Interoperable Query Processing with Multiple Heterogeneous Knowledge Servers[1]

Louiqa Raschid and **Yahui Chang** and **Bonnie J. Dorr**
Institute for Advanced Computer Studies
University of Maryland
{louiqa | yahui | bonnie }@umiacs.umd.edu

Keywords: heterogeneous knowledge bases, cooperation in heterogeneous systems, interoperable query processing, information mediation, canonical representations, F-logic rules.

Abstract: This paper describes a technique for information mediation when multiple heterogeneous knowledge and data servers are to be accessed during query processing. One problem is building an intelligent interface between each knowledge server (KS) and its processor (KP); and the second is to provide interoperability among multiple KP/KS so that a query may be answered using information from multiple sources. We present example scenarios which highlight these problems and then outline query mapping and transformation techniques that are applicable. The techniques for solving the interoperability problems involve representations in some canonical form. This includes a canonical representation (CR) corresponding to each KP/KS pair and a merged CR (MCR) to represent the mapping among the CRs. The MCR and CRs include relevant information obtained from a source query, and heterogeneous mapping (het-map) information, for all possible mappings among the multiple servers. The knowledge in the canonical form must be represented so that it can be easily during query transformation. We use an example of translating queries from an object schema to a relational schema to illustrate typical knowledge that must be represented in some canonical form. We use a high level logical language, F-logic, to represent the heterogeneous mapping (het-map) and query transformation information as a set of declarative rules, in the canonical form.

1 Introduction

The task of combining knowledge from multiple heterogeneous data and knowledge servers places great demands on the interface between the query processors and the knowledge servers. Intelligent interfaces must support interoperability during query processing, so that information may be shared among the multiple servers, in a manner that is transparent to the users.

In this paper, we propose some techniques to solve this problem and outline some tasks that must be accomplished. Our approach is based on using an appropriate canonical representation (CR) to capture the information relevant to the task of providing interoperability during query processing. The canonical form must be able to represent the information needed to provide a mapping among the servers and to produce transformed queries. This canonical form must be independent of the query languages and the data models in use by the multiple servers.

We address two problems of *information mediation* (IM) that are encountered when combining knowledge from heterogeneous knowledge servers and providing interoperable query processing: (1) intelligent query processing within the context of a single system, *i.e.*, within one Knowledge Processor / Knowledge Server (KP/KS) pair; and (2) intelligent and interoperable query processing across multiple KP/KS. We refer to figure 1 to illustrate these two problems.

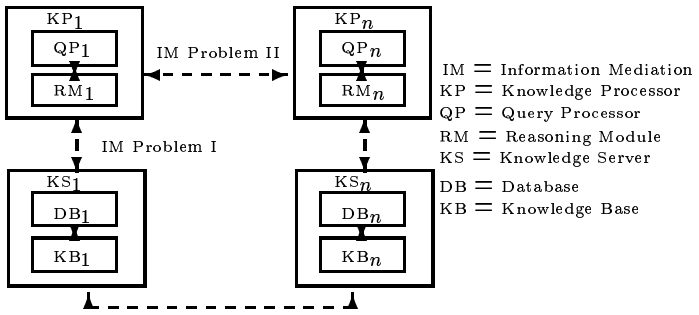


Figure 1: An Architecture for Information Mediation with Multiple Heterogeneous Servers

The first information mediation problem is to provide an intelligent interface that accepts a query from a user and to modify this query to one that is more informative and/or more appropriate for the particular knowl-

[1]This research has been partially supported by the National Science Foundation under grant IRI-9120788, by the DARPA Basic Research Program under grant N00014-92-J-1929, by the Army Research Office under contract DAAL03-91-C-0034 through Batelle Corporation, and by the Army Research Institute under contract MDA-903-92-R-0035 through Microelectronics and Design, Inc.

edge server. The user may be unaware that the query is inappropriate, or that there is a mismatch between the query and the information encoded in the corresponding KS. This problem is compounded when the user poses queries against one of several heterogeneous servers. We resolve this mismatch by constructing a *canonical representation* (CR) that is built through assimilation of information for each KP/KS pair. This representation is used to provide a modified version of the query that is more appropriate, allowing intelligent retrieval. This problem is analogous to knowledge acquisition; in this research context, we are achieving knowledge acquisition by constructing a precompiled CR, through assimilation of information from each KS.

The second information mediation problem is providing an interface to a number of knowledge servers. To pose a query correctly, the user must be aware of all possible distinctions between individual KP/KS pairs. This includes knowledge of the different query processing languages, the representation models of the different knowledge servers and the different schema. In addition to these syntactic differences, the user must also be aware of the semantic differences that may exist among the KP/KS pairs. These differences may stem from mismatched representations and generalizations, inconsistencies in representation, equivalent representations or other forms of dependencies. We discuss techniques for mapping a user's query from one system to the information from the knowledge base of another system, to produce a modified query. In order to achieve this mediation, so that query processing is interoperable, and knowledge is shared, we must take advantage of the knowledge in each CR and capture the heterogeneous mapping information among multiple CRs in a *merged canonical representation* (MCR).

Query processing techniques have not always been successful in building intelligent interoperable interfaces. Information is often stored as *ad hoc* rules and may be used when answering queries but the user has no insight into how or why the rules were used. The canonical form formalizes the knowledge needed to modify queries or generate new queries. We use an example of translating queries from an object schema to a relational schema to illustrate typical knowledge that must be represented in some canonical form. Modern theories in the field of machine understanding and translation of language bring much to bear on the issues that are relevant to developing a mapping between the input query representation (KP) and the underlying data/knowledge base representation (KS), (and *vice versa*). The development of a single merged canonical representation for the current problem is analogous to the task of developing an *interlingua* or language-independent form by machine-translation researchers (see, *e.g.*, Dorr (1987, 1990, 1993)). Thus, the techniques we develop for solving these problems have been influenced from techniques that have been successfully applied in these fields of research.

There are several components involved in providing an interoperable query processing capability. One important task is extracting semantics from some source query, (after the query is resolved against the particular schema for which it was formulated) and representing this extracted information in some canonical form. In addition, information extracted from each of the schemas (and their data models) must also be represented in some canonical form. An important component of this information is the heterogeneous mapping (het-map) information. In its simplest form, this mapping information provides a structural mapping among entities in the canonical form. If we consider the particular case of mapping from an object database, into a relational database, then het-map must provide, at the least, for each attribute of an object, the corresponding relation from which this attribute may be retrieved.

The canonical representation will thus include relevant information obtained from a source query, and information on possible mappings among multiple heterogeneous schema. This knowledge must be represented so that it can be easily used during query transformation. We propose using a high level logical language, F-logic, to represent this information as a set of declarative rules, in the canonical form.

This paper is organized as follows: section 2 presents example scenarios for the two problems in providing an intelligent interface and providing interoperability among multiple knowledge servers. Section 3 introduces the use of a canonical representation (CR) and a merged canonical representation (MCR) to solve these two problems. Section 4 uses an example of mapping objects and attributes between an object schema and equivalent relational schema to illustrate the heterogeneous mapping (het-map) information that must be represented in the MCR in some canonical form. Section 5 uses an example XSQL-like query to illustrate the knowledge that must be extracted from a query and represented in a canonical form. Section 6 uses a high-level language F-logic (as formulated by Kifer and Lausen (1989) and Kifer *et al.* (1990)) to represent the heterogeneous mapping (het-map) and query transformation information, as a set of declarative rules, in the canonical form. The approach is summarized in section 7. We compare our work with related research in the area, and we discuss future research.

2 Examples of Transforming Queries from Object to Relational Schema

As an example of our approach to query transformation, we use a sample object schema (appendix A) and three *equivalent* relational schema (appendix B) to provide examples of query transformation, going from object schema to relational schema. In the object schema of appendix A, each node is an object. The broken arcs represent an inheritance hierarchy among the objects, and the direction of the arrow represents a class to sub-class relationship. A solid arc represents a named attribute of some object which is itself another object. For example, an arc (labeled LeasedVehicle) between the objects Company and Vehicle, corresponds to LeasedVehicle, an attribute of each object Company, referring to some object Vehicle.

Since the schema contain equivalent information, we

can focus on the problem of extracting the relevant information from the source query and using relevant mapping information from the different schema to produce a transformed query. The source query for the object schema is an XSQL-like query, i.e., an extension of SQL to access object data bases. The XSQL queries we use are fairly simple and only differ from SQL queries in the query path expressions in the where clause. We will describe them briefly and refer the reader to Kifer *et al.* (1992) for details of the language. An XSQL query takes the following form, where X, Y and Z are objects Company, GasolineBoat and Company, respectively:

Query 1

Select Z
from Company X, GasolineBoat Y, Company Z $\cdots$
where X.LeasedVehicle[Y].Manufacturer[Z] $\cdots$

This query, when applied to the object schema of appendix A, selects all objects Z, of Company, such that each object X, of Company, has an attribute LeasedVehicle which is an object Y (specified in the **where** clause). Further, each Y is a GasolineBoat (specified in the **from** clause), and the object to be retrieved, Z, is a Company which is the Manufacturer of each GasolineBoat Y (specified in the **where** clause). As seen in the object schema of appendix A, GasolineBoat does not have an attribute Manufacturer and has to inherit this from the class Vehicle, in the related class hierarchy.

There are three relational schemas in appendix B. In RDBMS 1, relations R1 and R2 have all the Company and YachtClub information, R3 has all the Auto information, (DieselAuto and GasolineAuto) and R4 has all the Boat information (DieselBoat and GasolineBoat). In RDBMS 2, R1 and R2 are as before. R3 has the Model and Manufacturer information on each Vehicle (DieselBoat, DieselAuto, GasolineBoat, GasolineAuto) and R4 and R5 have the remaining information on Auto and Boat, respectively. RDBMS 3 differs from RDBMS 2 in that the Model and Manufacturer information in R3 is not identified based on the VehicleId information, but is identified based on Engine information. The three schemas are fairly straightforward representations for the object schema. However, the subtle differences in their structure must be reflected in the appropriate mappings, to generate equivalent queries.

To obtain an SQL query in RDBMS 1, which is equivalent to *Query 1*, for those manufacturers of GasolineBoat leased by some company, we use the knowledge that the VehicleId values in relation R1 (for the leased vehicles) may correspond to either boats or automobiles. These VehicleId values refer to information in R4, which includes Manufacturer information on both GasolineBoat and DieselBoat. We must select only those tuples in R4 corresponding to GasolineBoat. This is indicated by the value of the Type attribute. To obtain the manufacturer information we need relation R4 which has only boats. An equivalent SQL query would be as follows:

Query 2

Select Manufacturer
from R1 R4
where R1.LeasedVehicle = R4.VehicleId and
R4.Type = GasolineBoat

To obtain a query equivalent to *Query 1* in RDBMS 2, we use the knowledge that the VehicleId values in R1 may be either boats or automobiles. To identify the boats, we refer to R5 which has information on GasolineBoat and DieselBoat. However, R5 does not have any Manufacturer information. Manufacturer information is in relation R3, but this relation includes information for both boats and automobiles. An equivalent query would be as follows:

Query 3

Select Manufacturer
from R1 R3 R5
where R1.LeasedVehicle = R5.VehicleId and
R5.Type = GasolineBoat
and R5.VehicleId = R3.VehicleId

The difference between RDBMS 2 and RDBMS 3 is that the Manufacturer information in R3 is not identified based on the VehicleId but on the Engine. This will change just the **where** clause in *Query 3* such that relations R3 and R5 are joined on Engine, and we do not give this query here. Suppose we now consider a query that is structurally similar to *Query 1*, wrt the object schema, as follows:

Query 4

Select Z
from YachtClub X, GasolineBoat Y, Company Z $\cdots$
where X.OwnedVehicle [Y].Manufacturer [Z] $\cdots$

Although *Query 4* is similar to *Query 1*, from the object schema of appendix A, we know that all OwnedVehicles must be in GasolineBoat. Consequently, VehicleId values in relation R2 refer only to GasolineBoat and we can use these values to directly identify the Manufacturer information from R3. Thus, in RDBMS 2, unlike the previous *Query 3*, we do not need R5 to identify the GasolineBoat, and we can obtain Manufacturer from just R2 and R3 as follows:

Query 5

Select Manufacturer
from R2 R3
where R2.LeasedVehicle = R3.VehicleId

However, in RDBMS 3, although we have the knowledge that the VehicleId values in R2 refer only to GasolineBoat, in R3, the Manufacturer information cannot be obtained based on the value of the VehicleId. We need the Engine information for the GasolineBoat. We use the VehicleId values in R2 to identify GasolineBoat in R5, and use the Engine information to obtain Manufacturer information from R3. Thus, we cannot use the knowledge that all OwnedVehicles are in GasolineBoat, during translation. The corresponding query will be very similar to *Query 3*, and is as follows:

Query 6

Select Manufacturer
from R2 R3 R5
where R2.LeasedVehicle = R5.VehicleId and
R5.Engine = R3.Engine

Although the translation to obtain the SQL queries appears to be straightforward, it is clear that the translation process must identify the relevant knowledge from (1) the source XSQL queries, (2) the object schema (3) the equivalent relational schema, and (4) the mapping information between these schema. Our research addresses the problem of defining an appropriate canonical representation which will assist in the task of obtaining this

KS$_1$	**Project**
	time frame (completion dates, schedules)
	personnel needed (skills, experience)
	cost estimates (to charge for a project)
	components needed (may be recursively defined)
	component usage
KS$_2$	**Personnel Employment**
	skills
	salary
	current work
	availability
	Personnel Work History
	experience (historical)
	salary (historical)
	skills
	availability
KS$_3$	**Components**
	cost estimates (for purchase)
	availability (inventory, location)
	METHOD reliability (dependent on usage, time frame, project,..)
	subcomponents (may be recursively defined)
	METHOD equivalent components (dependent on usage, project, reliability ...)

Figure 2: Information Encoded in Three Knowledge Servers

knowledge from the queries and schemas as well as be in a form that supports the translation process. The importance of such a canonical representation increases, as the queries and the mappings increase in representational power.

We now consider a different and more complex example of sharing information among three knowledge servers, KS_1, KS_2 and KS_3, shown in figure 2. The servers provide information on Projects, Personnel and Components, respectively. We do not make the assumption that the servers share equivalent knowledge. Thus, unlike in the previous example, there is no direct mapping among equivalent objects; however, mapping information is needed to facilitate information sharing.

Suppose KP_1 requests specific information on the reliability of a single component. KP_3 would return one or more reliability figures. However, the KP_3/KS_3 pair may have the knowledge that there is a dependency between these reliability figures and the particular *usage* of this component, in a project in KS_1. Another possibility is that these reliability figures vary as components are updated, and thus depend on the *time frame* of a project in KS_1. Ideally, we would return a modified query that indicates that the *usage* in a project or the *time frame* is needed from KS_1, to obtain the appropriate reliability figures.

Suppose a query posed by KP_1/KS_1 attempted to identify the employees needed to staff a particular project. This information must be obtained from the KP_2/KS_2 pair. There are two ways of identifying the personnel here, one based on their work history and one based on their present employment. If the personnel are identified on the basis of their skills in the Personnel Work History relation in KS_2, then for these employees one must determine their employment status, availability and salary. This may require further access to the Personnel Employment relation in KS_2. Further, their availability may be dependent on the status of some project on which they are currently employed, as specified in KS_1. This form of knowledge, used to modify the queries, does not belong in the category of structural mappings between equivalent objects in different schemes, as was the case in the previous example. However, this information must be represented in some canonical form for use during query transformation. Although this paper does not address how such information is represented, we included this example to describe the variety of information that must be encoded in some canonical representation.

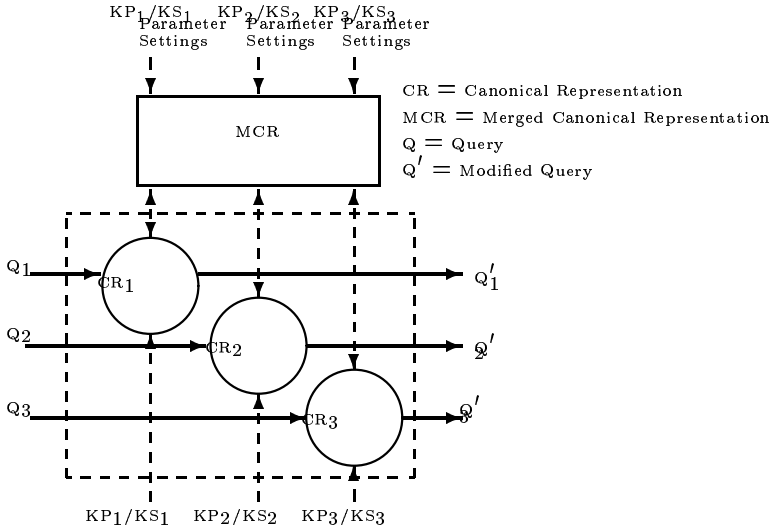


Figure 3: Query Mapping and Transformation Using Canonical Representations

3 Using Canonical Representations in Query Transformation

Assuming that some appropriate canonical form has been identified, Figure 3 illustrates how the individual CRs and the composite MCR (obtained from the merging of possible mappings among the individual CRs) are used. The simpler problem involves a query to a single knowledge server KS_i and so uses information from a single canonical representation CR_i serving that KP_i/KS_i pair. It involves a mapping specified along some appropriate horizontal axis of figure 3. The result of transforming the query using the CR_i is a modified query, Q_i'.

The solution to the problem of interoperable query processing and knowledge sharing involves the mapping into, and out of, the MCR specified along the vertical axis of figure 3. The input is a query Q_i which is understood by a single CR_i that represents a KP_i/KS_i pair. However, answering this query requires understanding and accessing information from the other KP/KS pairs via the MCR. Thus, we generate appropriate query forms for the other KP/KS pairs, using the knowledge in the

MCR and the corresponding CRs. The output is a modified query, Q'_i, relevant to the particular pair, KP_i/KS_i, as well as other queries Q'_j, Q'_j, Q'_k, *etc.* relevant to the other CRs. Thus, to provide interoperability, we make use of the heterogeneous mapping (het-map) information in the MCR as well as the CRs. The advantages of using a CR and a merged CR (MCR) are that it plays the dual role of capturing the information needed for producing modified queries and that it specifies how to utilize this information to transform the queries. Typically, this information may be present as a set of *ad hoc* rules, and used during query processing. However, the user would have no knowledge of the reasons that these rules were applied and exactly why and which modified query was produced and answered.

In the rest of this paper, we will use several examples to illustrate the type of mapping information that must be captured in the MCR when equivalent information is represented among multiple schemas. We also discuss the knowledge that must be obtained from a source query, relevant to the process of query transformation. We then illustrate how a high level logical formalism (F-logic) may be used to develop such a canonical representation.

4 Mapping Information for Query Transformation Among Heterogeneous Models

There are several components involved in the query transformation process. Clearly, one important task is extracting semantics from some source query that is to be transformed, into a representation in some canonical form. The query must be resolved against the particular schema for which the query is formulated. This will be discussed in a later section, making use of an XSQL-like query. In addition to the semantics that is extracted from a query, there is a wealth of information that must be extracted, for each of the schemas (and their data models), which must also be represented in some canonical form. An important component of this information is the heterogeneous mapping (het-map) information. In its simplest form, this provides a structural mapping among entities in the canonical form. If we consider the particular case of transforming an XSQL-like query, against an object database, into an SQL-like query against a relational database, then this heterogeneous mapping (het-map) will provide, at the least, for each attribute of an object, the corresponding relation from which this attribute may be retrieved.

Clearly, the exact nature of the heterogeneous mapping information is largely determined by the different data models. One of the goals of our research is to determine an appropriate canonical representation so that this information may be represented, independent of the particular data models. In this paper, however, we will limit ourselves to an object model and a relational model, and use the example schema to identify example mappings and the rationale. We do not imply that these examples are complete. We describe this heterogeneous mapping (het-map) information in English, and in a possible canonical form (based on a high-level logic) in a later section.

Case 1	Transformation depends on start object S0	
	Case 1.1	Use S0, attribute to produce a list as follows: {(relation name RN, attribute name AN)}
	Case 1.2	Same as 1.1 except that S0 is also used to identify a subset of tuples of relation RN. Produce a list {(rel name RN, attr name AN, selection criterion for RN based on S0)}
Case 2	Transformation depends on end object E0	
	Case 2.1 Does not depend on S0.	Use E0, attribute to produce a list as follows: { (relation name RN, attribute name AN)}
	Case 2.2 Also depends on start object S0.	Same as 2.1 except that S0 is also used to identify a subset of tuples of relation RN. Produce a list {(rel name RN,attribute name AN, selection criterion for RN based on S0)}
	Case 2.3 Does not depend on S0 subset of tuples of RN.	Same as 1.2 except that E0 is used instead of S0 to identify a subset of tuples of RN. Produce list {(RN, AN, selection criterion for RN based on E0)}
	Case 2.4 Also depends on start object S0.	Same as 2.3 except that S0 is also used to identify a subset of RN. Produce list {(RN, AN, selection criterion for RN based on S0 selection criterion for RN based on E0)}
Case 3	Transformation depends on path objects PO in path between S0 and E0	
Case 4	Most general case where attribute of object is an object	

Figure 4: Cases for Mapping Between an Object Schema and a Relational Schema

Each of the following cases corresponds to a different scenario for mapping between an object schema and a relational schema. The scenarios are straightforward. In each case, we assume that in the object schema, we are interested in obtaining values for some attribute corresponding to a particular object, a *start object* (SO). If the attribute is not actually structurally associated with the start object, then it must be associated with an *end object* (EO) and there will be a class hierarchy associating these two objects. Finally, we may also refer to *path objects* (PO) occurring in the path between the start

and end objects. The task of extracting the SO, EO and PO information from an example XSQL-like query is discussed in the next section. In the corresponding relational schema, the heterogeneous mapping information must identify the appropriate tuples of one (or more) relation(s) and the corresponding attribute(s) involved. As we mentioned earlier, we only consider the simple case of a single start object (SO) and attribute. Extensions to include multiple attributes, and combining information among multiple objects, when the attribute of an object is itself an object is discussed by Raschid *et al.* (1993).

Figure 4 summarizes the different cases that are to be discussed in detail in this section.

Case 1 – Depends on SO

In this scenario, the organization of the corresponding relational schema dictates that the transformation is dependent only on the start object SO, (and is independent of the end object EO or path objects PO, if any). Thus, we obtain the correct mapping directly from the (start object, attribute) pair. Consider an example pair from the object schema of Figure 2, say (DieselAuto, Model). There are two possibilities, as follows:

Case 1.1 – Depends on SO

In this first case, the het-map uses (SO, attr-SO) to obtain a relation name RN and an attribute name AN. The attribute AN must be an attribute of the relation RN. All tuples in RN correspond to the start object SO, and attribute AN has the values for attribute attr-SO, e.g., the values for Model for DieselAuto. For example, if there is a relation containing information on all diesel autos, then the het-map transformation, may produce the following relation and attribute pair:

(RN ≡ RelDieselAuto, AN ≡ AttrModel)

Instead of a single pair, het-map may produce a list of {(RN, AN)}. For example, if the information on diesel autos were stored in separate relations, for hatchbacks and sedans, this would produce the following:

{(RN ≡ RelDieselSedan, AN ≡ AttrModel), (RN ≡ RelDieselHatchback, AN ≡ AttrModel)}.

Case 1.2 – SO provides an additional constraint

In this second case, the het-map information for (SO, attr-SO) provides a relation RN, and an attribute AN of that relation RN, corresponding to the attribute attr-SO. However, all tuples in RN do not correspond to the start object SO. Thus, SO provides an additional selection criterion, to select those tuples in relation RN which actually correspond to SO. Attribute AN-SO is a second attribute of relation RN and tuples of RN which are in a range of specified values for this attribute, AN-SO, correspond to the tuples for object SO. The attribute values of AN then correspond to the values for attribute attr-SO, for the selected tuples in RN.

Suppose DieselAuto and DieselBoat are stored together in a single relation DieselVehicle. Then, an example of the het-map transformation is to produce the following:

(RN ≡ RelDieselVehicle, AttrAN-SO, range-of-values(for DieselAuto) in AttrAN-SO
= {DieselAuto}, AN ≡ AttrModel)

Suppose instead that all DieselAuto objects are actually identified by whether they are hatchbacks or sedans in the relation DieselVehicle. Then het-map will produce the following:

(RelDieselVehicle, AttrAN-SO, range-of-values(for DieselAuto) in AttrAN-SO
= {HatchBack, Sedan}, AttrModel)

Case 2 – Depends on EO

For this scenario, the het-map information will indicate that the transformation is dependent on the end object, EO. Suppose we consider the start object, attribute pair, (DieselAuto, Model). The attribute Model is actually structurally associated with the end object (EO) Vehicle. The het-map information may indicate that the relational schema is such that the corresponding relations and attributes are dependent on (Vehicle, Model), as opposed to (DieselAuto, Model). This is the scenario in Cases 2.1 and 2.3. Further, it is possible that the SO, DieselAuto, is a further constraint on the relation (indicating further selection of some appropriate tuples which are actually diesel autos). This is the situation in Cases 2.2 and 2.4.

Case 2.1 – Depends on EO

This is similar to Case 1.1, and het-map will use the (EO, attribute) pair of (Vehicle, Model), instead of the (SO, attribute) pair of (DieselAuto, Model), to obtain a list of relation attribute pairs, {(RN, AN)}, for example, (RelVehicle, AttrModel). Such a situation will result when all the vehicles in the Vehicle relation also happen to be diesel autos, and so there is no necessity to select only a subset of those tuples from the Vehicle relation that correspond to DieselAuto.

Case 2.2 – SO provides an additional constraint

This corresponds to the SO, say Auto, being an additional constraint, in a variation of the previous case. In other words, the relational schema is such that the relation and attribute names are determined based on the EO, for example Vehicle, but all the tuples in this relation do not necessarily correspond to the SO, Auto.

Suppose we consider the set of possible path objects between a start object, say Auto and an end object, say Vehicle. They are as follows:

< Auto, DieselAuto, DieselVehicle, Vehicle >

< Auto, GasolineAuto, GasolineVehicle, Vehicle >

The het-map will use (Vehicle, Model) and (Auto, Model) and produce the following, indicating that only some of the tuples in the relation correspond to the SO, Auto. Auto provides an additional selection criterion based on the values of some attribute in the relation, as follows:

(RelVehicle, AttrAN-SO, range-of-values(for Auto) in AttrAN-SO
= {DieselAuto, GasolineAuto}, AttrModel).

Case 2.3 – EO provides additional constraint

This is similar to Case 1.2, except that the end object EO is used to identify the relation and attribute. However, all tuples in the relation do not correspond to EO, and there is an additional selection constraint on that relation, to identify the tuples corresponding to EO. Thus, the het-map transformation for the (EO, attribute) pair will produce a list of
{(RN, AN-EO, range-of-values(for EO) in AN-EO, AN)}.

Suppose we consider a situation where all the Vehicle data are stored with other objects, in a relation AllItems. There is an attribute which has an identifying value, say Vehicle, to identify all vehicles in the relation AllItems. In addition, all the vehicles in that relation happen to be autos. An example is as follows:

(RelAllItems, AttrAN-EO, range-of-values(for Vehicle) in AttrAN-EO = {Vehicle},
AttrModel).

Case 2.4 – Both SO and EO provide additional constraint

This case is similar to 2.3, in that het-map uses the EO and attribute to obtain a relation and attribute pair. In addition, there is a selection constraint which restricts some of the tuples of the relation, based on an attribute, AN-EO, to identify all the tuples corresponding to the EO. Finally, the start object, SO, also provides an additional selection constraint, based on attribute AN-SO, to identify the tuples corresponding to SO. Het-map will produce the following:

{(RN, AN-EO, range-of-values(for EO) in AN-EO, AN-SO, range-of-values(for SO) in AN-SO, AN)}.

This corresponds to the case where all Vehicle data are stored in a relation AllItems with other items. An attribute with value Vehicle identifies the vehicles in this relation. However, all the vehicles do not correspond to the object DieselAuto. We need to identify the tuples corresponding to objects that are in both Vehicle and DieselAuto. To explain the need for two selection constraints based based on EO and SO, note that there may be multiple class hierarchies associated with Vehicle. DieselAuto may only participate in one such hierarchy. Similarly, DieselAuto may participate in other hierarchies that do not include Vehicle. An example is as follows:

(RelAllItems, AttrAN-EO, range-of-values(for Vehicle) in AttrAN-EO = {Vehicle},
AttrAN-SO, range-of-values (for Auto) in AttrAN-SO = {DieselAuto, GasolineAuto},
AttrModel).

Case 3

This is the most general case, and we may consider the previous cases to be degenerate cases of this case. In this scenario, het-map uses the start object SO, the end object EO, and any of the path objects, $P_1 \cdots P_n$ in the transformation. For each pair of elements in this path, het-map may use the pair of objects to obtain a transformation corresponding to combining (joining) pairs of relations.

The previous Case 1 will be a degenerate case where het-map ignores the path objects and only uses the start object SO and attribute to produce a transformation on an individual relation. Similarly, the previous Case 2 is also a degenerate case where het-map ignores the path objects and uses the end object EO and attribute to produce a transformation on an individual relation, with possible additional constraints provided by SO. Finally, Case 3 will consider all or some of the path objects PO and produce a transformation which will involve combining multiple relations. Case 3 may also be applied in conjunction with Case 2 as will be seen.

For each object in a pair of objects in the path between the start and end object, say P_i and P_j, het-map may produce a pair (RN, AN-Join) or a quadruple (RN, AN-Sel, range of values for AN-Sel, AN-Join). In the case of a pair, the relation RN_i for P_i and RN_j for P_j are joined over the corresponding attributes AN-Join. If it is a quadruple, then the range of values for AN-Sel is used to select some tuples from each relation RN_i and RN_j, after which the two relations are joined. The het-map must also consider the possibility that this joining path information may be incomplete wrt each pair in the path in which case objects in the path must be skipped in order to form the next pair. Consider the following paths between Auto and Vehicle in a class hierarchy:

< Auto, DieselAuto, DieselVehicle, Vehicle >

< Auto, GasolineAuto, GasolineVehicle, Vehicle >

There will be joining path information for each pair in each path only in the case where information on each of the (four) objects in each path are stored separately in individual relations. Now, het-map will produce three transformations representing joining path information for each path. However, suppose that information on DieselAuto and GasolineAuto is combined and stored together, grouped by the autos being either Hatchback or Sedan. The information on both DieselBoat and DieselAuto is also combined and stored in a relation DieselVehicle. The information on Vehicle is in a relation AllItems.

For the pair (Auto, DieselAuto) in the join path, there may be no joining path information since these these do not correspond to individual relations. Het-map then skips this first pair in the path and considers the pair (Auto, DieselVehicle) in the join path. This may produce the following pair where the first element is a list of 2 elements:

{(RelHatchback, AttrAN-Sel, range-of-values(for Auto) in AttrAN-Sel = {DieselAuto},
AN-Join ≡ AttrDAId),

(RelSedan, AttrAN-Sel, range-of-values(for Auto) in AttrAN-Sel = {DieselAuto},
AN-Join ≡ AttrDAId) }

(RelDieselVehicle, AN-Join ≡ AttrDAId)

Here, in the resulting query, some subset of tuples are selected from the relations RelHatchback and RelSedan, and then joined with tuples of RelDieselVehicle, over some particular attribute.

For the next pair in the path, (DieselVehicle, Vehicle), we may obtain the following pair:

(RelDieselVehicle, AttrAN-Sel, range-of-values(for Auto) in AttrAN-Sel= {DieselAuto},
AN-Join ≡ AttrDVId)

(RelAllItems, AN-Join ≡ AttrDVId).

Finally, het-map would use some sub-case from the previously discussed Case 2 to obtain the transformation for the pair (Vehicle, Model), for example,

(RelAllItems, AttrAN-EO, range-of-values(for Vehicle) in AttrAN-EO = {Vehicle},
AttrModel).

Case 4

This is the het-map transformation when the attribute of an object is itself an object, for example, a query involving the Manufacturer of a Vehicle which is the OwnedVehicle of a YachtClub. In the next section, we consider how this information is extracted from an XSQL-like query, but the het-map transformations for combining multiple paths is not discussed here and we refer the reader to Raschid *et al.* (1993).

5 Extracting Query Transformation Information from an XSQL query

In this section we consider the steps in parsing an XSQL-like query to extract information relevant to the query transformation process. We assume that similar information can be extracted from queries in other languages. An XSQL query is of the following form:

Select
from object varname, object varname ⋯
where qpe

The *query path expression* (qpe) in the **where** clause of the query is of the following form:

X.attr-X [Y].attr-Y [Z].attr-Z [A].attr-A [B] ⋯

where X, Y, Z, etc., are object classes, attr-X, attr-Y, etc., are attribute labels, and X.attr-X [Y] is read as *the attribute attr-X for the object X, is an object Y*.

The XSQL query and, in particular, the qpe of the query is resolved against the object schema, represented in F-logic in our case. A procedure **Extract** will operate on the list of objects in the **from** clause of the query, the qpe of the query, and the schema. The output will be a set of *query path structures* (qps). The set of qps obtained from an XSQL query will correspond to the mapping information from this query, in the canonical representation (CR). The qps in the CR will be used in the next step, which is the mapping between data models, using the merged canonical representation (MCR),

Extract: Extracting information from X.attr-X[Y].attr-Y[Z]		
$Extract_1$	Object type of Y known in **from**	
	$Extract_{1,1}$	X.attr-X returns object Y′ and Y′ ≡ Y
	$Extract_{1,2}$	X.attr-X returns object Y′ and Y′ ≠ Y
$Extract_2$	Object type of Y not known in **from**	
	$Extract_{2,1}$	X.attr-X returns some object Y′ and Y′ has some subclass(es) Y which have attribute attr-Y
	$Extract_{2,2}$	X.attr-X returns some object Y′ but none of the subclass(es) Y of Y′ has attribute attr-Y. However, Y and Y′ may inherit attribute attr-Y

Figure 5: Summary of the Extract Procedure

for the purpose of generating some translated query. For simplicity, we consider an XSQL query with a single qpe in the **where** clause.

Each qps structure in the CR is as follows:

[start-object,
list-type = {sub | sup},
< list >,
end-object,
attribute,
return-object(end-object,attribute)]

In each qps structure, the list-type indicates the existence of a path (based on some class hierarchy) between the objects start-object, say X, and end-object, say X′, in the F-logic schema. The direction of traversal of this hierarchy is noted by sub or sup. This is followed by the list of objects that are traversed in this path. The start-object and end-object, X and X′, may be explicitly specified in the corresponding XSQL query or they are obtained from the F-logic schema, by the **Extract** procedure. The end-object is a particular object X′ with attribute attr-X as specified in the qpe. Finally, we also determine the return-object that is returned based on X′ and attr-X. This return object may not be identical to the Y specified in the **from** clause. If indeed it is different from Y, then **Extract** must continue and generate further qps until the complete path from say X to Y is determined. There may be several qps for each object identified as start-object. Note that this start-object is just the start of a list of objects in the qpe, and is different from the start object SO, mentioned in the het-map information, in the previous section. Figure 5 summarizes the operation of the procedure **Extract**.

We only describe the procedure using examples and refer the reader to Raschid *et al.* (1993) for details.

Extract

Each sub-qpe

X.attr-X [Y].attr-Y [Z]

of the XSQL query will be processed in turn. It will first be resolved against the schema to obtain a path from X to some X′ with attribute attr-X. It

will return one (or more) *query path structure* (qps), as follows:

[X, sup, <list-superclass$_X$ >, end-object$_X$ $\equiv$ X$'$, attr-X, return-object(X$'$,attr-X) $\equiv$ Y$'$]

For example, the qpe DieselBoat.Manufacturer will generate the following qps:

[DieselBoat, sup, <DieselBoat, DieselVehicle, Vehicle>, Vehicle, Manufacturer, Company]

This procedure **Extract** will now terminate if X.attr-X does not return an object. Otherwise, we continue as follows to determine a path to Y:

Extract$_1$ – object type of Y known in from

Extract$_{1,1}$ – if $Y \equiv Y'$

Do nothing since the object returned is exactly Y, and continue processing. An example is the following query:

Query 7
Select *
from GasolineBoat Y
where YachtClub.OwnedVehicle[Y] $\cdots$

The object Y which is the return-object(YachtClub,OwnedVehicle) is exactly GasolineBoat, and this matches the object specified for Y in the query.

Extract$_{1,2}$ – if $Y \neq Y'$ **but is known in from**

An example is the following query:

Query 8
Select *
from DieselBoat Y Company Z
where Company.LeasedVehicle [Y].Manufacturer [Z] $\cdots$

It will generate the following set of qps:

[Company, nil, NULL, Company, LeasedVehicle, Vehicle]

[Vehicle, sub, <Vehicle, DieselVehicle, DieselBoat>, DieselBoat, _ , _]

To explain, return-object(Company, LeasedVehicle) is Vehicle, but the type for Y specified in the **from** clause of the XSQL query is DieselBoat. The second qps correctly obtains this object for Y. Processing continues with the qpe DieselBoat.Manufacturer [Z].

There is a possibility that during this procedure, paths between objects will be traversed multiple times and may lead to redundant computation. This must be avoided. For example, when **Extract** continues processing DieselBoat.Manufacturer [Z], the following qps is generated:

[DieselBoat, sup, <DieselBoat, DieselVehicle, Vehicle>, Vehicle, Manufacturer, Company]

As can be seen, the path between DieselBoat and Vehicle is traversed twice and could lead to redundant computation.

Extract$_2$ – object type of Y unknown in from

Extract$_{2,1}$

The example query is as follows:

Query 9
Select *
from
where Company.LeasedVehicle [Y].Registration

Here, the object type of Y is unknown. The object Y$'$ which is the return-object(Company,LeasedVehicle) is Vehicle. From the schema, and the attribute Registration associated with Y, the procedure **Extract** will infer that there is a path from Y$'$ to two objects (Y) which have the attribute Registration. Thus, we determine that Y may be either DieselBoat or GasolineBoat. The corresponding qps will be generated.

Extract$_{2,2}$

Again, we present this case using the following example query:

Query 10
Select *
from
where YachtClub.OwnedVehicle [Y].Manufacturer [Z].attr-Z

Here, the object type of Y is not specified. From the schema, we determine Y$'$, the return-object(YachtClub,OwnedVehicle) to be the object GasolineBoat. However, GasolineBoat does not have attribute Manufacturer nor does it have any sub-class with this attribute. The procedure **Extract** will subsequently process the sub-qpe GasolineBoat.Manufacturer, and determine that there is a path in the schema from GasolineBoat to the object Vehicle, and the object Vehicle has the attribute Manufacturer. Appropriate qps must be generated. Again, with this situation, there is the possibility of redundant path traversal.

6 Representation of the Query Transformation and Heterogeneous Mapping Information

The information on mapping between the different schema, corresponding to heterogeneous data models, and the rules for query transformation, must be represented in some canonical form. The first step is to express this information in a high level language. We chose the F-logic language as formulated by Kifer and Lausen (1989) and Kifer *et al.* (1990) for a number of reasons. F-logic can be used to describe an object-oriented database, it has the ability to manipulate high order concepts and there is work underway to build an interpreter for the language (see, *e.g.*, Lefebvre *et al.* (1992)). We briefly present the syntax and semantics of F-logic and then describe the data structures and a few of the transformation rules. Raschid *et al.* (1993) provide additional details. We are continuing research on the appropriate canonical form for representing this information.

6.1 Syntax of F-logic

We borrow this brief description from Lefebvre *et al.* (1992). In F-logic, the *instance term* $o : c$ means that the object o is an instance of class c. A *data term* $o[m@a_1, \cdots, a_n \rightarrow v; m'@a_1, \cdots, a_p \leftrightarrow \{v', v''\}]$ means that the value of the functional method m with arguments a_1 to a_n for the object o is a set containing the values v' and v''. If a method m has no arguments, "@" will be omitted. The symbol $\leftrightarrow$ indicates a set-valued method.[2] Note that other values could also be members of this set, and that the expression above does *not* restrict the value of m' for o to be $\{v', v''\}$; it only indicates *some of* the values of the corresponding set. An object can be denoted by a constant, or a term. For example, $dept(cs)$ is a valid object identifier. An *atomic data term* is a data term referring to only one method. Notational conventions allow us to write $o[m' \leftrightarrow v]$ instead of $o[m' \leftrightarrow \{v\}]$ for a single element of a set-valued method; the expression $o[m \rightarrow v; n \rightarrow v']$ is equivalent to $o[m \rightarrow v] \wedge o[n \rightarrow v']$; the expression $o : c[m \rightarrow v]$ is equivalent to $o : c \wedge o[m \rightarrow v]$.

An F-logic program consists of a set of data declarations(data or instance terms) and a set of *deduction rules*. A deduction rule has a *head*, which is a data term, and a *body*, which is a conjunction of data and instance terms. Disjunction and negation are allowed in the body of rules. Deduction rules can be used in a way similar to Datalog rules(or Prolog clauses), i.e., the head of a rule defines a derived method, the value of which can be found by evaluating the body.

6.2 Data Structures for Heterogeneous Mapping and Transformation

We present the data structures (in the canonical representation) corresponding to transforming a query fragment in the XSQL language (against an object-oriented data model defined using F-logic) to a query against a corresponding relational database. The query fragment that we discuss here has a simple query path expression (qpe) and produces a single query path structure (qps). We do not consider combining several query path structures (see Raschid *et al.* (1993) for further work on this problem).

```
MPS[begin-class → X,
    list-class ↔ P,
    end-class@P → X',
    attr-X → A,
    return-object@X', A → Y]

HT-mapping(X, D)
   [reachable-attr ↔ A,
    includes ↔ X',
    mappings ↔ map(X, A, D)[rel-attr ↔ (R, Z)],
    constraints ↔constraint(R, X)
                  [attr → AN-C, range → S ]]

CR-query(MPS, D)
   [rel-attr ↔ (R, AN),
    constraints ↔ (R, AN-C,S),
    join-path@P → L]
```

A mapping path structure (MPS) represents the semantics extracted from a simple XSQL query path structure (qps), as discussed in the previous section. It has methods *begin-class, list-class, end-class, return-object*, etc., corresponding to the qps structure. The heterogeneous mapping (HT-mapping) captures the het-map information between the object-oriented and relational schema. Its structure is discussed in detail in the next section. CR-query corresponds to the transformed query in the canonical representation (CR). CR-query will be used to generate the appropriate relational query for a particular database schema D (the actual generation is not discussed here). The method *rel-attr* of CR-query will return one or more pairs of relation and attribute names. For each pair that is returned, if there is additional selection criteria, then the method *constraints* returns the appropriate attribute name and the range of values for the selected tuples to qualify. The *join-path* is a method that provides the join information for the Case 3 of the het-map (not discussed in detail). Other structures, e.g., lists are omitted. Recall that several possible heterogeneous mappings between relational and object-oriented schema were discussed previously and presented as several cases. We discuss the canonical representation of these cases in this section.

6.3 Heterogeneous Mapping Algorithm and F-logic Rules

The heterogeneous mapping and transformation algorithm that we present here is intended to provide a flavor of the types of transformations that can be expressed and is not intended to represent a complete algorithm for transforming XSQL queries. It can best be expressed by the decision table in appendix C, where each row corresponds to one of the sub-cases of possible het-map transformations discussed previously.

Information extracted from the XSQL query, in the form of qps are represented by the MPS. This and the relevant mapping information, represented in HT-mapping, are used in the decision procedure. The MPS encodes a decision procedure based on the begin class X, end class X', and the attribute of interest A specified in the XSQL query. For both X and X', HT-mapping has the relevant mapping information for a particular database D, in this case each sample database is a relational database.

Each row in the table corresponds to one of the subcases of possible het-map transformation previously discussed. We note that case 3 is a fairly complex case involving a sequence of joins in the relational query and the details are omitted. The decision table is implemented using a set of rules in F-logic. These rules occur in 3 groups. Ident-Rel-Attr is a group of rules which represents the het-map transformation that provides the corresponding relation and attribute name in the CR-query data structure. Sel-Constr is a group of rules which specifies selection criteria that must first be satisfied by the tuples of the relations specified in Ident-Rel-Attr rules (or the Join-Path rules). Join-Path is a group of rules which specify relations which must be joined,

[2] This symbol is different from the one used in the original paper.

corresponding to the more complicated Case 3 of the het-map transformations that have been discussed. This third group of rules is not described here.

Case 1.1 is selected when the het-map information, encoded by MPS and HT-mapping in the canonical representation, indicate that given the begin class X, A is a *reachable* attribute. In other words, the corresponding relation and attribute names are directly obtained. The following F-logic rule in the group Ident-Rel-Attr may be applied in this case and will provide one or more (relation, attribute) pair(s) in the corresponding CR-query structure:

```
CR-query(MPS, D)
    [rel-attr ↔ (R, AN) ] ←
     MPS[begin-class → X , attr → A] ∧
     HT-mapping(X, D)[reachable-attr ↔ A,
                      mappings ↔ map(X, A, D)
                                 [rel-attr ↔ (R, AN)]]
```

From our example, an instantiation of this rule for the pair (DieselAuto, Model) may be as follows:

```
CR-query(MPS, D)
    [rel-attr ↔ (RelDieselAuto, AttrModel) ] ←
     MPS[begin-class → DieselAuto , attr → Model] ∧
     HT-mapping(DieselAuto, D)
         [reachable-attr ↔ Model,
          mappings ↔ map(DieselAuto, Model, D)
                     [rel-attr ↔ (RelDieselAuto, AttrModel)]]
```

For Case 1.2, the MPS and HT-mapping indicate that given the begin class X, A is a reachable attribute. However, HT-mapping indicates there is an additional selection *constraint* that (the tuples corresponding to) X must satisfy. Thus, in addition to applying the previous rule, the following rule in the group Sel-Constr must also be applied. This rule will generate a constraint in the corresponding CR-query structure, as follows:

```
CR-query(MPS, D)
    [constraints ↔ (R, AN-C, S)] ←
     MPS[begin-class → X] ∧
     CR-query(MPS, D)[rel-attr ↔ (R, AN)] ∧
     HT-mapping(X, D)[constraints ↔ constraint(R, X)
                                    [attr → AN-C, range → S]
```

If we consider the example from the previous section, corresponding to Case 1.2, one particular instantiation for the two rules from Ident-Rel-Attr and Sel-Constr is as follows:

```
CR-query(MPS, D)
    [rel-attr ↔ (RelDieselVehicle, AttrModel) ] ←
     MPS[begin-class → DieselAuto , attr → Model] ∧
     HT-mapping(DieselAuto, D)
         [reachable-attr ↔ Model,
         mappings ↔ map(DieselAuto, Model, D)
                    [rel-attr ↔ (RelDieselAuto, AttrModel)]]

CR-query(MPS, D)
    [constraints ↔
     (RelDieselVehicle, AttrAN-C, {Hatchback, Sedan }) ] ←
      MPS[begin-class → DieselAuto] ∧
      CR-query(MPS, D)
          [rel-attr ↔ (RelDieselVehicle, Model)] ∧
      HT-mapping(DieselAuto, D)
          [constraints ↔ constraint(RelDieselVehicle, DieselAuto)
                  [attr → AN-C, range → {Hatchback, Sedan}S]
```

After this transformation, we see that CR-query for this particular database and query is as follows:

```
CR-query(MPS, D)
    [rel-attr ↔ (RelDieselVehicle, AttrModel) ]
     [constraints↔
      (RelDieselVehicle, AttrAN-C, {Hatchback, Sedan }) ]
```

It should be straightforward to generate a selection query which selects the values of the attribute AttrModel from those tuples of the relation RelDieselVehicle such that the range of values in the attribute AttrAN-C is {Hatchback, Sedan}.

Next, we consider Cases 2.1 and 2.3, which both use the same rule in the group Ident-Rel-Attr. The decision table entries correspond to the situation where the MPS and HT-mapping indicate that the attribute A is not a reachable attribute for the start class X but is reachable for the end class X′. Further, the start class *includes* the end class, or in other words, all tuples in the relation(s) corresponding to the end class also correspond to the start class. The following F-logic rule in the group Ident-Rel-Attr may be applied in this case and will provide a relation, attribute pair in the corresponding CR-query structure:

```
CR-query(MPS, D)
    [rel-attr ↔ (R, AN)] ←
     MPS[begin-class → X, list-class ↔ P,
         end-class@P → X′ , attr → A] ∧
     not(HT-mapping(X, D)[reachable-attr ↔ A]) ∧
     HT-mapping(X′, D)[reachable-attr ↔ A,
                       mappings ↔ map(X′, A, D)
                                  [rel-attr → (R, AN)]] ∧
     not(HT-mapping(X, D)[constraints ↔ constraint(R, X)]))∧
     HT-mapping(X, D)[includes ↔ X′]
```

The decision table entries for Cases 2.1 and 2.3 correspond to the situation where the MPS and the HT-mapping indicate that although there is no constraint on the begin class, there may be a constraint on the end class. If there actually is a constraint (Case 2.3), then the following rule in the group Sel-Constr will be applied and will generate a constraint in the corresponding CR-query structure. (Complete details are presented by Raschid *et al.* (1993).)

```
CR-query(MPS, D)
    [constraints ↔ (R, AN-C, S)] ←
     MPS[list-class ↔ P, end-class@P → X′] ∧
     CR-query(MPS, D)[rel-attr ↔ (R, AN)] ∧
     HT-mapping(X′, D)[constraints ↔ constraint(R, X′)
                                     [attr → AN-C, range → S]
```

7 Summary

We have proposed the use of a canonical representation (CR) to address interoperable query processing with heterogeneous databases. We use examples of extracting knowledge from an XSQL-like query and mapping between heterogeneous schema to illustrate the knowledge that must be in the CR. We use a high-level declarative language F-logic, to demonstrate a possible form for the CR.

Over the last few years, research in schema integration has addressed the issue of heterogeneous data models. However, that research has focussed on integration issues rather than mapping among schema. It has also not addressed the problem of query transformation, assuming that queries will be expressed against an integrated schema. This is different from our goals, since we do not assume schema integration among the multiple servers, nor the use of a single query language.

Kent (1991) classifies mismatches in heterogeneous databases as domain mismatches (discrepancies among data values, units of measure, data types, etc.) and schema mismatch (different constructs in the data models representing the same concept). They use a language called the IPL (Iris Programming Language) to express integrator functions to solve the mismatch. They do not address the problem of query transformation. Semantic heterogeneity, where discrepancies may occur with the same data model, is investigated by Krishnamurthy *et al.* (1991). The goal of their research is achieving schema integration (transparent to the users) and supporting the ability to update a multidatabase. The language they propose is an extension to a Horn language. Our research differs from theirs in that we consider multiple data models and query languages. Kim and Seo (1991) provide an exhaustive list of conflicts that may occur in a multidatabase environment, expressed in the context of the relational data model.

The research that is closest to our work is described by Lefebvre *et al.* (1992) and Qian (1992). Lefebvre *et al.* (1992) consider the the problem of interoperable query processing, but their approach is limited to the relational data model. F-logic is used to express the mapping information among multiple relational schema and to express the algorithm for query transformation. Our work can be seen as an extension of their approach, to include heterogeneous models. Qian (1992) suggests that a language which has minimal *representation bias* may express mismatch in representation among heterogeneous schema. The paper describes an approach to interoperable query processing for different schema based on the same data model (entity relationship model), but should be applicable to heterogeneous data models as well.

Research in combining deductive and object-oriented data models is also relevant (see, *e.g.*, Barsalou and Gangopadhay (1992), Gardarin and Valduriez (1992), and Jeusfeld and Jarke (1991)). The Conceptbase system by Jeusfeld and Jarke (1991) addresses the problem of integrity checking through expressing many object-oriented constructs in the form of a deductive database system. Gardarin and Valduriez (1992) propose a language that is suitable for relational, deductive and object-oriented data models. Although the goals of these researchers are different, they provide insight into knowledge that is relevant to the query transformation process. Finally, in SIMS (see Arens and Knoblock (1992)), information sharing is facilitated through using the LOOM knowledge representation schema to construct a global schema for each application domain. Queries are proposed against the global schema and the SIMS system formulates a plan for query evaluation. The sub-queries are evaluated by LIM (interface modules) which build LOOM schema for each of the databases. This research, too, does not address the issues of query transformation and interoperable query processing.

Our current research has been limited to equivalent information resident in heterogeneous servers. Thus, the mapping information in the CR has been straightforward. We plan to extend our research to include knowledge of dependencies, constraints, etc. where the information in the CR may no longer be equivalent among the servers. As an example, in the LOOM system, a concept is described by structural characteristics, a classification hierarchy, and its relationship with other concepts. It will be a challenge to provide interoperability for a LOOM schema, in conjunction with relational and object-oriented schema. Most research in heterogeneity has focussed on structural mismatch (among concepts). However, most knowledge models support rules, constraints, methods, etc. Interoperable query processing must support these constructs in heterogeneous servers, and our research will consider techniques to represent this knowledge in a canonical form.

8 Bibliography

Arens, Y. and C. Knoblock (1992) "Planning and reformulating queries for semantically-modeled multidatabase systems," *Proceedings of the International Conference on Knowledge Management.*

Barsalou, T. and D. Gangopadhay (1992) "M(DM): An open framework for interoperation of multimodel multidatabase systems," *Proceedings of the International Conference on Data Engineering.*

Dorr, B. (1987) "UNITRAN: A Principle-Based Approach to Machine Translation," AI Technical Report 1000, Master of Science thesis, Department of Electrical Engineering and Computer Science, Massachusetts Institute of Technology.

Dorr, B. (1990) "Solving Thematic Divergences in Machine Translation," *Proceedings of the 28th Annual Conference of the Association for Computational Linguistics*, University of Pittsburgh, Pittsburgh, PA, 127–134.

Dorr, B. (1993) "Interlingual Machine Translation: A Parameterized Approach," *Artificial Intelligence* 63:1&2.

Gardarin, G. and P. Valduriez (1992) "ESQL2: An object-oriented SQL with F-logic semantics," *Proceedings of the International Conference on Data Engineering.*

Jeusfeld, M. and M. Jarke (1991) "From relational to object-oriented integrity simplification," *Proceedings of the Second International Conference on Deductive and Object-Oriented Databases.*

Kent, W. (1991) "Solving domain mismatch and schema mismatch problems with an object-oriented database programming language," *Proceedings of the International Conference on Very Large Data Bases.*

Kifer, M. and G. Lausen (1989) "F-logic: A higher-order language for reasoning about objects, inheritance and scheme," *Proceedings of the ACM Sigmod Conference.*

Kifer, M., G. Lausen, and J. Wu (1990) "Logical foundations of object-oriented and frame-based languages," Technical Report 90/14.

Kifer, M., W. Kim, and Y. Sagiv (1992) "Querying object-oriented databases," *Proceedings of the ACM Sigmod Conference.*

Kim, W. and J. Seo (December, 1991) "Classifying schematic and data heterogeneity in multidatabase systems," *IEEE Computer*, pages 12–18.

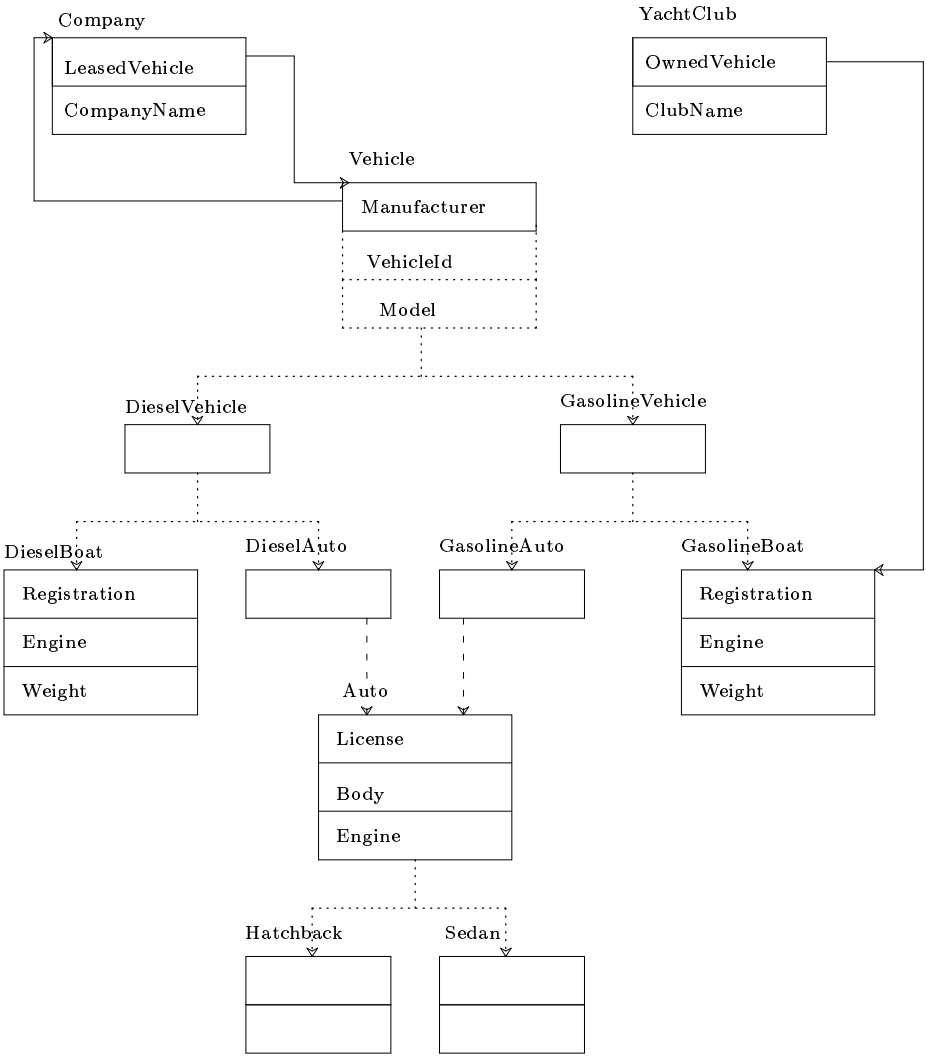
Krishnamurthy, R., W. Litwin, and W. Kent (1991) "Language features for interoperability of databases with schematic discrepancies," *Proceedings of the ACM Sigmod Conference.*

Lefebvre, A., P. Bernus and R. Topor (1992) "Query transformation for accessing heterogeneous databases," *Joint International Conference and Symposium on Logic Programming, Workshop on Deductive Databases.*

Qian, X. (1992) "Semantic interoperation via intelligent mediation," *Personal communication.*

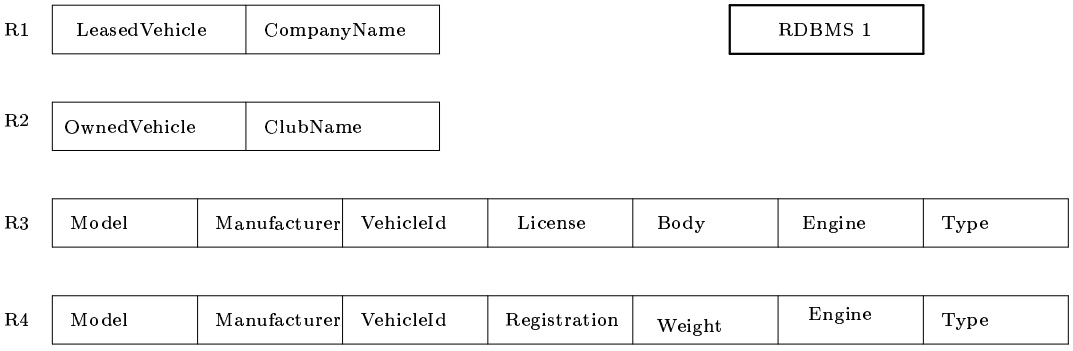
Raschid, L., Y. Chang, and B. Dorr (1993) "Query transformation among heterogeneous data models: a prototype based on F-logic," *In preparation.*

A Example Object Schema

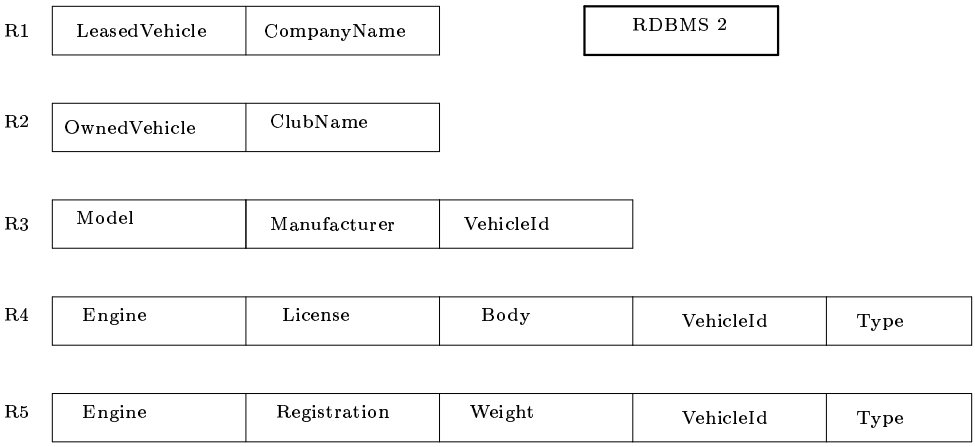


B Corresponding Relational Schema

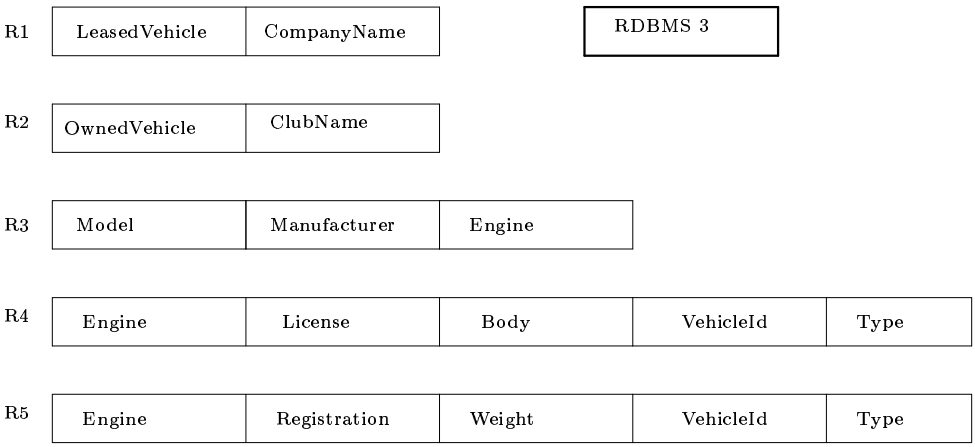
Relational Schema 1:



Relational Schema 2:



Relational Schema 3:



C Decision Table for Heterogeneous Mapping Algorithm

	begin-class X			end-class X′	
	reachable-attr	constraints	includes	reachable-attr	constraints
1.1	Y	N			
1.2	Y	Y			
2.2	N	Y		Y	N
2.4	N	Y		Y	Y
2.1	N	N	Y	Y	N
2.3	N	N	Y	Y	Y
3	N	N	N	Y	