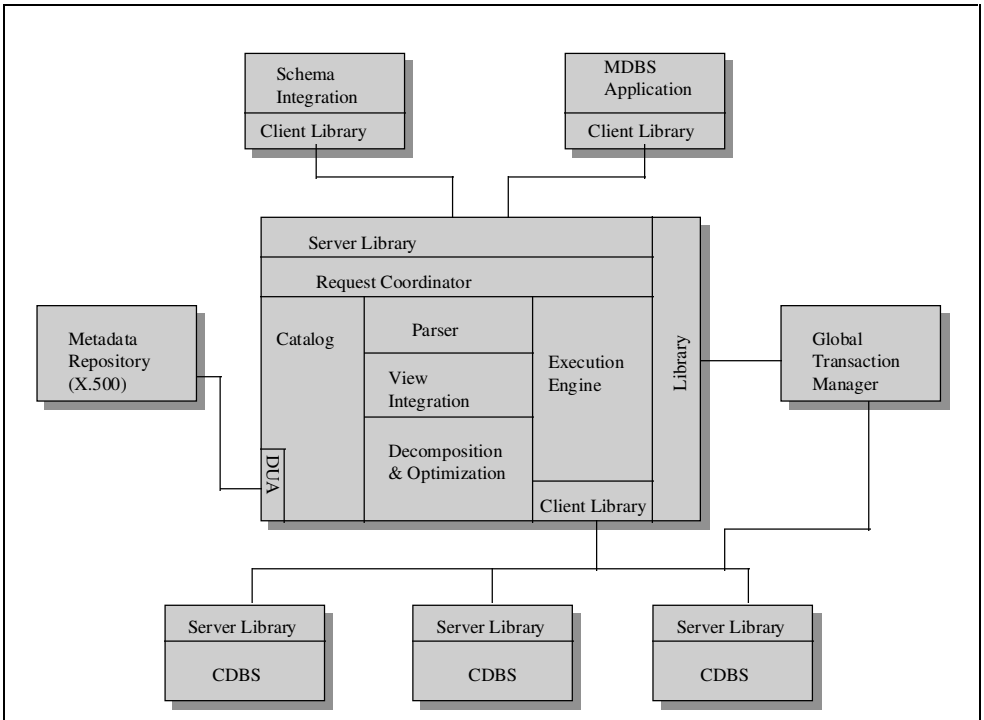

Figure 1: CORDS MDBS Architecture

via a library of interface functions called the MDBS Client Library. A MDBS Server performs DBMS functions, such as query processing and optimization, transaction management, and security, at the global level. A MDBS Server connects to a CDBS through a Server Library which accepts SQL requests from the MDBS, interacts with the CDBS through its normal application program interface, and then translates the response into the form expected by the MDBS. CDBSs currently supported by the prototype include the Empress[1], Oracle[2], and DB2/6000 relational systems, the IMS[3] hierarchical database system, and VAX DBMS[4] network database system. A key difference between the CORDS MDBS and a commercial DBMS is its reliance on services provided by the surrounding environment, in particular security services, transaction management services, and an information repository service to maintain the MDBS Catalog.

The *MDBS Catalog* is the central repository for metadata needed by the multidatabase system. Three classes of metadata are required: schemas, mappings, and descriptions of CDBSs. Two types of schemas are stored: export schemas and MDBS schemas. An export schema defines the data made available to the MDBS from a CDBS and MDBS schemas define collections of data at the MDBS level which are drawn from the exported data. The mappings needed to transform export schema objects into MDBS

---

[1] Empress is a trademark of Empress Software Corporation.

[2] Oracle is a trademark of Oracle Corporation.

[3] DB2/6000 and IMS are trademarks of International Business Machines Corporation.

[4] VAX DBMS is a trademark of Digital Equipment Corporation.

	Attribute	Relation	Schema
Data Type	X		
Scale	X		
Precision	X		
Default Value	X		
Name	X	X	X
Key		X	
Isomorphism		X	X
Union Compatibility		X	
Abstraction Level		X	X
Missing Data		X	
Integrity Constraint	X		

Table 1: Schema Conflict Classification

schema objects are created during the schema integration process. As shown in Figure 1, the schema integration tools are viewed as an application by the MDBS.

3. Schema Integration Environment

The process of schema integration in the CORDS MDBS takes schemas from a set of CDBSs and produces one or more integrated views of the available data. We do not define a single all-encompassing *global schema* but instead define *MDBS schemas* to provide the data for individual applications, or groups of applications. MDBS schemas are equivalent to *federated schemas* as defined by Sheth and Larson [14]. MDBS schemas are made up of virtual global relations we call *MDBS Views*.

MDBS Views are views that span multiple heterogeneous databases. They are like relational views in that they are not physically materialized but rather are stored as mappings which are invoked whenever an MDBS View is accessed. The syntax for MDBS Views extends the standard SQL view definition facility with support for *attribute contexts* and *transformation functions*. Attribute contexts are used to describe the semantics of the attributes and transformation functions are used to resolve several types of schema conflicts.

3.1. Schema Conflicts

The key issue in schema integration is the resolution of conflicts among the schemas. A schema integration method, therefore, can be viewed as a set of steps to identify and resolve conflicts. Schema conflicts represent differences in the semantics different schema designers associate with syntactic representations in the data definition language so it is not practical to attempt to fully automate the schema integration process. Our approach is to develop a method and a set of tools to assist the schema integrator. Figure 2 contains a set of example CDBS schemas which are used to illustrate the types of schema conflicts discussed below. The schemas describe data for the management of individual research projects within CORDS at three of the member universities. SQL table definitions of the CDBS relations are given in Appendix A.

The first step in developing a schema integration method is to identify and classify the schema conflicts handled during integration. A number of classifications have been proposed, for example Kim and Seo [8] and Missier and Rusinkiewicz [10]. Our approach, as shown in Table 1, classifies conflicts according to two dimensions: location (columns in Table 1) and type (rows in Table 1). The location of a conflict, given that our integration model is based on the relational data model, can be either in an attribute, within a relation or within a schema (that is involve multiple relations). Our set of conflict types, based on the categorization of Missier and Rusinkiewicz, include the following:

1. **Data type** conflicts occur when semantically equivalent attributes have different types. For example, the identifiers for research projects in **CDBS1**, "Project.*Id*", and in **CDBS2**, "ProjectInfo.*ProjId*", are of type integer and char(10), respectively.
2. **Scale** conflicts arise when semantically similar attributes use different units of measure. For instance, the project budgets in **CDBS2**, "ProjectInfo.Budget", and **CDBS3**, "Proj.pBudget", are in thousands of dollars and in dollars, respectively.
3. **Precision** conflicts arise when different granularity's are used for semantically equivalent attributes. For example, email addresses in **CDBS1**, "Members.Email", and in **CDBS2**, "Personnel.Email", are local addresses and full Internet addresses, respectively.
4. **Default value** conflicts occur when attributes defined in the same domain carry different default values from that domain. For example, the default values for project lengths in **CDBS1** ("Project.Length") and **CDBS3** ("Proj.pLength") are 0 years and 1 year, respectively.
5. **Name** conflicts occur when either semantically similar objects carry different names (synonyms), or when semantically unrelated objects carry the same name (homonyms). For instance, project participants are called "Members" in **CDBS1** and "Personnel" in **CDBS2**.
6. **Key** conflicts occur when two or more relations model semantically equivalent objects but have semantically different keys. For example, the key for the project participant table is "Members.Phone" in **CDBS1** and "Personnel.EmpNo" in **CDBS2**.
7. **Schema isomorphism** conflicts arise when a different number of attributes, or relations, are used to describe semantically similar objects. For instance project participants' addresses are described by the combination of the attributes "Members.Street", "Members.City" and "Members.AptNo" in **CDBS1** and by the single attribute "Personnel.Address" in **CDBS2**.
8. **Union compatibility** conflicts occur between two relations if there is a mismatch in the number, or in the domains, of the attributes. For example, the relations "Deliverables" in **CDBS2** and "Milestones" in **CDBS3** are not union compatible.

CDBS1 - Queen's University

Project (*Id*, Name, PI, Description, Budget, Length)
Members (Name, Position, EMail, *Phone*, Office, Street, City, AptNo, Wage)
MileStones (*Name, ProjectId*, ExpDelDate, ActDelDate)

CDBS2 - University of Michigan

ProjectInfo (*ProjId*, Name, Leader, Budget, StartDate, EndDate)
Personnel (ProjId, *EmpNo*, Name, Address, Phone, OfficeNo, Email, Salary)
Deliverables (*Name, ProjId*, DeliveryDate)

CDBS3 - University of Western Ontario

Proj (*pId*, pName, pDescription, pLeader, pStatus, pBudget, pLength)
Employees (*eNum*, eProj, eName, eAddress, ePhone, eEmail, eSalary)
Students (*sNum*, sProject, sName, sAddress, sPhone, sOffice, sEmail, sSalary)
Milestones (*mName, mProjectId*, mDescription, mDate)

** Primary keys for the relations are shown in *italics.*

Figure 2: Example CDBS Schemas

9. **Abstraction level incompatibility** conflicts occur when semantically similar objects are described at different levels of abstraction. For example, project participants are described as either "Employees" or "Students" in **CDBS3** and as the more general entity "Members" in **CDBS1**.
10. **Missing data** conflicts arise when an object is described in one schema by a subset of the attributes used in the other schema. For example the relation "Deliverables" in **CDBS2** contains a subset of the attributes in "Milestones" in **CDBS3**.
11. **Integrity constraint** conflicts arise when there are different integrity constraints for semantically similar objects. This type of conflict is not considered in this paper but is discussed elsewhere [3].

3.2. Schema Integration Method

Our schema integration method decomposes schema integration into a number of tasks which may be linked together into an iterative process as shown in Figure 3. We expect that a user may return to any of the previous steps and, using the feedback obtained by moving through the process, either modify or augment the schemas at any stage. The steps break down conflict resolution according to the locations of the conflicts.

3.2.1. Export

Exporting a local schema (or portion of a local schema) makes that schema, and its corresponding data, available to the MDBS. The export task involves two activities: translating the local schema into its corresponding representation in the common data model, and providing the *contexts* through which the data can be interpreted.

Attribute contexts support the resolution of attribute level conflicts by providing a mechanism to describe the semantic properties of the attributes, for example data type, scale, precision, etc. The *context* of an attribute consists of a number of *facets* where each facet corresponds to a semantic property. The facets included in the context are

1. *uniqueness* - constraint specifies that two tuples cannot have the same value for the attribute;
2. *cardinality* - constraint on the number of values that can be present in the attribute (just NULL or NOT NULL in relational);
3. *type* - set of values from which an attribute may draw;
4. *precision* - specifies the granularity of the data values;
5. *scale* - specifies the interpretation of the values, for example unit of measurement for numeric and language or code for nonnumeric;
6. *default value* - specifies the default value assumed if no value is provided.

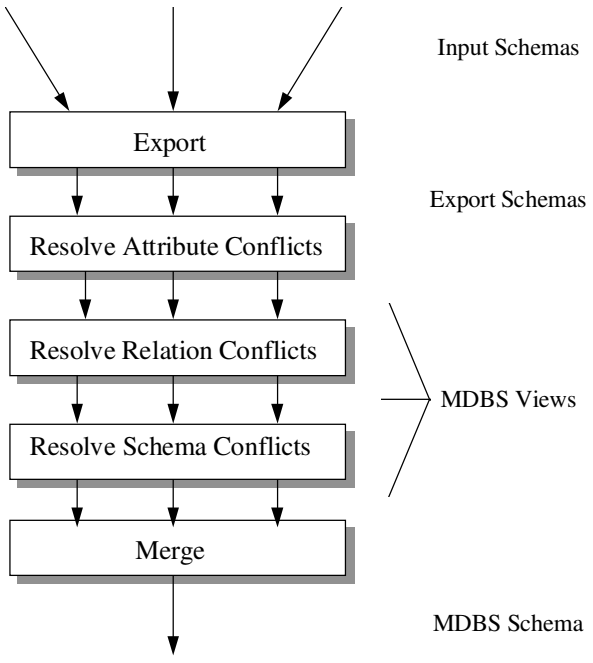

Figure 3: Schema Integration Process

3.2.2. Resolve Attribute Level Conflicts

Corresponding attributes from the various export schemas are identified and mapped to a MDBS attribute using the MDBS View definition. Name conflicts are resolved by this mapping. The other types of attribute level conflicts, which involve the context facets, are resolved by the introduction of *transformation functions* to map the export attribute contexts to the MDBS attribute context. A transformation function is a user-defined function which is applied to the data from a CDBS to convert it into the format expected by the MDBS user. For example, a transformation function may be defined to convert a data value expressed in inches into one expressed in feet, or transformation function might be used to combine a number of attributes into a single attribute to resolve a schema isomorphism conflict.

3.2.3. Resolve Relation Level Conflicts

Relations from the various export schemas that represent semantically equivalent objects are identified and conflicts that occur within the scope of these relations are resolved using transformation functions.

3.2.4. Resolve Schema Level Conflicts

Schema level conflicts involve more than one relation within a single export schema. Conflicts of this type are usually resolved by combining the relations into a single logical relation using a join or union within the MDBS View statement.

3.2.5. Merge

A MDBS View definition is created that combines the corresponding MDBS Views created from the individual export schemas in the previous steps.

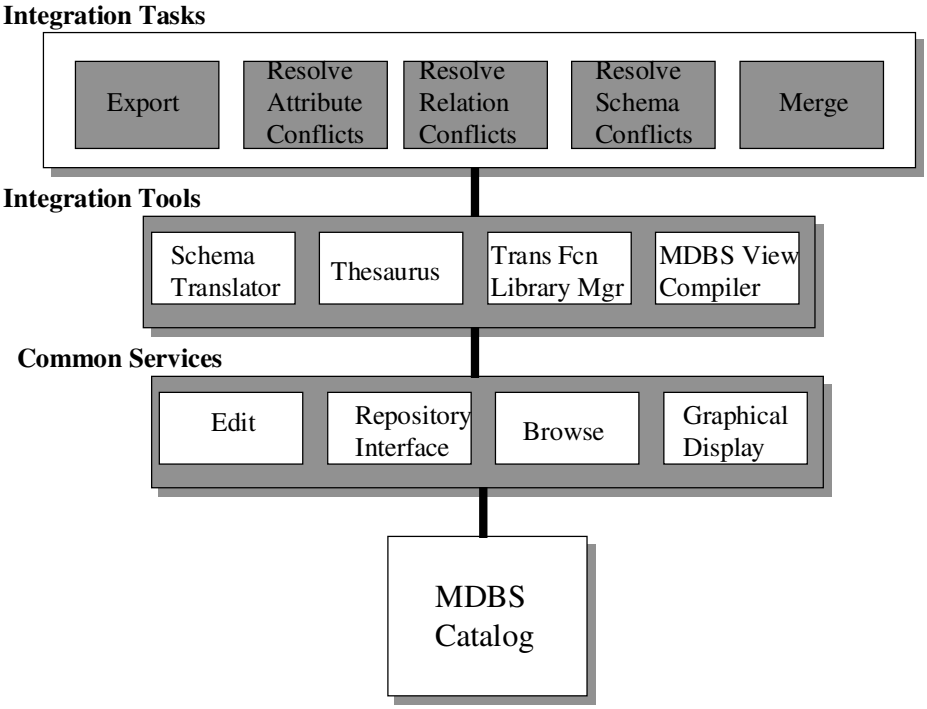

Figure 4: Schema Integration Toolkit

3.3. Schema Integration Toolkit

The CORDS Schema Integration Toolkit is a set of tools and services to support a MDBS DBA using the schema integration method described above to create MDBS Schemas. It has an AIX Windows graphical interface and was developed on an RS/6000 machine. It runs as an application of the CORDS MDBS. The structure of the prototype is shown in Figure 4. A user performing the integration tasks described in Section 3.2 uses one or more of the available integration tools to carry out each task. The integration tools, in turn, are implemented using one or more of the Common Services provided by the environment.

The integration tools available include the following:

1. **Schema Translator**: The schema translator is a tool that automates the translation from one data model to another. It takes as input a file containing the local schema expressed in terms of the local data model. The output is a file containing the schema expressed in terms of the desired data model. Currently we support translation among schemas defined in relational, hierarchical, network and object-oriented models. The schema translator automatically generates basic export table definitions from the translated schema which can then be manually edited, augmented with facet information, then submitted to the MDBS. The translation process is described in detail elsewhere [1].

2. **Thesaurus**: The thesaurus is used to help with name conflict resolution. It stores information about relationships, in particular synonyms, among object names. Users are able to add information to the thesaurus as new names and relationships are encountered. The thesaurus analyzes a schema expressed in the common data model and highlights possible relationships among names in the schema with names currently stored in the thesaurus. It may also be used in conjunction with the browser.
3. **Transformation Function Library Manager**: A number of basic transformation functions have been implemented in the schema integration environment; for example, functions that convert integers to character strings or convert from one character length to another. These functions are maintained in a library by the Transformation Function Library Manager. Descriptions of the transformation functions are stored in the MDBS Catalog and may be accessed using the Browser. In resolving a conflict, the user searches the Catalog for an appropriate transformation function and, if one is found, a call to that function is included in the MDBS View definition. Transformation functions are later invoked from the library by the MDBS. If a suitable transformation function is not found in the library the user can invoke the Editor, create and compile new transformation functions, and then add them to the library.
4. **MDBS View Compiler**: The MDBS View Compiler parses and checks an MDBS View definition and then stores the appropriate information in the MDBS Catalog.

The Common Services available in the Schema Integration Environment are used by the integration tools or may be invoked directly by a user. They include the following:

1. **Editor**: A common editor is used to create and modify MDBS View definitions and transformation functions.
2. **Catalog Interface**: The Catalog Interface is a set of routines used to access the MDBS Catalog. The implementation of the Catalog and the interface are described elsewhere [9].
3. **Browser**: The MDBS catalog browser allows users to browse and query the catalog to find out what objects are currently available via the MDBS. The browser facilitates the selection of names and attribute types during the exportation stage in order to simplify the integration mappings. Filters allow users to locate items with particular properties. The implementation of the browser is described in detail elsewhere [9].
4. **Graphical Interface**: A common graphical interface is used by all the tools to facilitate easy movement through the steps of the schema integration process.

4. Scenario for Schema Integration

In this section, we provide a sample scenario to show how the tools and method can be used to combine data from different sources into a single logical view. To illustrate, we use the sample component database schemas shown in Appendix A. Personnel information is stored local to each CDBS. For our scenario, we show how this information may be logically combined in a step-by-step manner so that it may be accessed as a single logical relation.

4.1. Export

Export schemas are defined using an extended SQL. An export table definition includes the export table name (which must be unique within the MDBS), the attribute names and types, and information about the component data source (the site, the database and the table where the actual data resides). Unique names are created by concatenating site name, schema name and table name. Additional semantic information may be added to the attributes by the inclusion of facets. The current prototype allows for the specification of cardinality, domain, range, scale (units) and default values.

Initial versions of export tables are produced from the CDBS schemas by the Schema Translator. They are then refined by the DBA using the editor supplied with the schema integration toolkit. Once an export schema has been defined, the DBA submits it to the MDBS where it is parsed and stored in the MDBS catalog.

Figure 5, Figure 6 and Figure 7 show the export tables which are selected for inclusion in our sample MDBS view. These tables contain semantically equivalent information, namely information about people involved in research projects at different sites. The browser facilities may have been used to locate the participating export tables.

```
EXPORT TABLE UMichiganMembers (
  ProjId        char(10),
  Name          char(40),
  EmpNo         integer
    (Cardinality = "not Null"),
  Address       char(80),
  Phone         integer,
  Salary        dollar
    (Units = "US dollars"),
  OfficeNo      char(6),
  Email         char(30)
)
from umichigan.CDBS2.Personnel;
```

Figure 5: Michigan Export Table

```
EXPORT TABLE QueensMembers (
  Name          char(40),
  Position      char(20)
    (Domain = (student, faculty, research)),
  Wage          dollar
    (Units = "canadian dollars"),
  Email         char(30),
  Phone         integer
    (Cardinality = "not Null"),
  Office        integer,
  Street        char(40),
  City          char(20),
  AptNo         char(10)
) from queensu.CDBS1.Members;
```

Figure 6: Queen's Export Table

```
EXPORT TABLE WesternStudents (
  Num           integer
    (Cardinality ="not Null"),
  Project       integer,
  Name          char(30),
  Address       char(60),
  Phone         integer,
  Office        integer,
  Salary        dollar
    (Units="canadian dollars"),
  Email         char(60)
) from westernu.CDBS3.Students;

EXPORT TABLE WesternEmployees (
  Num           integer
    (Cardinality ="not Null"),
  Project       integer,
  Name          char(40),
  Address       char(80),
  Phone         integer,
  Salary        dollar
   (Units="canadian dollars"),
  Email         char(60)
) from westernu.CDBS3.Employees;
```

Figure 7: Western Export Tables

4.2. Resolve Attribute Level Conflicts

The next step is to identify the attributes that are to be included in the integrated schema. For our integrated view, we include the name, address, email address, and salary information for each project member. The Thesaurus tool is used to discover potential correspondences among attributes based on the names used for the attributes. Name conflicts are then resolved by mapping the export attributes to a common generic name using the MDBS View definition statement.

Once corresponding attributes are identified the Transformation Function Library Manager is used to analyze the contexts of the attributes and to suggest transformation functions, if required, to map the export attributes to the view attributes. If an appropriate transformation function is not already available in the library, the DBA uses the editor to create a new transformation function and then includes the new function in the library. Figure 8, Figure 9 and Figure 10 show the creation of four views (one corresponding to each of the export tables) in which several attribute conflicts are resolved through the use of transformation functions.

4.3. Resolve Relation Level Conflicts

We next identify the relation level conflicts. Resolution of these conflicts are shown in Figures 8, 9 and 10. Name conflicts can be resolved by renaming via the MDBS view definition. The schema isomorphism conflict concerning the address information across the three CDBSs is resolved by applying a transformation function "StringConcat" to the three fields in the Queen's table to produce a single address field.

To resolve the key conflict present in our example, Phone, the key in CDBS1.Members, is used as an employee identifier. Since employee numbers may not be unique across universities an attribute to identify the university is added to the views with the "EXPLICIT" function and the key becomes the combination of "University" and "EmpId".

```
CREATE VIEW MichiganView (
  Name          char(80),
  Address       char(100),
  University    integer,
  EmpId         integer
    (Cardinality = "not Null"),
  Email         char(60),
  Salary        dollar
    (Units = "US dollars")
)
AS SELECT  ToChar80(Name), ToChar100(Address), EXPLICIT(1), EmpNo,
          ToChar60(Email), UStoCanDollar(Salary)
  FROM UMichiganMembers;
```

Figure 8: Michigan View - Resolve Attribute and Relation Conflicts

```
CREATE VIEW  QueensView (
   Name          char(80),
   Address       char(100),
   University    integer,
   EmpId         integer
     (Cardinality="not Null"),
   Email          char(60),
   Salary        dollar
    (Units = "canadian dollars")
)
AS SELECT ToChar80(Name), StringConcat(Street, City, AptNo),
          EXPLICIT(2), Phone, ToChar60(Email), Wage
   FROM QueensMembers;
```

Figure 9: Queen's View - Resolve Attribute and Relation Conflicts

4.4. Resolve Schema Level Conflicts

There is an abstraction level incompatibility between the Western views in Figure 10 which consider employees and students separately, and the other two CDBSs which use a single, more general, relation to capture both types of project participants. The two Western views are combined into a more general one in Figure 11.

4.5. Merge

The intermediate views are merged together into a final MDBS View in Figure 12.

```
CREATE VIEW WesternStdView (
  Name          char(80),
  Address       char(100),
  University    integer,
  EmpId         integer
    (Cardinality="not Null"),
  Email         char(60),
  Salary        dollar
    (Units = "canadian dollars")
)
AS SELECT ToChar80(Name), ToChar100(Address), EXPLICIT(3), Num,
          Email, Salary
  FROM  WesternStudents;

CREATE VIEW WesternEmpView (
  Name          char(80),
  Address       char(100),
  University    integer,
  EmpId         integer
    (Cardinality = "not Null"),
  Email          char(60),
  Salary        dollar
   (Units = "canadian dollars")
)
AS SELECT ToChar80(Name), ToChar100(Address), EXPLICIT(3), Num,
          Email, Salary
  FROM WesternEmployees;
```

Figure 10: Western Views - Resolve Attribute and Relation Conflicts

```
CREATE VIEW WesternView (
   Name          char(80),
   Address       char(100),
   University    integer,
   EmpId         integer,
   Email         char(60),
   Salary        dollar
     (Units = "canadian dollars")
)
AS SELECT Name, Address, University, Num, Email, Salary
   FROM WesternStdView
   UNION
   SELECT Name, Address, University, Num, Email, Salary
   FROM WesternEmpView;
```

Figure 11: Western Combined View - Resolve Schema Conflicts

```
CREATE VIEW CORDSMembers (
   Name          char(80),
   Address       char(100),
   EmpId         integer,
   University    integer,
   Email         char(60),
   Salary        dollar
     (Units = "canadian dollars")
)
as SELECT Name, Address, PhoneNumber, University, Email, Salary
   FROM QueensView
   UNION
   SELECT  Name, Address, EmpId, University, Email, Salary
   FROM MichiganView
   UNION
   SELECT Name, Address, EmpId, University, Email, Salary
   FROM  WesternView;
```

Figure 12: Final Merged View

5. Related Work

Johannesson and Jamil [7] observe that, in most approaches to schema integration in the literature [4,6,11,12,16], the schema integration process can be divided into three major phases: schema comparison, schema conforming and schema merging. *Schema comparison* involves analyzing and comparing schemas in order to determine correspondences, in particular different representations of semantically equivalent objects. *Schema conforming* transforms schemas in order to increase their similarity, and *schema merging* produces the integrated schema. The schema integration method presented here contains these three activities though not in distinct phases. The schema comparison and conforming activities are blended throughout the export and conflict resolution steps of our approach.

Schema integration methods may be grouped according to whether they are procedure-based or assertion-based. *Assertion-based* approaches [16], use assertions to state correspondences between constructs in the different CDBS schemas. The assertions are then used to guide the integration. An advantage of this type of approach is that assertions can be "model-independent". *Procedure-based* approaches [4,6,11,12] manipulate the input schemas directly. Integration is typically performed by creating *views* on top of the input schemas. Approaches of this type assume a particular common data model. The method discussed in this paper is a procedure-based approach.

Our work is different from most methods presented in the literature, with the exception of Pegasus [12], in that it is part of a working MDBS system and it deals with

the complete range of activities associated with schema integration from schema translation of heterogeneous schemas through to schema merging.

Our notion of attribute contexts is based on work by Sciore, Siegel and Rosenthal [13] which uses semantic contexts to support interoperability. They introduce the concept of a *semantic value*, which is a data value together with its associated context, as the unit of information exchange between information systems. This exchange is handled by a new system component called a context mediator. We use contexts for a similar purpose, namely to describe the semantic properties, or facets, of a data value. The context information is used to guide the user in locating, and/or creating, appropriate transformation functions which perform some of the duties of the context mediator.

6. Summary

Semantic heterogeneity is one of the more difficult issues in MDBSs. The CORDS MDBS provides the CORDS Schema Environment to deal with the problem. We have described the schema integration method and integration toolkit that compose the environment. Schema integration is a complex process. Our approach has not been to try and automate integration but rather to provide the user with an organized approach and a set of tools that operate within a common environment.

Our schema integration method is based on the concept of MDBS Views, an extension of relational views, which span multiple heterogeneous CDBSs. MDBS Views are a mechanism with which a user may perform the logical integration of data and subsequently transparently access the data in the CDBSs. We provide a classification of possible schema conflicts according to both type and location. The schema integration method then uses the classification to divide the integration process into a series of steps. The process is illustrated by a schema integration scenario and example.

An important aspect of the schema integration toolkit is that it supports the user through all phases of the integration process within a common working environment. The integration tools share a number of common services and the metadata for all phases of the integration is stored in a common repository, the MDBS Catalog. The tools automate some tasks in the integration process and support the user in performing others.

References

[1] R. Abu-Hamdeh, J. Cordy and P. Martin. Schema Translation using Structural Transformation. *Proc. of CASCON' 94: IBM Centre for Advanced Studies Conference*, 202 - 215 (October 1994).

[2] G. Attaluri, D. Bradshaw, N. Coburn, P.-A. Larson, P. Martin, A. Silbershatz, J. Slonim and Q. Zhu. The CORDS Multidatabase Project. *IBM Systems Journal 34(1)*, 39 - 62 (1995).

[3] P. Baliga. *Computing Constraints on Multidatabase Views*. M.Sc. Thesis, Dept. of Computing and Information Science, Queen's University (1994).

[4] C. Batini, M. Lenzerini and S. Navathe. A Comparative Analysis of Methodologies for Database Schema Integration. *ACM Computing Surveys 18(4)*, 323 - 364 (1986).

[5] M. Bauer, N. Coburn, D. Erickson, P. Finnigan, J. Hong, P.-A. Larson, J. Pachl, J. Slonim, D. Taylor and T. Teory. A Distributed System Architecture for Distributed Application Environment. *IBM Systems Journal 33(3),* 399 - 425 (1994).

[6] U. Dayal and H. Hwang. View Definition and Generalization for Database Integration in a Multidatabase System. *IEEE Trans. on Software Engineering 10(6)*, 628 - 645 (1984).

[7] P. Johannesson and M. Jamil. Semantic Interoperability Context Issues and Research Directions. *Proc. of the Second International Conference on Cooperative Information Systems*, 180 - 191 (May 1994).

[8] W. Kim and J. Seo. Classifying Schematic and Data Heterogeneity in Multidatabase Systems. *IEEE Computer 24(12),* 12 - 18 (Dec. 1991).

[9] P. Martin and W. Powley. Storing MDBS Catalog Information in an X.500 Directory Service. *Proc. of CASCON'94, IBM Centre for Advanced Studies 1994 Conference*, 216 - 226 (October 1994).

[10]P. Missier and M. Rusinkiewicz. Extending a Multidatabase Manipulation Language to Resolve Schema and Data Conflicts. *Proc. of the Workshop on Interoperability of Database Systems and Database Applications*, 19 - 37 (October 1993).

[11]A. Motro. Superviews: Virtual Integration of Multiple Databases. *IEEE Trans. on Software Engineering 13(7)*, 785 - 798 (1987).

[12]A. Raffi, R. Ahmed, M. Ketabchi, P. DeSmedt and W. Du. Integration Strategies in the Pegasus Object-Oriented Multidatabase System. *Proc. of the 25th Hawaii International Conference on System Sciences*, 323 - 334 (1992).

[13] E. Sciore, M. Siegel and A. Rosenthal. Using Semantic Values to Facilitate Interoperability Among Heterogeneous Information Systems. *ACM Trans. on Database Systems 19(2)*, 254 - 290 (June 1994).

[14] A. Sheth and J.A. Larson. Federated Database Systems for Managing Distributed, Heterogeneous, and Autonomous Databases. *ACM Computing Surveys 22(3),* 183 - 236 (Sept. 1990).

[15]A. Sheth. Semantic Issues in Multidatabase Systems. *ACM Sigmod Record 20(4),* 5 - 9 (December 1991).

[16]S. Spaccapietra, C. Parent and Y. Dupont. Model Independent Assertions for Integration of Heterogeneous Schemas. *VLDB Journal 1(2)*, 81 - 126 (1992).

Appendix A: Schema Integration Example

CDBS1 - Queen's University

```
CREATE TABLE Project
   Id            integer NOT NULL,
   Name          varchar(40),
   PI            char(40),
   Description   char(80),
   Budget        integer,
   Length        integer;

CREATE TABLE Members
   Name          char(40),
   Position      char(20),
   Wage          dollar,
   Email         char(30),
   Phone         integer NOT NULL,
   Office        integer,
   Street        char(40),
   City          char(20),
   AptNo         char(10);

CREATE TABLE MileStones
   Name          char(30) NOT NULL,
   ProjectId     integer, NOT NULL,
   ExpDelDate    date,
   ActDelDate    date;
```

CDBS2 - University of Michigan

```
CREATE TABLE ProjectInfo
   Name          char(40),
   ProjId        char(10) NOT NULL,
   Leader        char(40),
   Budget        integer,
   StartDate     date,
   EndDate       date;

CREATE TABLE Personnel
   ProjId        char(10),
   EmpNo         integer NOT NULL,
   Name          char(40),
   Address       char(80),
   Phone         integer,
   Salary        dollar,
   OfficeNo      char(6),
   Email         char(30);
```

```
CREATE TABLE Deliverable
  ProjId          char(10) NOT NULL,
  Name            char(80) NOT NULL,
  DeliveryDate    date;
```

CDBS3 - University of Western Ontario

```
EXPORT TABLE Proj
  pId             integer NOT NULL,
  pName           char(80),
  pDescription    char(100),
  pLeader         char(40),
  pStatus         char(20),
  pBudget         integer,
  pLength         integer;

CREATE TABLE Employees
  eNum            integer NOT NULL,
  eProj           integer,
  eName           char(40)
  eAddress        char(80),
  ePhone          integer,
  eSalary         dollar,
  eEmail          char(60);

CREATE TABLE Students
  sNum            integer NOT NULL,
  sProject        integer
  sName           char(30),
  sAddress        char(60),
  sPhone          integer,
  sOffice         integer,
  sSalary         dollar,
  sEmail          char(60);

CREATE TABLE Milestones
  mProjectId      integer NOT NULL,
  mName           char(60) NOT NULL,
  mDescription    char(100),
  mDate           date;
```