Final release of the system prototype for query management

(FINAL, 29/04/2005)

Abstract – The purpose of this document is to describe the most important features of the final release of the system prototype for query management in the Sewasie system, and to give an account of the techniques for query management within Sewasie in the context of several brokering agents.
# Document information

<table>
<thead>
<tr>
<th>Document ID code</th>
<th>D3.5</th>
</tr>
</thead>
<tbody>
<tr>
<td>Keywords</td>
<td>Query Manager, Query reformulation, Brokering Agent, Query Manager Prototype</td>
</tr>
<tr>
<td>Classification</td>
<td>FINAL</td>
</tr>
<tr>
<td>Date of reference</td>
<td>29/04/2005</td>
</tr>
<tr>
<td>Distribution level</td>
<td>Sewasie Partners of the Sewasie consortium, and EU commission</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Editor</th>
<th>Maurizio Lenzerini</th>
<th>UNIROMA</th>
</tr>
</thead>
<tbody>
<tr>
<td>Authors</td>
<td>Maurizio Lenzerini</td>
<td>UNIROMA</td>
</tr>
<tr>
<td>Reviewer</td>
<td>Christoph Quix</td>
<td>AACHEN</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Date</th>
<th>Version</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>28/02/2005</td>
<td>1</td>
<td>Draft version</td>
</tr>
<tr>
<td>21/04/2005</td>
<td>2</td>
<td>Draft version</td>
</tr>
<tr>
<td>29/04/2005</td>
<td>3</td>
<td>Final version</td>
</tr>
</tbody>
</table>

## Copyright notices

© 2002 SEWASIE Consortium. All rights reserved. This document is a project document of the SEWASIE project. All contents are reserved by default and may not be disclosed to third parties without the written consent of the SEWASIE partners, except as mandated by the European Commission contract IST-2001-34825 for reviewing and dissemination purposes.

All trademarks and other rights on third party products mentioned in this document are acknowledged as owned by the respective holders.
# Contents

1 Executive summary 4

2 Query management in the context of SEWASIE 5

3 Query processing within one SiNode: Summary 5

4 Query processing in the presence of one brokering agent: Summary 8
   4.1 The Mechanical scenario 10
   4.2 Query expansion 11
      4.2.1 Query expansion Example 12
   4.3 Evaluation of the EXPAtoms 13
   4.4 Evaluation of the expanded query 13

5 Single Class Query Processing 13
   5.1 Mapping Table and Data Conversion 14
   5.2 Object Identification: Join Conditions 14
   5.3 Data Reconciliation: Resolution Functions 15
      5.3.1 Homogeneous Attributes 16
   5.4 Object Fusion: Full Disjunction 16
      5.4.1 Full Disjunction Computation 16
   5.5 Query rewriting 17
      5.5.1 Atomic constraint mapping 17
      5.5.2 Atomic constraint mapping for Homogeneous Attribute 17
      5.5.3 Select-list computation 18

6 System prototype for Query Management 19
   6.1 Play Maker 20
      6.1.1 Expander 20
      6.1.2 Unfolder 20
   6.2 Query Agent 21
   6.3 SiNode Agent 22

7 Query management with more than one brokering agent 22
   7.1 Framework 26
   7.2 Semantics 26
   7.3 Interactions between mappings in P2P systems 26
   7.4 Query answering in P2P systems 26

8 Conclusions 26

A Query in XML Format 28
1 Executive summary

This document illustrates the final prototype implementation of the techniques designed for query management within the Sewasie system.

The first release of the prototype was described in the Deliverable D3.3, and was concerned with query management within a single SINode.

The second release, described in Deliverable D3.4, dealt with the problem of query management in the case where different SINodes are connected to a so-called brokering agent. Starting from a specified brokering agent, the query agent initiates a process constituted by a series of interactions with the various brokering agents, with the goal of getting information on the matter of interest from the SINodes managed by these brokering agents. It follows that the query agent should be able both to interact with relevant brokering agents, and to ask the various SINodes the right information.

In this document, we address the case of multiple brokering agents. The interaction between different brokering agents can be effectively realized only if the brokering agents are somehow mutually linked by means of mappings. Thus, put in a simple way, the problem we are addressing is the one of answering queries posed to a network of brokering agents, each one linked to the other ones in unpredictable and dynamic way. We will give an account of both the basic ideas to solve this problem, and the corresponding techniques and their formal properties.
2 Query management in the context of SEWASIE

As we illustrated in [30], in Sewasie, the query agent is the carrier of the user query from the user interface to the SINodes, where concrete data are located. One basic task of the query agent is to interact with the brokering agents, in order to solve the query. Starting from a specified brokering agent, the query agent initiates a process constituted by a series of interactions with the various brokering agents, with the goal of getting information on the matter of interest from the SINodes managed by these brokering agents (see Section 6 of [28]).

A typical interaction between a query agent and a brokering agent may imply that the brokering agent will provide directions to relevant SINodes and information on SINode contents, or reference the query agent to other brokering agents. The query agent will then move to such SINodes and query them, or may move on to the other brokering agents to ask them for directions again. During this process, the query agent is informed by the brokering agents about which SINodes contain relevant data, so that the query agent may access such SINodes, collect partial answers and integrate them. It follows that the query agent should be able to:

1. interact with relevant brokering agents,
2. ask the various SINodes the right information, according to a specific query plan.

Item (2) above was the subject of Deliverable D3.3, produced in May 2003, where we dealt with the problem of answering a query posed to an SINode. This task is carried out by the query manager of the SINode of interest, and is characterized as follows: Given a query posed in terms of the virtual global view associated to the SINode, derive a query plan that is able to correctly access the data sources under the control of that SINode, and execute the query according to this plan.

The design of techniques to be used by the query agent in order to interact with a given brokering agent was addressed in Deliverable D3.4, produced in May 2004. These techniques formed the basis for the second release of the system prototype for query management.

The purpose of the present document is address the case of multiple brokering agents. We will give an account of both the basic ideas to solve this problem, and the corresponding techniques and their formal properties.

In the next section (Section 3), we recall the most important features of query processing within one SINode. Obviously, this is a crucial building block for query processing in Sewasie. Section 4 is devoted to a summary of the techniques designed for the interaction between the query agent and one brokering agent. The description of such an interaction directly provides an account for query processing in Sewasie in the case of one brokering agent. The case of multiple brokering agents is the subject of Section 5, where both the basic ideas to solve this problem, and the corresponding techniques are described.

3 Query processing within one SINode: Summary

A query issued to an SINode is managed by the query manager associated to such SINode. The **Query Manager** of an SINode 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.

For all the issues regarding the structure of the SINodes, especially those related to the representation of the ontology managed by SINodes, we refer the reader to [29].

We conceive a single SINode as a data integration system based on a global schema. In other words, each SINode combines the data residing at different sources, and provide the external user with a unified view of these data. Such a unified view is represented by the global schema (also called Global Virtual View), and provides a reconciled view of all data, which can be queried...
by the user. Obviously, one of the main tasks in the design of an SINode is to establish the mapping between the sources and the global schema. Such a mapping should be suitably taken into account in formalizing the notion of SINode.

It follows that, from the perspective of the query manager, the main components of an SINode are the global schema, the sources, and the mapping. Therefore, we formalize a SINode as follows. An SINode $I$ is a triple $\langle G, S, M \rangle$, where

- $G$ is the **global schema**, expressed in a language $L_G$ over an alphabet $A_G$.
- $S$ is the **source schema**, expressed in a language $L_S$ over an alphabet $A_S$. The alphabet $A_S$ includes a symbol for each source accessible by the SINode.
- $M$ is the **mapping** between $G$ and $S$.

Next, we discuss how the three components are specialized in the context of Sewasie.

**Global schema** The global schema is expressed in terms of ODLI3 (see Deliverable [29]).

**Source schema** The source schema is again expressed in terms of classes. Therefore, the language $L_S$ in our context is again ODLI3, and the alphabet $A_S$ is the set of local class name. It is important to note that local classes are internal entities representing external sources, in the sense that each local class has a link with a corresponding external source, where external data are located. The correspondence between local classes and external sources is realized by means of a further layer (that we can call the **wrapping layer**), that associates to each local class a query over the external sources.

**Mapping** In Sewasie, the various SINodes follow the Global-as-view (GAV) approach. In the GAV approach, the mapping $M$ associates to each element $g$ in $G$ a query $q_S$ over $S$. The following properties further characterize how the mapping is specified in Sewasie.

- To every global class $C$ a set of local classes is associated. Only data coming from such local classes will be used to get instances for the class $C$.
- Sources are considered sound, which implies that the global classes may have more instances than those retrieved from the data of the local sources, according to the mapping.
- The query associated to $C$ retrieves the data ensuring that, in every instance of the global schema, no two objects with the same value of the key of $C$ exist. This property is particularly important, because it implies that the mapping incorporates data cleaning/transformation functions, that are activated when data are loaded into the global classes.

As for the semantics of an SINode, a database (DB) for a schema $G$ is simply a collection of sets of objects, one set for each global class in the alphabet of $G$. Note that, since every global class has a key, there is an obvious correspondence between objects of a global class $C$ and tuples of the relation represented by $C$: indeed, there is one object for every value of the key.

In order to assign semantics to an SINode $I = \langle G, S, M \rangle$, we start by considering a **local database for** $I$, i.e., a database $D$ that conforms to the local classes of $S$. Based on $D$, we now specify which is the information content of the global schema $G$. We call **global database** for $I$ any database for $G$.

A global database $B$ for $I$ is said to be **legal with respect to** $D$, if:

- $B$ is legal with respect to $G$, i.e., $B$ satisfies all the constraints of $G$;
- $B$ satisfies the mapping $M$ with respect to $D$. 
The notion of \( B \) satisfying the mapping \( M \) with respect to \( D \) depends on the approach adopted for assigning meaning to the mapping. In the GAV approach, the one adopted in our context, the mapping \( M \) associates to each element \( g \) in \( G \) a query \( q_S \) over \( S \). Therefore, a GAV mapping is a set of assertions, one for each element \( g \) of \( G \), of the form

\[
q_S \rightsquigarrow g
\]

**Definition 1** A database \( B \) satisfies the assertion \( q_S \rightsquigarrow g \) with respect to a local database \( D \) if

\[
q_S^D \subseteq q_B^D
\]

where \( q_B^D \) is the result of evaluating the query \( q_S \) over the source database \( D \).

Queries to \( I \) are posed in terms of the global schema \( G \), and are expressed in a query language \( L_Q \) over the alphabet \( A_G \). A query is intended to provide the specification of which data to extract from the virtual database represented by the SINode. We will restrict our attention to unions of conjunctive queries. In other words, the language \( L_Q \) in the context of Sewasie is the language of unions of conjunctive queries. Formally, a conjunctive query \( q \) of arity \( n \) is written in the rule-based form

\[
q(x_1, \ldots , x_n) \leftarrow \text{body}(x_1, \ldots , x_n, y_1, \ldots , y_m)
\]

where:

- \( q \) belongs to a new alphabet \( Q \) (the alphabet of queries),
- \( \text{body}(x_1, \ldots , x_n, y_1, \ldots , y_m) \) is a conjunction of atoms involving the variables \( x_1, \ldots , x_n \), \( y_1, \ldots , y_m \), and a set of constants from \( I \), and
- the predicate symbols of the atoms are in \( G \).

Note that, since all variables \( x_1, \ldots , x_n \) in the head appear also in the body, we are dealing with safe conjunctive queries.

The answer to a conjunctive query \( q \) of arity \( n \) over a database \( DB \), denoted \( q^B \), is the set of \( n \)-tuples of constants \( (c_1, \ldots , c_n) \), such that, when substituting each \( c_i \) for \( x_i \), the formula

\[
\exists (y_1, \ldots , y_n). \text{body}(x_1, \ldots , x_n, y_1, \ldots , y_m)
\]

evaluates to true in \( DB \). Note that the answer to \( q \) over \( DB \) is a relation whose arity is equal to the arity of the query \( q \). Finally, the answer to a union of conjunctive queries over a database \( B \) is the union of the answers to the component conjunctive queries.

We now specify the semantics of queries posed to an SINode. As we said before, such queries are expressed in terms of the symbols in the global schema of \( I \).

**Definition 2** If \( q \) is a query of arity \( n \) and \( DB \) is a database, we denote with \( q^{DB} \) the set of tuples (of arity \( n \)) in \( DB \) that satisfy \( q \).

**Definition 3** Given a local database \( D \) for \( I \), the answer \( q^{I,D} \) to a query \( q \) with respect to \( I \) and \( D \), is the set of tuples \( t \) such that \( t \in q^B \) for every global database \( B \) that is legal for \( I \) with respect to \( D \), i.e. such that \( t \) is an answer to \( q \) over every database \( B \) that is legal for \( I \) with respect to \( D \). The set \( q^{I,D} \) is called the set of certain answers to \( q \) in with respect to \( I \) and \( D \).

The process carrying out query answering has been the subject of Deliverable [31]. Here, we simply recall that query processing within one SINode is constituted by four phases. The first phase expands the query according to the integrity constraints in the global schema. In the second phase, the atoms in the expanded query are materialized by taking into account the mapping to the local classes. 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: this is done in the third phase. Finally, 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.
4 Query processing in the presence of one brokering agent: Summary

In Sewasie, the brokering agent holds knowledge about both a set of SINodes, and a set of other brokering agents. When a query is posed to the system, the query agent starts its process by interacting with the brokering agent that is able to provide answers to the query. In this section, we describe the interaction between the query agent and the brokering agent, under the assumption that the brokering agent is the only one in the system.

We define the structure of the Sewasie system in the case where one brokering agent is available. For the purpose of this deliverable, the most important components of the brokering agent are the Brokering Agent Ontology, and the mapping between such an ontology and the SINodes known to the brokering agents.

A Sewasie system \( \mathcal{W} \) is constituted by:

- A Brokering Agent Ontology \( \mathcal{O} \), which is a schema expressed in ODLI3, with alphabet \( \mathcal{A}_\mathcal{O} \).
- A set \( \mathcal{N} \) of SINodes, each one of the form stated in Section 2.
- A set \( \mathcal{M}_\mathcal{O} \) of GAV mapping assertions between \( \mathcal{O} \) and \( \mathcal{N} \), where each assertion associates to an element \( g \) in \( \mathcal{O} \) a query \( q_{\mathcal{N}} \) over the global schemas of a set of SINodes in \( \mathcal{N} \).

Intuitively, the ontology is the intensional representation of the information provided by the brokering agent to the external world, whereas the mapping specifies how such an intensional representation relates to the data managed by the SINodes. The following properties further characterize how the mapping is specified.

- To every class \( C \) of \( \mathcal{O} \), a (possibly empty) set of classes of the global schemas of a set of SINodes in \( \mathcal{N} \) is associated. More precisely, to every class \( C \) in \( \mathcal{O} \) the associated query \( q_{\mathcal{N}} \) over the global schemas of a set of SINodes in \( \mathcal{N} \) is characterized as follows:
  - \( q_{\mathcal{N}} \) is a union of atoms, where each atom is a class that appears in the global schema of an SINode \( N \) in \( \mathcal{N} \).
  - Only data coming from such classes will be used to get instances for the class \( C \).
- \( q_{\mathcal{N}} \) incorporates a suitable cleaning function that ensures that, in every instance of \( \mathcal{O} \), no two objects with the same value of the key of \( C \) exist.

In order to assign semantics to a Sewasie System \( \mathcal{W} = (\mathcal{O}, \mathcal{N}, \mathcal{M}_\mathcal{O}) \) (where \( \mathcal{N} = \{N_1, \ldots, N_n\} \)), we start by considering a local database for \( \mathcal{W} \), i.e., a database \( \mathcal{D} \) that

- is the disjoint union of \( n \) databases \( D_1, \ldots, D_n \),
- each \( D_i \) is a local database for \( N_i \), i.e., conforms to the local classes of the SINodes in \( \mathcal{N} \).

Based on \( \mathcal{D} \), we now specify which is the information content of the BAO \( \mathcal{O} \). We call BA database for \( \mathcal{W} \) any database for \( \mathcal{O} \). A BA database \( \mathcal{B} \) for \( \mathcal{W} \) is said to be legal with respect to \( \mathcal{D} \), if:

- \( \mathcal{B} \) is legal with respect to \( \mathcal{O} \), i.e., \( \mathcal{B} \) is coherent with the implicit and the explicit constraints of \( \mathcal{O} \);
- there is a set of databases \( \mathcal{B}_{\mathcal{N}} = \{B_1, \ldots, B_n\} \) for \( N_1, \ldots, N_n \) respectively, such that
  - each \( B_i \) is a legal database for \( N_i \) wrt \( D_i \),
  - \( \mathcal{B} \) satisfies each assertion in \( \mathcal{M}_\mathcal{O} \) wrt \( B_{\mathcal{N}} \).
The notion of $B$ satisfying the mapping $M_O$ with respect to $B_N$ depends on the approach adopted for assigning meaning to the mapping. In the GAV approach, the one adopted in our context, the vmapping $M_N$ associates to each element $g$ in $O$ a query $q_G$ over the schemas $g_1, \ldots, g_n$ of the SINodes $N_1, \ldots, N_n$. Therefore, a GAV mapping is a set of assertions, one for each element $g$ of $O$, of the form

$$q_G \sim g$$

Let $W = (O, N = \{N_1, \ldots, N_n\}, M_O)$ be a Sewasie system, let $B_N = B_1, \ldots, B_n$ be $n$ global databases for $N_1, \ldots, N_n$, and let $D$ be a local database for $W$. A BA database $B$ for $W$ satisfies the assertion $q_G \sim g$ with respect to $B_N$ if

$$q_G^{B_N} \subseteq g^B$$

where $q_G^{B_N}$ is the result of evaluating the query $q_G$ over the global databases $B_N = B_1, \ldots, B_n$.

Roughly speaking, the logical characterization of GAV is therefore through the first order assertions

$$\forall x \ q_G(x) \rightarrow g(x)$$

It is interesting to observe that the implicit assumption in many GAV proposals is the one of exact views. Indeed, in a setting where all the views are exact, there are no constraints in the global schema, and a first order query language is used as $L_{M,S}$, a GAV data integration system enjoys what we can call the “single database property”, i.e., it is characterized by a single database, namely the global database that is obtained by associating to each element the set of tuples computed by the corresponding view over the sources. This motivates the widely shared intuition that query processing in GAV is easier than in LAV. However, it should be pointed out that the single database property only holds in such a restricted setting.

In particular, the possibility of specifying constraints in $\mathcal{G}$ greatly enhances the modeling power of GAV systems, especially in those situations where the global schema is intended to be expressed in terms of a conceptual data model, or in terms of an ontology [26]. In these cases, the language $L_{Q}$ is in fact sufficiently powerful to allow for specifying, either implicitly or explicitly, various forms of integrity constraints on the global database.

Queries managed by the query agent are expressed in terms of the brokering ontology $O$, and are formulated in a query language $L_{Q}$ over the alphabet $A_{O}$. Queries are communicated by the Sewasie User Interface in the form described in Deliverable 6.3.

A query is intended to provide the specification of which data to extract from the virtual database represented intensionally by the brokering agent ontology. As in the case of a single SINode, we will restrict our attention to unions of conjunctive queries. In other words, the language $L_{Q}$ in the context of Sewasie is the language of unions of conjunctive queries. We remind the reader that the definition of conjunctive query is reported in Section 3 (subsection 3.3). Here, we simply recall that a conjunctive query $q$ of arity $n$ is written in the rule-based form

$$q(x_1, \ldots, x_n) \leftarrow body(x_1, \ldots, x_n, y_1, \ldots, y_m)$$

where:

- $q$ belongs to a new alphabet $Q$ (the alphabet of queries),
- $body(x_1, \ldots, x_n, y_1, \ldots, y_m)$ is a conjunction of atoms involving the variables $x_1, \ldots, x_n$, $y_1, \ldots, y_m$, and a set of constants from $\Gamma$,
- the predicate symbols of the atoms are in $\mathcal{G}$.

Also, we remind the reader that the answer to a union of conjunctive queries over a database $DB$ is the union of the answers to the component conjunctive queries over the same database.

We now specify the semantics of queries posed to a Sewasie system $W$. As we said before, such queries are expressed in terms of the symbols in the Brokering Agent Ontology $O$ of $W$. 
Given a local database $D$ for $W$, the answer $q^{W:D}$ to a query $q$ with respect to $VV$ and $D$, is the set of tuples $t$ such that $t \in q^B$ for every BA database $B$ that is legal for $W$ with respect to $D$. The set $q^{W:D}$ is called the set of certain answers to $q$ with respect to $W$ and $D$.

Note that, from the point of view of logic, finding certain answers is a logical implication problem: check whether it logically follows from the information on the sources that $t$ satisfies the query.

The algorithm for processing a query posed over a Brokering Agent Ontology should take into account that the data to be collected are managed by the SINodes known to the Brokering Agent. In turn, the data managed by each SINode are actually stored in a set of data sources that are connected to the SINode by means of the internal mapping under the control of the SINode itself.

It follows that query processing should be based on two query reformulation steps:

1. Reformulation w.r.t. the BA ontology: this step should reformulate the query in terms of the SINodes known by the Brokering Agent; as we will discuss in section 6.1 this reformulation is performed at the level of Brokering Agent by the Playmaker component.

2. Reformulation w.r.t. the SINode ontologies: this step should reformulate each of the SINode query obtained in the first step in terms of the data sources known by the SINodes; as we will discuss in section 6 this reformulation is performed at the level of a single SINode Agent by the SINode Query Manager.

Taking into account all the above observations, we can conclude that query processing is constituted by the following phases:

1. Query expansion: the query posed in terms of the brokering agent ontology is expanded to take into account the explicit and implicit constraints in the brokering agent ontology O. Relevant subqueries (called EXPAtoms) are extracted from the expanded query (Subquery identification step); an EXPAtom is a Single Class Query (SCQ), i.e., a query related on a single Global Class of the BA-GVV.

2. Evaluation of the EXPAtoms: the atoms in the expanded query are materialized by taking into account the mapping from the classes in the GVVS of the SINodes to the classes of the BA-GVV.

As an EXPAtom is a Single Class Query, the EXPAtom Processing is a Single Class Query Processing. In particular, EXPAtom is a SCQ on a mapped Global Class, i.e., a Global Class mapped (by means of its mapping table) in the classes of the SINodes known by the Brokering Agent.

3. Evaluation of the expanded query: when all its atoms are materialized, the expanded query is submitted to the Query Engine to obtain the answer of the original query.

In the following, we give an account of the most important features of each of the above specified phases. Then, in section 6 we will discuss how this query processing has been implemented in the final release of the prototype for Query Manager, in particular by using a step by step example of how the query is managed along the process. In order to consider an example, next section introduces The Mechanical scenario.

4.1 The Mechanical scenario

In the Mechanical Domain we have the following Scenario (see figure 1). First of all, for each ontology (Brokering Agent and SINodes) we consider the Global Virtual View (GVV) and the related mappings. The Brokering Agent GVV (BA-GVV) is mapped (mapping $m_1$) into the GVVs of the underlying Sinodes. Then each SINode-GVV is mapped (mapping $m_2$) into the schemata of the underlying Data Sources.

The complete ontologies (in the SEWASIE format) are at the following links:
In this deliverable, we will give an example of mappings in section 5.1.

4.2 Query expansion

Query expansion amounts to rewriting the query $Q$ posed to the Brokeraing Agent Ontology $O$ into a new query $Q^\prime$, in such a way that all the knowledge about the constraints in $G$ has been “compiled” into $Q^\prime$. The goal of the rewriting step in our approach is made explicit in the following definition. The definition makes use of the notions reported in the following definition.

Let $W = \langle O, N = \{N_1, \ldots, N_n\}, M(O) \rangle$ be a Sewasie system, and let $B_N = B_1, \ldots, B_n$ be $n$ global databases for $N_1, \ldots, N_n$. Then

- the retrieved BA database $M(O)(B_N)$ is the BA database obtained by computing, for every element of $O$, the query associated to it by $M(O)$ over the global databases $B_1, \ldots, B_n$;
the certain answers $q^{W,B_N}$ to a query $q$ with respect to $W$ and $B_N$, is the set of tuples $t$ such that $t \in q^B$ for every BA database $B$ that is legal with respect to $O$, and satisfies each assertion in $M_O$ wrt $B_N$.

Let $W = \langle O, N = \{N_1, \ldots, N_n\}, M_O \rangle$ be a Sewasie system, and let $Q$ be a query (union of conjunctive queries) over $O$. Then $Q$ is called a perfect rewriting of $Q$ wrt $W$ if, for every global database $B_N = B_1, \ldots, B_n$ (where $B_1, \ldots, B_n$ are $n$ global databases for $N_1, \ldots, N_n$), $Q^{W,B_N} = Q^{M_O(B_N)}$.

We remind the reader that, according to what we said before, we assume that, for every $B_N$, the retrieved BA database $M_O(B_N)$ satisfies all key constraints.

Let $W = \langle O, N = \{N_1, \ldots, N_n\}, M_O \rangle$ be a Sewasie system, and let $Q$ be a query (union of conjunctive queries) over $O$. Then $Q_1$ is called a perfect rewriting of $Q$ wrt $W$ if, for every global database $B_N = B_1, \ldots, B_n$ (where $B_1, \ldots, B_n$ are $n$ global databases for $N_1, \ldots, N_n$), $Q^{W,B_N} = Q^{M_O(B_N)}$.

As we said before, we assume that the mapping is defined in such a way that no key constraint is violated when we load data from the local classes to the global classes. It follows that key constraints are taken into account by the global class materializer, in particular through the cleaning strategy it implements. Therefore, key constraints in $O$ can be safely ignored in expanding the query.

The algorithm for computing the perfect rewriting of $Q$ wrt $W$ that we use in Sewasie is reported in [27]. This algorithm is based on the results presented in [7], but is adapted to the fact that the ontology $O$ used in the brokering agent is expressed in ODL instead of the relational model.

4.2.1 Query expansion Example

We consider an example of query formulated by the End User Query tool which generates a conjunctive query; this query is expressed in an internal format by using a XML format.

The XML format for our query is in the Appendix A; for our discussion we can consider the query in an SQL-like format:

```
SELECT C.NAME, C.CAPITAL_STOCK, C.REGION, C.ADDRESS, C.SUBCONTRACTOR, C.COMPANY_ID
FROM company C, list_of_category L, Processes_plastic_and_rubber P
WHERE C.COMPANY_ID = L.COMPANY_ID
AND P.CATEGORY_ID = L.CATEGORY_ID
AND C.CAPITAL_STOCK > '50'
AND C.REGION LIKE 'VENETO'
AND C.SUBCONTRACTOR LIKE 'yes'
```

The Query expansion process produces the following EXPQuery and set of EXPAtoms:

**EXPQuery:**

```
SELECT r2.NAME, r2.COMPANY_ID, r2.CAPITAL_STOCK, r2.REGION, r2.SUBCONTRACTOR, r2.ADDRESS
FROM scq1 r1, scq2 r2, scq3 r3
WHERE r1.CATEGORY_ID = r3.CATEGORY_ID AND r2.COMPANY_ID = r3.COMPANY_ID
UNION
SELECT r2.NAME, r2.COMPANY_ID, r2.CAPITAL_STOCK, r2.REGION, r2.SUBCONTRACTOR, r2.ADDRESS
FROM scq4 r1, scq2 r2, scq3 r3
WHERE r1.CATEGORY_ID = r3.CATEGORY_ID AND r2.COMPANY_ID = r3.COMPANY_ID
UNION
SELECT r2.NAME, r2.COMPANY_ID, r2.CAPITAL_STOCK, r2.REGION, r2.SUBCONTRACTOR, r2.ADDRESS
FROM scq5 r1, scq2 r2, scq3 r3
WHERE r1.CATEGORY_ID = r3.CATEGORY_ID AND r2.COMPANY_ID = r3.COMPANY_ID
UNION
SELECT r2.NAME, r2.COMPANY_ID, r2.CAPITAL_STOCK, r2.REGION, r2.SUBCONTRACTOR, r2.ADDRESS
FROM scq6 r1, scq2 r2, scq3 r3
WHERE r1.CATEGORY_ID = r3.CATEGORY_ID AND r2.COMPANY_ID = r3.COMPANY_ID
UNION
```

**EXPAtoms:**

```
```
The EXPQuery is an union of conjunctive queries (in the above example, we show just some of these conjunctive queries); for our query, EXPQuery contains a conjunctive query for each subclass of Processes_plastic_and_rubber class; for example, for the Mould_making subclass we have

\[
\begin{align*}
\text{SELECT } & \text{r2.NAME, r2.COMPANY_ID, r2.CAPITAL_STOCK, r2.REGION, r2.SUBCONTRACTOR, r2.ADDRESS} \\
\text{FROM } & \text{scq1 r1, scq2 r2, scq3 r3} \\
\text{WHERE } & \text{r1.CATEGORY_ID=r3.CATEGORY_ID AND r2.COMPANY_ID=r3.COMPANY_ID}
\end{align*}
\]

with \text{scq1: SELECT CATEGORY_ID FROM Mould_making}).

4.3 Evaluation of the EXPAtoms

An EXPAtom is a Single Class Query on a mapped Global Class, i.e., a Global Class mapped in the classes of the SINodes known by the Brokering Agent. To explain the evaluation of the EXPAtoms we need to take into account how the mapping from the classes in the global schemas of the SINodes to the classes of the Brokering agent ontology are specified. We follow the approach proposed for the MOMIS, for the Single Class Query Processing, discussed in section 5.

4.4 Evaluation of the expanded query

The expanded query (union of conjunctive queries) is now evaluated over the BA database computed by the EXPAtoms Processing. This expanded query is submitted to the Query Engine, and the resulting tuples are given as result of the overall process of query evaluation.

5 Single Class Query Processing

In this section, we concentrate on the evaluation of a Global Class Query (GCQ). First of all, for each global class \( C \) of the BA-GVV we define the query \( q_N \) related to \( C \). Then, the evaluation of GCQ can be performed by unfolding GCQ w.r.t. to the classes of the underlying Sinodes (in the following we will refer to these classes as local classes).

As stated before,

- to every class \( C \) of \( O \), a (possibly empty) set of classes of the global schemas of a set of SINodes in \( N \) is associated. More precisely, to every class \( C \) in \( O \) the associated query \( q_N \) over the global schemas of a set of SINodes in \( N \) is characterized as follows:
  - \( q_N \) is a union of atoms, where each atom is a class that appears in the global schema of an SINode \( N \) in \( N \).
  - Only data coming from such classes will be used to get instances for the class \( C \).
- \( q_N \) incorporates a suitable cleaning function that ensures that, in every instance of \( O \), no two objects with the same value of the key of \( C \) exist.

We follow the approach proposed for the MOMIS system where \( q_N \) is defined on the basis of the following elements:
**Mapping Table**: define the mapping between the global class attributes and the local class attributes.

**Join condition**: We assume that there exists a Join Condition between each pair of overlapping relations to identify tuples corresponding to the same object and fuse them.

**Full Disjunction**: $q_N$ is defined in such a way that it contains a unique tuple resulting from the merge of all the different tuples representing the same real world object. This problem is related to that of computing the natural outerjoin of many relations in a way that preserves all possible connections among facts [48]. Such a computation has been termed as full disjunction by Galindo Legaria [45].

### 5.1 Mapping Table and Data Conversion

The Mapping Table is a table whose columns represent the local classes, which belong to the Global Class and whose rows represent the global attributes. An element $MT[GA][LC]$ represents the set of local attributes of $LC$ which are mapped onto the global attribute $GA$. How local attributes are mapped onto the global attribute $GA$ is defined by means of **Data Conversion Functions**: for each element $MT[GA][LC]$ that is not a null we define a Data Conversion Function, denoted by $MTF[GA][L]$, which represents how the local attributes of $L$ are mapped to the global attribute $GA$. The function $MTF[GA][L]$ must be a function that is executable/supported by the local source of the class $L$. For example, for relational sources, $MTF[GA][L]$ is an SQL value expression; the following defaults hold: if $MTF[GA][L] = LA$ then $MTF[GA][L] = LA$ and, if $MTF[GA][L]$ contains more than one string attribute, then $MTF[GA][L]$ is the string concatenation.

Figure 2 shows part of the Mapping Table of the global class Company of Brockering Agent GVV in the Mechanical Scenario. The global class Company groups the local class Company of SINode1 and the local class Company of SINode2; note that, at the level of Brockering Agent, a local class is a class of the underlying SINode GVVs. At the level of a SINode, we have that (we consider SINode2), the global class SN2.company is mapped into the local class S1.aziende (where S1 is the Data Source TUTTOSTAMPI) and in the local class S2.company (where S2 is the Data Source DEFORMAZIONE).

### 5.2 Object Identification: Join Conditions

Merging data from different sources requires different representations of the same real world object to be identified; this process is called **object identification** [47]. The topic of object identification is currently a very active area of research and there is significant contribution both from the artificial intelligence [49] and database communities [40,42] on this problem.

To identify instances of the same object and fuse them we introduce **Join Conditions** among pairs of local classes belonging to the same global class. Given two local classes $L_1$ and $L_2$ belonging to $C$, a Join Condition between $L_1$ and $L_2$, denoted with $JC(L_1, L_2)$, is an expression over $L_1.A_i$ and $L_2.A_j$, where $A_i (A_j)$ are global attributes with a not null mapping in $L_1$ ($L_2$); the set of attributes $L.A$ used to express the join conditions for $L$ is called **Join Attribute Set** of $L$ and it is denoted with $JA(L)$.

The set of join conditions between pairs of local classes of the global class $C$, called **Join Map**, is denoted with $JM$; a **join condition** between $L_1, L_2 \in L$ will be denoted with $JM L_1 L_2$.

The designer can explicitly define the Join Map $JM(C)$ or can use some predefined expressions; for example, the actual implementation of the system allows the designer to define a join condition which holds up for each pair of local classes belonging to a the Global Class as follows: selecting the global attributes $JA_1, \ldots, JA_n$ the following equality join condition ($EJC$) holds up for each pair of local classes $L_1$ and $L_2$ belonging to the Global Class:

$$EJC(L_1, L_2) : L_1.A_1 = L_2.A_1 \text{ and } \ldots \text{ and } L_1.A_n = L_2.A_n$$
5.3 Data Reconciliation: Resolution Functions

After having solved the object identification problem, we have to perform the fusion of data coming from different sources taking into account the problem of inconsistent information among sources [41, 44, 47, 46]. In the context of MOMIS we use Resolution Function [47]. A Resolution Function [47] may be defined for each global attribute mapping onto local attributes coming from several local sources, in order to solve data conflicts due to different local attribute values; in this way we can define what value shall appear in the result. Our system provides some standard kinds of resolution functions:

- **Random function**: this function results in having the content of the one of the local attribute randomly chosen.

- **Aggregation functions for numerical attributes**: SUM, AVG, MIN, MAX, . . .

- **Precedence function**: The highest informational quality value on the basis of an information quality model.

- **Coalesce function**: The first not null value among the local attributes values is the function result.
5.3.1 Homogeneous Attributes

If the designer knows that there are no data conflicts for a global attribute mapped onto more than one source (that is, the instances of the same real object in different local classes have the same value for this common attribute), he can define this attribute as an **Homogeneous Attribute**; this is the default in our system. Of course, we do not need to define a Resolution Function for homogeneous attributes. A global attribute mapped onto one source is a particular case of homogeneous attribute.

In our example, we define all the global attributes as Homogeneous Attributes except for the Global Attribute Address where we use the Precedence Function

\[
\text{precedence}(\$\{\text{SI-\textit{NMAgent1}.company.ADDRESS.main}\}, \$\{\text{SI-\textit{NMAgent2}.company.ADDRESS.main}\})
\]

5.4 Object Fusion: Full Disjunction

In our approach, for each global class, \( q_N \) is defined by means of the Full Disjunction [45] that has been recognized as providing a natural semantics for data merging queries [48]. The informal definition of Full Disjunction is the following: Computing the natural outerjoin of many relations in a way that preserves all possible connections among facts. We apply this definition to our context, then, given a global class \( C \) composed of \( L_1; L_2; \ldots; L_n \), the instance of \( C \) is the full-disjunction of \( L_1; L_2; \ldots; L_n \), computed on the basis of the **Join Conditions** (in other words, we use the Join Conditions instead of the natural outer join). In the case of two classes, the Full Disjunction corresponds to the full outer join.

5.4.1 Full Disjunction Computation

Our goal is to compute the Full Disjunction by means of an SQL query (in particular, by means of the outerjoin operator) by using results and techniques proposed in the literature [48, 45]. In [48] was demonstrated that there is a natural outerjoin sequence producing the full disjunction if and only if the set of relation schemes forms a connected, acyclic hypergraph. If we apply the framework of [48] to our context, it is easy to verify that a Global Class with more than 2 local classes is a cyclic hypergraph, then we cannot use the algorithm proposed in [48].

An important point in our definition of the full-disjunction is the following: to ensure that, in every instance of the Ontology, no two objects with the same value of the key of \( C \) exist. To this end, we assume that

1. each \( L \) contains a key
2. all the **join conditions** are on key attributes
3. all the join attribute are mapped in the same global attribute, say \( X \).

Then, it can be demonstrated that:

1. \( X \) is a key of \( C \)
2. the full disjunction can be computed as follows: (this SQL query is called **FDQuery**):

\[
\begin{align*}
\text{select} & \ast \\
\text{from} & (L_1 \text{ full join } L_2 \text{ on JM}(L_1,L_2)) \\
& \text{full join } L_3 \text{ on } (JM(L_1,L_3) \text{ OR JM}(L_2,L_3)) \\
& \text{...} \\
& \text{full join } L_n \text{ on } (JM(L_1,L_n) \text{ OR JM}(L_2,L_n) \text{ OR } ... \text{ OR JM}(L_n,1,L_n))
\end{align*}
\]
5.5 Query rewriting

The query rewriting method depends on the approach used to model the mappings between the GVV and the local schemata: MOMIS uses a global-as-view (GAV) approach, then the global query is rewritten by means of unfolding, that is, by expanding each atom of the global query according to its definition in the mapping. In the following example we describe our query rewriting method.

Given a global class \( C \), related to the local class \( L_1, L_2, \ldots, L_n \), we consider a Single Global Query \( Q \) over \( C \):

\[
Q = \text{select } <Q\text{-select-list}> \text{ from } G \text{ where } <Q\text{-condition}>
\]

where \( <Q\text{-condition}> \) is a Boolean expression of positive atomic constraints: \((GA_1 \text{ op } \text{value})\) or \((GA_1 \text{ op } GA_2)\), where \( GA_1 \) and \( GA_2 \) are attributes of the global class \( C \).

As an example, we consider scq2 (see page 12)

\[
\text{scq2: SELECT NAME,CAPITAL_STOCK,REGION,ADDRESS,SUBCONTRACTOR,COMPANY_ID FROM company WHERE CAPITAL_STOCK > '50' AND REGION LIKE 'VENETO' AND SUBCONTRACTOR LIKE 'yes'}
\]

The query rewriting process is made up of the following steps:

1. Atomic constraint mapping
2. Select-list computation

The output of the query rewriting process is a set of Single Class Queries over the SINOde GVVs:

\[
\text{FDAtom} = (\text{SINode}, Q = \text{select } <Q\text{-select-list}> \text{ from } C \text{ where } <Q\text{-condition}>)
\]

where \( C \) is a Global Class of the SINOde.C GVV. These queries will be denoted by \( \text{FDAtom} \) (in fact, these queries will be used as atoms in the Full Disjunction expression).

5.5.1 Atomic constraint mapping

In this step, each atomic constraint of \( Q \) is rewritten into one that can be supported by the local class.

The atomic constraint mapping is performed on the basis of mapping functions defined in the Mapping Table (see section 5.1). Moreover, the atomic constraint mapping depends on the definition of Resolution Functions for global attributes (see section 5.3); for example, if the numerical global attribute \( GA \) is mapped onto \( L_1 \) and \( L_2 \), and we define AVG function as resolution function, the constraint \((GA = \text{value})\) cannot be pushed at the local sources, because of the AVG function has to be calculated at a global level, the constraint may be globally true but locally false. In this case, the constraint is mapped as true in both the local sources. On the other hand, if \( GA \) is an homogeneous attribute the constraint can be pushed at the local sources.

5.5.2 Atomic constraint mapping for Homogeneous Attribute

For homogeneous attributes the constraint mappings are defined as follows:

- An atomic constraint \((GA \text{ op value})\) is mapped onto the local class \( L \) as:

\[
(MTF[GA][L] \text{ op value}) \text{ if } MTF[GA][L] \text{ is not null and}
\]

\[
\text{true otherwise}
\]

\[
\text{where op operator is supported into } L
\]
An atomic constraint \((GA_1 \text{ op } GA_2)\) is mapped onto the local class \(L\) as:

\[
(MTF[GA_1][L] \text{ op } MTF[GA_2][L]) \quad \text{if } \quad MTF[GA_1][L] \text{ and } MTF[GA_2][L] \text{ are not null}
\]

\[
\text{true} \quad \text{otherwise}
\]

By default, each operator used in the global query is supported into a local class: the designer can define, the operators which are not supported for each local class. The current implementation of the system assumes that each operator, \(OP\), used in the global query is supported into a local class, i.e. a constraint including \(OP\) can be solved in local class. As a future work, we plan to extend our framework by considering a general mechanism for translating constraint queries such as the one proposed in [43].

For non homogeneous attributes the constraint mappings must be performed on the basis of the resolution functions; in general, an atomic constraint defined for non homogeneous attributes cannot be rewritten in the local sources and is rewritten as true (this is the method currently adopted in our system).

As an example, for the atomic constraints in scq2, by considering the mapping table in figure 2, we have

- \(\text{CAPITAL\_STOCK} > '50'\) (\text{CAPITAL\_STOCK} is an homogeneous attribute):
  - in \(SN_1\) is mapped in \(\text{CAPITAL\_STOCK} > '50'\)
  - in \(SN_2\) is mapped in \text{true}

- \(\text{REGION LIKE 'VENETO'}\) (\text{REGION} is an homogeneous attribute):
  - in \(SN_1\) is mapped in \(\text{REGION LIKE 'VENETO'}\)
  - in \(SN_2\) is mapped in \(\text{REGION LIKE 'VENETO'}\)

- \(\text{SUBCONTRACTOR LIKE 'yes'}\) (\text{SUBCONTRACTOR} is an homogeneous attribute):
  - in \(SN_1\) is mapped in \text{true}
  - in \(SN_2\) is mapped in \(\text{SUBCONTRACTOR LIKE 'yes'}\)

### 5.5.3 Select-list computation

The select-list of a FDAtom over the local class \(L\) is computed by considering the union of

1. the global query attributes in \(<\text{Q\_select\_list}>\) with a not null mapping in \(L\)
2. the Join Attribute Set of \(L J A(L)\). i.e. the set of attributes used to express the join conditions for \(L\) (see section 5.2)
3. the global attributes in \(<\text{Q\_condition}>\) with a not null mapping in \(L\)

This set of global attributes is the transformed in the corrispondin set of local attributes on the basis of the mapping table.

Now, the query rewriting process is completed and the set of FDAtoms is computed; for example, for \(scq2\) we obtain:

\[
\text{FDATOM1} = (SN_1, \text{SELECT COMPANY\_ID, NAME, REGION, ADDRESS, CAPITAL\_STOCK}}
\text{FROM company}
\text{WHERE ((CAPITAL\_STOCK) > ('50') and (REGION) like ('VENETO'))})
\]
FDATOM2 = ( SN2, SELECT COMPANY_ID, NAME, REGION, ADDRESS, SUBCONTRACTOR
FROM company
WHERE ((REGION) like ('VENETO') and (SUBCONTRACTOR) like ('yes'))
)

A FDAtom FDAtom over L is sent and executed to the SINode including the class L; the
answer is transformed by applying the Data conversion functions related to L and the result of
this conversion is materialized in a temporary table; we denote this temporary table with FDAtom.
In realta’ il nome di questa tabella e’ assegnato in modo univoco dal sistema.

When all the FDAtom answers have materialized, we fuse and reconcile them into the global
answer as follows:

1. we compute the full disjunction as discussed in section 5.4; in particular, we apply the
FDQuery introduced at page 16 by considering as select list of this query all the local
attribute that are mapped into the global attribute in the select list of the original query
(<Q_select-list>). As an example, for scq2:

```
select FDATOM1.COMPANY_ID AS COMPANY_ID_1,
FDATOM2.COMPANY_ID AS COMPANY_ID_2,
FDATOM1.NAME AS NAME_1,
FDATOM2.NAME AS NAME_2,
FDATOM1.REGION AS REGION_1,
FDATOM2.REGION AS REGION_2,
FDATOM1.ADDRESS AS ADDRESS_1,
FDATOM2.ADDRESS AS ADDRESS_2,
FDATOM1.CAPITAL_STOCK AS CAPITAL_STOCK_1,
FDATOM2.SUBCONTRACTOR AS SUBCONTRACTOR_1
from FDATOM2 full outer join FDATOM1
on (((FDATOM1.COMPANY_ID) = (FDATOM2.COMPANY_ID)))
```

2. we apply the Resolution Functions, as follows:
   - for Homogeneous Attributes (e.g. REGION) we can take one of the related values (indifferently REGION_1 or REGION_2);
   - for non Homogeneous Attributes (e.g. ADDRESS) we apply the related Resolution Functions (in this case the precedence function).

6 System prototype for Query Management

In this section we will discuss how the proposed query processing has been implemented in the
final release of the prototype for Query Manager, in particular by using a step by step example
of how the query is managed along the process.

Figure3 shows the functional architecture of the last release of the system prototype for Query
Management. The starting point of the query processing is a query, formulated by the Query Tool
Interface.

The components involved in the Query Processing are:

**Playmaker** (section 6.1)

This is the component of the Brokering Agent which is in charge to perform the reformulation
of the query w.r.t. the BA ontology.

**Query Agent** (section 6.2)

The coordination of all the query processing is performed by the Query Agent, which ac-
cepts the query from the Query Tool Interface (or the Negotiation Tool), interacts with both
the Brokering Agent and the SINode Agents, and returns the result as a materialized view in the SEWASIE_DB.
**SINode Agent** (section 6.3)

One of the modules of the SINode Agent, the **SINode Query Manager**, accepts a query and reformulates it according to the semantics of the SINode Ontology, and returns the result to the Query Agent.

### 6.1 Play Maker

The **Playmaker** component performs the reformulation of the query w.r.t. the BA ontology. The **Playmaker** has two components: the **Expander** and the **Unfolder**.

#### 6.1.1 Expander

This component performs the Query expansion discussed in section 4.2: the query posed in terms of the BA GVV is expanded by taking into account the constraints in the BA Ontology, so that the expanded query (EXPQuery) can be processed by ignoring constraints.

The output of this component is:

1. EXPQuery: this query will be in the message sent to the Query Agent . . .
2. the EXPAtoms: these queries will be the input of the **Unfolder** component

#### 6.1.2 Unfolder

The Unfolder component performs, for each EXPAtom, the query unfolding and rewriting discussed in section 5.

The output of this component is, for each EXPAtom,

1. the full disjunction expression **FDQuery**: this query will be sent to the Query Agent.
2. the atoms of FDQuery, **FDAAtoms**: these queries will be sent to the Query Agent.
3. the **resolution functions** related to the attributes involved in the EXPAtom; these functions will be sent to the Query Agent.

Once the execution of the Play Maker is completed, the output of the Play Maker computation is sent from the BA to the QA with a single message.

### 6.2 Query Agent

In the following the 3 steps of the Query Processing of the Query Agent are discussed.

1. **Parallel Execution of the FDAtoms.**

   For each FDAtom (Parallel Execution):
   
   - **INPUT**: FDAtom
   - **MESSAGES**: from QA to SINode Agent
   - **OUTPUT**: a table storing the FDAtom result in the SEWASIE_DB

   We said that a FDAtom is a query on a single class of an SINode, then a message is sent from QA to this SINode Agent (this SINode Agent is uniquely determined by the name of the global class used in the FDAtom). The message contains both the FDAtom query (an SQL expression) and the name of the table (and other parameters for the JDBC connection to the SEWASIE_DB) where the FDAtom result will be stored.

2. **Fusion.** For each EXPATom (Parallel Execution):

   - **INPUT**: FDAtoms, FDQuery, Resolution Functions
     
     (a) Execution of FDQuery (Full Disjunction of the FDAtoms)
     (b) Application of the Resolution Functions on the result of previous action
   
   - **OUTPUT**: a view storing the EXPAtom result in the SEWASIE_DB

   We said that the unfolding of an EXPAtom is a set of FDAtoms. Then, once the execution of all FDAtoms related to an EXPAtom is completed (when all the FDAtom results related to an EXPAtom are available and stored in the related tables), we can perform the Full Disjunction of these FDAtoms in order to obtain the EXPAtom result; moreover, Resolution Functions are applied at this point. In this execution, the Query Agent uses the SQL Engine of the SEWASIE_DB and the EXPAtom result is stored as a materialized view in the SEWASIE_DB.

3. **Final result.** For each EXPATom (Parallel Execution):

   - **INPUT**: Output of the FUSION step
     
     (a) Execution of the Expanded Query
   
   - **OUTPUT**: Final Query result view stored in the SEWASIE_DB

   Once the execution of all EXPATom is completed and their result is in the SEWASIE_DB, we can execute the Expanded Query by means of the SQL Engine of the SEWASIE_DB and then we obtain the query result as a materialized view in the SEWASIE_DB. A this point, the Query Agent will send a message to the Query Tool Interface with the name of this materialized view.
6.3 SINode Agent

One of the modules of the SINode Agent, the SINode Query Manager, accepts a query and reformulates it according to the semantics of the SINode Ontology, and returns the result to the Query Agent.

We discussed the SINode Query Manager prototype in the Deliverable D3.3 First release of the system prototype for query management. In this last version of the prototype, the main new aspects are from an architectural point of view: the SINode Query Manager is fully integrated within an SINode Agent. In particular, a query sent to the SINode Agent is executed by the SINode Query Manager and the result of this execution is stored in the common repository (SEWASIE_DB).

7 Query management with more than one brokering agent

In this section, we describe the main approach and the main techniques we have devised to deal with the case of several interacting brokering agents.

Since we have already described the Sewasie approach dealing with one brokering agent, here we will simply address all issues related the interaction of several brokering agents, without referring to the behavior of a single brokering agent.

The interaction between different brokering agents can be effectively realized only if the brokering agents are somehow mutually linked by means of mappings. Thus, put in a simple way, the problem we are addressing is the one of answering queries posed to a network of brokering agents, each one linked to the other ones in unpredictable and dynamic way.

Figure 4 shows the typical situation we are referring to. We have a network of brokering agents, each with a set of mappings to other brokering agents. Each brokering agent is also linked to a set of SINodes by means of “local” mappings. A query is posed to a single brokering agent, and such brokering agent is supposed to answer the query by both accessing the SINodes which is linked to, and by using the mappings to other brokering agents.
Note that this is exactly the scenario envisioned in what is called peer-to-peer data integration (see figure 5). Thus, we will use in the following a terminology where the term “peer” will replace the term “brokering agent”.

In recent years, the issue of cooperation, integration, and coordination between information nodes in a networked environment has been addressed in different contexts, including data integration [20], the Semantic Web [17], Peer-to-Peer and Grid computing [2, 16], service oriented computing and distributed agent systems [24, 18]. Put in an abstract way, all these systems are characterized by an architecture constituted by various autonomous nodes (called sites, sources, agents, or, as we call them here, peers) which hold information, and which are linked to other nodes by means of mappings.

Here, we study data integration in such Peer-to-Peer (P2P) systems. Each peer in the system provides part of the overall information available from a distributed environment, without relying on a single global view, and acts both as a client and as a server in the system. Moreover, the various nodes adopt a suitable infrastructure for managing information. The P2P paradigm was made popular by Napster, which employed a centralized database with references to the information items (files) on the peers. Gnutella, another well-known P2P system, has no central database, and is based on a communication-intensive search mechanism. More recently, a Gnutella-compatible P2P system, called Gridella [1], has been proposed, which follows the so-called Peer-Grid (P-Grid) approach. A P-Grid is a virtual binary tree that distributes replication over community of peers and supports efficient search. P-Grid’s search structure is completely decentralized, supports local interactions between peers, uses randomized algorithms for access and search, and ensures robustness of search against node failures.

As pointed out in [15], current P2P systems focus strictly on handling semantic-free, large-granularity requests for objects by identifier, which both limits their utility and restricts the techniques that might be employed to distribute the data. These current sharing systems are largely
limited to applications in which objects are described by their name, and exhibit strong limitations in establishing complex links between peers. To overcome these limitations, data-oriented approaches to P2P have been proposed recently [16, 2, 15]. For example, in the Piazza system [15], data origins serve original content, peer nodes cooperate to store materialized views and answer queries, nodes are connected by bandwidth-constrained links and advertise their materialized views to share resources with other peers.

Differently from the traditional setting, integration in data-oriented P2P systems is not based on a global schema. Instead, each peer represents an autonomous information system, and information integration is achieved by establishing P2P mappings, i.e., mappings among the various peers. Queries are posed to one peer, and the role of query processing is to exploit both the data that are internal to the peer, and the mappings with other peers in the system. To stress the data-oriented nature of the framework, we assume that the various peers export data in terms of a suitable schema, and mappings are established among such peer schemas. A peer schema is therefore intended to export the semantics of information as viewed from the peer.

One of the main issue in formalizing data oriented P2P systems is the semantic characterization of P2P mappings. In this section, we argue that, although correct from a formal point of view, the usual approach of resorting to a first-order logic interpretation of P2P mappings (followed e.g. by [9, 16, 2]), has several drawbacks, both from the modeling and from the computational perspective. In particular we analyze three central desirable properties of P2P systems:

- **Modularity**: i.e., how autonomous are the various peers in a P2P system with respect to the semantics. Indeed, since each peer is autonomously built and managed, it should be clearly interpretable both alone and when involved in interconnections with other peers. In particular, interconnections with other peers should not radically change the interpretation of the concepts expressed in the peer.

- **Generality**: i.e., how free we are in placing connections (P2P mappings) between peers. This is a fundamental property, since actual interconnections among peers are not under the control of any actor in the system.

- **Decidability**: i.e., are sound, complete and terminating query answering mechanisms available? If not, it becomes critical to establish basic quality assurance of the answers returned by the system.

We show that these desirable properties are weakly supported by approaches based directly on FOL semantics. Indeed, such approaches essentially consider the P2P system as a single flat logical theory. As a result, the structure of the system in terms of peers is actually lost and remote interconnections may propagate constraints that have a deep impact on the semantics of a peer (see Section 7.3). Moreover, under arbitrary P2P interconnections, query answering under the first-order semantics is undecidable, even when the single peers have an extremely restricted structure. Motivated by these observations, several authors proposed suitable limitations to the form of P2P mappings, such as acyclicity, thus giving up generality to retain decidability [16, 19, 11].

To overcome the above drawbacks, we propose a new semantics for P2P systems, with the following aims:

- **We want to take into account that peers in our context are to be considered autonomous sites that exchange information. In other words, peers are modules, and the modular structure of the system should be explicitly reflected in the definition of its semantics.**

- **We do not want to limit a-priori the topology of the mapping assertions among the peers in the system. In particular, we do not want to impose acyclicity of assertions.**

- **We seek for a semantic characterization that leads to a setting where query answering is decidable, and possibly, polynomially tractable.**
We base our proposal of a new semantics for P2P systems on epistemic logic, and we show that the new semantics is clearly superior to the usual FOL semantics with respect to all three properties mentioned above.

The new semantics is weaker than FOL, but it nicely reflects the modularity of the system. Let us illustrate this aspect with the help of an example. In Figure 6 an example is shown with three brokering agents. The schema of the first one, called $P_1$, contains a generalization that partitions the instances of Parent into Mother and Father. Also, we assume that the data stored in the SNodes connected to $P_1$ implies that $d$ is a Parent and $e$ is a Father. The concept Father in $P_1$ is mapped to the concept Man of the brokering agent $P_2$, whereas Mother in $P_1$ is mapped to the concept Woman of $P_2$. Finally, both Man and Woman in $P_2$ are mapped to the concept Person of the brokering agent $P_3$. We now analyze how the brokering agent $P_3$ answers the query Person($x$).

Obviously, $e$ is an answer to the query in both semantics. As for $d$, we distinguish between the two semantics. In FOL, we have to consider two possible models of $P_1$ in order to answer correctly to the query. According to the first model (Figure 7), $d$ is an instance of Mother, and therefore an instance of Woman, and therefore an instance of Person. According to the second model (Figure 8), $d$ is an instance of Father, and therefore an instance of Man, and therefore an instance of Person. Since in both cases, $d$ is an instance of Person, $d$ is an answer to the query under the FOL semantics.

In our new semantics, $P_1$ does not know whether $d$ is an instance of Father or Mother. It follows that no knowledge is transferred from $P_1$ to $P_2$ about $d$. Consequently, no knowledge is transferred from $P_2$ to $P_3$ about $d$, and $d$ is not an answer to the query (see Figure 9).

As for the method for answering queries, for fairly general P2P systems, we devise a top-down query answering algorithm that is based on a recursive (Datalog) reformulation of the query posed to one of the peer of the P2P system, and that is polynomial time with respect to the size of data stored in the peers. Notably, our technique can be applied every time peers are able to do query answering based on reformulation; this ability is at the base of several recent results on data integration [7, 4, 3].
The rest of the section is organized as follows. In subsection 7.1, we introduce a framework that captures a very general architecture for P2P systems, and then, in subsection 7.2, we define both the first-order and the epistemic semantics of such a framework. In subsection 7.3, we discuss the issues of modularity, generality, and decidability under the two semantics, and in subsection 7.4 we study decidability and complexity of query processing in P2P systems under the epistemic semantics. Finally, we draw some conclusions in subsection 8.

### 7.1 Framework

inputPODS2004paper/2-framework

### 7.2 Semantics

inputPODS2004paper/3-semantics

### 7.3 Interactions between mappings in P2P systems

inputPODS2004paper/4-modularity

### 7.4 Query answering in P2P systems

inputPODS2004paper/5-query-processing

### 8 Conclusions

In this document we have summarized the approach to query processing adopted in Sewasie, and we have provided an account of our techniques for dealing with the case of multiple brokering agents.
In particular, in Section 5 we have proposed a new semantics for P2P systems, and have argued that it is more suitable than the commonly adopted semantics based on FOL. We have also presented a sound, complete, and terminating procedure that, under general conditions, can generate a Datalog program that returns the certain answers to a query posed to the P2P system. Notice that our query answering technique can immediately be applied in all contexts in which each peer is able to compute the perfect reformulation of queries, and in particular in the architecture of Sewasie.

According to the FOL semantics, Person(d) is true in all cases, and therefore is a certain answer to \( \{ x \mid \text{Person}(x) \} \)

**Figure 8: FOL semantics: model 2**
Figure 9: Epistemic semantics

According to the epistemic semantics, Person(d) is not a certain answer to \( \{ x \mid \text{Person}(x) \} \), since neither \( K_2 \text{Man}(d) \) nor \( K_3 \text{Woman}(d) \) is valid, and therefore, \( K_4 \text{Person}(d) \) is not valid too.

A Query in XML Format

The following is an example query in XML format:

```xml
<query id="sid://MechanicBA/query#1A0F4736CBEB6D867ED637C913D874170">
  <select>
    <variable name="X_0" label="name of company" type="VARCHAR(360)" />
    <variable name="X_1" label="company id of company" type="INTEGER" />
    <variable name="X_2" label="capital stock of company" type="FLOAT" />
    <variable name="X_3" label="region of company" type="VARCHAR(360)" />
    <variable name="X_4" label="subcontractor of company" type="VARCHAR(360)" />
    <variable name="X_5" label="address of company" type="VARCHAR(360)" />
  </select>
  <where>
    <term predicate="interface#globalSource.company">
      <variable name="X" />
    </term>
    <term predicate="attribute#globalSource.company.NAME.main">
      <variable name="X" />
      <variable name="X_0" />
    </term>
    <term predicate="attribute#globalSource.company.COMPANY_ID.main">
      <variable name="X" />
      <variable name="X_1" />
    </term>
    <term predicate="attribute#globalSource.company.CAPITAL_STOCK.main">
      <variable name="X" />
      <variable name="X_2" />
    </-term>
    <builtin predicate="greater">
      <variable name="X_2" />
      <constant value="50" />
    </builtin>
    <term predicate="attribute#globalSource.company.REGION.main">
      <variable name="X" />
      <variable name="X_5" />
    </term>
  </where>
</query>
```
References


