# An ASP-Based Data Integration System

Nicola Leone, Francesco Ricca, and Giorgio Terracina

Dipartimento di Matematica, Università della Calabria, 87030 Rende, Italy
{leone,ricca,terracina}@mat.unical.it

**Abstract.** The task of an information integration system is to combine data residing at different sources, providing the user with a unified view of them, called *global schema*. Simple data integration scenarios have been widely studied and efficient systems are already available. However, when some constraints are imposed on the quality of the global data, the integration process becomes difficult and, often, it may provide ambiguous results. Important research efforts have been spent in this area, but no actual system efficiently implementing the corresponding techniques is available yet. This paper is intended to be a step forward in this direction; it proposes a new data integration system, based on Answer Set Programming (ASP) and many optimizations, allowing to carry out consistent query answering (CQA) over massive amounts of data.

## 1 Introduction

The task of an *information integration system* is to combine data residing at different sources, providing the user with a unified view of them, called *global schema*. Users formulate queries over the global schema, and the system suitably queries the sources, providing an answer. Users are not obliged to have any information about the sources.

Recent developments in IT such as the expansion of the Internet have made available to users a huge number of information sources, generally autonomous, heterogeneous and widely distributed: as a consequence, information integration has emerged as a crucial issue in many application domains, e.g., distributed databases, cooperative information systems, data warehousing, or on-demand computing.

However, information integration is, in general, an extremely complex task. Both state-of-the-art commercial software solutions (e.g., [1]) and academic systems (see e.g. [2,3] for a survey) fulfill only partially the ambitious goal of integrating information in complex application scenarios. Moreover, comprehensive, formal methodologies and coherent tools for designing information integration systems are still missing.

The main objectives a data integration system should address are the following:

1. **A comprehensive information model**, through which the knowledge about the integration domain can be easily specified. The possibility of defining expressive integrity constraints (ICs) over the global schema, the precise characterization of the relationship between global schema and the local data sources, the formal definition of the underlying semantics, as well as the use of a powerful query language, are mandatory for the specification of complex integration applications.

E. Erdem, F. Lin, and T. Schaub (Eds.): LPNMR 2009, LNCS 5753, pp. 528–534, 2009.

2. **Capability of dealing with data that may result inconsistent** with respect to global ICs. Even if some solutions to the problem of query answering for inconsistent data (e.g., [3,4,5]) have been already proposed, they did not produce so far effective and scalable system implementations, mainly due to the high computational complexity of the problem.
3. **Advanced information integration algorithms**; these should provide a formal correspondence between the data integration system and the expected query answers, especially in the handling of inconsistent data.
4. **Mass-memory-based evaluation strategies**. In fact, real world scenarios involve massive amounts of data; thus, techniques that require all-in-memory evaluations would not provide the needed scalability.

The system proposed in this paper is an attempt to address the above issues; it starts from the experience we gained in the INFOMIX [6] project to overcome some limitations experienced in real-world scenarios. In fact, it is based on Answer Set Programming (ASP) and exploits datalog-based methods for answering user queries, which are sound and complete with respect to the semantic of query answering. This guarantees meaningful data integration and solves issue 1. It incorporates a number of optimization techniques that "localize" and limit the inefficient computation, due to the handling of inconsistencies, to a very small fragment of the input. This allows obtaining fast query-answering, even in such a powerful data-integration framework, thus solving issue 2. The problem of consistent query answering (CQA) is reduced to cautious reasoning on disjunctive datalog programs, which allows to effectively compute the query results precisely, by using state-of-the-art disjunctive datalog systems. The formal query semantics is captured also in presence of inconsistent data. This solves issue 3. Finally, the system adopts DLV$^{DB}$ [9,11] as internal query evaluation engine, which allows for mass-memory evaluations and distributed data management features, and solves issue 4.

In the following sections we introduce some details on the key components of the system and its overall architecture.

## 2 Key System Components

In our setting, a data integration system $\mathcal{I}$ is a triple $\langle \mathcal{G}, \mathcal{S}, \mathcal{M} \rangle$, where $\mathcal{G}$ is the global schema, which provides a uniform view of the information sources to be integrated, $\mathcal{S}$ is the source schema, which comprises the schemas of all the sources to be integrated, and $\mathcal{M}$ is the mapping establishing a relationship between $\mathcal{G}$ and $\mathcal{S}$. $\mathcal{G}$ may contain integrity constraints (ICs). $\mathcal{M}$ is a *Global-As-View* (GAV) mapping [5], i.e., $\mathcal{M}$ is a set of logical implications $\forall x_1 \cdots \forall x_n . \Phi_{\mathcal{S}}(x_1, \ldots, x_n) \supset g_n(x_1, \ldots, x_n)$, where $g_n$ is a relation from $\mathcal{G}$, $n$ is the arity of $g_n$, $\Phi_{\mathcal{S}}$ is a conjunction of atoms on $\mathcal{S}$ and $x_1, \ldots, x_n$ are the free variables of $\Phi_{\mathcal{S}}$. Each global relation is thus associated with a *union of conjunctive queries* (UCQs).Both $\mathcal{G}$ and $\mathcal{S}$ are assumed to be represented in the relational model, whereas $\mathcal{M}$ is represented as a set of datalog rules.

As an example consider a bank association that desires to unify the databases of two branches. The first database models managers by using a table $man(code, name)$ and employees by a table $emp(code, name)$, where code is a primary key for both tables.

The second database stores the same data in table $employees(code, name, role)$. Suppose that the data has to be integrated in a global schema with two tables: $m(code)$, and $e(code, name)$, having both code and name as keys and the inclusion dependency $m[code] \subseteq e[code]$, indicating that manager codes must be employee codes. GAV mappings are defined as follows:

$$e(C, N) :- emp(C, N). \quad e(C, N) :- employee(C, N, \_).$$

$$m(C) :- man(C, \_). \qquad m(C) :- employee(C, \_, man).$$

If $emp$ stores $(e1, john)$, $(e2, mary)$, $(e3, willy)$, $man$ stores $(e1, john)$, and $employees$ stores $(e1, ann, man)$, $(e2, mary, man)$, $(e3, rose, emp)$, it is easy to verify that, while the source databases are consistent w.r.t. local constraints, the global database obtained by evaluating the mappings violates the key constraint on $e$ (e.g. both $john$ and $ann$ have the same code $e1$ in table $e$). Basically, when data are combined in a unified schema with its own integrity constraints the resulting global database might be inconsistent; any query posed on an inconsistent database would then produce an empty result.

In this context, user queries must be re-modelled according to the mappings and violated constraints, in order to compute *consistent* answers, i.e. answers which consider as much as possible of *correct* input data. This task is accomplished in our system by the CQA Rewriter component.

Another key component of the system is the Query Evaluator, which is in charge of actually evaluating, over source databases, the queries posed over the global schema. These two key components are described in some detail in the following subsections.

### 2.1 CQA Rewriter

In the field of data-integration several notions of consistent query answering have been proposed (see [3] for a survey), depending on whether the information in the database is assumed to be *correct* or *complete*. Basically, the incompleteness assumption coincides with the *open world assumption*, where facts missing from the database are not assumed to be false. In our system, we assume that sources are complete; as argued in [8], this choice strengthens the notion of minimal distance from the original information.[1] Moreover, there are two important consequences of this choice: integrity restoration can be obtained by only *deleting tuples*; and, computing CQA for conjunctive queries remains *decidable* even for arbitrary sets of denial constraints and inclusion dependencies [8].

More formally, given a global schema $\mathcal{G}$ and a set $C$ of integrity constraints, let $\mathcal{DB}$ and $\mathcal{DB}^r$ be two global database instances. $\mathcal{DB}^r$ is a *repair* [8] of $\mathcal{DB}$ w.r.t. $C$, if $\mathcal{DB}^r$ satisfies all the constraints in $C$ and the instances in $\mathcal{DB}^r$ are a maximal subset of the instances in $\mathcal{DB}$. Basically, given a conjunctive query $Q$, *consistent answers* are those query results that are not affected by axioms violations and are true in any possible repair [8]. Thus, given a database instance $\mathcal{DB}$ and a set of constraints $C$, a conjunctive query $Q$ is consistently true in $\mathcal{DB}$ w.r.t. $C$ if $Q$ is true in every repair of

[1] It is worth noting that, in relevant cases like denial constraints, query results coincide for both correct and complete information assumptions.

$\mathcal{DB}$ w.r.t. $C$. Moreover, if $Q$ is non-ground, the consistent answers to $Q$ are all the tuples $\overline{t}$ such that the ground query $Q[\overline{t}]$ obtained by replacing the variables of $Q$ by constants in $\overline{t}$ is consistently true in $\mathcal{DB}$ w.r.t. $C$. Note that, in this setting, the problem of computing consistent answers to queries in the case of denial constraints and inclusion dependencies (the most common schema constraints) belongs to the $\Pi_2^P$ complexity class [8]. The CQA Rewriter, takes as input a conjunctive query $Q$, a set of integrity constraints $C$, and a global database $\mathcal{DB}$ and builds both an ASP program $\Pi_{cqa}$ and a query $Q_{cqa}$, such that: $Q$ is consistently true in $\mathcal{DB}$ w.r.t. $C$ iff $Q_{cqa}$ is true in every answer set of $\Pi_{cqa}$, in symbols: $\Pi_{cqa} \models_c Q_{cqa}$.

Just to show an example on how the rewriter works, consider the example introduced previously; the program obtained by rewriting the global constraints is:

$$\overline{e}(X,Y) \vee \overline{e}(X,Z) :- e(X,Y), e(X,Z), Y <> Z.$$
$$\overline{e}(X,Y) \vee \overline{e}(Z,Y) :- e(X,Y), e(Z,Y), X <> Z.$$

$$e^r(X,Y) :- e(X,Y), not\ \ \overline{e}(X,Y).$$
$$m^r(X) :- m(X), e^r(X,\_).$$

Here, the disjunctive rules guess atoms to be cancelled for satisfying key constraints, whereas the following rules remove atoms violating also referential integrity constraints, and build repaired relations. Note that the minimality of answer sets guarantees that deletions are minimized. This sub-program is then fed along with the mappings and the user query to the query evaluator (see next Section).

Suppose now that we ask for the list of manager codes; since both $m^r(e1)$ and $m^r(e2)$ are in all the answer sets of the resulting program, both $m(e1)$ and $m(e2)$ are derived as the consistent answers.

It is important to point out that our rewriting procedure has been devised in such a way that the evaluation of produced ASP programs is complexity-wise optimal according to the complexity classification of constraints and queries of [8]. Indeed, polynomial, co-NP and $\Pi_2^P$ queries are dealt with by exploiting normal (stratified) programs, head-cycle-free, and non-head-cycle-free programs, respectively.

### 2.2 Query Evalutator

The core query evaluation engine of our integration system is DLV$^{DB}$ [9]. It is a DLP evaluator born as a database oriented extension of the well known DLV system [10]. It has been recently extended [11] for dealing with unstratified negation, disjunction and external function calls.

The main peculiarities of DLV$^{DB}$ related to the data integration system are:

- It allows the handling of (possibly distributed) massive amounts of data; in fact, it exploits ODBC connections to link the ASP program to database relations and implements an evaluation strategy working mostly onto the database, where input data reside. More precisely, it carries out the grounding of the logic program completely on the DBMS, and then loads in main memory only the minimal amount of data necessary for the generation of stable models, thus handling disjunction

and unstratified negation. With respect to the data integration setting, this strategy perfectly fits the system's needs; in fact, if input data is "clean" and no global constraints are violated, the query answering process can be completed during the grounding, and no data must be loaded in main memory. On the contrary, if some constraint is violated, only conflicting data must be loaded in main memory; this amount of data is usually much smaller than the overall input size.

- GAV mappings defining the integration system can be directly evaluated without further elaboration; this provides a direct correspondence between what the designer specifies and what the system should evaluate. This simplifies the application of optimization and rewriting techniques.
- DLV$^{DB}$ extends ASP with external function calls; this is, in general, particularly suited for solving inherently procedural sub-tasks. External functions must be defined as stored functions in the database coupled with DLV$^{DB}$. In the context of data integration, the possibility of calling external functions provides rich capabilities in cleaning input data. In fact, it allows to embed, in a declarative setting, purely procedural tasks such as string manipulation. As an example, consider the integration of two databases representing dates in different formats: one as dd/mm/yy, and the other one as mm/dd/yy. In order to guarantee consistency in the integrated database, one of these representations must be transformed and this can be easily carried out by string manipulation. This procedurally simple task would be tedious in a purely declarative setting.
- DLV$^{DB}$ embodies some query-oriented optimization strategies, like magic-sets, query unfolding, and static filtering (see [12] for a survey), capable of significantly improving query evaluation performances. In the context of data integration, such optimizations find their perfect application when queries contain constants, since they allow to "localize" the computation and drastically reduce the amount of data to reason about.

## 3 The Integration System

The general architecture of the proposed system is shown in Figure 1. It is intended to simplify both the integration system design and the querying activities by exploiting a user-friendly GUI. Specifically, at design time, the user can:

- Graphically design the global schema and the mappings (which we recall are expressed by UCQs) between global relations and source schemas.
- Specify data transformation rules on source data; these can be implemented by suitable functions defined in the working database as stored functions.
- Specify global constraints, in order to define quality parameters that global integrated data must satisfy.

At query time, the user can exploit a QBE-like interface to express queries over the global schema; these are internally expressed in datalog as UCQs. The "plain" query is then elaborated by the CQA Rewriter which takes into account both mappings and global constraints to express the query over the sources and to handle inconsistencies

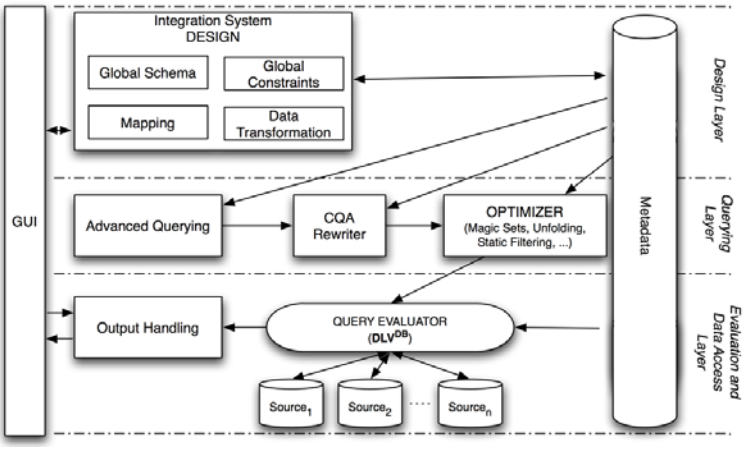


**Fig. 1.** System Architecture

possibly involving the query answers; the output of the CQA Rewriter is then a (possibly disjunctive) datalog program which is fed to the Optimizer for further elaboration. The Optimizer applies rewriting strategies which aim at pushing down selections directly onto the sources and at "localizing" over conflicting data as much as possible of the needed reasoning. Finally, the optimized program is fed to the Query Evaluator ($\text{DLV}^{DB}$) which executes the grounding phase totally on the DBMS and loads in main-memory only data strictly necessary to resolve conflicts. The output of this evaluation is then the query answer, which is proposed graphically back to the user.

## References

1. Hayes, H., Mattos, N.: Information on demand. DB2 Magazine 8(3) (2003)
2. Halevy, A.Y.: Data integration: A status report. In: 10th Conference on Database Systems for Business, Technology and Web (BTW 2003), pp. 24–29 (2003)
3. Bertossi, L., Hunter, A., Schaub, T. (eds.): Inconsistency Tolerance. LNCS, vol. 3300. Springer, Heidelberg (2005)
4. Bravo, L., Bertossi, L.: Logic programming for consistently querying data integration systems. In: Int. Joint Conference on Artificial Intelligence (IJCAI 2003), pp. 10–15 (2003)
5. Lenzerini, M.: Data integration: A theoretical perspective. In: Proc. PODS 2002, pp. 233–246 (2002)
6. Leone, N., et al.: The INFOMIX System for Advanced Integration of Incomplete and Inconsistent Data. In: Proc. ACM SIGMOD 2005, pp. 915–917 (2005)
7. Arenas, M., Bertossi, L.E., Chomicki, J.: Consistent Query Answers in Inconsistent Databases. In: Proc. PODS 1999, pp. 68–79. ACM Press, New York (1999)
8. Chomicki, J., Marcinkowski, J.: Minimal-change integrity maintenance using tuple deletions. Information and Computation 197(1-2), 90–121 (2005)
9. Terracina, G., Leone, N., Lio, V., Panetta, C.: Experimenting with recursive queries in database and logic programming systems. Theory and Practice of Logic Programming (TPLP) 8(2), 129–165 (2008)

10. Leone, N., Pfeifer, G., Faber, W., Eiter, T., Gottlob, G., Perri, S., Scarcello, F.: The DLV System for Knowledge Representation and Reasoning. ACM Trans. Comput. Log. 7(3), 499–562 (2006)
11. Terracina, G., De Francesco, E., Panetta, C., Leone, N.: Enhancing a DLP system for advanced database applications. In: Calvanese, D., Lausen, G. (eds.) RR 2008. LNCS, vol. 5341, pp. 119–134. Springer, Heidelberg (2008)
12. Minker, J. (ed.): Foundations of Deductive Databases and Logic Programming. Morgan Kaufmann Publishers, Inc., Washington (1988)