An Object-Oriented Query Language Interface to Relational Databases in a Multidatabase Database Environment

Susan D. Urban Taoufik Ben Abdellatif

Department of Computer Science and Engineering
Arizona State University
Tempe, AZ 85287-5406
urban@asuvax.eas.asu.edu

Abstract

This paper presents an approach to query processing in a multidatabase system that uses an object-oriented model to capture the semantics of other data models. The object-oriented model is used to construct a global schema, defining an integrated view of the different schemas in the environment. The model is also used as a self-describing model to build a meta-database for storing information about the global schema. A unique aspect of this work is that the object-oriented model is used to describe the different data models of the multidatabase environment, thereby extending the meta database with semantic information about the local schemas. Structural mappings between the global schema and each local schema are then easily supported. An object algebra provides a query language for expressing global queries, using the structural mappings to translate object algebra queries into SQL queries over local relational schema. The object algebra can be used to directly retrieve temporarily-stored data from the object-oriented database or to transparently retrieve data from local sources using the translation process described in this paper.

1 Introduction

Access to multiple database systems has become a challenging issue to the database community [Litw86]. As described in [Sheth90], a multidatabase system can have several different architectures. In loosely-coupled architectures, the user is responsible for understanding the different access languages of the individual databases. In tightly-coupled architectures, the user is typically provided with a global conceptual schema (GCS) to provide a high-level integrated view of the database systems to be accessed, hiding the structural differences between different schemas and giving the user the impression that he/she is accessing a single database. A common query language for accessing data using the GCS can then be provided. The relational model and extended entity relationship models have often been used for forming global schemas [Sheth90]. These models, however, have limited semantic capabilities. Ideally, the model used for forming a GCS should be expressive enough to fully address the similarities and differences between the different schemas of the multidatabase environment.

This research investigates an approach to query translation in a multidatabase system that uses an object-oriented model to capture the semantics of other data models [Abde94]. The object-oriented model is used to construct a GCS, defining an integrated view of the different schemas in the environment. The object-oriented model is used as a self-describing model [Rous85] so that it is also used to build a meta-database for storing information about the GCS. A unique aspect of this work is that the object-oriented model is also used to describe the different data models of the multidatabase environment, thereby extending the meta database with semantic information about the local schemas. With the GCS and local schemas all represented in an object-oriented form, structural mappings between the GCS and each local schema are easily supported. An advantage of this approach is that a high-level object-oriented query language can be used to express queries over the GCS. The structural mappings are then used to generate appropriate queries in the query language of the local data models.

As part of the objectives of this work, the CORAL semantic data model was used as the basis for the description of the GCS, the local database schema, and the structural mappings between the GCS and the local schemas [Urba91a, Urba91b]. To limit the scope of this work, this research has focused on providing object-oriented access to the relational model only. The ONTOS object-oriented database [ONTO92] system was then used to build the meta database over which global queries are expressed. An object algebra

1063-6927/94 $03.00 © 1994 IEEE

[Shaw90] provides a query language for expressing global queries, using the structural mapping details to translate object algebra queries into SQL queries over the local relational schemas. The advantage of using an object algebra is that the object-oriented database can be viewed as a blackboard for temporary storage of local data (as viewed by the GCS) and for establishing relationships between different databases. The object algebra can be used to directly retrieve temporarily-stored data from the blackboard (i.e., object-oriented database) of the GCS or to transparently retrieve data from local sources using the translation process described in this paper.

The remainder of this paper is organized as follows. A brief overview of related work appears in Section 2. A discussion of the semantic framework then follows in Section 3, including an overview of the self-describing data model and the way in which it is used to describe local schemas and global-to-local mappings. A specific example is also presented together with an evaluation of the mapping framework. Section 4 provides an overview of the object algebra that is used as the query language for this research. Section 5 describes the query translation process, illustrating how the semantic framework is used to generate SQL queries. The paper concludes in Section 6 with a summary and description of future work.

2 Related Work

An excellent survey of the current state of multidatabase systems research is presented by Sheth in [Shet90]. In [Shet90], Sheth indicates that multidatabase systems take different approaches for solving the problem of semantic heterogeneity. Their approaches are reflected through the use of different architectures, different data manipulation languages, different approaches to the resolution of semantic conflicts and different query processing schemes.

As an example, Mermaid [Temp87] uses a relational model to represent global schema and uses language translation, data translation, and schema translation to resolve semantic problems using its data dictionary. DQS [Belc88], also based on the relational model, uses an additional schema to store information about mapping definitions which are carried out according specific rules at each site.

The ER and EER models have also been used as common data models in ADDS [Brei86], OMNIBASE [Rusi89], and SCOOP [Spac82]. Both models have been weak, however, in deriving external views. Furthermore, the ER model is not capable of fully describing schemas expressed using semantic and object-oriented data models [Salt91]. The multidatabase system Multibase [Land82] uses the DAPLEX functional data model [Ship81] to represent its global schema. To resolve the data incompatibilities that result from accessing data jointly from different databases, Multibase develops rules that are included with each federated view and are stored in auxillary databases.

More recent approaches have used semantic and object-oriented data models for integrating heterogeneous databases. Bertino's work in [Bert91] describes a method that creates object-oriented views of the systems to be integrated. In [Cast91] the author suggests the augmentation of the semantic level of the local schemas so that implicit semantics can be made explicit through the use of the data model BLOOM. The *Carnot Multidatabase System* [Coll91] is an association of two tools: 1) an integration tool used to assist in integrating different systems by taking advantage of a set of semantic services provided by the *Cyc knowledge system* [Lena90], and 2) a translation tool that uses a collection of mapping rules to translate queries to and from a global schema. The InHead project [Doyl91] takes a different approach to resolving semantic problems by seeking the help of artificial intelligence tools in association with an object oriented data model. By using a blackboard architecture, the queries are posted and the knowledge sources for each database cooperate to resolve the query.

A more formal approach to data model mapping is presented in [Kali90], where denotational semantics is used to define data model translations. An approach such as that in [Kali90] can theoretically be used to formally define the translation process defined in this paper as well as any of the processes described in the research projects above.

3 Semantic Framework

An important aspect in the development of a query processing system for a multidatabase is management of metadata of the GCS and of the local schemas to which they map. This section describes the meta data representation that we have established using a self-describing semantic data model.

3.1 A Self-Describing Semantic Data Model

CORAL is the semantic data model that has been used as a self-describing model in this research. The top portion of the diagram in Figure 1 illustrates the features of the semantic data model in a self-describing mode. In the model, circles represent classes. Bold lines represent superclass/subclass relationships and thin lines represent class properties. There are two main model constructs: CLASS and PROPERTY. A *class* can either be a SIMPLE_CLASS with atomic and printable values or an ABSTRACT_CLASS representing objects that require explicit creation and deletion.

As indicated in Figure 1, an ABSTRACT_CLASS can have properties that are either multi-valued (double arrow head) or single-valued (single arrow head). This information

is kept in a boolean-valued attribute *sv/mv*. The fact that ABSTRACT_CLASSs have properties is modeled by the *property* attribute of ABSTRACT_CLASS. Viewing properties as function definitions, properties map objects in the domain to objects in the range. As a result, the inverse of *properties* is the *domain* attribute of the PROPERTY class. Each PROPERTY object also has a *range* which is a class object. The range of a PROPERTY object can therefore be a SIMPLE_CLASS or an ABSTRACT_CLASS.

Figure 1 also shows that a property can be specified as non-null through the use of the *required* boolean-valued attribute. If a property object represents a one-to-one correspondence between the domain and the range of the property definition, the *unique* attribute would indicate this semantics. Properties that are required are indicated in Figure 1 by the letter *R* following the property name. Inverse properties are indicated in Figure 1 through the use of the *inverse-of* attribute of the PROPERTY class.

An ABSTRACT_CLASS can also have composite keys, where keys are used for external identity. KEY objects are composed of many PROPERTY objects. Finally, classes can be grouped into CATEGORYs, where a category is a logical grouping of a superclass (*category_owner*) and its subclasses (*category_members*) to form groups of disjoint and/or required subclasses.

3.2 Description of Local Schemas

The self-describing schema of the CORAL semantic model in the top portion of Figure 1 provides the basic representation for storing global conceptual schemas in this research. CORAL is also used to describe the models of the databases that are integrated as part of this work, thus creating a database for storing semantic information about the schemas to be integrated. This approach also establishes a basis for describing structural mappings between the global conceptual schema and the local schemas.

The bottom half of Figure 1 represents the CORAL description of relational schemas. Each relational database is composed of RELATIONs, ATTRIBUTEs, and KEYs. Each RELATION has a required *name*. A RELATION can have several keys and attributes as indicated by the *has_keys* and the *has_attributes* properties, respectively. The KEY, which is a collection of attributes, can be a foreign key or a primary key of a relation that is used elsewhere as a foreign key, as indicated by the *pk_to_fk* attribute. Each key object includes the required boolean attribute *primary_key* to specify if it is a primary key. Each ATTRIBUTE has a required *name* attribute and can be specified as non-null through the use of the *required* attribute. One-to-one attributes can be specified using the *unique* attribute. ATTRIBUTEs have a *domain* field that specifies the attribute type. ATTRIBUTEs also include a dependency field that specifies if the attribute is functionally dependent on the primary key of the relation.

3.3 Structural Mappings

Given the self-describing model presented in Figure 1 together with the description of the relational schemas, the structural mapping needed to support global query processing based on an object-oriented view of local schemas can now be presented. As shown by the dashed lines in Figure 1, an ABSTRACT_CLASS in the global schema can map to one or more ATTRIBUTEs in a relational schema. The mapping occurs through the ATTRIBUTE_CONSTRUCT class so that additional semantics can be attached to the mapping using the *and/or* attribute. If the *and/or* attribute is set to "and", then the ABSTRACT_CLASS maps to a concatenation of relational attributes, otherwise the class maps to either one of the indicated attributes.

An ABSTRACT_CLASS can also correspond directly to one or more RELATIONs. The RELATION_CONSTRUCT class specifies whether an ABSTRACT_CLASS maps to a union of RELATIONs or a join of RELATIONs through the *union/join* attribute. In addition, a PROPERTY in the global schema can map to one or more ATTRIBUTEs in a local relational database. Note that it is not necessary to explicitly include the case where a property maps to a relation. When a property does map to a relation, the mapping represents the case when the property range is an ABSTRACT_CLASS. Since the mapping for an ABSTRACT_CLASS object is already established, the mapping process can make use of the PROPERTY *range* attribute. In the case where a property in the object-oriented view is associated with an attribute that serves as a foreign key, the property is mapped directly to the foreign key attribute. This additional information is needed to resolve joins in the case where there is more than one foreign key between two relations. If the property range is a SIMPLE_CLASS, the PROPERTY simply maps to the collection of ATTRIBUTEs that it corresponds to.

As an example of the structural mapping, Figure 2 presents three different databases that hold information about students in three different universities. In LCSA, Student refers to both graduate and undergraduate students. The *level* attribute of Student indicates the class standing of the Student. A Course can have a prerequisite and can be taught by more than one Faculty. In LCSB, Student refers to undergraduate students only. The *status* attribute of Course indicates whether the instructor is a faculty member or a graduate student. Finally, LCSC shows the course taught and taken by Graduate_Student, as well as the committee for each Graduate_Student. Each Committee relation has a *chair* attribute that refers to the faculty chair of the committee and a *subject* attribute that refers to the subject of the research.

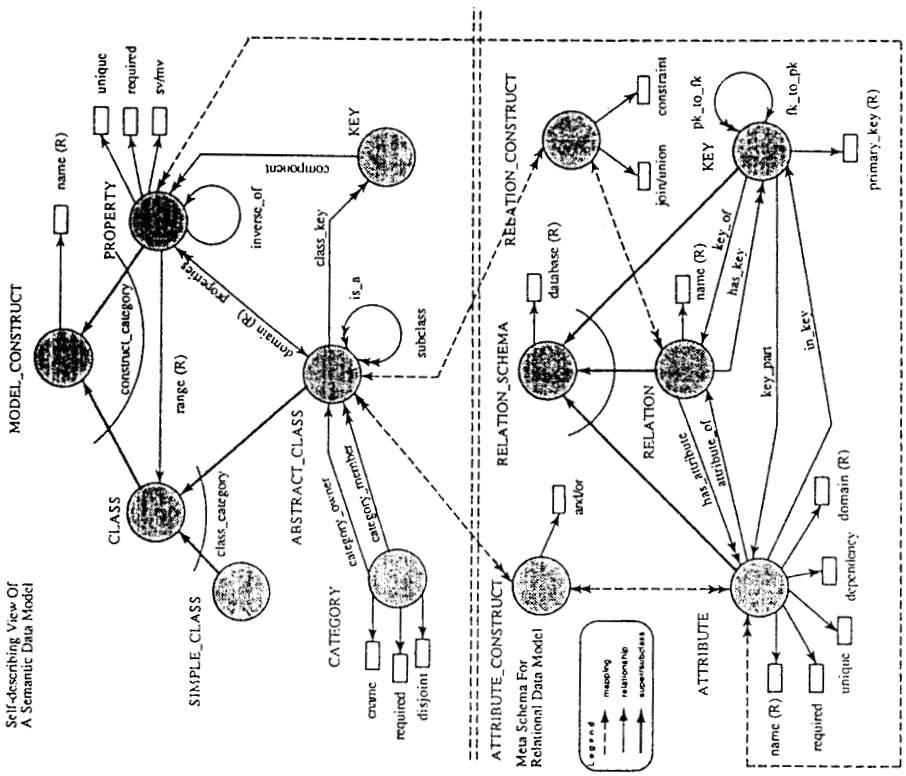

Figure 1: Global-to-Local Schema Mappings Using a Self-Describing Data Model

Figure 3 presents the object-oriented view of the three relational schemas LCSA, LCSB, and LCSC, using the semantic CORAL model. The details of the mapping from the GCS to the local schemas are stored using the structural mapping in Figure 1. As an example, classes can map to either attributes or relations. The Student class in the GCS is therefore mapped to the Student relation in LCSA, the Student relation in LCSB, and the Student relation in LCSC. The Address class in GCS is mapped to a concatenation of the attributes *street*, *city*, *state*, and *zip* of the Student relation in LCSA (using the "and" setting the of ATTRIBUTE_CONSTRUCT class) and to the attribute *address* of the Graduate_Student relation in LCSC. Note that the Faculty class maps to the Faculty relation in LCSA, the *taught_by* or *advisor* attributes in LCSB, and either one of the *chair*, *member1*, *member2*, or *dchair* attributes in LCSC (using the "or" setting of the ATTRIBUTE_CONSTRUCT class). An interesting case is also found in the Graduate class mapping. The Graduate class maps to the Student relation in LCSA with the constraint that the value of the *level* attribute in relation is graduate. This information is needed to support queries over the Graduate class and proper translation to the LCSA schema. This additional information is made available through the *constraint* attribute in the RELATION_CONSTRUCT class.

4 Object Algebra Overview

In order to query a global schema such as that presented in Figure 3, the EQUAL set-oriented object algebra has served as the basis for this work. EQUAL [Shaw90] is an object algebra associated with the object-oriented data model ENCORE.

The operations of the EQUAL object algebra are *select*, *image*, *project*, *ojoin*, *nest*, *unnest*, and *flatten*. In addition, the algebra supports traditional set operations, such as *intersection*, *union*, and *difference*. The set of operators also include the *coalesce* and *dupeliminate* operators to handle the occurrence of duplicate objects.

The *select* operator is used to return a subset of objects from a larger set of objects. In Figure 4, *select* returns the subset of male faculty members from the faculty class. The *flatten* operation is used to remove the extra level of nesting that is produced when the function in the *image* operation returns a set of objects. In Figure 4, flatten is used to change the result of operation s_1 from a set of sets of student objects to a set of student objects.

The *ojoin* operator is used to create relationships between objects from two different collections of objects. Operation s_4 in Figure 4 creates a join between a collection of students and a collection of faculty members. The *project* operator is

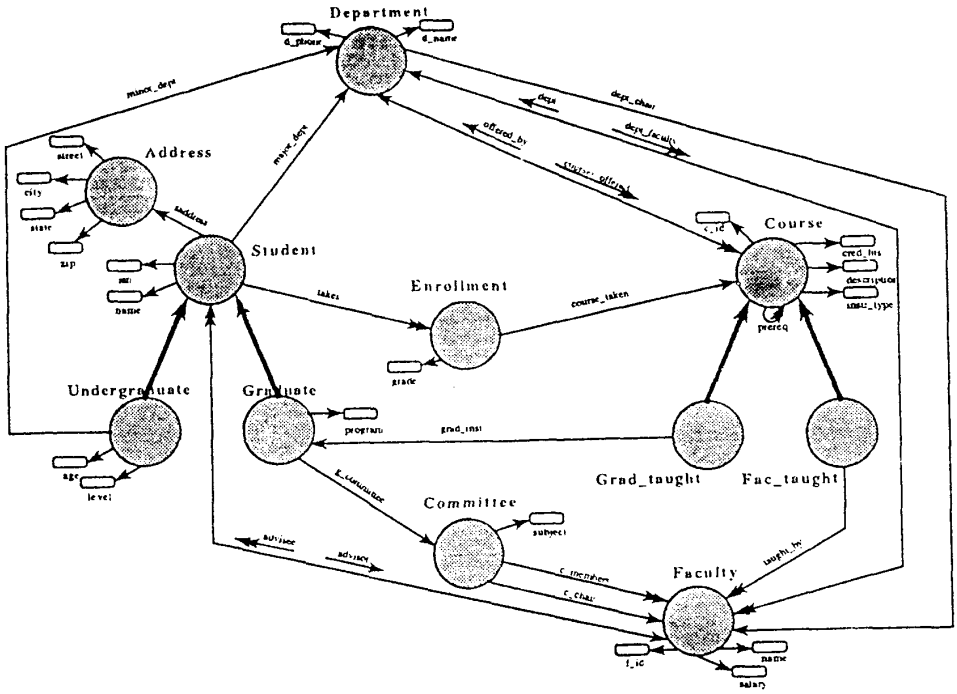

Figure 3: The Semantic Data Model Schema for the University Example

similar to the *image* operator except that it can apply more than one function to each object in the queried collection. In Figure 4, *project* applies functions to extract student and faculty names.

The *dupeliminate* operation is used to remove the duplications created as side effects of applying different combinations of object algebra operations. In our example dupeliminate will only leave one object of a subcollection that is identical to the result of s_5.

The final example in Figure 4 is the *nest* operation. *Nest* is used to manipulate sets of tuples by combining tuples to create set-valued attributes. Although not shown in the example, the *unnest* operation is mainly used after a project or ojoin operator to normalize multivalued attributes.

5 Object Algebra to SQL Translation

This section describes the process of translating object algebra queries over the global schema to SQL. The translation process is carried out by first performing an intermediate simplification process. The global to local schema mapping in Figure 1 is then used to finalize the mapping process. In the following sections, we provide an overview of the simplification and mapping processes.

5.1 The Simplification Phase

The simplification process is performed by translating the object algebra internal representation into an intermediate data structure. Most of the simplification process is concerned with simplifying nested queries. The object algebra operations can be nested in three different ways. Queries can be sequential, with one query using the result of another. One query can also be nested in another query as input. Finally, one query can have another query as part of its predicate or as part of its function. Throughout the simplification process, the following additional processes are performed: 1) elimination of object-oriented operations that are irrelevant to the SQL translation, and 2) analysis of path expressions to identify implicit join conditions that must be made explicit in the SQL form.

Since the purpose of the translation process is to create SQL queries from object algebra queries, we create an intermediate structure for the object algebra queries to assist in the translation process. The intermediate data structure, has the following form:

SPJ(Input_Collection, Projection_Collection, Joining_List, Predicate_list, Grouping_list, Distinct_Flag)

Local Schema A

Student	*ssn*, name, birthdate, street, city, state, zip, level, d_name
Course	*c_id*, description, credit_hours
Prerequisite	*c_num, prereq_cnum*
Offered_Courses	*c_num, dname*
Enrollment	*c_id, ssn*, grade
Faculty	*f_id*, name, salary, d_name
Teacher	*c_id, f_id*
Department	*d_name*, d_phone, d_chair

Local Schema B

Student	*ssn*, name, age, level, major, minor, advisor
Course	*c_num*, c_name, taught_by, status
Link	*ssn, c_num*, grade

Local Schema C

Graduate_Student	*ssn*, last_name, first_name, address, program, dname
Course	*c_name*, c_num, d_name, cred_hrs
Course_Taken	*ssn, c_name*, grade
Course_Taught	*ssn, c_name*
Committee	*ssn*, chair, member1, member2, subject
Department	*dname*, dphone, dchair

Figure 2: Relational Schemas for Local Schemas A, B, and C

SPJ indicates that the intermediate structure represents a basic select-project-join query. The select-project-join format of the intermediate structure therefore translates a sequence of object algebra operators into a form that is more amenable to translation to SQL. The *Input_Collection* is used to store the collection of classes to be joined. Similarly, *Projection_Collection* refers to the properties to be projected. The *Joining_List* stores join information extracted from path expression or from explicit joins such as those found in the *ojoin* operation. The *Predicate_List* includes the predicates and conditions that range over the *Input_Collection*. The *Grouping_List* is a list of all the properties that operations group on, such as those found in the *nest* operation. Finally, the *Distinct_Flag* is set to *true* to indicate that duplicate values should be eliminated (and set to *false* otherwise).

As an example of the simplification process, the following *image* query has a *select* operation as nested input:

$$
\begin{aligned}
s_1 &:= select(faculty, \lambda s\ s.sex = "M") \\
s_2 &:= image(s_1, \lambda s\ s.advisee) \\
s_3 &:= flatten(s2) \\
s_4 &:= ojoin(student, s_3, ST, FC, \lambda s\ \lambda f\ s.advisor =_0 f) \\
s_5 &:= project(s_4, \lambda s\ < (a_1, s.ST.sname), \\
&\quad (a_2, s.FC.name) >) \\
s_6 &:= dupeliminate(s_5, 1) \\
s_7 &:= nest(s_6, a2).
\end{aligned}
$$

Figure 4: Object Algebra Example

$$
\begin{aligned}
s_1 &:= image(select(student, \lambda s \\
&\quad s.major\_dept.dept\_chair.name = "JohnSmith"), \\
&\quad \lambda ss\ ss.sname)
\end{aligned}
$$

Since the above query represents a selection on the student class followed by a projection operation, the objective of nested input simplification is to collect the select and project information into a more generic form. As part of the simplification process, we are also confronted with the use of the path expression in the select operation. Path expressions represent implicit joins between classes where each entry in the joining list includes join class names between parenthesis followed by the joining property. Using the syntax of the intermediate data structure, the *image* operation above is simplified into the following form:

	s_1 := SPJ(
Input Collection:	{Student, Department, Faculty},
Select List:	{sname},
Joining List	{<(Student, Department), major_dept>, <(Department, Faculty), dept_chair>},
Predicate List:	{name = "John Smith"},
Grouping List:	{},
Disjoint Flag:	false).

The above structure indicates that the solution to the query involves a join between the Student, Department, and Faculty classes on the attributes indicated, with a selection on the chair name.

5.2 The Mapping Phase

After the object algebra queries are transformed into the intermediate data structure, mapping rules are applied to generate SQL queries. The mapping rules are used in conjunction with the global-to-local mappings described in Figure 1. In addition to the language conversion rules, the

mapping rules are used to identify which sites should receive different subparts of the query. The mapping rules use information from the SPJ representation together with the meta data to determine which relations and attributes should be used to generate the SQL queries. In some cases, ambiguities must be resolved when classes and properties from the object-oriented view can potentially map to more than one construct in the relational schemas. The joining list in the SPJ structure provides information that is used to resolve ambiguities. Inverse relationships in the meta data for the object-oriented view are also used to resolve ambiguities. Since space limitation do not allow the enumeration of the rules of the mapping process, an example is presented below to illustrate the different ways in which Figure 1 is used to support the mapping process. The complete algorithm for the mapping process can be found in [Adbe94].

Example: Give the names of all students majoring in computer science along with the names of their advisors.

$$
\begin{aligned}
stud\_fac &:= ojoin(Student, Faculty, S, F, \\
&\quad \lambda s\ \lambda f\ (s.major\_dept.d\_name = \text{"}cs\text{"}) \wedge \\
&\quad (s.advisor =_0 f)) \\
result &:= project(stud\_fac, \lambda s \\
&\quad < (Stud\_name, s.S.sname), \\
&\quad (Fac\_name, s.F.name) >)
\end{aligned}
$$

The internal structure of the above operation is the following:

result := SPJ(
Input Collection: {Student, Faculty, Department},
Select List: {sname, name},
Joining List: {<(Student, Department), major_dept>,
<(Student, Faculty), advisor>},
Predicate List: {d_name = "cs"},
Grouping List: {},
Distinct Flag: false)

SQL query for local relational database A:
The SQL query for database A cannot be generated because there is no advisor information in this relational database.

SQL query for local relational database B:

$$
\begin{aligned}
result2 := \ & select \quad name, advisor \\
& from \quad Student \\
& where \quad major = \text{"}cs\text{"};
\end{aligned}
$$

SQL query for local relational database C:

$$
\begin{aligned}
result3 := \ & select \quad g.first\_name, g.last\_name, c.chair \\
& from \quad Graduate\_Student\ g, Committee\ c \\
& where \quad g.ssn = c.ssn\ and\ g.dname = \text{"}cs\text{"}
\end{aligned}
$$

The query sent to LCSB shows that only the Student relation is needed to retrieve the information needed for both student and advisor names. Since Student in the GCS maps to the Student relation in LCSB and Department in the GCS maps to either the *major* or *minor* attributes in the Student relation, the *major_dept* property in the joining list of the SPJ representation is used to resolve the mapping to the *major* attribute in the Student relation (and thus no join is required in LCSB). In a similar manner, since the Faculty class in the GCS maps to either the *taught_by* or the *advisor* attributes is LCSB, the *advisor* attribute in the joining list is used to resolve the mapping to the *advisor* attribute in the Student relation.

The query sent to database C uses the information in the Committee relation to get advisor information since there is an explicit mapping from the *advisor* property in the GCS to the *chair* attribute of the Committee relation in LCSC. The mapping information within the relational component for database C further indicates that Committee and Graduate_Student are joined based on the *ssn* foreign key in the committee. Note that an explicit join between Student and Department is not required since there are no attributes to project from the Department relation. The selection on the department name can occur directly within the Student relation. This example illustrates that using the same generic structural mapping in Figure 1, the queries generated for individual sites vary according to the specific mapping instances that exist for each local schema.

6 Summary and Conclusions

This paper has presented an approach to query translation in a multidatabase environment that uses an object-oriented model as a global conceptual schema together with an object algebra for expressing queries over the global view. A unique aspect of this work is that the object-oriented model is used as a self-describing model to store information about the global schema. The object-oriented model is also used to describe the relational schemas to which the global schema maps, including the mapping relationships. Simplification and mapping rules are then used to transform queries over the object-oriented view of the data into appropriate queries over the relational schemas. The advantage of this approach is that the object-oriented algebra used within this work can be used to retrieve data directly from the object-oriented database *or* to transparently retrieve data from the local schemas using the object-oriented view.

As future research, we are expanding the translation process to support other types of local schemas, such as network schemas. Another direction involves more extensive examination of the object identity problem for the case where information about an object may exist in several local databases. Our longer term objectives are to develop a complete environment that uses an object-oriented database system as a blackboard for temporary storage of data retrieved from local databases.

Acknowledgments

This research was partially supported by a grant from Bull Worldwide Information Systems.

References

[Abde94] Abdellatif, T. B. , *A Query Transformation Subsystem to Resolve Semantic Heterogeneity*, Dept. of Comp. Sci. and Eng., Arizona State Univ., to be completed, Spring 1994.

[Belc88] Belcastro, B., Potkowski, V., Kaminski, A., Kowalewski, W., Mallamaci, M., Mezyk, C. L., Mortardi, S., Scorocco, T., Staniszkis, F. P., and Turco, G., "An Overview of the Distributed Query System DQS", *Proc. Int. Conf. on Extending Database Technologies*, Venice, Mar. 1988, *in Lecture Notes in Computer Science*, vol, 303, Springer-Verlag, NY, pp. 170-189.

[Bert91] Bertino, E., "Integration of Database Systems Using an Object-Oriented Approach," in *Proc. of the 1st Int. Workshop on Interoperability in Multidatabase Sys.*, Kyoto, Apr., 1991, pp. 22-29.

[Brei86] Breitbart, Y., Olson, p., and Thompson, G., "Database Integration in a Distributed Heterogeneous Database System," *Proc. of the 2nd Int. Conf. on Data Eng.*, Los Angeles, Feb. 1986, pp. 301-310.

[Cast91] Castellanos, M., Saltor, F., and Garcia-Solaco, M., "A Canonical Model for the Interoperability among Object-Oriented and Relational Databases," Ozsu, Dayal, and Valduriez Ed., *Int. Workshop on Distributed Object Management*, Edmonton, Canada, Aug. 1992, pp. 365-370.

[Coll91] Collet, C., Huhns, M. N., and Shen, W. M., "Resource Integration Using a Large Knowledge Base Carnot", *Computer*, Dec. 1991, pp. 55-62.

[Doyl91] Doyle, W. and Kerschberg, L., "An Intelligent Heterogeneous Autonomous Database Architecture for Semantic Heterogeneity Support," *Proc. 1st Int. Workshop on Interoperability in Multidatabase Sys.*, Kyoto, Japan, Apr. 1991, pp. 152-155.

[Kali90] Kalinichenko, L., "Methods and Tools for Equivalent Data Model Mapping Constructs," *Advances in Database Technology - EDBT '90*, in *Lecture Notes in Computer Science*, Springer-Verlag, Germany, no. 416, pp. 92-119.

[Land82] Landers, T. and Rosenberg, R., "An Overview of Multibase," *Distributed Databases*, H. J. Schneifer, Ed., North Holland, The Netherlands, 1982, pp. 153-184.

[Lena90] Lenat, D. and Guha, R. V., "Building Large Knowledge-Based Systems: Representation and Inference in the Cyc Project, " Addison-Wesley, Reading, Mass., 1990.

[Litw86] Litwin, W. and Abdellatif, A., "Multidatabase Interoperability, "*IEEE Computer 12*, Dec. 1986, pp. 10-18.

[ONTO92] *ONTOS Reference Manual*, Release 2.2, ONTOS, Inc., 1992.

[Rusi89] Rusinkiewicz, M., et al., "OMNIBASE: Design and Implementation of a Multidatabase System," *Proc. of the 1st Int. Conf. on Parallel and Dist. Processing*, Dallas, TX, May 1989, pp. 162-169.

[Rous85] Roussopoulos, N. and Mark, L., "Schema Manipulation in Self-Describing and Self-Documenting Data Models, "*Int. Journal of Computer Sciences*, 1985, pp. 1-26.

[Salt91] Saltor, F., Castellanos, M., and Garcia-Solaco, M., "Suitability of Data Models as Canonical Models for Federated Databases," *SIGMOD Record*, Dec. 1991, pp. 44-48.

[Shaw90] Shaw, G. M. and Zdonik, S., "A Query Algebra for Object-Oriented Databases," *Proc. of the 6th Int. Conference on Data Eng.*, Los Angeles, Feb. 1990, pp. 154-162.

[Shet90] Sheth, A. P. and Larson, J. A., "Federated Database system for Managing Distributed, Heterogeneous, and Autonomous Databases, "*ACM Computing Surveys*, vol. 22, no. 3, Sep. 1990, pp. 183-236.

[Ship81] Shipman, D. "The Functional Data Model and the Data Language DAPLEX," *ACM Transactions on Databases Sys.*, Mar. 1981, pp. 140-173.

[Spac82] Spaccapietra, S., "An Approach to Effective Hetero- geneous Database Cooperation,"*Distributed Data Sharing Systems*, Van de Reit and Litwin (ed.), North-Holland, The Netherlands, 1992, pp. 209-218.

[Temp87] Templeton, M. Brill, D. Dao, S. Lund, E. Ward, P. Chen, A. and MacGregor, R., "Mermaid-A Front-End to Distributed Heterogeneous Databases,"*Proc. of the IEEE*, May 1987, pp. 695-708.

[Urba91a] Urban, S. D. and Wu, J., "Resolving Semantic Heterogeneity Through the Explicit Representation of data Model Semantics , "*SIGMOD Record*, Dec. 1991, pp. 55-58.

[Urba91b] Urban, S. D., "A Semantic Framework for Heterogeneous Database Environments," *Proc. of the 1st Int. Workshop on Interoperability in Multidatabase Sys.*, Kyoto, Japan, Apr. 1991, pp. 245-252.