Query Translation for Heterogeneous Information Management

Fabrice Jouanot, Christophe Nicolle, Nadine Cullot
Laboratoire LE2I
Faculté des Sciences Mirande
Université de Bourgogne
BP 138
21004 Dijon, FRANCE
Email : {jouanot, cnicolle, ncullot}@khali.u-bourgogne.fr

1 Introduction

Sharing, integrating and transparently accessing information across heterogeneous systems increase the need of new translation methods.

Heterogeneity may take different forms including hardware, operating systems, data models (structural representation, semantic and data manipulation) and internal physical organization of models.

Semantic heterogeneity that results from differences in static (structures and data) and dynamic (queries) aspects of data models hinders the management of heterogeneous information.

A solution is to provide direct translation paths between pair of data models. This solution is best suited to loosely coupled FDBS described by Sheth in [10] or to data migration methods. In this case, users can access information through their native data models. This solution may require a number of translators (schema, query and data) that is quadratic in terms of the number of models. Katz & al. in [5] give an algorithm to generate relational query from codasyl procedure. Their method analyses data flow to create data semantic accesses. These accesses are used to define equivalences between codasyl DML (Data Manipulation Language) and SQL components. Bell & al. in [1] propose a symetric method to map concepts from SQL to codasyl DML.

Alternatively, semantic heterogeneity can be resolved by integrating all local schemes into one global schema. This solution is best suited to distributed and tightly coupled federated database systems. In this approach, the global schema is expressed in a common data model in which users of the federation express their queries [10]. An Object Broker Request usually decompose global queries into global sub-queries for each local databases. Translations from global sub-queries to local sub-queries are required if global and local languages are different. Keim in [6] presents a system which allows the integration of relational databases (SQL) in a distributed system using an object oriented global data model (SOQL). The query translation process uses a meta-information base which contains mapping between local and global schema. For this, there is a canonical representation of queries to resolve heterogeneity problems between SQL and SOQL. Chang & al. in [2] propose a similar approach for query translation between relational and object oriented databases. They use a knowledge base built during schema translation where semantic and structural equivalences are stored. An intermediate canonical form is used during the translation of SQL to SOQL. This method is best suited for legacy applications.

Another way to manage heterogeneous information is to use multi-database systems. These systems are composed of a non-integrated set of databases. Data access is done by using common query language which contains data localisation information. Florescu & al. in [4] present a mutli-database system based on ODMG an object oriented data model. Queries are expressed in OQL (Object Query Language). This method uses schema mapping dictionnaries for query translation. A Local Application Interface (LAI) is used for information localisation and translation of OQL queries into local queries. Nevertheless, this method needs the definition of specific mapping rules for each LAI combined with all heterogneous databases.

Section 2 presents an overview of the system TIME which allows data models translation in cooperative information systems (CIS). In section 3, we focus on TIRE, a module of TIME, which ensures query translation for heterogeneous infomation management. Section 4 concludes this article and presents our future work.

2 Overview of TIME

First, we present the CIS architecture of the TIME project (Traducteur Intelligent avec Meta-modèle Extensible). Figure 2 depicts an environment which includes tools for the cooperation of existing information systems. Each participating IS includes a "cooperative interface" which helping the resolution of semantic heterogeneity among the components. The interface is made of a schema acquisition module, a communication module, a local data access module and data translation modules.

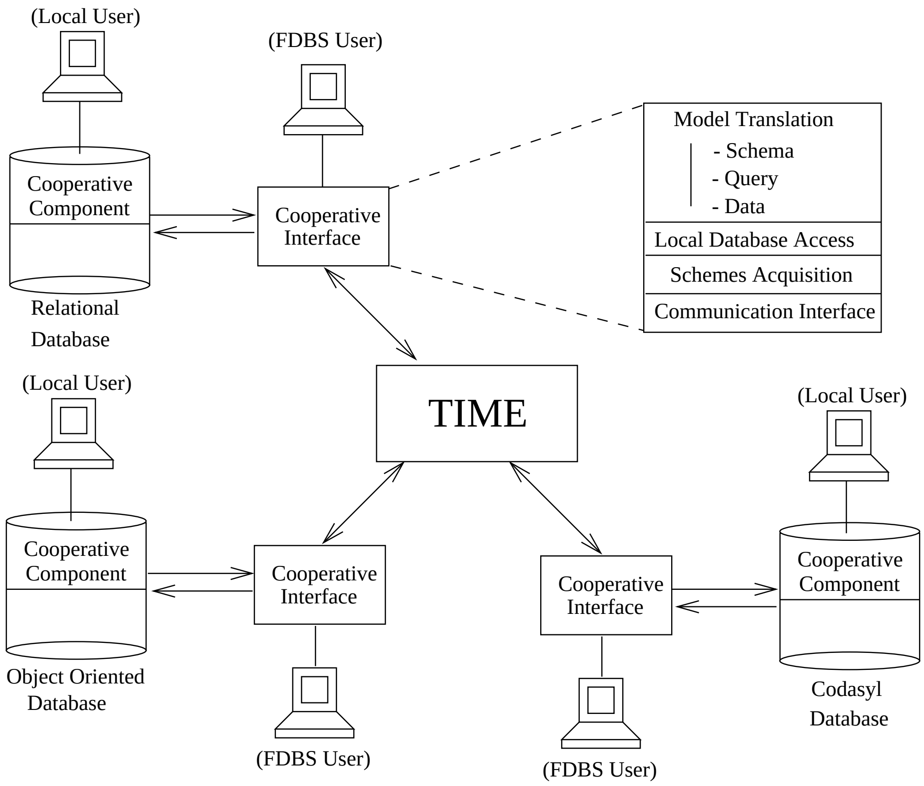

Figure 1: CIS Conceptual architecture

The schema acquisition module is used by an IS to exchange schema with other members of the federation. It implements import and export protocols between sites. A component uses the import protocols to acquire a subset of schemes from other components. The communication module is used to exchange information between components while the local data access module is used to manipulate the local data base. Finally, the translation module which is the focus of the TIME project, contains tools for schema transformation, query translation between sites and the transformation of database instances between sites.

Our objectives are to bridge the gap of heterogeneous data representation providing an extensible meta model to describe the syntax and semantic of various data models, and, to allow reusability of translation rules sharing code generation among different data model translators.

To achieve these goals we propose a data translation methodology. TIME contains two modules : TISE, a module for schema translation, and TIRE, a module for query translation.

TISE uses a metamodel which defines a set of metatypes (fig.2). These metatypes are able to represent concepts of various data models to modelize real world objects. Metatypes are organized in an inheritance hierarchy to capture similarities among data model concepts. As in object oriented model, an inheritance link between two metatypes is intended for reuse of structural properties as well as meta-data constraints. We use inheritance as a mechanism for extending the meta model by specializing existing metatypes to define new metatypes. The goal of reusability is achieved by defining and by associating data translation and meta-schema transformation rules to inheritance links of the generalization lattice. Note that there are two types of rules: "mapping rules" which are used to map data model schema to the meta model schema and "transformation rules" which are used within the meta model to convert meta schemes. Reusability is accomplished by using the structure of the inheritance generalization to share translation

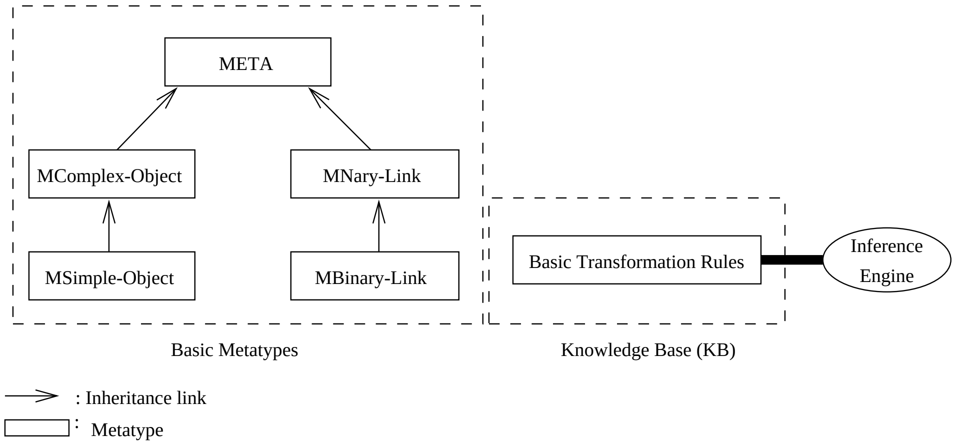

Figure 2: Structure of TISE

rules and code between metatypes. The resulting model and rules are used as a foundation for a tool which can automatically or semi-automatically generate data model compiler. Now, TISE is implemented in C++ and handles four data models (Relational, Codasyl, ERC+ and Object Oriented data models). The description of this module is not the goal of this paper. A complete description of TISE can be found in [8, 9].

Query translation and formatting data is insured by a module of TIME called TIRE presented in the following section.

3 Query Translation

Now, we present TIRE an extensible tool for query translation. In part one, the architecture of TIRE is briefly described. In part two, we define the concept of meta-query, which is a structure associated with meta-model to translate query. The fact and rule bases used as knowledge base during the translation are described respectively in part three and four. The last part shows a simple example of translation in TIRE.

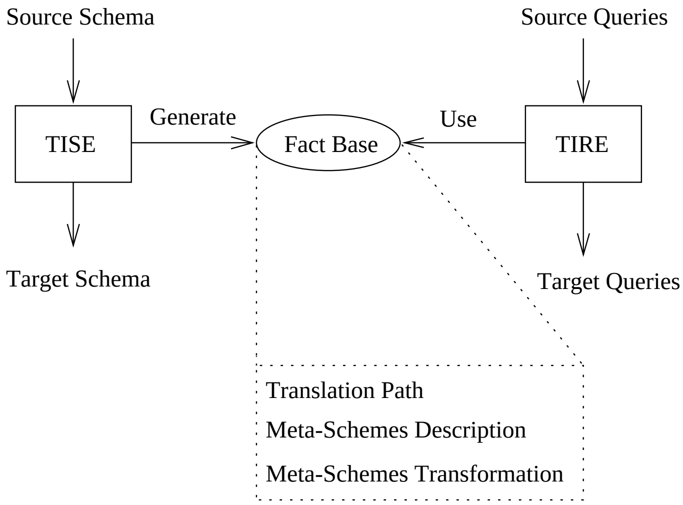

Figure 3: Correspondences beetween TISE and TIRE

3.1 Principle

TIRE is a system which allows the transformation of queries from a source DML to a target DML. In a learning step, TIRE uses as input a fact base generated by the schema transformation phases of TISE.

As shown in figure 3, the fact base is composed of

- a lattice of translation paths which guides the translation of queries,
- all meta-schemas which appear during the translation process of TISE,
- keys information about the transformation on these meta-schemas.

We define a meta-query as a query which uses only metatypes of TISE. A meta-query characterizes a node of the lattice of translation, it can manipulate different meta-schemas appearing in the schema transformation phases of TISE. The structure of a meta-query is shown bellow.

Like in TISE, we use the prefix *meta* to define operations and concepts in TIRE. There are three parts in the query tanslation process :

- **Part 1**: Translation of a source query into a meta-query in TIRE. This is done by a Down Optimisation Module (DOM) which flattens complex queries.
- **Part 2**: Transformation of an intermediate meta-query into a target meta-query.
- **Part 3**: Translation of a target meta-query into a target query expressed in the target model. This is done by a Up Optimisation Module (UOM) which unflattens the meta-query on target schema.

The general architecture of TIRE is depicted in figure 4.

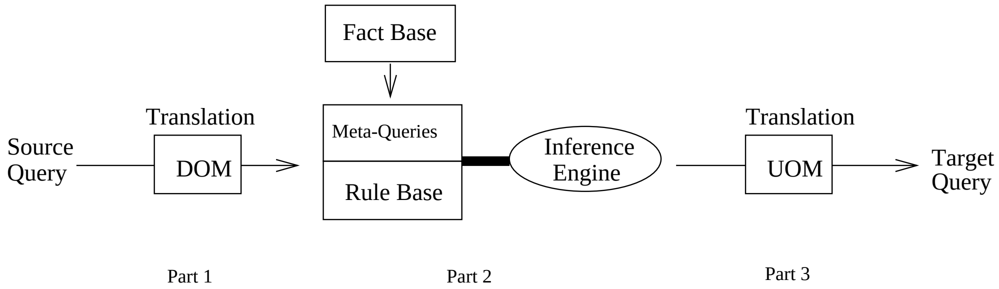

Figure 4: Architecture of TIRE

3.2 Structure of a Meta-Query

In TIRE, meta-query is expressed in a form called Canonical Meta-Query (CMQ) whose structure is independent of end-users data model languages. There is a CMQ for each intermediate meta-schema in the schema translation path. A Canonical Meta Query handles a set of predefined metatypes. Two different CMQ have two different sets of metatypes (the intersection of these sets could be not null). For example CMQ(SO_BL) is a query which handles a meta-schema composed of instances of MSimple-Object and MBinary-Link metatypes. A CMQ is generally composed of six concepts :

CMQ(X)=[Extension,Declaration,Join,Operation,Predicate,ECR]

- Extension specifies type and format of information which must appear in the final result.
- Declaration specifies a list of variables which are in the source query or which are generated during transformation process.
- Join specifies a set of implicit or explicit joins which appear during the transformation process.
- Operation specifies particular operators used in various data manipulation languages like INSERT, UPDATE, etc. This last parameter is not developed because we restrict the translation to interrogation queries (Due to interoperable systems where modification are done by local users).
- Predicates specify a set of constraints which use comparison operations between two CQE parameters or an CQE parameter and a constant.

- CQE (Complex Query Expression) is an expression of a complex query path of the type 'car.own.name'. These expressions appear usualy in meta-schema when metatypes is near object oriented concepts. The parameter CQE is composed of :

$$\text{CQE(i)} = [\ \text{Start\_Object, Path\_Object, Final\_Object}\ ]$$

  - The first element Start_Object describes the first object in the left side of a query path expression. It is an instance of a metatype or a variable.
  - The second element Path_Object describes the path to reach the right side object of an expression. It describes sub-paths of the general path.
  - Final_Object defines the right side of an expression.

3.3 Structure of the Fact Base

The fact base built by schema transformation phase contains mapping between meta-schemas. The fact base formally describes how information of a source schema is step by step converted and represented in a target schema. The definition of a mapping in the fact base is represented by MS_{Sx-Sy}. It represents the various schemas and mapping between a source schema Sx and a target schema Sy. It is composed of a tuple (S, MP, CMQ) for each step of transformation. The parameter S defines the schema representation, MP defines the mapping between this representation S and the last one, CMQ indicates which meta-query handles the schema S. For example, the fact base for translating relational model into object oriented model for specific schema is composed as follow :

$$\begin{aligned}\text{MS}_{SQL-OSQL} = \{\ & (\text{S}_{SQL}, \oslash, \text{CMQ(SQL)}),\\ & (\text{S}_{Rel}, \text{MP}_{SQL\rightarrow Rel}, \text{CMQ(Rel)}),\\ & (\text{S}_{SO\_BL}, \text{MP}_{REL\rightarrow SO\_BL}, \text{CMQ(SO\_BL)}),\\ & (\text{S}_{CO\_BL}, \text{MP}_{SO\_BL\rightarrow CO\_BL}, \text{CMQ(CO\_BL)}),\\ & (\text{S}_{CO\_BL\_NL}, \text{MP}_{CO\_BL\rightarrow CO\_BL\_NL}, \text{CMQ(CO\_BL\_NL)}),\\ & (\text{S}_{CO\_NL\_OIL}, \text{MP}_{CO\_BL\_NL\rightarrow CO\_NL\_OIL}, \text{CMQ(CO\_NL\_OIL)}),\\ & (\text{S}_{OO\_NL\_OIL}, \text{MP}_{CO\_NL\_OIL\rightarrow OO\_NL\_OIL}, \text{CMQ(OO\_NL\_OIL)}),\\ & (\text{S}_{OO\_OIL}, \text{MP}_{OO\_NL\_OIL\rightarrow OO\_OIL}, \text{CMQ(OO\_OIL)})\}\end{aligned}$$

Each query on this source relationnal schema follows the same translation path. Another object oriented transformation of a different relational schema may certainly have a different translation path: TISE provides the translation path during the learning step. We don't develop how are built parameters S and MP because S is also composed of instances of metatypes and MP is built automatically by meta-schema transformation rules for each instance transformation (see [7]).

3.4 Meta-Query transformation rules

Translation from a CMQ to another CMQ is realised by a query transformation rule which uses the fact base. For each path between two CMQ, a couple of transformation rules is provided to ensure the bidirectional translation (possible by backward processing on the fact base). As in TISE, the knowledge base is composed of two types of rules. First the basic set of query transformation rules, which are associated with canonical meta-query and which handle only instances of basic metatypes and non basic rules to handle CMQ on non basic metatypes.

3.4.1 Basic transformation rules

In this section, we describe unformally the basic query transformation rules. We can reuse existing generic rules to limit the definition of too many rules with instanciation of metatypes during the translation process.

- R_b(SO_L, CO_L), this generic rule transforms a CMQ which handles instances of SO and Link metatypes into a CMQ which handles instances of CO and Link metatypes. If instances of SO are gouped into one instance of CO then the ECR parameters which described the SO are removed, but new variables are created in the parameter Declaration of the new CMQ. In the parameter JOIN, the links are modified. This generic rule is used for the basic transformation rules R_b(SO_BL, CO_BL) and R_b(SO_NL, CO_NL) by example.

- R_b(CO_L,SO_L), this generic rule creates new binary links in parameters ECR and JOIN. New simple objects are created in ECR, all instances of complex object change their type into simple object. This rule is used for the basic transformation rules R_b(CO_BL, SO_BL) and R_b(CO_NL, SO_NL).

- R_b(O_BL, O_NL), the corresponding meta-schema transformation rule transforms a set of instances of BL into one instance of NL. It transforms the source CMQ by removing all binary links, and creating new instances of Nary-Link. This generic rule is used for the basic transformation rules R_b(SO_BL, SO_NL) and R_b(CO_BL, CO_NL).

- R_b(O_NL, O_BL), new instances of binary links are created, with new instances of object. The source CMQ is modified to handle these new instances, parameters Join, ECR and Declaration are modified. The instances of MNary link are removed into Join and ECR. This generic rule is used for the basic transformation rules R_b(SO_NL, SO_BL) and R_b(CO_NL, CO_BL).

3.4.2 Non basic transformation rules

These rules allow the transformation of canonical meta-query which handle new non basic metatypes into CMQ which handle existing metatypes. We note R_s (for Specifics Rules). In our example, we define two rules for the relational data model one from the metatype Rel to SO and BL metatypes and one inversely.

- R_s(REL, SO_BL), this rule transforms all join operators into binary links. The predicate parameter is analysed to detect comparison between primary and foreign keys. This comparison are transformed into the join parameter by binary links.

- R_s(SO_BL, REL), this rule transforms all binary links into explicit joins in the predicate parameter. New variables are created in the parameter declaration. At the end of this rule, the join parameter is empty.

Meta-query transformation rules for object oriented model is built in the same manner. We don't present these rules in this paper (see [7]).

3.5 Example of query translation

In this section we present the query translation process of TIRE. During the schema transformation, TISE generates a dictionary of mappings which is used by TIRE as a fact base. In this example, we show the translation of a relational query like SQL to object oriented query like OSQL. The source query is : "Give the name of all persons which own a yellow car". In SQL we can express this query by :

```
SELECT P.Name
FROM PERSON P, CAR C
WHERE P.PersonN#=C.Owner AND C.Color="Yellow"
```

The first step is to map SQL query into source meta-query. This is done in part one. We obtain :

```
CMQ(SQL) [ Extension [ECR(1) ],
           Declaration [V(ECR(2), P), V(ECR(3), C) ],
           Join [ ],
           Predicate [ECR(4) = ECR(5), ECR(6) = C(String, 'Yellow')¹]
           ECR [ ECR(1) [ V(P)², ∅, A(Name)³ ],
                 ECR(2) [ O(Relation, PERSON)⁴, ∅, ∅ ],
                 ECR(3) [ O(Relation, CAR), ∅, ∅ ],
                 ECR(4) [ V(P), ∅, A(PersonN#) ],
                 ECR(5) [ V(C), ∅, A(Owner) ],
                 ECR(6) [ V(C), ∅ A(Color) ] ],
           OPERATION [] ]
```

The following step the translation of this canonical meta-query into a canonical representation which handle instances of MRelation metatype. The intermediate meta-query transformations are detailed in appendix. Finally, the target OSQL query is obtained by transforming CMQ(OSQL) (Part 3).

[1] where C is a constant
[2] where P is a variable
[3] where Name is an attribute
[4] where PERSON is a relational object

```
SELECT P.Name
FOR EACH C.OWN P, CAR C
WHERE C.Color="Yellow"
```

4 Conclusion and futur works

In this paper we have described TIME, a methodology for translating multiple data models in cooperative database systems. TIME is composed of two sub-modules: TISE for schema translation and TIRE for query translation. We have focused our presentation on TIRE, where query processing between heterogeneous data models is carried out by mapping rules. A meta-query is transformed step by step, using the mapping rules. The composition of these rules constitues a translation path. Meta-queries and mapping rules form a lattice. TIRE is extensible, it allows the addition of a new DML and its associated maping rules from a source query to a meta-query and inversely. To reduce query translation cost, it achieves reusability of existing translation paths by sharing mapping rules. Information on metaqueries and rules are contained in a knowledge base. The fact base used by TIRE is created by an automatic learning step during the schema translation process.

This work is limited to the process of SQL for the relational model and OSQL for object oriented data model. It needs to be checked for other DML, and to be implemented in C++ like TISE.

References

[1] Bell D., Grimson J., *Distributed Database System*, International Comuter Science Series, Addison-Wesley Publishing Company, 1992

[2] Chang Y., Raschid L., Dorr B.J., *A Survey of Approaches to Achieve Interoperability with Multiple Databases*, Technical Report, Institute for Advanced Computer Studies, University of Maryland, April 8, 1994.

[3] Demurjian S.A., Hsiao D.K., *Towards a Better Understanding of Data Models Through the Multilingual Data system*, IEEE Transaction on Software Engineering, pp 946-958, Vol.14, No.7, July 1993.

[4] Florescu D., Raschid L., Valduriez P., *Using Heterogeneous Equivalences for Query Rewriting in Multidatabase Systems*, Submited to COOPIS 1995.

[5] Katz R.H., Wong E. , *Decompiling Codasyl LMD into Relational Queries*, TODS Vol.7 No.1, pp 1-23, 1981.

[6] Keim D.A., Kriegel H.P., Miethsam A. , *Query Translation Supporting the Migration of Legacy Database into Cooperative Information Systems*, Proceedings of the 2nd International Conference on Cooperative Information Systems, pp 203-214, 1994.

[7] Nicolle c. : *Traduction multi-modèles dans les systèmes d'information coopératifs*, Mémoire de Thèse, Université de Bourgogne, 22 Janvier 1996, Dijon, France.

[8] Nicolle C., Benslimane D., Yetongnon K. : *Multi-Data Models Translation in Interoperable Information Systems*, Proceeding of CAISE96, Lecture Notes in Computer Science, Springer-Verlag Eds, May 20-24, 1996.

[9] ϒNicolle C., Cullot N., Yetongnon K., *A meta-model based methology for translating data models in interoperable information systems*, Minitrack on Heterogeneous Database Interoperability, The Second Annual Americas Conference on Information Systems Phoenix, Arizona, August 16-18, 1996 ϒϒ

[10] Sheth A.P., Larson J.A. , *Federated Database Systems, for Managing Distributed, Heterogeneous, and Autonomous Databases*, pp183-235, ACM Computing Surveys, Vol.22, No.3, September 1990.

Appendix

In this appendix, we present the intermediate steps of meta-query transformation rules. These steps complete the example of the section on meta-query translation.

Part 2 : Step 1. This step removes syntactic components and analyses the PREDICATE parameter to discover mapping between primary and foreign keys. In this case, the system transforms this component of PREDICATE into a component of the JOIN parameter. Moreover, during this step, the instances change their type from Relation to MRelation.

```
CMQ(MRel) [ Extension [ ECR(1) ],
            Declaration [ V(ECR(2), P),
                          V(ECR(3), C) ],
            Join [ ECR(4) = ECR(5) ],
            Predicate [ ECR(6) = C(String, 'Yellow')]
            ECR [ ECR(1) [ V(P), ∅, A(Name) ],
                  ECR(2) [ O(MRelation, PERSON), ∅, ∅ ],
                  ECR(3) [ O(MRelation, CAR), ∅, ∅ ],
                  ECR(4) [ V(P), ∅, A(PersonN#) ],
                  ECR(5) [ V(C), ∅, A(Owner) ],
                  ECR(6) [ V(C), ∅ A(Color) ] ],
            OPERATION [] ]
```

Part 2 : Step 2. The next step transforms this canonical meta-query into a CMQ which handles instances of MSimple-Object and MBinary-Link. In this step, the query transformation rule Rt(Q1,MRel,Q2,SO_BL) transforms all joins into binary links. New variables are created by the system to represent the new binary links. In our example, following information in the meta-schema translation dictionary, inclusion dependencies between PERSON and CAR are transformed into the binary link "OWN".

```
CMQ(SO_BL) [ Extension [ ECR(1) ],
             Declaration [ V(ECR(2), P),
                           V(ECR(3), C),
                           V(ECR(5), O)],
             Join [ With O on (P, C) ],
             Predicate [ ECR(4) = C(String, 'Yellow')]
             ECR [ ECR(1) [ V(P), ∅, A(Name) ],
                   ECR(2) [ O(MSimple-Object, PERSON), ∅, ∅ ],
                   ECR(3) [ O(MSimple-Object, CAR), ∅, ∅ ],
                   ECR(4) [ V(C), ∅ A(Color) ] ],
                   ECR(5) [ O(MBinary-Link, OWN), ∅, ∅ ],
             OPERATION [] ]
```

Part 2 : Step 3. The next transformation step is important. The rule employed in this step, detects semantic enrichments in the meta-schema mapping (i.e. grouping a set of linked simple object into one complex object). In this case, the link in the Join parameter is removed and all variables concerning the simple objects linked are transformed by a variable of the resulting complex object. Our example is too simple to show the semantic enrichment. All simple objects are only transformed into complex objects.

```
CMQ(CO_BL) [ Extension [ ECR(1) ],
             Declaration [ V(ECR(2), P),
                           V(ECR(3), C),
                           V(ECR(5), O)],
             Join [ With O on (P, C) ],
             Predicate [ ECR(4) = C(String, 'Yellow')]
             ECR [ ECR(1) [ V(P), ∅, A(Name) ],
                   ECR(2) [ O(MComplex-Object, PERSON), ∅, ∅ ],
                   ECR(3) [ O(MComplex-Object, CAR), ∅, ∅ ],
                   ECR(4) [ V(C), ∅ A(Color) ] ],
                   ECR(5) [ O(MBinary-Link, OWN), ∅, ∅ ],
             OPERATION [] ]
```

Part 2 : Step 4. The next step detects the transformation of instances of MBinary-Link into instances of MNary-link. We obtain a new canonical meta-query which handles MComplex-Objects, MBinary-Links and MNary-Links. In our example the binary link "own" becomes a Nary link.

Part 2 : Step 5. Next, the system transforms the CMQ(OC_BL_NL) into a new CMQ(OC_OIL_LN) to detect query on inheritance link. The binary link "Own" don't become an inheritance link. Thus, there is no transformation in our example.

Part 2 : Step 6. Next, the system transforms all complex objects into MOObject metatype. The CMQ(OC_OIL_LN) is transformed into a new canonical meta-query CMQ(OO_OIL_LN) where all instances of MComplex-Object become instances of MOObject. There are on the following form :

```
CMQ(OO_OIL_LN) [ Extension [ ECR(1) ],
                 Declaration [ V(ECR(2), P),
                               V(ECR(3), C),
                               V(ECR(5), O)],
                 Join [ With O on (P, C) ],
                 Predicate [ ECR(4) = C(String, 'Yellow')]
                 ECR [ ECR(1) [ V(P), ∅, A(Name) ],
                       ECR(2) [ O(MOObject, PERSON), ∅, ∅ ],
                       ECR(3) [ O(MOObject, CAR), ∅, ∅ ],
                       ECR(4) [ V(C), ∅ A(Color) ] ],
                       ECR(5) [ O(MNary-Link, OWN), ∅, ∅ ],
                 OPERATION [] ]
```

Part 2 : Step 7. The last step of the transformation in TIRE is a transformation of all instances of MNary-Link into MOObject. Following the meta-schema transformation, links are added into existing objects as composition links or are transformed into instances of MOObject. In our example, the binary-link "OWN" becomes a reference attribute defined in ECR. In the Declaration, the variable O is removed and variable references are modified. The Join parameter is modified to represent the reference between two objects. The ECR with parameter O is renamed with the variable defining the object which is attribute reference. We obtain the following CMQ(OO_OIL) :

```
CMQ(OO_OIL) [ Extension [ ECR(1) ],
              Declaration [ V(ECR(5), P),
                            V(ECR(3), C),
              Join [ With ECR(5) on (ECR(2), ECR(3)) ],
              Predicate [ ECR(4) = C(String, 'Yellow')]
              ECR [ ECR(1) [ V(P), ∅, A(Name) ],
                    ECR(2) [ O(MOObject, PERSON), ∅, ∅ ],
                    ECR(3) [ O(MOObject, CAR), ∅, ∅ ],
                    ECR(4) [ V(C), ∅, A(Color) ] ],
                    ECR(5) [ V(C), ∅, R(ECR(2),OWN)],
              OPERATION [] ]
```

Part 2 : Step 8. Last, we transform the CMQ(OO_OIL) into CMQ(OSQL). In this step, due to implicit join, the Join parameter is empty, the connection between PERSON and CAR is realised directly in the definition C. We obtain :

```
CMQ(OSQL) [ Extension [ ECR(1) ],
            Declaration [ V(ECR(5), P),
                          V(ECR(3), C),
            Join [ ],
            Predicate [ ECR(4) = C(String, 'Yellow')]
            ECR [ ECR(1) [ V(P), ∅, A(Name) ],
                  ECR(2) [ O(Class, PERSON), ∅, ∅ ],
                  ECR(3) [ O(Class, CAR), ∅, ∅ ],
                  ECR(4) [ V(C), ∅, A(Color) ] ],
                  ECR(5) [ V(C), ∅, R(ECR(2),OWN)],
            OPERATION [] ]
```