SEWASIE
Semantic Webs and AgentS in Integrated Economies
IST-2001-34825

WP3
Task T3.1
Deliverable D3.3

First release of the system prototype for query management

(FINAL, 30/05/2003)

Abstract – The purpose of this document is to present a prototype implementation of a general techniques for query management within one SINode in the Sewasie system.

Zoran Majkić
UNIROMA

Document information

Document ID code	D3.3		
Keywords	Query Expander, Query reformulation, Single-class atoms		
Classification	**FINAL**	Date of reference	**30/05/2003**
Distribution level	**Sewasie Partners of the Sewasie consortium, and EU commission**		

Editor	Zoran Majkić	UNIROMA
Authors	Maurizio Lenzerini	UNIROMA
	Zoran Majkić	UNIROMA
Reviewer	Domenico Beneventano	UNIMO

Version history		
Date	*Version*	*Description*
20/05/2003	DRAFT 1	Draft version
30/05/2003	FINAL	Final version

Contents

1 Executive summary

This document illustrates a prototype implementation of the technique designed for query management within one SINode in the Sewasie system. Query processing within one SINode is constituted by the following phases: Query Expansion, Global class materialization, Local class materialization, and Evaluation of the expanded query.

In our prototype, we have built one module for each of the above phases. After a brief introduction to query processing within one SINode, the subsequent sections are devoted to a brief explanation of the various modules. In the document, we also illustrate an example, which is the basis for the first demo of the system.

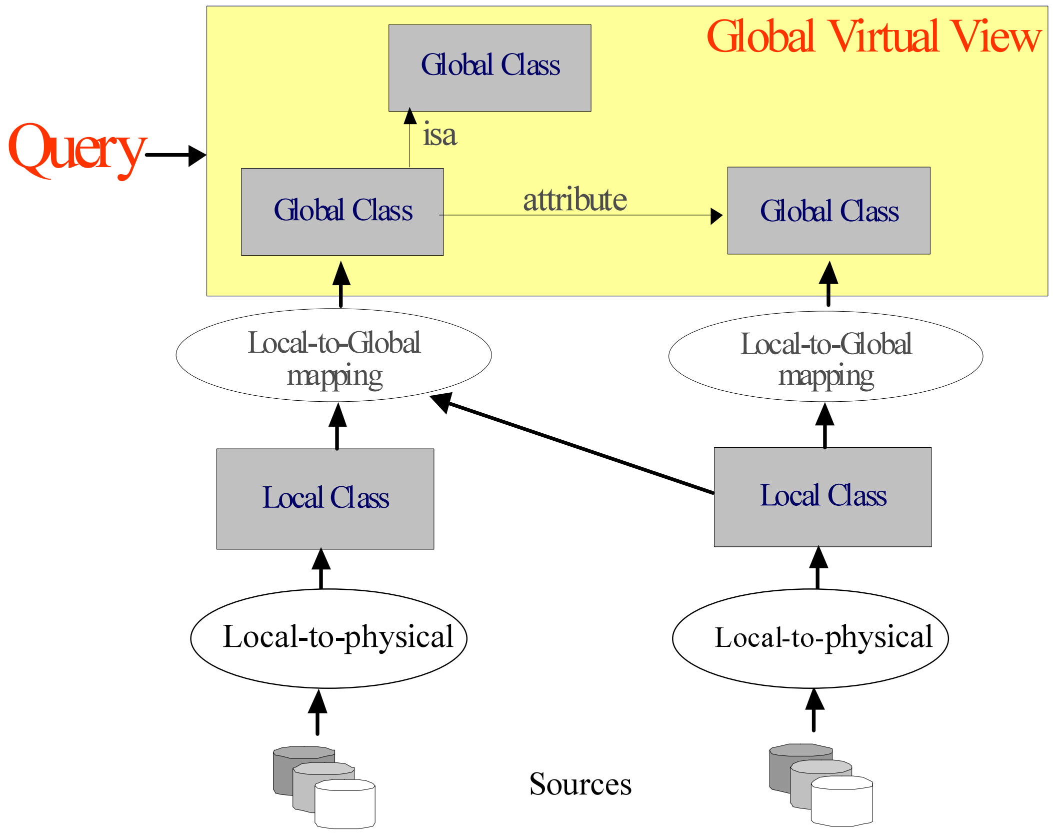

Figure 1: Structure of one SINode

2 Query processing within one SINode

The Query Manager is the coordinated set of functions which take an incoming query, define a decomposition of the query according with the mapping of the global virtual view of the SINode onto the specific data sources available and relevant for the query, sends the queries to the wrappers in charge of the data sources, collects their answers, performs any residual filtering as necessary, and finally delivers whatever is left to the requesting query agent.

The structure of an SINode is illustrated in figure 1.

Details about the structure of an SINode, and about the technique for query processing within one SINode can be found in the following Sewasie deliverables:

- Deliverable D2.1: Specification of the general framework for the multilingual semantic enrichment processes and of the semantically enriched data stores. University of Modena and Reggio Emilia.
- Deliverable D3.2.A: Techniques for query reformulation – Part A. University of Rome “La Sapienza”.

Query processing within one SINode is illustrated in Figure 2. From this figure, one can see that query processing is constituted by the following phases:

1. **Query Expansion**: the query is expanded to take into account the integrity constraints in the global schema.
2. **Global class materialization**: the atoms in the expanded query are materialized by taking into account the mapping to the local classes.

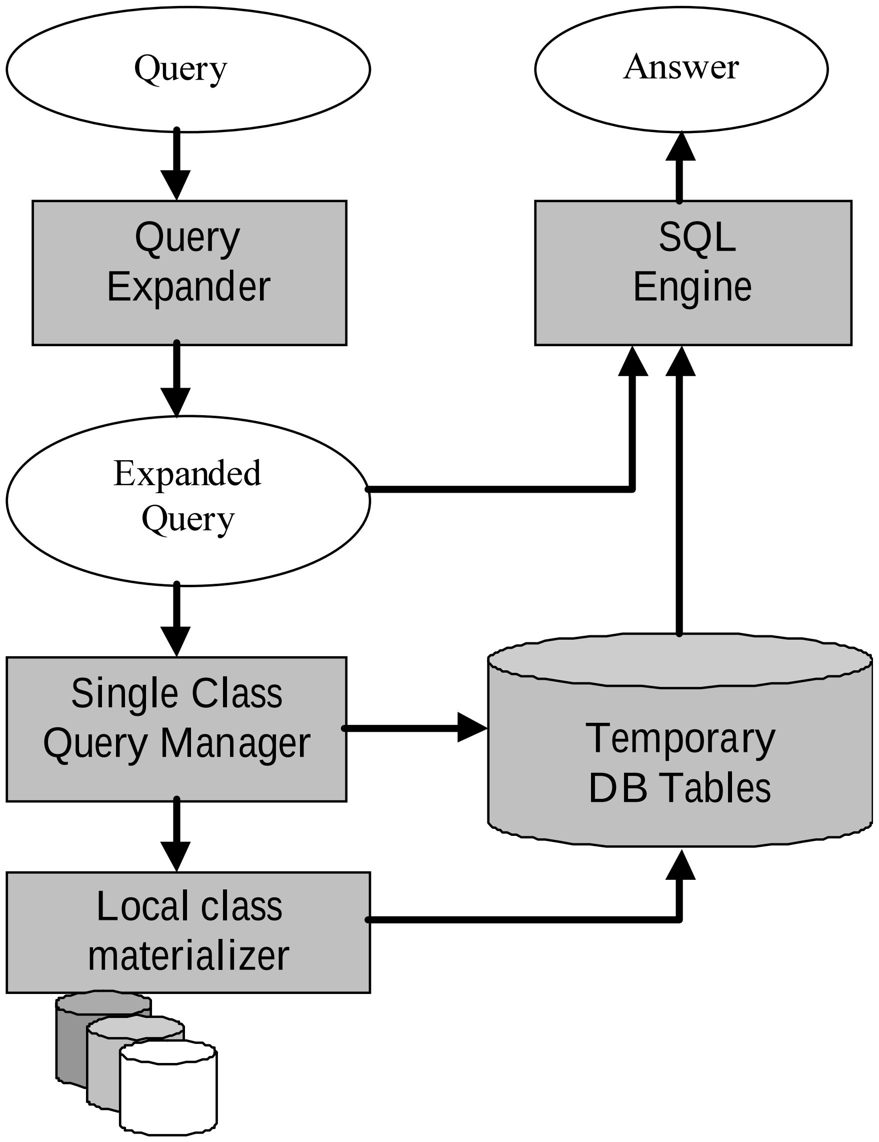

Figure 2: Query processing within one SINode

3. **Local class materialization**: in order to materialize the global classes, another materialization step is required, namely the one that retrieves the data from the external sources, and stores them into the local classes.

4. **Evaluation of the expanded query**: when all the global classes that are relevant for the query are materialized, the expanded query is submitted to the SQL Engine to obtain the answer of the original query.

In our prototype, we have built one module for each of the above phases. The following sections are devoted to a brief explanation of the various modules. We also refer to an exampe on which the demo of the system is based.

The prototype has been developed in Java 1.3 (compliant with Java 1.2 version), and uses Microsoft SQL Server as DBMS.

3 Query expander

3.1 Description

Query expansion amounts to rewriting the query Q posed to a global schema $\mathcal{G}$ into a new query Q', in such a way that all the knowledge about the constraints in $\mathcal{G}$ has been "compiled" into Q'. We remind the reader that the intial query is a conjunctive query, whereas the output query is a union of conjunctive queries (which can obviously be expressed in Datalog form).

3.2 Demo

We first describe the global schema of the demo in the relational form. Such a schema results from the tanslation of a corresponding object-oriented schema. The relational schema is the following (in the expression of a foreign key constraint, numbers indicate the positions of the attributes):

```
Category(CatCode,Description,Sector)
Enterprise(Name,Description,Address,PhoneNo,Email,Web,Contact)
BusinessOrganization(Name,Province,BuiltYear,NoEmployees,Turnover)
BusinessOrganizationCat(Name,CatCode)
Manufacturer(Name,Owners,WorkingHours)
```

```
KeyConstraint(Category,1)
KeyConstraint(Enterprise,1)
KeyConstraint(BusinessOrganization,1)
ForeingKey(BusinessOrganization,1,Enterprise,1)
KeyConstraint(BusinessOrganizationCat,1)
ForeignKey(BusinessOrganizationCat,1,BusinessOrganization,1)
ForeignKey(BusinessOrganizationCat,2,Category,1)
KeyConstraint(Manufacturer,1)
ForeignKeyInclusion(Manufacturer,1,BusinessOrganizationCat,1)
```

The input query aims at retrieving name and addresses of enterprises whose category has a sector named "IT" (information technology). The SQL version of the query is:

```
SELECT e.Name, e.Address
FROM   Enterprise e, BusinessOrganization b, BusinessOrganizationCat c, Category d
WHERE e.Name=b.Name and e.Name=c.Name and b.Name=c.Name and c.CatCode=d.CatCode
      and d.Sector='IT';
```

The Datalog version is:

```
Q(X13,X3):-BusinessOrganization(X13,X9,X10,X11,X12),
           BusinessOrganizationCat(X13,X15),
           Category(X15,X16,"IT"),Enterprise(X13,X2,X3,X4,X5,X6,X7).
```

The result of the expansion of the query is the following Datalog query:

```
Q(X13,X3):-BusinessOrganization(X13,_,_,_,_),BusinessOrganizationCat(X13,X15),
           Category(X15,_,"IT"),Enterprise(X13,_,X3,_,_,_,_).
Q(X13,X3):-BusinessOrganizationCat(X13,X15),BusinessOrganizationCat(X13,_),
           Category(X15,_,"IT"),Enterprise(X13,_,X3,_,_,_,_).
Q(X13,X3):-BusinessOrganizationCat(X13,X15),Category(X15,_,"IT"),
           Enterprise(X13,_,X3,_,_,_,_),Manufacturer(X13,_,_).
Q(X13,X3):-BusinessOrganizationCat(X13,X15),Category(X15,_,"IT"),
           Enterprise(X13,_,X3,_,_,_,_).
```

It should be noted how the query expansion step has “compiled” all the foreign key constraints in the new query. As we said before, the expanded query can now be processed by ignoring the constraints in the schema. The result on the correctness of the expansion process ensures us that we do not loose any information by doing so.

4 Global class materializer

4.1 Description

The expanded query is processed in order to single out the global classes that need to be materialized in order to answer the query. Besides deciding which global classes to materialize, this module also computes, for each such global class C, the query that should be evaluated in order to get the correct instances of C. Finally, this module is in charge of computing the temporary relations holding the instances resulting from the materialization of the chosen global classes. This process is obviously guided by the mapping between the global classes and the local classes.

4.2 Demo

In our demo, the global classes to be materialized, together with the query to be issued on such classes for the materialization, is shown in the following (each query is given a name, that is the name of the temporary relation to be used in the final process of query evaluation):

```
Global Class Query OurRelation_00:
SELECT R.Name, R.Province, R.BuiltYear, R.NoEmployees, R.Turnover
FROM BusinessOrganization R;

Global Class Query OurRelation_01:
SELECT R.Name, R.CatCode
FROM BusinessOrganizationCat R;

Global Class Query OurRelation_02:
SELECT R.CatCode, R.Description, R.Sector
FROM Category R
WHERE (R.Sector = "IT");

Global Class Query OurRelation_03:
SELECT R.Name, R.Description, R.Address, R.PhoneNo, R.Email, R.Web, R.Contact
FROM Enterprise R;

Global Class Query OurRelation_20:
SELECT R.Name, R.Owners, R.WorkingHours
FROM Manufacturer R;
```

5 Local class materialization

As we said before, one of the task of the global class materializer is to single out the local classes that are relevant for the evaluation of the expanded query.
Since we are following the Global-as-views approach, this task is straightforward: in order to single out the local classes that are relevant for the evaluation of the expanded query, it is sufficient to look at the mapping from the global classes mentioned in the expanded query to the local classes.

Thus, the mapping tells us which are the relevant local classes for the evaluation of the query: such local classes should now be materialized, by accessing the corresponding sources.

In the current protoype, the local class materialization is done in an ad-hoc manner. The second release of the prototype will address this issue in more detail.

6 Evaluation of the expanded query

6.1 Description

The expanded query (union of conjunctive queries) is expressed in SQL, and is then evaluated over the global database computed by the global class materializer. This expanded query is submitted to the SQL Engine, and the resulting tuples are given as result of the overall process of query evaluation within the SINode.

6.2 Demo

The final query is the following (note that the names of the global class queries are used in the definition of the query):

```
SELECT R1.Name, R4.Address
FROM OurRelation_00 R1, OurRelation_01 R2, OurRelation_02 R3, OurRelation_03 R4
WHERE (R1.Name = R2.Name) and (R2.Name = R4.Name) and (R2.CatCode = R3.CatCode)
UNION
SELECT R1.Name, R4.Address
FROM OurRelation_01 R1, OurRelation_01 R2, OurRelation_02 R3, OurRelation_03 R4
WHERE (R1.Name = R2.Name) and (R2.Name = R4.Name) and (R2.CatCode = R3.CatCode)
UNION
SELECT R1.Name, R4.Address
FROM OurRelation_20 R1, OurRelation_01 R2, OurRelation_02 R3, OurRelation_03 R4
WHERE (R1.Name = R2.Name) and (R2.Name = R4.Name) and (R2.CatCode = R3.CatCode)
UNION
SELECT R1.Name, R3.Address
FROM OurRelation_01 R1, OurRelation_02 R2, OurRelation_03 R3
WHERE (R1.Name = R3.Name) and (R1.CatCode = R2.CatCode)
```