Synchronization of Queries and Views Upon Schema Evolutions: A Survey

LOREDANA CARUCCIO, GIUSEPPE POLESE, and GENOVEFFA TORTORA,
University of Salerno

One of the problems arising upon the evolution of a database schema is that some queries and views defined on the previous schema version might no longer work properly. Thus, evolving a database schema entails the redefinition of queries and views to adapt them to the new schema. Although this problem has been mainly raised in the context of traditional information systems, solutions to it are also advocated in other database-related areas, such as Data Integration, Web Data Integration, and Data Warehouses. The problem is a critical one, since industrial organizations often need to adapt their databases and data warehouses to frequent changes in the real world. In this article, we provide a survey of existing approaches and tools to the problem of adapting queries and views upon a database schema evolution; we also propose a classification framework to enable a uniform comparison method among many heterogeneous approaches and tools.

CCS Concepts: • **General and reference** → **Surveys and overviews**; • **Information systems** → **Database views**; *Data exchange*; *Mediators and data integration*; • **Software and its engineering** → *Software maintenance tools;*

Additional Key Words and Phrases: DB schema evolution, query synchronization, view synchronization

ACM Reference Format:
Loredana Caruccio, Giuseppe Polese, and Genoveffa Tortora. 2016. Synchronization of queries and views upon schema evolutions: A survey. ACM Trans. Database Syst. 41, 2, Article 9 (May 2016), 41 pages.
DOI: http://dx.doi.org/10.1145/2903726

1. INTRODUCTION

During the life cycle of an information system, it is often necessary to modify the schema of the underlying database. This might occur either to correct previous design and implementation errors or to adapt the information system to changes in the real world. This is a well-known and critical problem, named *Schema Evolution*, which has drawn the attention of many researchers in the database community. In fact, there are many artifacts depending on the old version of a database schema, which might need to be modified in order to continue work on the new schema. In particular, queries and views might no longer work properly if the schema update operations concern constructs of the old schema on which they were defined. Thus, they need to be synchronized to the new schema.

There are many other contexts in which it is necessary to synchronize queries and views upon schema evolution operations, for example, in data warehouses, data integration systems, web services, and mashup development [Moro et al. 2007].

The problem of synchronizing queries and views upon a schema evolution has been faced since 1982, and it has been considered a major bottleneck in system conversion

Authors' addresses: L. Caruccio, G. Polese, and G. Tortora, Department of Computer Science, University of Salerno, Via Giovanni Paolo II, nr. 132, 84084, Fisciano, Italy; emails: {lcaruccio, gpolese, tortora}@unisa.it.

ACM 0362-5915/2016/05-ART9 $15.00
DOI: http://dx.doi.org/10.1145/2903726

[Shneiderman and Thomas 1982a]. Due to the wide variety of application domains where the problem appears, many different names have been used: query-program conversion [Shneiderman and Thomas 1982a], view/query rewriting [Rundensteiner et al. 1999], change propagation [Melnik 2004], mapping adaptation [Velegrakis et al. 2003b], and cotransformations within the broader research context of bidirectional transformations [Czarnecki et al. 2009; Terwilliger et al. 2012]. A more formal definition has been given by Rundensteiner et al. [1997] in the context of Data Integration, where the view synchronization problem has been defined as a dynamic process to adapt the view definition upon capability changes of an information source, that is, changes involving the addition, the deletion, or the renaming of database constructs. In this article, we will use the term "query/view synchronization," since the problem definition has been expanded to some contexts in which the concept of query is more appropriate than that of view.

Although the query/view synchronization problem has proven to be of vital importance, the shortage of proper automated tools has not made it practical so far [Bernstein and Melnik 2007; Curino et al. 2008b; Hick and Hainaut 2006]. Moreover, it is not economically sustainable to manually rewrite queries and views, since it would require a considerable amount of programmers' work. Thus, the research community has focused its attention on the development of approaches capable of mitigating the inherent complexity of the synchronization process, such as (1) approaches applying updates in a *lazy* fashion or (2) approaches capable of *tolerating* the presence of queries and views working on previous versions of a database schema. *Lazy* update approaches have been proposed to cope with delayed synchronization processes [Ferrandina et al. 1994], whereas *tolerating* approaches assume that only new applications are relevant, while old ones are discarded. When old applications have to be maintained, schema versioning provides an alternative solution to the query/view synchronization problem [De Castro et al. 1997; Grandi and Mandreoli 2003; Jørgensen and Böhlen 2007; Jensen and Böhlen 2002]. The adoption of schema versioning lets old applications continue work with the old schema, whereas new applications will work with the new schema.

Query/view synchronization is also crucial in the area of application management, to refactor application programs accessing evolved database schemas, since programs mainly embed queries [Li 1999; Ram and Shankaranarayanan 2003].

Although in the literature there are several approaches facing the query/view synchronization problem in the broader sense, there are still many synchronization processes based on specific policies, cases, and/or domain-specific issues. In order to analyze and classify the existing approaches in a uniform and inclusive way, in this article, we survey existing query/view synchronization approaches providing a framework to describe, classify, and systematically compare them. In particular, the survey aims to pursue the following goals: (1) to analyze existing query/view synchronization approaches; (2) to provide a comprehensive and classified list of them, useful for researchers, database designers, and database tool vendors; and (3) to help users select the approaches more suitable for their purposes.

The article is organized as follows. In Section 2, we characterize the query/view synchronization problem as a specialization of the main schema evolution problem and describe some application domains. In Section 3, we present our framework for the classification of the existing approaches, which are clustered in (1) operation-based approaches, (2) mapping-based approaches, and (3) hybrid approaches, described in Sections 4, 5, and 6, respectively. Finally, a discussion and final remarks are given in Section 7.

2. THE QUERY/VIEW SYNCHRONIZATION PROBLEM

Often, organizations have to cope with the evolution of information systems at several stages of their life cycle. This may happen when a system is first released, since bugs or incomplete functionalities may arise in this phase; or, the system might successively need to evolve in order to reflect changes in the real world, which might also entail the evolution of the underlying database.

Definition 1. Let S be a database schema, and $Inst(S)$ be the set of possible instances of S; an *evolution* of S is the result of one or more changes to the data structures, constraints, or any other artifact of S, that is, changes modifying the contents of its system catalog. They might consist of *simple schema modifications*, such as the addition, deletion, or renaming of an attribute, of a constraint, or of a relation, and/or *composed schema modifications*, such as join, partition, and decomposition [Lerner 2000]. In what follows, we denote with $S \rightarrow S'$ the evolution of the schema S into S', where S and S' are called schema versions.

Example 1. Let us consider the following schema S:

$$ATMBranchPoint(\underline{ATMcode}, address, branch, bank, positioning), \quad (1)$$

where the underlined attribute is the primary key. The schema represents the code, address, branch, and bank of an ATM point, whereas the attribute *positioning* indicates whether the ATM point is located internally or externally to the branch location.

Let us also consider two evolutions $S \rightarrow S'$ and $S' \rightarrow S''$, with:

$$\begin{aligned} S' :\quad & ATMPoint(\underline{ATMcode}, address, branch, positioning) \\ & Branch(\underline{branch}, bank) \\ \text{and} \quad & \\ S'' :\quad & ATMPoint(\underline{ATMcode}, address, branch) \\ & Branch(\underline{branch}, bank). \end{aligned} \quad (2)$$

In $S \rightarrow S'$, the relation *ATMBranchPoint* is decomposed into the relations *ATMPoint* and *Branch*, whereas in $S' \rightarrow S''$, the attribute *positioning* is dropped from *ATMPoint*.

The impact of schema modifications on the database instances can be characterized through the concept of *information capacity* [Hull 1986; Miller et al. 1993]. The latter specifies whether the set $Inst(S')$ is equivalent to $Inst(S)$, extends $Inst(S)$, or reduces it [Bernstein and Melnik 2007]. For instance, when an attribute is dropped from a database schema, the corresponding information is lost; hence, $Inst(S')$ reduces $Inst(S)$.

Definition 2. The information capacity variation associated to a given schema evolution $S \rightarrow S'$ can be evaluated by means of a function $g : Inst(S') \rightarrow Inst(S)$, according to which the evolution is:

—*Capacity increasing*, if the function g is surjective; that is, for each instance in the set $Inst(S)$, there is at least one corresponding instance in the set $Inst(S')$.
—*Capacity preserving*, if the function g is bijective; that is, there exists an instance in the set $Inst(S)$ if and only if there exists a corresponding instance in the set $Inst(S')$.
—*Capacity reducing*, if the function g is injective; that is, for each instance in the set $Inst(S)$, there exists at most one corresponding instance in the set $Inst(S')$.

According to Definition 2, the evolution $S \rightarrow S'$ of Example 1 is *capacity preserving*, whereas the evolution $S' \rightarrow S''$ is *capacity reducing*.

Capacity-reducing variations are the most critical ones, since they entail a loss of information that might irremediably invalidate some database components, like queries and application programs. However, the other two types of variations might require adaptations of several database components in order to make them continue work on the new schema version. The change operations applied to a database schema, and the problems deriving from them, fall in the context of the so-called Schema Evolution problem.

In the last few decades, the schema evolution problem has increasingly drawn the attention of database researchers. In fact, although in traditional information systems the database schema was designed to accept any future requirement changes, this assumption has become unrealistic, especially with the advent of Web Information Systems, due to their distributed nature, which yields an even stronger pressure toward changes. However, what has made this problem a critical one is the impact of the schema evolution operations on queries and applications, since it has been estimated that each evolution step might affect up to 70% of the queries operating on the schema [Curino et al. 2008], which must consequently be reworked.

In this scenario, the query/view synchronization (QVS in the following) problem is to adapt all the queries and views defined on the old version of a database schema, in order to make them also work on the evolved one.

Definition 3. Let Q be the set of queries and views defined on a database schema S; upon a schema evolution $S \rightarrow S'$, the QVS problem consists of finding a transformation t of Q, which produces a set Q' of queries and views on S', such that the semantics of Q' on S' preserves the semantics of Q on S. If such a transformation exists, we say that there exists a synchronization $Q \rightarrow Q'$, or even that Q' represents a legal rewrite of Q.

As an example, let us consider the following view on the database schema S of Example 1:

$$\begin{aligned} &IsInternal(ATMcode, bank) \\ &\quad \leftarrow ATMBranchPoint(\underline{ATMcode}, address, branch, bank, positioning) \wedge \\ &\qquad \wedge\ positioning = \text{“}internal\text{”}, \end{aligned} \tag{3}$$

which extracts the tuples $IsInternal(ATMcode, bank)$ for all the internal ATMs. Clearly, even such a simple view needs to be synchronized or rewritten upon each of the schema evolutions defined in Equation (2).

Since it is too expensive to synchronize queries and views manually, research in this area attempts to derive methods and tools providing users with automated support for the QVS problem. Also, it would be desirable to give the user the illusion of defining queries and views on an older version of the schema even though it has evolved over time [Lakshmanan et al. 1993].

As shown in Figure 1, upon a schema evolution (Figure 1(b)), it is necessary to synchronize both the application programs (Figure 1(a)) and the database instance (Figure 1(c)) [Ram and Shankaranarayanan 2003]. In our study, we will focus on the synchronization of application programs and, in particular, of the queries and views. The reader interested in the instance update problem may refer to the related work section of Lerner [2000] for a short survey.

There are two basic strategies to specify a schema evolution, one that describes the procedure to transform S into S', and one that first specifies S' and then finds schema correspondences between S and S'. Moreover, there are hybrid strategies mixing the characteristics of the two previous ones. As a consequence, we can have the following three types of schema evolution approaches:

(1) *Operation based*
(2) *Mapping based*
(3) *Hybrid*

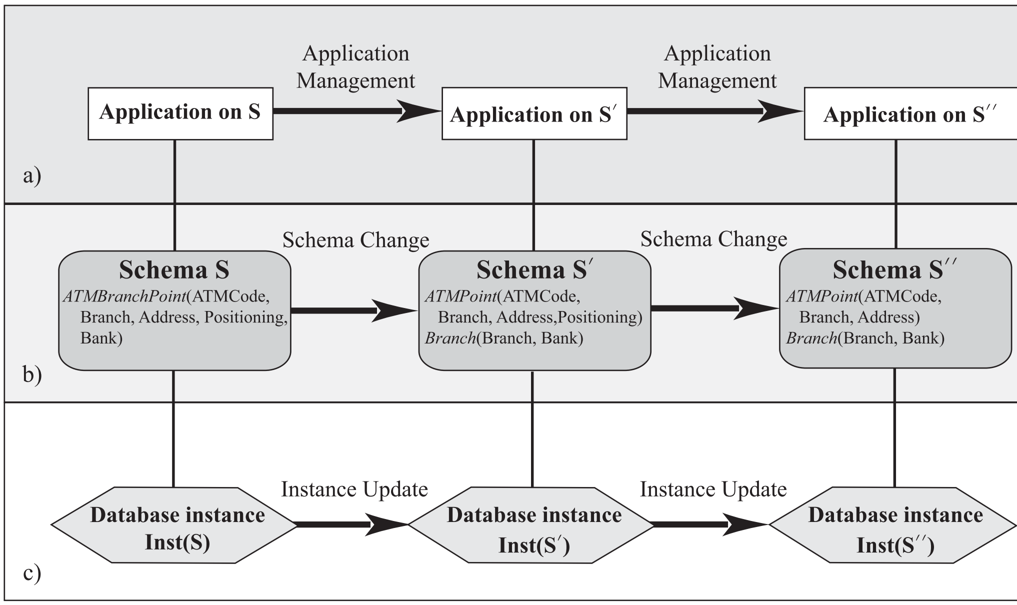

Fig. 1. A schema evolution scenario: (a) application management, (b) schema change, (c) instance update.

Operation based includes approaches based on the *editing process*, that is, approaches that define schema modification commands to implement each type of supported changes. More specifically, the *operation-based* approaches define several modification operations to specify the effects of single modifications on both schemas and instances (see Banerjee et al. [1987], Zicari [1991], Roddick et al. [1993], Peters and Özsu [1997], and Moro et al. [2007]). Such approaches enable the management of both simple and composed schema modifications.

As an example, by using the schema modification operators defined in the SMO language [Curino et al. 2008b], an *operation-based* specification of the schema evolution $S' \rightarrow S''$ of Figure 1 is

$$DROP\ \ COLUMN\ \ positioning\ \ FROM\ \ ATMPoint. \tag{4}$$

Mapping based includes approaches based on the *editing result*, that is, approaches that allow one to modify the schemas as necessary and then compare the two schema versions [Lerner 2000]. Thus, they mainly focus on the detection of correspondences between schema versions, which can be represented by means of *mappings* [Bertino 1992; Lakshmanan et al. 1993; Lerner 2000; Bernstein et al. 2000; Bernstein and Rahm 2000; Melnik 2004; Bernstein 2003; Bernstein and Melnik 2007; Velegrakis et al. 2004a]. Formally, a mapping m between two schemas S and S' is a set of assertions of the form $q_S \rightsquigarrow q_{S'}$, where q_S and $q_{S'}$ are queries over S and S', respectively, with the same set of distinct variables, and $\rightsquigarrow \in \{\subseteq, \supseteq, \equiv\}$. An assertion $q_S \subseteq q_{S'}$ is a *sound* mapping, meaning that q_S is contained in $q_{S'}$ w.r.t. $S \cup S'$; an assertion $q_S \supseteq q_{S'}$ is a *complete* mapping, meaning that $q_{S'}$ is contained in q_S w.r.t. $S \cup S'$; and an assertion $q_S \equiv q_{S'}$ is an *exact* mapping, when it is sound and complete [Haase and Motik 2005]. While a schema determines a set of possible database instances, $Inst(S)$, the mappings between S and S' are subsets of $Inst(S) \times Inst(S')$ [Ten Cate and Kolaitis 2009]. Using the mapping concept, the schema evolution $S' \rightarrow S''$ expressed in Equation (4) would be

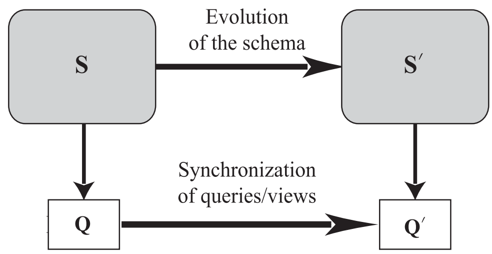

Fig. 2. The query/view synchronization problem.

described by the following schema mapping:

$$\begin{aligned}&\forall ATMcode, Address, Branch, positioning(\\ &\quad ATMPoint(\underline{ATMcode}, Address, Branch, positioning) \\ &\quad \rightarrow ATMPoint(\underline{ATMcode}, Address, Branch) \\ &\quad),\end{aligned} \tag{5}$$

which represents a FullTGD mapping.[1]

In general, *operation-based* approaches exploit the advantage of knowing a priori how the schema can evolve and the effects of each evolution. Unfortunately, since the possible modifications are well defined, they pose a limit on the possible evolutions of the schema. On the other hand, *mapping-based* approaches allow us to handle every type of modification [Melnik 2004], but without a complete view of the effects that a single modification will have on the schema. Finally, by combining the characteristics of both operation-based and mapping-based approaches, *hybrid* approaches exploit advantages and strength points of both basic types.

2.1. The QVS Process

The QVS process is graphically represented as shown in Figure 2. The horizontal arrow between S and S' represents the schema evolution $S \rightarrow S'$, and the horizontal arrow between Q and Q' represents the QVS $Q \rightarrow Q'$. The vertical arrows between S and Q, and S' and Q', respectively, indicate that Q and Q' are sets of queries and views defined on S and S', respectively.

Unfortunately, it is not always possible to find a synchronization for all the queries and views upon a schema modification, especially when the schema evolution is *capacity reducing* (see Lakshmanan et al. [1993] and Rundensteiner et al. [1997]) or when the schema is considerably altered (see Barklund et al. [1997] and Polese and Vacca [2009a]). In these cases, it is necessary to decide whether the query/view must be dropped or the schema evolution inhibited, in order to preserve the correct functioning of all the queries and views. To this end, Papastefanatos et al. [2006] defined the concept of policy guiding the synchronization process. In particular, they defined three types of policies: (1) *propagate*, (2) *block*, and (3) *prompt*, which prescribe how to handle the portions of the view definitions affected by the schema modification. In particular, the policy *propagate* prescribes to apply changes and synchronize Q; *block* forbids changes; and *prompt* prescribes to ask the DBA for the action to be undertaken. Although the

[1]A FullTGD mapping represents a schema mapping specified by using a complete set of source-to-target tuple-generating dependencies (source-to-target tgds) [Fagin et al. 2005]. A tuple-generating dependency is a first-order logic formula of the form $(\forall x)(\varphi(x) \rightarrow (\exists y)\psi(x, y))$, in which $\varphi(x)$ is a conjunction of atoms such that the variables of each atom are among those in x and each variable x occurs in at least one of the atoms of $\varphi(x)$; and $\psi(x, y)$ is a conjunction of atoms whose variables are those in x and y. FullTGDs represent a subclass of TGDs in which the existential quantifier is on the right side of the formula [Kolaitis 2005].

last policy might appear the most suitable, one should not abuse it in order to keep the QVS process sufficiently automated.

While the concept of policy concerns the application of schema changes, Rundensteiner et al. [1997] propose a different solution focusing on the view constructs, by introducing the *View Evolution Parameters (VEPs)*; these are a class of parameters through which it is possible to define how to handle the single components of a view during the synchronization process, by specifying a priori whether the component (e.g., an attribute) is replaceable or whether it is dispensable. In addition, the same authors introduced the *View Extent*[2] *Parameter (VE)* [Rundensteiner et al. 1997] associated to a view Q specifying a condition on the extent of a view Q' in order for Q' to be considered an acceptable synchronization of Q. In other words, the *VE* parameter ϕ, $\phi \in \{\equiv, \subseteq, \supseteq, \approx\}$ specifies a priori whether the extent of Q' must be equivalent ($\equiv$), be included ($\subseteq$), include ($\supseteq$), or approximate ($\approx$) the extent of Q for Q' to be considered a legal rewrite of Q. In practice, the *VE* parameter establishes the relationship that must hold between the projections of Q' and Q on their common attributes, that is,

$$\pi_{Attr(Q)\,\cap\,Attr(Q')}(Q') \;\; \phi \;\; \pi_{Attr(Q)\,\cap\,Attr(Q')}(Q), \tag{6}$$

where $\phi \in \{\equiv, \subseteq, \supseteq, \approx\}$, and ϕ is the usual projection operator.

Example 2. Let

$$\begin{aligned} &AtmBrPos(ATMcode, branch, positioning) \\ &\qquad \leftarrow ATMPoint(\underline{ATMcode}, address, branch, positioning) \end{aligned} \tag{7}$$

be a view defined on the schema S' of Figure 1, which extracts the code, the reference branch, and the positioning of each ATM. If there is a *VEP* associated to the view *AtmBrPos* specifying that the attribute *positioning* is dispensable, then the schema evolution $S' \rightarrow S''$ is feasible, and consequently, if the *VE* associated to the same view is $\equiv$, then the view

$$\begin{aligned} &NewAtmBrPos(ATMcode, branch) \\ &\qquad \leftarrow ATMPoint(\underline{ATMcode}, address, branch) \end{aligned} \tag{8}$$

can be considered a synchronization of *AtnBrPos*, since

$$Attr(Q) \cap Attr(Q') = \{ATMcode,\ branch\}$$

and

$$\begin{aligned} &\pi_{\{ATMcode, branch\}}(AtmBrPos(ATMcode, branch, positioning) \equiv \\ &\qquad \pi_{\{ATMcode, branch\}}(NewAtmBrPos(ATMcode, branch)). \end{aligned} \tag{9}$$

Another problem to be considered upon an evolution $S \rightarrow S'$ is the choice of the schema version on which to start the synchronization process. In fact, a different sequencing of schema evolution/synchronization operations might lead to different synchronization results. To this end, the QVS problem may reduce to finding the schema version S_i from a schema evolution sequence $S_1, \ldots, S_i, \ldots, S_n$ (resulting after several schema evolutions) that is more similar to the schema S'. Such a schema will be the one from which to start the next synchronization step [Koeller and Rundensteiner 2005].

Example 3. Let us consider the schema evolution $S' \rightarrow S''$ recalled in Example 2. If S'' undergoes a new evolution $S'' \rightarrow S'''$ in which the previously dropped attribute

[2]The view extent is the usually adopted term indicating the result set of a view statement, that is, the materialized view.

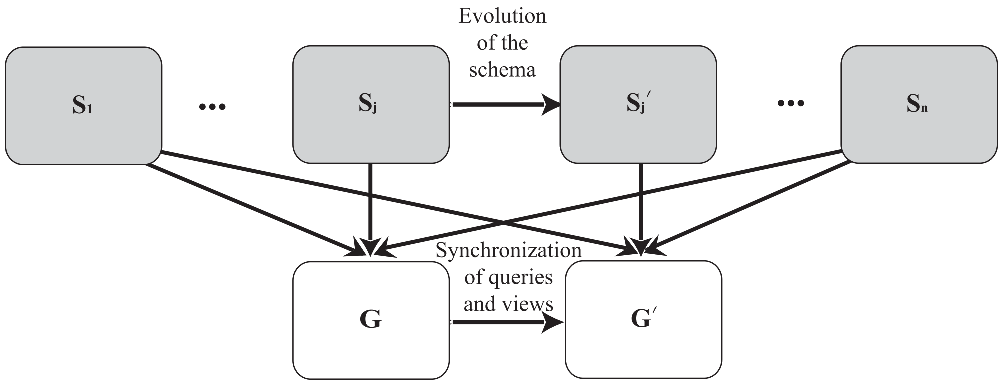

Fig. 3. The QVS problem in the Data Integration Systems context.

positioning is readded, the best way to synchronize the view *NewAtmBrPos* given in Equation (8) is to start the synchronization process from the schema version S' instead of S'', because there is no means to recover the values of the attribute *positioning* after it has been dropped.

2.2. Application Domains

Let us discuss some application domains in which either the QVS problem has been faced or it can potentially arise. This will help the reader better understand the peculiarities of some surveyed approaches, since they have been developed for solving specific problems of one application domain and hence show features that are not found in other approaches. Moreover, exploring new potential application domains for the QVS problem might stimulate further research and applications, for which the proposed survey might help reuse characteristics of existing approaches, avoiding reinventing the wheel. This is somehow what has happened in the past, since new approaches have been devised to solve problems for which existing approaches might have been suitable.

We will discuss the QVS problem in the context of the following application domains: *Data Integration Systems*, *Web Data Integration Systems*, *Web Services*, *Mushups*, *Data Exchange Systems*, and *Data Warehouses*.

Data Integration Systems (DISs) enable the correlation of concepts belonging to distinct data sources, which are independent and used in different contexts. They allow us to collect data from different sources and merge them into an integrated view [Lenzerini 2002; Miller et al. 2001; Seligman et al. 2010; Caruccio et al. 2014; Deufemia et al. 2014]. An integrated view G (also called global or mediation view) is defined from a set of data sources $S_1, \ldots, S_n$ by creating mappings from each of them onto the global view, that is, $Q^{S_i} \to G, \forall i = 1, \ldots, n$.

In this context, data sources are particularly dynamic; hence, it is of vital importance to have an automated process for QVS. However, when the modifications to the data sources are relevant, this might affect the schema of the global view, which entails the reiteration of the algorithm for source schema integration and the update of the mappings between the new global view and some (possibly all) data sources [Velegrakis et al. 2004a; Melnik et al. 2003a].

Figure 3 shows how the QVS problem in the DIS domain can be viewed as a specialization of the more general problem of QVS, where the evolution of a source schema yields the necessity to synchronize the integrated view G. In particular, the synchronization of G is accomplished by rewriting the mappings between the modified sources and G. In Figure 3, the horizontal arrow between S_j and $S_j{}'$ represents the evolution of a generic source schema S_j (i.e., $S_j \rightarrow S_j{}'$), whereas the horizontal arrow between G and G' represents the synchronization of the integrated view G (i.e., $G \rightarrow G'$). Other arrows ending in G and G' represent the fact that G and G' are defined by sets of mediation queries on sources $S_1, \ldots, S_n$.

Web Data Integration Systems are DISs encompassing both structured and semistructured data containers available on the web. Thus, in the Web Data Integration Systems context, we need to consider many new issues, which make their design and management particularly complex [Madhavan et al. 2007]. For instance, given the high number of information sources available on the web and their rapid growth, the scalability of such systems becomes highly critical. Moreover, we also need to consider the heterogeneity of web information sources and their modification frequency, which makes automatic synchronization processes even more vital. For this reason, several approaches focus on the creation of peer-to-peer logical relations between information sources by means of mediation mappings [Rahm et al. 2005; Halevy et al. 2003], thus avoiding the creation of the global view.

Web Services represent a programming paradigm enabling the extraction and the integration of data from heterogeneous information systems [Hansen et al. 2003]. The web service paradigm is based on the use of open standards for systems integration. In particular, the XML standard is used for preparing messages and structuring data, the SOAP protocol is used for data transfer, the WSDL language is used for describing the available services, and the UDDI standard is used for defining the available services. In the web service programming paradigm, developers often need to query and translate XML messages originating from Web Services or directly from the databases in which they are stored [Moro et al. 2007].

Mashups are web applications combining third-party data and services [Weiss and Gangadharan 2010]. They originate from the necessity to integrate contents, functionalities, and structured or semistructured information available on the web, as Open API or reusable services. There are several programming workbenches for developing mashups, for example, Yahoo's Pipes, Google Maps Editor, Microsoft popfly, and so forth [Yu et al. 2008]. They aim to facilitate mashup development even for unexperienced users. One fundamental requirement for mashups is that the integration of contents must occur dynamically, upon specific runtime requests from the users. The development and management of this type of application is heavily based on data integration methods, that is, methods specifying how data are related by means of mappings. The complexity of such tasks is tackled either by limiting the data integration types (e.g., by using standard ISBN objects, latitude/longitude) or by providing complex architectures for data integration (see Thor et al. [2007]).

Data Exchange Systems. Both web services and mashups can be considered as special cases of data exchange systems, in which the focus is on the problem of transforming instances of a source schema into those of a target one, by means of a mapping relating the two schemas [Fagin 2006; Kolaitis 2005]. In fact, data exchange, also known as data conversion, represents the problem of taking data structured according to a source schema and restructuring and translating them according to a target schema.

Figure 4 shows how the QVS problem can be adapted to the context of data exchange systems, where the evolution of the source schema leads to the necessity to synchronize the target schema T. In particular, the synchronization of T depends on the rewriting of the mapping relating S and T. In Figure 4, the horizontal arrow between S and S' represents the evolution of the source schema (i.e., $S \rightarrow S'$), the horizontal arrow between T and T' represents the QVS $T \rightarrow T'$, and the horizontal arrow between Q

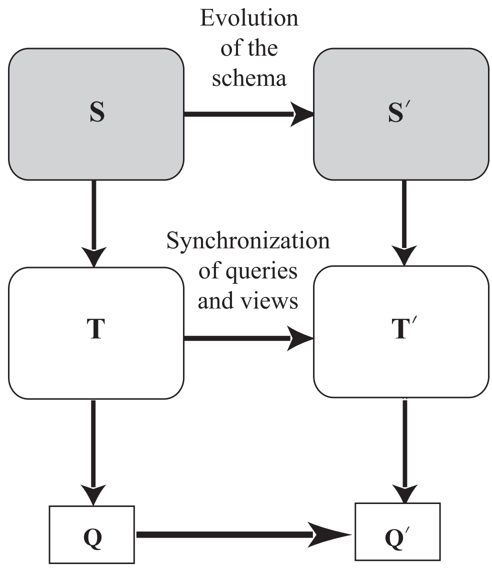

Fig. 4. The QVS problem in the Data Exchange Systems context.

and Q' represents the QVS $Q \rightarrow Q'$. The vertical arrows ending in T and T' represent that T and T' are defined by sets of queries/views (represented as mapping) from the source, and the vertical arrows ending in Q and Q' represent the fact that Q and Q' are sets of queries/views defined on T and T', respectively.

A further generalization of this scenario is given by *bidirectional transformation (bx)*, which is a mechanism for maintaining the consistency of two (or more) related sources of information and can be used to manage coevolutions of database schemas and schema-dependent programs [Terwilliger et al. 2012]. Thus, a *bx* between two sources of information *A* and *B* (e.g., a database source and view, two different software models, or the input and output of a program) comprises a pair of unidirectional transformations like those occurring in data exchange systems: one from *A* to *B*, and another from *B* back to *A* [Czarnecki et al. 2009]. It is easy to figure out that the QVS problems highlighted in the context of data exchange systems also apply to the *bidirectional transformation* context.

Data Warehouses (DWs) are specialized databases mainly used to support business decisions. They store data collected from several operational databases and possibly from several external information sources. The development of Data Warehouses can be accomplished either by defining global views on a data source or by loading data from different data sources onto a reconciled database by means of ETL (Extraction, Transformation, and Loading) procedures. Moreover, DWs are based on a multidimensional data model, in which data represent *facts* associated to numerical *measures* that can be analyzed along several *dimensions* [Pedersen and Jensen 2001]. Analyses on Data Warehouses are mainly performed by means of OLAP (OnLine Analytical Processing) queries and Data Mining techniques. Since the evolution of a source data schema might corrupt the mappings between the DW and the modified source, it can be easily figured out how the QVS problem might be crucial also in this application domain. In fact, upon the evolution of a source schema, it might be necessary to synchronize some of the queries used to construct the global view, or some ETL procedures in case a reconciled schema is constructed. However, if the evolution of the source schema also affects the structure of the global view or the reconciled schema, it might also be necessary to synchronize OLAP queries and/or Data Mining applications defined on the DW [Bellahsene 2002].

3. THE PROPOSED CLASSIFICATION FRAMEWORK

From the previous discussion, it is evident that the existing QVS approaches differ in the way they face single problem issues within the whole synchronization process. Thus, it would be desirable to define a framework characterizing each approach in terms of its capabilities and its proper context of use, so as to highlight differences and similarities among them and their peculiarities.

The proposed framework is structured in two groups of parameters: *structural issues* and *semantic issues*.

Structural Issues. The parameters in this group characterize how the QVS problem is tackled by a single approach:

—*Type of approach*. This parameter indicates the implementation level of the approach, that is, whether it is a tool/system, a programming language, a nonimplemented approach, and so forth.
—*Application area*. This parameter indicates the application domain for which the approach has been created and in which it is mainly used.
—*Supported models*. Since some approaches for QVS are devised for a specific data model, whereas others can be applied to more or all possible data models, this parameter indicates the supported data model(s).
—*Resource / technique*. This parameter specifies the resources and techniques used, such as whether the approach exploits meta-knowledge, meta-reasoning, algorithms, guidelines, and so forth.
—*QV definition language*. Queries/views are specified through a language, such as SQL, or a mapping. Moreover, in some QVS approaches, the expressive power of the language has been enriched in order to enhance the specification of the whole synchronization process. Thus, this parameter reports the type of the query/view definition language.
—*Schema changes language*. The evolution is managed through a specific language for modifying schemas and/or for catching versions' correlations. Thus, this parameter indicates the language used by an approach to specify the evolution of a schema.
—*Managed schema changes*. Independently of the specific schema change language, an approach can manage different sets of modifications that can be *simple modifications*, such as the addition, the deletion, or the renaming of an attribute, of a constraint, or of a relation, or *compound modifications*, such as join, partition, decomposition, and so forth. Thus, this parameter indicates the type of schema changes addressed by the approach.

Semantic Issues. The parameters in this group concern the aspects of the QVS problem of a single approach faced by each approach:

—*Automation level of the synchronization*. A fully automated synchronization is a desirable feature. However, some approaches provide a partial automation. This parameter indicates whether an approach permits a *semi* or *complete* automation level of synchronization.
—*Management of information loss*. The QVS problem is particularly difficult when the schema evolution leads to a loss of information, as in the case of capacity-reducing changes. Some approaches specifically focus on the management of changes yielding loss of information. This parameter indicates whether and how an approach manages the information loss.

Structural issues						
Type of approach	Application area	Supported models	Resource - Technique	QV definition language	Schema changes language	Managed changes

Semantic issues								
Automation level of the synchronization	Management of information loss	Global/Local synchronization	Transparency of evolution	Choice among several synchronization	Evaluation of evolution impact	Traceability of changes	Detection of QVs to be synchronized	Propagation to view extent

Fig. 5. The headers of the tables "structural issues" and "semantic issues" of the classification framework.

—*Global/local synchronization*. An approach can enable the definition of synchronization policies/modalities that are *local* to a single query/view or can enable the definition of *global* policies/modalities applied to all the queries and views. This parameter indicates how a specific approach defines its global/local synchronization.
—*Transparency of evolution*. The transparency of evolution enables users to pose queries/views based on a (possibly old) version of the schema, even though the schema has evolved to a different state [Lakshmanan et al. 1993]. This parameter indicates whether and how an approach manages the transparency of evolution.
—*Choice among several synchronizations*. In some cases, there is not a single legal synchronization. For this reason, it is useful to choose among allowable synchronizations in order to get better and/or less expensive synchronizations. Thus, this parameter indicates whether and how an approach manages the choice among several synchronizations.
—*Evaluation of evolution impact*. The schema evolution is an error-prone process. For this reason, it would be desirable to have mechanisms enabling users to evaluate the impact of a schema evolution, so as to let them estimate costs and benefits of schema changes [Maule et al. 2008]. Thus, this parameter indicates whether and how an approach supports the user in the evaluation of the evolution impact.
—*Traceability of changes*. In some contexts, the schema evolution is not a supervised and/or monitored process. In those cases, it is important to monitor the occurrence of schema modifications upon an evolution. Thus, this parameter indicates whether and how traceability of changes is supported.
—*Detection of QVs to be synchronized*. In large-scale systems, the huge number of queries/views defined on one or more sources makes the detection of the queries/views to be synchronized an extremely hard task. Thus, this parameter indicates whether and how an approach tackles the detection of queries/views to be synchronized.
—*Propagation to view extent*. A view is said to be materialized when its data (i.e., its extent) are computed and persistently stored [Bellahsene 2002]. When there are materialized views to be synchronized, it is useful to adapt the view extent to the accomplished synchronization. Thus, this parameter indicates whether and how an approach manages the synchronization propagation to the view extent.

According to this framework, we will classify the existing approaches and will collect them in two tables (see Figure 5), one for structural issues and another for semantic issues.

Since each approach is based on one of the three schema evolution strategies described in Section 2 (i.e., *operation based*, *mapping based*, and *hybrid*, respectively), we have grouped the surveyed approaches based on these three categories. Thus, each of the following three sections focuses on the classification of approaches falling in the same category. At the end of each section, the characteristics of the surveyed approaches are compared by means of a summary table structured according to the framework parameters (Figures 7, 9, and 11). Finally, in order to let the readers quickly access the approaches better suiting their needs and/or interests, the table in Figure 6 maps each

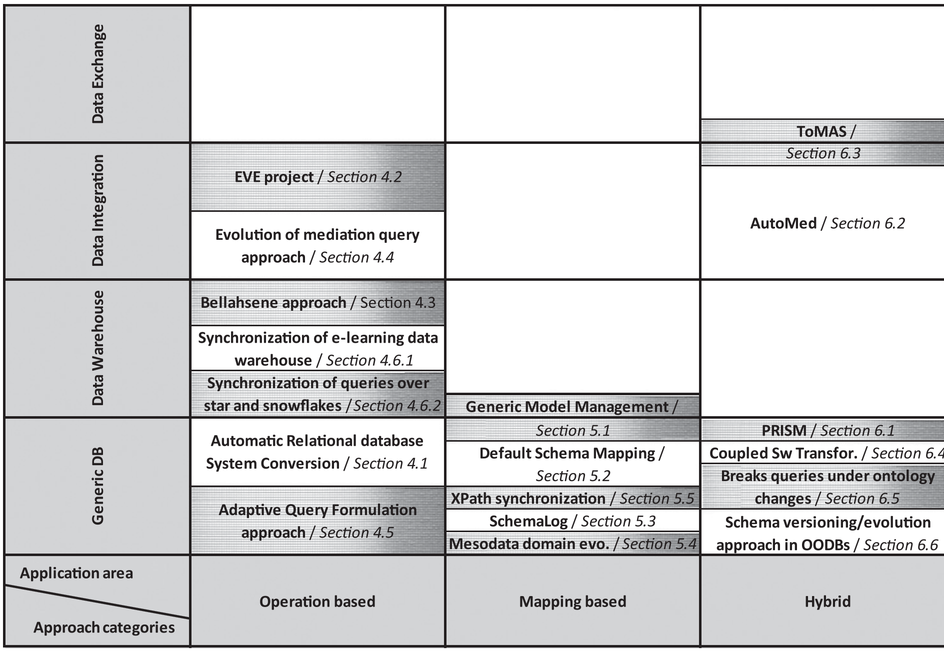

Fig. 6. The approaches within their application area.

surveyed approach to the section in which it is described, and to the *application areas* where it has been originally defined.

4. OPERATION-BASED APPROACHES

The approaches in this area aim to define a sequence of modification operations transforming a given version of a schema into a target one.

4.1. The Automatic Relational Database System Conversion (ARSC)

One of the main concerns upon a modification of a database schema is how to rewrite queries and application programs to make them work also on the modified schema. This problem represents the bottleneck of conversion systems and has been first tackled within the *Automatic Relational Database System Conversion* approach [Shneiderman and Thomas 1982a, 1982b], which is not aimed at creating a software product but at proving that automatic conversion is feasible. Indeed, the approach has not been implemented yet, and it belongs to the class of operation-based approaches, since it determines synchronization modalities based on the modifications that are possible on the database schema. The authors defined a set of 15 transformations, including both simple schema modifications (e.g., change the name of an identifier; create/delete a relation or an attribute) and composite schema modifications (e.g., promote/demote a primary key, affect functional dependencies, and so forth), which have been defined for the relational data model and are applicable only to relational data schemas satisfying 4NF. Moreover, such transformations have been classified as *information preserving*, *data dependent*, and *program dependent*. A transformation is said to be *information preserving* if it is possible to guarantee that the application of the transformation does not cause information loss. A transformation is *data dependent* if the stored database must be checked to verify whether the transformation is consistent with the target system. As an example, the deletion of an attribute is a transformation that might violate constraints of the target system, such as when the deleted attribute is part of a foreign key. When such a situation occurs, the DBA must decide whether to block the transformation or whether to modify the source database (e.g., changing its integrity constraints) so that the transformation can continue. Finally, a transformation is *program dependent* if it is not guaranteed that a QVS can be achieved. As an example, the renaming of an attribute or of a relation clearly guarantees the possibility of transforming the queries/views into equivalent ones, whereas this is not guaranteed in case of deletion of an attribute. Thus, when the transformation is *program dependent,* the DBA must check whether it prevents the transformation of some query or view into an input/output equivalent one, in which case he or she must decide whether the transformation is acceptable. Since the process of checking *data dependence* and *program dependence* properties is extremely costly, the authors advocate the implementation of efficient techniques to reduce such costs.

Such a classification is important, since it determines how to manage the synchronization process. In fact, if a transformation is data independent, then the target schema can be immediately constructed, and if it is program independent, the query rewriting rules can be automatically applied.

As an example, let us consider the evolution $S \rightarrow S'$ described in Figure 1, where the relation *ATMBranchPoint* is decomposed in the two relations *ATMPoint* and *Branch*. Such a transformation can be defined in ARSC through the following statement:

DECOMPOSE ATMBranchPoint INTO
ATMPoint(<u>ATMcode</u>, address, branch, positioning),
Branch(<u>branch</u>, bank)

and it is classified as *information preserving*, *data independent*, and *query program independent*. In fact, this transformation does not cause loss of information, and it is not necessary to check the consistency with respect to the logical format of the target system. As a consequence, the transformation can be automatically applied, but it might require the rewriting of queries and views involving the *AtmBranchPoint* relation. In fact, while the query to construct the view in Equation (3) needs to perform a selection and a projection on *AtmBranchPoint*, after the schema evolution, it must be transformed to first perform a natural join between *AtmPoint* and *Branch*.

The semantic issues of the approach can be summarized as follows:

—The synchronization process is described by using the defined transformations; their definition allows the *DBA* to identify the transformations yielding an *information loss*, in which case he or she must decide whether to permit or to inhibit the schema evolution. For this reason, the synchronization is *semiautomatic*.
—The *synchronization policy* is *global*; that is, it is applied to all the involved queries/views.

4.2. EVE

The Evolvable View Environment (EVE) has been built based on one of the most important studies performed in the context of schema evolution systems [Rundensteiner et al. 1997; Lee et al. 2002; Nica et al. 1998; Nica and Rundensteiner 1998; Koeller and Rundensteiner 2000; Lee et al. 1999a, 1999b; Koeller et al. 1998; Rundensteiner et al. 1998, 2000], in which the main concepts and definitions of the QVS problem have been provided. This study has also influenced most approaches that have been successively defined. Based on its early results, the EVE system has been implemented [Rundensteiner et al. 1999]. Its underlying approach is oriented to Data Integration (DI) architectures, facing the view synchronization problem by evolving views in response to a modification of the functionalities of an Information Source (IS). The EVE system supports simple schema modifications (add/delete/change_name of an attribute add/delete/change_name of a relation) and is based on an extensive use of metadata that must be specified during the system construction [Velegrakis et al. 2004a; Pedersen and Pedersen 2004a].

The authors propose the EVE architecture as a generic framework within which to perform view synchronization when the underlying ISs change their capabilities. In particular, the EVE architecture is divided into two spaces, namely, the information space, which is populated by a given number of heterogeneous ISs, and the view space, containing the user-defined views that are specific to the application context. Both of them use knowledge bases to store metadata concerning views and information sources. More specifically, a Meta Knowledge Base (MKB) stores metadata on source capabilities, their data model, and data content, whereas a View Knowledge Base (VKB) stores metadata on view definitions. They are both used in the view synchronization process. Moreover, the view space contains the view maintainer component of the EVE system, which takes into account the propagation of changes to view extents.

In EVE, the user can specify view synchronization policies a priori through Evolvable SQL (E-SQL), an extension of SQL, to guide the view synchronization process through both the *VE* and *VEP* parameters. The synchronization process is realized by means of several algorithms:

—The *Simple View Synchronization (SVS) algorithm* [Rundensteiner et al. 1997] concerning simple substitutions; for instance, when an attribute A of a relation R is deleted from an information source IS1, the View Synchronization Process (VSP) seeks the VKB and locates for each view affected by the modification (1) an acceptable definition based on the evolution preferences specified within the E-SQL definition, (2) the kind of functionality modification, and (3) the metaknowledge stored in the MKB.

—The *Complex View Synchronization (CVS) algorithm* [Nica et al. 1998] exploiting constraints of the MKB to handle complex substitutions; for instance, when a relation R is deleted, the VSP seeks the VKB, and for each affected view it changes its definition by replacing instantions of R with an expression derived from the MKB constraints.

—The *PrOject-Containment (POC) algorithm* [Nica and Rundensteiner 1998] seeks substitutions for the deleted attributes or relations used within a view definition by exploiting project containment constraints from the MKB. These express constraints between the relation or the attribute to be replaced and the replacing one.

Koeller and Rundensteiner [2000, 2005] propose a different synchronization policy to overcome the limitations of the previous synchronization algorithms, which takes into account only the last schema modification (One-Step algorithms) and may produce inappropriate views in situations like the one presented in Example 3. The proposed synchronization policy keeps views synchronized with their original definitions, in the context of a sequence of metadata modifications that can occur overtime. Such a new policy is automatically executed through the History-driven Algorithm *HD-VS (History-Driven View Synchronization)*, which executes three steps: backtracking in the view history, reapplication of part of the sequence of modifications to the metadata from the history, and reconstruction of part of the view-history graph in the process of reapplication of the metadata modifications [Koeller and Rundensteiner 2005].

A sample synchronization policy in EVE is provided in Example 2, where the a priori specification of the VEP and VE parameters on the view *AtmBrPos* given in Equation (7) permits to determine feasible synchronizations of it; based on them, the synchronization given in Equation (8) is considered feasible and can be automatically handled through the execution of the SVS algorithm.

The semantic issues of the approach can be summarized as follows:

—The synchronization process is based on the specification of policies based on View Evolution and View Extent parameters (*VEP* and *VE*, respectively).
—The latter are useful for an a priori specification of *information loss management* policies.
—VEP and VE increase the expressive power of SQL, enabling the specification of policies for *local synchronizations* through E-SQL.
—Thanks to the specification of VEP and VE, a *complete automation level of synchronization* is achieved.
—The EVE system determines how to *propagate changes to the extent* of views.

4.3. The Bellahsene Approach (BellApp)

This approach concerns problems related to the evolution of data warehouse systems, and in particular to the structural modification of views, upon modifications in the capabilities of the underlying source schemas [Bellahsene 2002]. Since a data warehouse can also be viewed as a set of materialized views on multiple data sources, this is highly related to the QVS problem.

BellApp always propagates schema modifications to views by means of a propagation policy, never blocking the synchronization process. This is based on the assumption that if a construct is deleted from the source schema, the propagation policy assumes that such construct is no more useful, and hence it is deleted from the view. The supported schema modifications fall into the category of *simple* modifications. For each possible modification on the basic relations of the source schema, the approach tries to derive the instance of the synchronized view from the materialization of the old view, aiming to avoid recomputing it from the basic relations on which the view is defined. To this end, the author defines the *containment control* ($Q_1 \subseteq Q_2$),[3] which represents the portions of the new view that are common to the old view, so as to recompute only the portion of the new view not contained in the old one. Thus, the problem reduces to finding the containment rewrite of the new view by using the old view. The basic idea here is to formulate the new view in terms of some queries that can be derived from existing materialized views.

As an example, let us consider again the evolution $S' \rightarrow S''$ described in Figure 1, and the following view

$$\begin{aligned} &IsBranchInternal(ATMcode, branch) \\ &\quad \leftarrow ATMPoint(\underline{ATMcode}, address, branch, positioning) \wedge \\ &\qquad \wedge\ positioning = \text{“}internal\text{”}, \end{aligned} \tag{10}$$

which extracts the tuples $IsBranchInternal(ATMcode, branch)$ for all the internal ATMs. Upon the deletion of the attribute *positioning*, BellApp replaces the affected clause within the selection condition of the *IsBranchInternal* definition with a tautology. As a consequence, the synchronized view will also include tuples that were previously discarded, as shown in the following:

$$\begin{aligned} &NewIsBranchInternal(ATMcode, bank) \\ &\quad \leftarrow IsBranchInternal(ATMcode, bank) \vee \\ &\qquad \vee\ (ATMPoint(\underline{ATMcode}, address, branch, positioning) \wedge \\ &\qquad \wedge\ \neg positioning = \text{“}internal\text{”}). \end{aligned} \tag{11}$$

[3]Definition of containment [Abitebul et al. 1995]: Let Q_1 and Q_2 be two queries defined on the schema S; we say that Q_1 is contained in Q_2, denoted by $Q_1 \subseteq Q_2$, if for each instance I of S, the answer set of Q_1 is a subset of the answer set of Q_2.

In this way, *NewIsBranchInternal* contains all the tuples previously materialized in the view *IsBranchInternal*, and new tuples extracted by the new subquery in which the condition *positioning* = *internal* is negated.

The semantic issues of the approach can be summarized as follows:

—The synchronization process presents a *complete level of automation*, based on the specified propagation policy.
—The *propagation to view extent* is tackled by the *containment control*, so permitting the derivation of the extent for the new view from the materialization of the old view.
—The *synchronization policy* is *global*.

4.4. The Evolution of Mediation Query Approach (EMQ)

Bouzeghoub et al. [2003] faced the problem of the evolution of mediation queries in Global-As-View (GAV) systems. The latter are particular DI systems, in which each relation in the global schema (also called mediation schema) is defined by queries (also called mediation queries) on the source schemas. For this reason, the evolution of mediation queries is a typical QVS problem, since one or more queries must be synchronized upon a source schema evolution. The authors have faced the QVS problem of this kind of system, aiming at preserving the integrity of mediation queries upon simple modification operations on the source schemas (e.g., insertion/deletion of attributes, relations, or referential constraints). The idea underlying their approach is to adapt an existing algorithm for *Mediation Query Generation*, namely, *MQG*, to work in a modular and incremental way, so as to synchronize all the mediation queries to the updated source schemas. In particular, *MQG* finds the GAV mappings (mediation queries) and generates metadata for both source relations and the operations needed to rewrite a mediation relation. The extended algorithm is named *Incremental Mediation Query Generation (IMQG)* [Kedad and Bouzeghoub 1999].

In order to guarantee modularity and incrementality, IMQG requires the mediation schema to store a chronology of all the design choices of the previous iterations. To this end, the authors define a representation of relationships among data sources and mediation relations, and of all the possible operations among them, by using the *operation graph*. The latter contains nodes representing relations, and edges representing possible operations. Moreover, they define a set of *propagation primitives* specifying updates and checks that must be executed on operation graphs and on mediation queries, so as to reflect update modification operations on the local schema. In this approach, the propagation of modifications on a source schema to a mediation level is accomplished by means of a global evolution process, which consists of two fundamental steps: the relation evolution step and the mediation query generation step. In order to synchronize the processes involved in such steps, the query generation process will only start when all the modification events notified by all the data sources have been propagated to the corresponding relations.

Successively, this approach has been extended to handle the evolution of mediation queries based on XML [Lóscio and Salgado 2004], in which the mediation schema is represented by means of the X-Entity model (an ER-like Language), and the evolution of the source data schema is based on a set of modification operations of the X-Entity schema [Roddick et al. 1993].

The semantic issues of the approach can be summarized as follows:

—The synchronization process is based on the construction of an operation graph, which contains relationships among data sources and mediation relations, and of all the possible operations among them. The graph permits the *detection of queries/views to be synchronized* (mediation queries) and their *local synchronization*, in order to avoid the redefinition of the whole mediation schema.

—The *automation level of the synchronization* is *complete*, and it is applied by Event-Condition-Action (ECA) rules based on propagation primitives.
—A defined *Lookup* process permits getting the *traceability of changes*.

4.5. The Adaptive Query Formulation Approach (AQF)

The adaptive query formulation approach (AQF) exploits a graph model to represent relations, views, constraints, and queries in a uniform way [Papastefanatos et al. 2006, 2007, 2009]. In particular, the entities of a database (e.g., relations, queries, views, and conditions of both the queries and the database) are modeled by means of nodes and edges of a directed graph. Other than representing the semantics of the database system, the graph allows us to predict the impact of a modification over the entire system.

The approach supports simple modifications (creation and deletion of relations, attributes, and conditions) and enables the definition of three types of policies: *propagate the modifications* (queries/views must be redefined), *block the modifications* (the intention here is to preserve the old semantics of the graph; hence, modification events must be blocked, or at least constrained, so as to preserve the old semantics), and *prompt* (the DBA interactively decides what to do). In other words, for each entity of the database, the approach prescribes how to handle each possible modification event by annotating the associated graph construct with the policy to be applied. The policy and the modifications activated on the graph constructs are specified locally to the views.

The synchronization of queries/views upon a modification event on a database schema is accomplished by determining the involved data structures and by applying their associated policies, which also determines how the graph will be affected. In this way, it is possible to define the actions to be applied (possibly automatically) by selecting the predominant policy among those defined for all the constructs affected by the modification. Notice that the application of a policy is itself considered as a new modification event.

The whole approach has been implemented within the HECATAEUS tool [Papastefanatos et al. 2008, 2010], which allows performing what-if analyses by graphically simulating the effects of modifications on the schema and their impact on queries, views, and the schema itself.

As an example, let us consider the evolution $S' \rightarrow S''$ described in Figure 1, where the attribute *positioning* is deleted from *ATMPoint*. After the evolution, all the queries/views referring to this attribute become syntactically invalid and need to be rewritten, as it happens for *AtmBrPos* given in Equation (7). If the DBA specified a "propagate" policy on the *attribute selection*, the view *NewAtmBrPos* given in Equation (8) could be automatically derived as a QVS of *AtmBrPos*.

The semantic issues of the approach can be summarized as follows:

—The synchronization process is based on the specification of policies on the graph constructs. The predefined policy specification is useful for the *management of information loss*, and it permits the *local synchronization to queries/views*.
—One type of policy invokes the *DBA supervision* in order to determine whether to permit or to inhibit a schema modification. For this reason, the *automation level of the synchronization* is *semi*.
—There is the possibility that more than one synchronization policy is simultaneously activated; to this end, the authors defined a guideline based on the "*policy prevalence*" *among several synchronizations*.
—The what-if analysis defined within the tool HECATAEUS permits one to *evaluate the impact of change*.

4.6. Domain-Specific Operation-Based Approaches

In the following, we discuss two ad hoc solutions created for specific domains.

4.6.1. The Synchronization of E-Learning Data Warehouses (eLDWSs). The QVS problem becomes particularly critical in the context of E-learning Data Warehouses (EDWs). This is a particular type of DW that is defined on E-learning Information Sources (EISs); it enables the integration of data in a single repository that is customized to user needs.

In this context, the QVS becomes critical because EISs are heterogeneous, distributed, and autonomous, and can contain a massive amount of information stored in local and geographical networks.

Studies reveal that a centralized architecture is inappropriate for this kind of system [Jalel 2007]. The author also proposes an EVE-based approach, exploiting the *Mobile Agent-e-DWMS architecture*, which uses mobile agent-based communication to realize some of the EVE's functions. Using this approach, the view synchronization process consists of determining legal rewrites of affected views, based on rules and constraints stored in the MKB. Such rules guide the definition of rewrites for the affected view components, according to the preference parameters stored in the VKB. With respect to the EVE canonical approach, this solution is based on a distributed structure and exploits static and mobile agent-based collaborative e-learning environments. The whole approach has been implemented by means of IBM aglets, which convey a better efficiency.

The approach inherits all the semantic issues highlighted for EVE. Moreover, the following semantic issue is specific for this approach:

—A dedicated Agent, namely, View Knowledge Base agent (*VKB agent*), permits the *identification of queries/views to be synchronized* in order to preserve the maximum number of view definitions.

4.6.2. The Synchronization of Queries Over Stars and Snowflakes (MDWS). Star and snowflake are two typical models used to describe Multidimensional schemas of Data Warehouses (MD-DWs): they both contain a Fact Table referencing a set of dimensional tables. Kaas et al. [2004] have systematically studied the evolution of star and snowflake schemas, focusing on the impact that their evolution produces on both exploration and aggregation queries; the authors specified eight different evolution operations: insertion and deletion of dimensions, levels, dimensional attributes, and measure attributes. Moreover, they provided a formal semantics for each type of modification operation for both star and snowflake schemas, describing the impact on existing navigation and aggregation queries.

Although the approach only provides a preliminary study toward the synchronization of navigation and aggregation queries on star and snowflake schemas, it can be considered a pioneer approach facing the problem of query synchronization in multidimensional data warehouses.

Thus, the only semantic issue of the synchronization process concerns the fact that the semantics of the evolution operations permit the *evaluation of the impact of the evolution process*.

4.7. Comparison of Operation-Based Approaches

The tables shown in Figure 7 summarize the *structural* and *semantic* issues of the operation-based approaches.

It can be easily noted that only a few operation-based approaches (two out of seven) have been completely implemented, whereas for the rest of them there are mostly prototype implementations or no implementation at all. Another interesting issue is that only two out of seven approaches have been employed in generic database

Structural issues / Approaches	Type of approach	Application area	Supported models	Resource - Technique	QV definition language	Schema change language	Managed changes
ARSC *Section 4.1*	Not implemented	Generic DB	Relational	System architecture	Default	Transform. primitives	add/delete attributes, merge/compose/decompose, change name, import/export/extract/remove dependency, promote/demote key
EVE *Section 4.2*	System	DI	Relational XML	SVS, CVS, POC, HD-SV	Evolvable SQL	Change operators	add/delete/change name attribute, add/delete/change name relation
BellApp *Section 4.3*	Prototype	DW	Relational	Containment rewritings	Extended SQL	Transform. primitives	add/delete/modify attributes, delete relation
EMQ *Section 4.4*	Prototype	DI (GAV)	Relational XML	IMGQ, ECA, X-Entity	Mediation queries	Change operators	add/remove attribute, add/remove relation, add_ref/remove_ref constraint
AQF *Section 4.5*	System	Generic DB	Relational	Graph model	SQL	Change events	add/delete attribute, add/delete/update condition, delete relation
eLDWS *Section 4.6.1*	Prototype	DW (e-learning)	Relational	Mobile agents	Evolvable SQL	Change operators	add/delete/change name attribute, add/delete/change name relation
MDWS *Section 4.6.2*	Not implemented	DW (multi-dim)	Star Snowflakes	N. H.	Aggregate SQL	Change operators	insert into fact/delete dimension, insert/delete level, connect/disconnect attribute from dimension level, add/delete

Semantic issues / Approaches	Automatism of synchronization	Management of information loss	Global/Local synchronization to QV	Transparency of evolution	Choice among several synchronization	Evaluation of evolution impact	Traceability of changes	Identification of QVs to be synchronized	Propagation to view extent
ARSC *Section 4.1*	Semi	DBA supervision	Global	N. H.	N. H.	N. H.	N. H.	N. H.	N. H.
EVE *Section 4.2*	Complete	VEP and VE	Local: Evolvable SQL	N. H.	N. H.	N. H.	N. H.	N. H.	View Maintainer
BellApp *Section 4.3*	Complete	N. H.	Global	N. H.	N. H.	N. H.	N. H.	N. H.	Containment control
EMQ *Section 4.4*	Complete	N. H.	Local: Operation graph	N. H.	N. H.	N. H.	Lookup process	Operation graph	N. H.
AQF *Section 4.5*	Semi	Policy specification	Local: Policy specification	N. H.	Policy prevalence	What-if analysis	N. H.	N. H.	N. H.
eLDWS *Section 4.6.1*	Complete	VEP and VE	Local: Evolvable SQL	N. H.	N. H.	N. H.	N. H.	VKB agent	N. H.
MDWS *Section 4.6.2*	N. H.	N. H.	N. H.	N. H.	N. H.	QV evolution semantics	N. H.	N. H.	N. H.

* N. H. = Not Handled

Fig. 7. Comparison tables of structural issues and semantic issues of the operation-based approaches.

applications, whereas the remaining ones have been used in either data integration or data warehousing.

Concerning the data model, most of the operation-based approaches are based on the relational one, but two of them have also been extended for the XML data model. One approach, namely, MDWS, is specifically targeted at stars/snowflakes, since it is specialized on evolutions of multidimensional data warehouse schemas. Another important characteristic of the operation-based approaches is that they are based on sufficiently different models and techniques, such as SVS, Containment rewritings, X-Entity, the graph model, mobile agents, and so forth.

In terms of the query/view definition language, one of the surveyed approaches uses SQL, whereas most of the remaining ones propose extensions of it. ARCS and EMQ do not use SQL-like languages. EMQ uses mediation queries, whereas ARCS uses the query/view definition language of the specific applications. As for schema change languages, the majority of approaches use change operators (four out of seven), whereas AQF uses change events. Finally, ARCS and BellApp use transformation primitives.

All the surveyed approaches have similar capabilities for change operations, mainly enabling the addition/deletion of attributes, relations, and constraints, while few of them also handle the modification of these schema constructs. Although not implemented, ARCS also handles more complex modifications, such as the composition/decomposition and merge of schema constructs. Finally, only MDWS manages completely different types of modifications, since they are applied to multidimensional schemas of data warehouses.

Concerning the semantic issues, most of the surveyed operation-based approaches provide a complete automated support to QVS (four out of seven), whereas ARSC and AQF invoke the DBA supervision in the QVS process in order to manage complex cases of schema evolution, such as those involving capacity-reducing modifications. However, the management of information loss has also been faced in EVE and eLDWS through the evolution and the extent parameters (VEP and VE), specified through E-SQL, also yielding a local synchronization of queries/views. The latter is also provided in AQF by means of policy specifications, and in EMQ by means of an operation graph. Finally, only ARSC and BellAPP provide global-level management of QVS.

The remaining semantic issues have been often raised to solve specific problems of the application domain for which an approach has been devised. For this reason, they appear as "not handled" in many surveyed approaches. In particular, none of the surveyed operation-based approaches handles transparency of evolution, since this issue is raised in mapping-based approaches. Only EMQ guarantees the traceability of changes through a lookup process. EMQ and eLDWS are the only approaches facing the problem of identifying the queries/views to be synchronized, which is vital in contexts in which a huge number of queries/views have to be managed. Another important issue is the propagation of evolution to view extents, which has been faced in EVE through the view maintainer, and in BellAP through the containment control. Finally, only AQF and MDWS enable the user/DBA to evaluate the evolution impact, through a what-if analysis in AQF, and through the evolution semantics analysis in MDWS. This is the only semantic issue handled in MDWS, since it is a nonimplemented study.

5. MAPPING-BASED APPROACHES

A schema mapping enables the transformation of data structured according to a specific model into data structured according to a different one, preserving their semantics. This methodology can be adopted in several contexts, such as *data integration*, where the schema mapping describes correspondences between two versions of a database schema. Schema mappings have the advantage of providing a description that is not limited to specific predefined modification operations, but to all the possible modification operations on a database schema.

5.1. The Generic Model Management (GMM)

Schema mappings have rapidly become popular, due to the possibility of applying them to many different contexts. However, this has revealed the necessity of developing a framework for efficiently managing them, together with operators for their manipulation. Model Management is the first proposal in this direction [Bernstein et al. 2000; Bernstein and Rahm 2000; Bernstein 2003] and focuses on two main concepts: schemas and mappings between them. After this first proposal, the Generic Model Management (GMM) has been introduced [Bernstein 2001; Melnik 2004; Bernstein and Melnik 2007], whose main goal is to reduce the programming work necessary to develop applications that solve metadata manipulation problems. In order to achieve this goal, GMM provides a set of algebraic operators enabling the generalization of transformation operations used through the application of metadata. The operators allow one to operate on both schemas and mappings. For this reason, GMM can be applied to many contexts, including data warehousing, e-commerce, relational-object wrapping, enterprise information integration, database portals, and report generators [Bernstein and Melnik 2007].

The operators of GMM range from *Match*, enabling the creation of a mapping between two input data schemas, to *ModelGen*, enabling the translation of schemas defined on different data models. Such operators can be applied only to the schemas and to the mappings between them. As a consequence, queries/views also must be defined according to these two concepts.

As shown in Figure 8, the QVS problem can be naturally seen as a GMM problem. The figure shows that, given a schema S and a mapping $map_{S-S'}$ defining its evolution

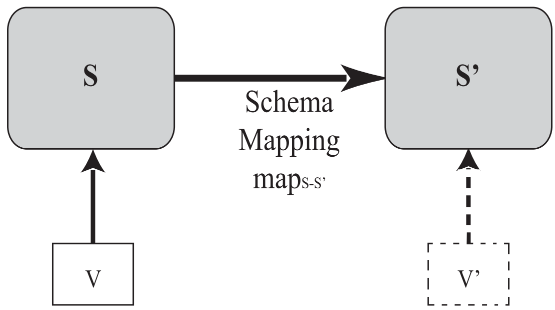

Fig. 8. The schema evolution problem according to the GMM view.

into a schema S', if V is the set of views defined on S, then the QVS problem is to find a set of views V' defined on S', representing a rewrite of V [Bernstein 2003; Bernstein and Melnik 2007]. This is done in a way independent from the specific modifications that have caused the schema evolution.

Two typical solutions to the QVS problem are the *direct* and the *inverse* solution, which differ in the direction of the mapping: when the mapping is from the source to the destination, we have a direct solution, and vice versa we have an inverse one. The former applies operators according to the following sequence of steps:

$map_{S-S'} = \textbf{\textit{Match}}(S, S')$
/* *Finds the mapping between schemas S and S'* */
$map_{V-S'} = \textbf{\textit{Compose}}(map_{V-S}, map_{S-S'})$
/* *Links views in V to the new schema S', composing the mapping between V and* */
/* *S with that between S and S'* */

In particular, this solution allows one to find the mapping defining the evolution $S \rightarrow S'$ by means of the *Match* operator, and then it creates the correspondences between the original views and the modified schema S', by means of the *Compose* operator. Vice versa, the *inverse* solution applies the operators according to the following sequence of steps:

$map_{S-S'} = \textbf{\textit{Match}}(S, S')$
/* *Finds the mapping between schemas S and S'* */
$map_{S'-S} = \textbf{\textit{Invert}}(map_{S-S'})$
/**Undoes the effects of* $map_{S'-S}$*/
$map_{S'-V} = \textbf{\textit{Compose}}(map_{S'-S}, map_{S-V})$
/* *Links view V to the new schema S'* */

In this solution, the operator *Inverse* is applied, which aims at reverting the direction of a mapping, undoing the effects of the previous mapping.

The two presented solutions show how operators defined in GMM can be applied to solve the QVS problem in simple scenarios. However, more complex application scenarios can be faced by applying new operators to these two types of solutions. For instance, when the evolution reduces the capacity of the data schema, as in the case of attribute deletion, then it can be decided either to delete the attribute from all the view definitions or to waive synchronizing the views containing it [Bernstein 2003]. In the last case, the operator *Diff* could be used to exclude all the views that cannot be mapped on the new data schema, whereas in the first case the DBA could define a function f to remove parts of the view definitions that have been deleted during the evolution, and apply f by means of the *Apply* operator.

The main advantage of GMM is that schemas, mappings, and generic operators are independent from the model, enabling the interoperability between different data models. Moreover, this does not limit the evolution of schemas, providing a set of manageable modification operations. The schema evolution is managed by trying to find the equivalence between the queries/views before and after the schema modification, overlooking the queries/views for which it is not possible to find such equivalence.

The *Rondo Programming Platform* [Melnik 2004; Melnik et al. 2003a, 2003b; Melnik 2005] represents the first software supporting GMM. It has been created to solve problems related to model management, schema evolution, reuse of views, and reintegration.[4] It implements all the GMM operators described in the literature, also providing a high-level programming environment. Other than supporting numerous models, the Rondo Platform provides a mechanism to introduce new modeling languages within the prototype.

The architecture of Rondo has a main component, called *Interpreter*, which allows one to organize the dataflow between operators. The latter can be defined by means of *Scripts* or *Applications*, whereas the models and the mappings are represented as objects structured in a shared metamodel and stored in a DBMS or a file system. The prototype also supports basic functionalities of SQL DDL, XML schemas, RDF schemas, SQL views, and UML. Moreover, it provides a Graphical User Interface (GUI) that allows receiving DBA's comments during the management of semiautomatic operations.

As an example, let us consider the following procedure in which the DBA imposes that queries/views containing removed information do not have to be synchronized:

$map_{S-S'} = \boldsymbol{Match}(S, S')$
/ Finds the mapping between schemas S and S′ */*
if $(!\mathbf{Contain}(V, \boldsymbol{Diff}(S, map_{S-S'})))$
/ Verify if there are not elements of S included into V, which are not */*
/ referenced in* $map_{S-S'}$ **/*
then
$map_{V-S'} = \boldsymbol{Compose}(map_{V-S}, map_{S-S'})$
/ Links views in V to the new schema S′, composing the mapping between V and */*
/ S with that between S and S′ */*

According to this procedure, the view *AtmBrPos* given in Equation (7) can be synchronized upon the evolution $S \rightarrow S'$ described in Figure 1, but not after the evolution $S' \rightarrow S''$.

The semantic issues of the approach can be summarized as follows:

—The synchronization process is based on the application of operators in the *direct* and *inverse* solution.
—The *automation of the synchronization* is *complete*.
—Queries based on older versions of a schema can be executed by using the *inverse solution*, which guarantees the *transparency of evolution*.
—The policy of *synchronization* is *global*.

5.2. Default Schema Mappings for Approximate View Synchronization (DSMS)

GMM-based approaches do not consider the possibility of performing synchronizations producing approximate views, mainly needed in the presence of capacity-reducing modifications. GMM-based approaches face these cases by either overlooking views referring to deleted constructs or adopting ad hoc solutions with the assistance of the DBA,

[4]The reintegration problem arises when a model is modified independently by several engineers or tools [Melnik et al. 2003a].

since he or she has the necessary knowledge to decide which parts of the view definition can be dropped.

The introduction of default schema mappings aims at understanding how to produce meaningful answers to queries relying on information lost during the schema evolution [Polese and Vacca 2009b; Lakshmanan et al. 1993], enabling the mapping-based QVS upon the occurrence of capacity-reducing modifications. The whole approach is based on the *inverse solution* of the GMM approach and originates from the idea that when information is lost upon a schema evolution, an attempt should be made to recover data from the source schema, even tolerating some errors. This can be done by means of *default mappings* [Polese and Vacca 2009b], a formalism based on default logic [Reiter 1980], which is suitable to express rules allowing exceptions. More formally, a default rule is a formula like

$$\frac{(\alpha(x) : \beta_1(x), \ldots, \beta_n(x))}{\gamma(x)}, \tag{12}$$

where $\alpha(x)$ is the default prerequisite, $\beta_1(x), \ldots, \beta_n(x)$, $n \geq 0$ are the justifications, and $\gamma(x)$ is the consequence. The formula says that if the prerequisite is true and there does not exist information contradicting the justifications, then it can be assumed that the consequence is also true.

A *Default Mapping (DM)* is defined as a triple $(S, S', \Sigma_{SS'})$, in which the first two elements represent the source and the target schemas, respectively, whereas $\Sigma_{SS'}$ represents the pair (B, D), where B is the set of mappings represented through the logic formalism of FullTGD, and D is the set of default rules.

As an example, let us consider the evolution $S' \rightarrow S''$ of Figure 1, and the following two default rules determining that the first three ATMs of a specific branch are internal, whereas the others are external:

$$\frac{(AtmPoint(ATMcode, address, branch) \wedge N \leq 3 : N = (lastNumber(ATMcode)))}{AtmBranchPoint(ATMcode, address, branch, \text{“}internal''\text{)}}$$

and

$$\frac{(AtmPoint(ATMcode, address, branch) \wedge N > 3 : N = (lastNumber(ATMcode)))}{AtmBranchPoint(ATMcode, address, branch, \text{“}external''\text{)}}.$$

The QVS of the view *AtmBrPos* given in Equation (7) can be accomplished by means of these default rules, which permit approximating the semantics of the original view, preserving the information in the attribute *positioning*.

The semantic issues of the approach can be summarized as follows:

—The synchronization process is based on the application of operators in the *direct* and the *inverse* solution.
—The *automation level of the synchronization* is *complete*.
—The *default mappings* enable the *management of information loss* by allowing one to define approximate queries/views.
—Queries based on older versions of a schema can be performed by using the *inverse solution*, which guarantees the *transparency of evolution*.
—The policy of *synchronization* is *global*.

5.3. The SchemaLog Approach (sLogS)

SchemaLog (sLogS) is a logic language developed for integrating *Heterogeneous DataBase Systems (HDBSs)* [Lakshmanan et al. 1993; Papoulis 1994; Andrews et al. 1996; Lakshmanan et al. 1997; Gingras et al. 1997]. It can be used for schema evolution purposes, since a schema evolution can be viewed as a special case of schema integration. With respect to the specific schema evolution problem, Polese and Vacca [2009a] aimed at guaranteeing an evolution transparency enabling users to submit queries on the schema version that they know, even if this is an old version. Approaches based on SchemaLog can achieve such a goal, except for capacity-reducing modifications, since they yield (meta-)information loss. In order to tackle such a problem, the authors use a cooperative approach to querying [Cuppens and Demolombe 1988], by proposing a query synchronization process based on Hintikka interrogative logic [Hintikka and Bachman 1991]. In particular, in the presence of capacity-reducing modifications, instead of providing direct answers, heuristics are exploited to produce approximate answers, which are submitted to the user or the DBA for approval. Such a process exploits a dialogue for information seeking, in which participants (user and system) aim at finding an adequate mapping between the query and the modified schema. The dialogue starts with a user-provided metaquery, from which the system will derive deductions. Moreover, the system will ask the user to provide missing information, starting a dialogue that will lead to the creation of new deductions and that will terminate only when the user receives an answer to the main metaquery or when he or she decides to stop.

As an example, let us consider the evolution $S \rightarrow S'$ of Figure 1 and the following metaquery:

? - Bank(x) ← AtmBranchPoint(x, “external”),

which extracts all the banks with an external ATM. According to the sLogS approach, the user can submit such query to S by ignoring the evolution, while the following process, representing a dialogue between the system and the user, will permit reinterpreting the query:

USER: ? - Bank(x) ← AtmBranchPoint(x, “external”)
SYSTEM: Error message: “AtmBranchPoint” doesn’t exist
USER: Principal metaquery - Can you map “AtmBranchPoint” in some relation?
SYSTEM: Is “Branch(branch, bank)” good?
USER: Yes
USER: ? - Bank(x) ← Branch(“external”, x)
SYSTEM: No values for x
USER: Principal metaquery - Which relations involve “external”?
SYSTEM: “AtmPoint(ATMcode, address, branch, “external”)”
USER: ? - Bank(x) ← Branch(y,x) ∧ AtmPoint(y, “external”)

The semantic issues of the approach can be summarized as follows:

—The synchronization process is based on the following idea: when it is not possible to achieve a correct QVS, the DBA needs to interact with the system to establish the correct query to be executed. To this end, the authors have based the synchronization process on the Hintikka interrogative logic.
—The *information loss is managed* using *cooperative query answering*.
—Consequently, the *automation level of the synchronization* is *semi*.
—An evolved schema is considered as a new schema that must be *mapped to its older version*, which guarantees *transparency of evolution*.
—The policy of *synchronization* is *global*.

5.4. Attribute Domain Evolution Using Mesodata (MesEv)

The concept of database evolution goes beyond evolutions related to structural modifications of the database schema. Particularly important is also the concept of evolution related to modifications of the schema semantics, identified as evolution of attribute domains. This type of evolution can be subdivided in the following classes of modifications: (1) modifications of the attribute representations; (2) modifications of domain constraints, for example, feasible minimum or maximum value; and (3) modifications of the meaning (perception).

Based on these considerations, de Vries and Roddick [2004]; De Vries et al. [2004], and de Vries and Roddick [2007] defined an approach for the relational data model by introducing complex data structures, which can be viewed as information on attribute domains, at a level between the data and the metadata. Such structures can also be used to support domain modifications. In other words, the mesodata layer extracts the domain in a separated level, namely, the *Mesodata Domain (Mdom)*, making it possible to access the semantics of information, even not directly from the database. In fact, it is not possible to directly access the mesodata type through the attributes; rather, it is possible to access the mesodata type through an external mapping on the database, which relates the intended concept and the data values.

Although this approach does not try to directly find a solution to the QVS problem, it allows one to reduce its complexity in case of modifications involving the domain of data, by introducing the mesodata level.

As an example, let us consider the attribute *ATMCode* from the relation *ATMBranch-Point* of Figure 1, which stores the ATM identification code. Let us also suppose that new standards prescribe that such codes (1) be transformed from the numeric to the alphanumeric format and (2) be started with the first letter of the city where they are located. In order to accomplish such modification without the mesodata layer, it would entail (1) adding a new CHAR attribute, (2) modifying the old values to be assigned to the new attribute, and finally (3) deleting the old attribute. Vice versa, a solution based on the use of the mesodata layer would only require the use of the mesodata type LIST, which will map existing integer values to new CHAR values as follows:

$$ATMCode = Mdom(1234) = {}'S1234'. \tag{13}$$

In this way, it will no longer be necessary to change the attribute *ATMCode* in the relation schema, since both the application and the operators will access it through the mesodata type. Moreover, both values will be accessible and valid.

The semantic issues of the approach can be summarized as follows:

—The synchronization process is based on the construction of an additional domain definition layer, which permits one to manage the domain evolution. This layer is composed of several data structures that are useful to define *mesodata layer mappings* between schema versions, guaranteeing *transparency of evolution*.
—For this reason, the *automation of the synchronization* is *complete*.
—The policy of *synchronization* is *global*.

5.5. The Synchronization of XPath Views (XPathS)

A particular class of data sources is represented by XML sources, whose popularity is growing, due to the necessity of modeling huge quantities of semistructured data available on the web. Such data sources are composed of documents containing data structured according to a schema defined through simple formalisms, like Document Type Definitions (DTDs), or more complex formalisms, like XML schemas.

In order to access data stored within XML documents, it is necessary to use a query language enabling the retrieval of elements from them. To this end, one of the most popular languages is XPath, which allows the specification of powerful queries. It is a Unix-like language, since the basic syntax of an XPath query resembles that of a file path in Unix. Thus, the access to parts of XML documents is realized by defining views on data by means of XPath. The schema of an XML document can either be contained in an external file identified by an URL or be embedded in the document itself. In both cases, a modification of the document might invalidate the views defined on it, requiring a synchronization process to adapt them to the new schema.

This problem has been analyzed by Pedersen and Pedersen [2004a, 2004b], who faced the automatic update of parameterized XPath queries. In their work, they aimed at finding modifications to the XML source schema and at updating XPath-based views to reflect such modifications. Moreover, they defined a prototype implementing basic algorithms for XPath query synchronization.

The approach for finding possible modifications to XML schemas is based on the analysis of query results. However, such queries might return empty results because of possible modifications to the schema, or they might return *incorrect* results. For this reason, the approach also provides an algorithm to detect such cases, by comparing the result of the current query with that of the previous one. Once a modification to the XML schema is found, a heuristic algorithm determines the query that best approximates the original one, by exploiting a policy that evaluates the similarity of queries, based on the similarity of their results.

Although the approach has been implemented for OLAP-based systems and XML data sources (Extended TARGIT system [Pedersen and Pedersen 2004b; Pedersen et al. 2008]), it provides general techniques that are applicable to all those situations in which there are views defined on XML data, as in Web services, B2B applications, and XML-based websites.

The semantic issues of the approach can be summarized as follows:

—The synchronization process is based on a heuristic algorithm that finds the best synchronization based on the *query results*. Consequently, the algorithm must *choose among several synchronization* options.
—The *query result* is analyzed, also to get the *traceability of changes*.
—The *automation of synchronization* is *complete*, and it applies approximations when necessary.
—The policy of *synchronization* is *global*.

5.6. Comparison of Mapping-Based Approaches

The tables shown in Figure 9 summarize the *structural* and *semantic* issues of the mapping-based approaches.

It immediately appears evident that there is only one completely implemented approach (XPathS) and one with a prototype implementation (GMM). Moreover, as opposed to operation-based approaches, all the surveyed mapping-based approaches have been employed in generic database applications, and most of them (three out of five) are based on the relational data model, except XPathS, which is based on XML, and GMM, which is applicable to all data models.

As with operation-based approaches, even in the mapping-based ones there are sufficiently different underlying models and techniques employed. As expected, since mappings and change operations are basically different paradigms, models and techniques used in these two types of approaches also are considerably different. In mapping-based approaches, they include direct/inverse operators, default rules, cooperative query answering, and so forth. Moreover, two out of the five mapping-based approaches use mapping-based queries/views definition languages, whereas two approaches use the languages of the specific applications. Only XPathS is based on XPath, since it is targeted at XML.

As for schema change languages, except for GMM and DSMS, which use algebraic mapping operators, each of the remaining three approaches uses a different mapping language, such as matching formulas and local and external mappings. Finally, concerning the managed changes, MesEv handles changes to attribute representations,

Structural issues / Approaches	Type of approach	Application area	Supported models	Resource - Technique	QV definition language	Schema change language	Managed changes
GMM *Section 5.1*	Prototype	Generic DB	All possible	Direct/inverse operators	Mapping	Algebrical mapping op.	All possible
DSMS *Section 5.2*	Not implemented	Generic DB	Relational	Inverse op. default rules	Default	Algebrical mapping op.	All possible
sLogS *Section 5.3*	Not implemented	Generic DB (heterogen.)	Relational	Coperative Query Ans.	Logical Mapping	Logical Mapping	All possible
MesEv *Section 5.4*	Not implemented	Generic DB	Relational	Data structures	Default	External mapping	Domain evolution: attribute representation changes, domain constraints changes, domain perception (meaning) changes
XPathS *Section 5.5*	System	Generic DB	XML	Heuristics	XPath	Matching formulas	All possible

Semantic issues / Approaches	Automatism of synchronization	Management of information loss	Global/Local synchronization to QV	Transparency of evolution	Choice among several synchronization	Evaluation of evolution impact	Traceability of changes	Identification of QVs to be synchronized	Propagation to view extent
GMM *Section 5.1*	Complete	N. H.	Global	Inverse solution	N. H.	N. H.	N. H.	N. H.	N. H.
DSMS *Section 5.2*	Complete	Default mappings	Global	Inverse solution	N. H.	N. H.	N. H.	N. H.	N. H.
sLogS *Section 5.3*	Semi	Coop. query answering	Global	Mapping to older version	N. H.	N. H.	N. H.	N. H.	N. H.
MesEv *Section 5.4*	Complete	N. H.	Global	Mesodata Layer mappings	N. H.	N. H.	N. H.	N. H.	N. H.
XPathS *Section 5.5*	Complete	N. H.	Global	N. H.	Query results	N. H.	Query results	N. H.	N. H.

* N. H. = Not Handled

Fig. 9. Comparison tables of structural issues and semantic issues of the mapping-based approaches.

domain constraints, and domain meanings, and the remaining four approaches handle all the possible changes.

Concerning semantic issues, except for sLogS, which provides partial automated support, the remaining ones all provide complete support. Only two approaches, namely, DSMS and sLogS, support the management of information loss, through default mappings and cooperative query answering, respectively. All the surveyed approaches provide a global synchronization strategy of queries/views.

Except for XPathS, all the surveyed mapping-based approaches handle transparency of evolution, but only GMM and DSMS use the same method, namely, the inverse solution. To this end, sLogs uses mappings to older database versions, whereas MesEv exploits mesodata layer mappings. Finally, for the remaining semantic issues, similar considerations made for operation-based approaches also apply to mapping-based ones, since they have been introduced for domain-specific problems. In fact, except for XPathS, which handles traceability of changes and choice of synchronization based on query results, the remaining surveyed approaches do not handle any of them.

6. HYBRID APPROACHES

The last class of QVS approaches are the so-called hybrid approaches, which try to exploit the strength points of both operation- and mapping-based approaches described earlier. Thus, in the following, we will describe approaches facing the QVS problem by merging concepts of mapping and of schema modifications guided by a set of predefined operations.

6.1. PRISM

PRISM is one of the tools designed in the context of the macro project Pantha Rei, which includes several research projects concerning schema evolutions and related data management problems [Curino et al. 2008]. In particular, the approach underlying PRISM aims at reducing the gap between ideal solutions to schema evolution and the real world [Curino et al. 2008a, 2008b, 2008c; Moon et al. 2008; Curino et al. 2009, 2010, 2013]. To this end, the authors provide several guidelines on what a system should offer to adequately support schema evolution and related problems. PRISM has been developed based on such guidelines, and it assists a DBA in the design of the schema evolution, guaranteeing the automatic processing of related queries. It accomplishes this task by devoting particular attention to information preservation, redundancy control, and reversibility.

In order to let the DBA specify modifications to the schema, PRISM provides a modification language exploiting Schema Modification Operators (SMOs), which represent a key element within the system. More specifically, an SMO is a function taking as input a relational data schema and an instance of it, and returning the modified version of them. For each SMO, it can be proved that it is possible to find a perfect and unique inverse.

The approach prescribes the conversion of SMOs in the logic language Disjunctive Embedded Dependencies (DEDs), aiming at exploiting the power of logic languages in query reformulation. DED enables the definition of forward and backward mappings that permit one to determine how to switch from the old version of a schema to the new one, and vice versa. Thus, PRISM can be considered as a hybrid approach, due to the use of SMOs and local mappings based on DED. The QVS is performed through the query reformulation algorithm Mixed And Redundant Storage (MARS) [Deutsch and Tannen 2003], exploiting a technique named Chase & Backchase, which finds equivalent queries by only using DED rules and executing the two phases Chase and Backchase. In particular, during the Chase phase, new conjunctions are added to the query based on DED rules, thus deriving a *universal plan*. Vice versa, during the Backchase phase, all the possible atoms are removed from the universal plan, so as to derive an equivalent query.

As an example, let us consider the evolution $S \rightarrow S'$ of Figure 1, which can be defined through the following operator:

DECOMPOSE TABLE ATMBranchPoint INTO
ATMPoint(<u>ATMcode</u>, address, branch, positioning),
Branch(<u>branch</u>, bank)

The modification is converted into the correspondent DED, which is used by the algorithm MARS in order to automatically reformulate the queries/views. For instance, the view given in Equation (3) is automatically synchronized into

$$\begin{aligned}&NewIsInternal(ATMcode, bank)\\&\quad\leftarrow(ATMPoint(\underline{ATMcode}, address, branch, positioning) \wedge\\&\qquad Branch(\underline{branch}, bank)) \wedge\\&\qquad \wedge\, ATMPoint.branch = Branch.branch \wedge\\&\qquad \wedge\, positioning = \text{“}internal\text{”}.\end{aligned}$$

The semantic issues of the approach can be summarized as follows:

—The synchronization process uses the query reformulation *algorithm* MARS, which rewrites queries based on DED mappings associated to the modification operators. For each operator, two DED mappings (forward and backward) have been defined, which determine the operator invertibility. For this reason, by using *DED mappings,* it is possible to get *transparency of evolution*.

—The evolution can be managed by users through an interface that contains *interface operators*, allowing one to *evaluate the evolution impact* and to *manage the information loss* through the *DBA supervision* prior to the evolution.

—The policy of *synchronization* is *global*.
—The *automation level of the synchronization* is *complete*.

6.2. The AutoMed Project

As already specified in Section 2.2, DI methodologies aim at integrating data from several local and autonomous data sources into a single schema, by using mappings between the global and the local schema. The AutoMed system represents a pioneer implementation of the *Both-As-View (BAV)* integration approach, in which mappings between N source schemas, $S_1, \ldots, S_n$, and a global schema S are defined as Step-by-Step pathways of N *transformation primitives* [Boyd et al. 2004]:

$T_1 : S_1 \rightarrow S; \ldots; T_n : S_n \rightarrow S.$

The BAV approach differs from its predecessors in that it enables the integration of data sources based on different data models. In particular, AutoMed supports a low-level, hypergraph-based data model, namely, the Hypergraph Data Model (HDM), which enables the representation of a schema by means of a triple $< Nodes, Edges, Constraints >$ [Poulovassilis and McBrien 1998]; AutoMed provides the Automated Intermediate Query Language (AIQL) for query definition.

The use of transformation primitives in AutoMed provides the basis for the integration of local schemas versus a global one, facilitating possible future evolutions of all of them [McBrien and Poulovassilis 1998, 1999a, 1999b; Brien and Poulovassilis 2001; Jasper 2002; McBrien and Poulovassilis 2002; Fan and Poulovassilis 2003; McBrien and Poulovassilis 2003]. Transformation primitives can be subdivided into *elementary primitives,* which apply to simple constructs of a schema (insertion, deletion, and renaming of nodes, edges, and constraints), and *complex primitives*, which can be derived through the application of a composition operator to elementary primitives. In general, the primitives of schema modification languages can be transformed into transformation primitives of AutoMed, enabling the definition of a transformation path from a source schema to the global one.

One of the strength points of AutoMed is the reversibility of transformations, which enables the reversibility of transformation paths. In order to apply a transformation primitive to a schema, it is necessary to define a query q specifying the semantics of the transformation. Thus, the user/DBA willing to perform a transformation of the schema should precisely know the characteristics and the semantics of the schema itself [Velegrakis et al. 2004a].

The QVS in AutoMed is accomplished by defining synchronization policies based on three types of evolutions. The first one, defined as *equivalence preserving*, prescribes that views be reformulated by using inversion and composition of transformations. The second one, defined as *contraction*, yields a reduction of the information capacity, and it prescribes the construction of an approximated view including only constructs belonging to all the sources. Finally, the last policy, defined as *extension*, requires the user intervention, or alternatively, the use of metaknowledge.

The semantic issues of AutoMed can be summarized as follows:

—The synchronization process is based on the application of transformation primitives defining the logical mapping concerning the evolution of a schema S_i into $S_i{}'$, and the application of synchronization policies based on the specific type of evolution.
—One category of policies invokes the *DBA intervention* in order to determine how to manage the schema evolution. For this reason, the *automation level of the synchronization* is *semi*.
—The reversibility of transformation primitives guarantees *transparency of evolution*.

—The policy of *synchronization* is *local*, because *transformation primitives* are applied to local schema mappings.

6.3. The ToMAS Tool

The Toronto Mapping Adaptation System (ToMAS) [Velegrakis et al. 2003a, 2003b, 2004a, 2004b] is a tool devised with the aim of adapting mappings upon the evolution of one of the involved data schemas. The problem of mapping adaptation is to keep consistency of mappings when a schema evolves, aiming to find semantic preserving rewrites of mappings [Velegrakis et al. 2003b]. Since this is a wide problem, in this work we will limit the discussion to the QVS aspects.

The general goal of the approach underlying ToMAS is to detect the mappings affected by the schema evolution and to deterministically generate semantically valid rewrites of them. Such rewrites ought to be consistent not only with the semantics and the structure of the affected schemas but also with past user choices, which belong to previous mappings. To this end, the authors provided several algorithms capable of computing the semantics of a schema upon a change, in an efficient and incremental way. The concept of incrementality is introduced since the approach describes a schema evolution as a sequence of primitive modifications, classified according to the following three categories: schema semantics modifications (insertion/deletion of schema constraints), schema structure modifications (insertion/deletion of schema constructs), and schema reorganization modifications (copy/renaming of schema constructs). In general, it is possible to find more than one semantically valid rewrite for a specific mapping, in which case the choice falls on the rewrite guaranteeing a minimal change, and the choice is made on the basis of this result.

It is worth noting the hybrid nature of this approach, since it aims to readapt mappings based on a specific classification of modification primitives.

The user/designer interacts with ToMAS through a visual interface enabling him or her to visualize schemas, mappings, and those mappings becoming inconsistent upon a modification. The user interface represents one of the components of the modular architecture of ToMAS, which also contains (1) an evolution engine (the core of the whole architecture), implementing algorithms for each type of modification; (2) a mapping analyzer, storing user-provided associations contained within the mappings; (3) a wrapper handling discrepancies among different schema models; and (4) a ranker classifying rewrites of mappings based on the concept of minimal change.

Finally, ToMAS can be applied effectively to other scenarios, such as data integration, data exchange, model management, physical data design, and so forth. Moreover, the tool has achieved excellent results in terms of performances, making it possible to use interactive applications in mapping adaptation even for big data schemas.

The semantic issues of ToMAS can be summarized as follows:

—The synchronization process is based on several *algorithms* that *detect the queries / views (represented as mappings) to be synchronized*, and it manages mapping rewrites that are consistent with the semantics of both the evolution and the existing mappings. The mappings are defined in terms of user-defined logical associations, provided through the *mapping analyzer* component, which guarantees a *local synchronization of queries / views*.
—The evolution can be managed by users through a *visual interface*, which allows them to *evaluate the evolution impact*.
—It is possible to have more semantically valid rewrites of mappings, so that the *choice among several synchronizations* will be based on the concept of *minimal changes*.
—The *automation of the synchronization* is *complete*.

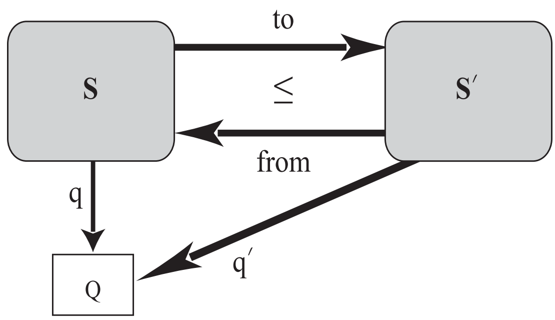

Fig. 10. Coupled transformation of schema query.

6.4. Coupled Software Transformation (CST)

Another important problem that must be tackled to ensure a correct evolution of software systems is the so-called *Coupled Software Transformation* [Cunha and Visser 2007a, 2007b; Visser 2008]. This concept highlights the fact that schema evolutions affect several artifacts of a system, which must be updated in a tightly coupled way to keep global consistency [Lämmel 2004]. A typical example of coupled transformation is the migration of a database instance coherently with a schema modification.

The project *2 Transformation Level (2TL)* aims at conceptualizing coupled transformations, yielding their formalization in terms of two level transformations: type level and value level. They enable the optimization of the query migration process, which can be described as a type-level transformation, from a type of source data to a type of destination data, assisted by a set of conversion functions *to* and *from* between source and destination data.

Finally, in CST, the QVS problem can be viewed as a special case of coupled transformations between the schema and the queries, where source and target schemas are viewed as the data types, and the transformations are modeled as inequalities between data types, through the two functions *to* (representation relation) and *from* (abstraction relation). Figure 10 shows a schematization of the coupled transformation schema query, where $S \leq S'$ means that the schema evolution $S \rightarrow S'$ preserves or increases the information capacity variation of the schema and that the query q' can be derived by composing q and $from$, that is, $q' = q \bullet from$.

The semantic issues of the approach can be summarized as follows:

—The synchronization process is based on the composition of *from* and *to* functions.
—The *automation level of the synchronization* is *complete*.
—The *bidirectionality of transformations* guarantees *transparency of evolution*.
—The policy of *synchronization* is *global*.

6.5. Breaks Queries Under Ontology Changes (BQOC)

Ontologies represent a methodology for knowledge representation. As opposed to databases, they conceptually model the real world by focusing on entities rather than on instances. Thus, an ontology contains entities, relationships among them, rules, axioms, and domain-specific constraints, and enables the extension of real-world knowledge by means of inference rules. Ontologies and instances together represent the Knowledge Base.

Evolution is a complex task even for ontologies, since it is necessary to evaluate the correctness of instances, applications, and services. In the context of ontology evolution, the research has focused on the creation and the management of mappings between different versions of the ontology, and on how to make instances consistent upon a modification. However, research has almost entirely overlooked the impact of a modification on the applications/services depending on it.

The first approach facing the problem of query breaking upon the modification of an ontology uses a log of ontology modifications, aiming to analyze and modify incoming RDF Data Query Language (RDQL) queries, based on modifications stored in the log [Liang et al. 2006]. Moreover, a system prototype accomplishing the following steps has been developed: (1) *capture* changes between two versions of the ontology (by means of the algorithm PROMPTDIFF [Noy and Musen 2002]); (2) *instantiate* modifications within the log of modifications; (3) *analyze* the queries submitted by the applications, verifying whether they are affected by modifications, before submitting them to the ontology; (4) *replace* affected queries with the new ones, allowing one to *update* the entities referred within the queries and those affected by modifications, then submitting them to the ontology; and (5) *enable* the ontology *to answer* the so-modified queries.

The semantic issues of the approach can be summarized as follows:

—The synchronization process uses a log of modifications in order to dynamically rewrite incoming RDQL queries with their synchronizations based on recorded changes. For this reason, the *changes log* guarantees *transparency of evolution*.
—The *automation level of the synchronization* is *complete*.
—The algorithm *PROMPTDIFF* provides the *traceability of changes*.
—The policy of *synchronization* is *global*.

6.6. Schema Versioning/Evolution Approach in Object-Oriented Databases (OOSVE)

Schema evolution can be considered a special case of *schema versioning*, where only the current version of a schema is maintained. In fact, in schema evolution, the goal is to ensure the possibility to perform schema modifications without losing existing data and preserving the semantics of queries/views, which is much more a reduced goal with respect to schema versioning, where the aim is to preserve the semantics of queries/views on any schema version.

Based on the premises earlier, Franconi et al. [2001, 2000] have defined a formal approach for handling schema versioning within the object-oriented data model. They focus on the definition of an extended model, on the formulation of interesting reasoning tasks to support schema evolution, and on the introduction of a code for solving schema versioning tasks. In this way, even though the approach was originally intended for supporting schema versioning, it can also be used in the context of QVS upon schema evolution.

First of all, the authors define a model enabling the representation of multiple schema versions. In particular, they define an evolving schema S as a set of class names (C_s), of attributes (A_s), and a partially ordered set of schema versions. A version of the schema S is defined through the application of a sequence of modifications to some previous version of the schema. Moreover, they define a mechanism enabling the definition of version coordinates that can be used as querying interfaces, or to refer to specific versions.

Finally, the authors define a method to assign a *possible legal state* to each database version. This is a state in which all the constraints imposed by the sequence of modifications performed from the initial version of the database schema are satisfied. Moreover, *data-level materialized views* are introduced upon a schema modification, in order to specify how to compile classes of the new version starting from the data of the previous one.

The semantic issues of the approach can be summarized as follows:

—The synchronization process is based on the definition of change semantics with a specific Description Logic (DL) language; it is possible to define a schema version

Structural issues / Approaches	Type of approach	Application area	Supported models	Resource - Technique	QV definition language	Schema change language	Managed changes
PRISM *Section 6.1*	System	Generic DB	Relational	DED mappings	SQL	SMO	add/drop/rename column, create/drop/rename/copy/merge/partition/decompose/join table
AutoMed *Section 6.2*	System	DI	All possible	HDM, Transf. composition	AIQL	Logical mappings	All possible
ToMAS *Section 6.3*	System	DI, DE	Relational	Minimal changes	Mapping	Modification primitives	add/remove constraints, schema pruning or expansion, schema restructuring
CST *Section 6.4*	Not implemented	Generic DB	All possible	Rewrite system	Point-free functions	Point-free functions	All possible
BQOC *Section 6.5*	Prototype	Generic DB	Ontologies	PROMPTDIFF	RDQL	schema matching	All possible
OOSVE *Section 6.6*	Not implemented	Generic DB	Object-oriented	Description Logic	Default	change operators	add/drop/change name of/change type of attribute, add/drop/change name of/change type of class, add/drop is-a

Semantic issues / Approaches	Automatism of synchronization	Management of information loss	Global/Local synchronization to QV	Transparency of evolution	Choice among several synchronization	Evaluation of evolution impact	Traceability of changes	Identification of QVs to be synchronized	Propagation to view extent
PRISM *Section 6.1*	Complete	DBA Supervision	Global	DED mappings	N. H.	Interface operator	N. H.	N. H.	N. H.
AutoMed *Section 6.2*	Semi	N. H.	Local: Transf. Primitives	Transformation reversibility	N. H.	N. H.	N. H.	N. H.	N. H.
ToMAS *Section 6.3*	Complete	N. H.	Local: mapping analyzer	N. H.	Minimal change	Visual interface	N. H.	Algorithms	N. H.
CST *Section 6.4*	Complete	N. H.	Global	Transformation bi-directionality	N. H.	N. H.	N. H.	N. H.	N. H.
BQOC *Section 6.5*	Complete	N. H.	Global	Changes log	N. H.	N. H.	PROMPTDIFF algorithm	N. H.	N. H.
OOSVE *Section 6.6*	Complete	N. H.	Global	Seq. of modif. on older vers.	N. H.	N. H.	N. H.	N. H.	N. H.

* N. H. = Not Handled

Fig. 11. Comparison tables of structural issues and semantic issues of the hybrid approaches.

through the application of a *sequence of modifications to a previous version*, which guarantees *transparency of evolution*.

—The *automation level of the synchronization* is *complete*.

—The policy of *synchronization* is *global*.

6.7. Comparison of Hybrid Approaches

The tables shown in Figure 11 summarize the *structural* and *semantic* issues of the hybrid approaches.

As could be expected, hybrid approaches mix the characteristics of both operation-based and mapping-based approaches. In fact, we find more implemented systems (three out of six), even though two of them (CST and OOSVE) are not implemented, and one has a prototype implementation. Also, for the application area, we can mix the considerations made for two other types of approaches; hence, the majority of approaches are applied to generic databases, except for AutoMed, which is applied in DI, and ToMAS, which is applied in DI and DE.

Concerning the data model, the six surveyed hybrid approaches can be divided into three categories: those supporting all the data models (AutoMed and CST), those supporting the relational one (ToMAS and PRISM), and those supporting specific data models, such as ontology (BQOC) and the object-oriented data model (OOSVE).

Also, hybrid approaches are based on many different models and techniques that have little in common with the other two classes of approaches. Thus, one important conclusion about this survey is that there has been no leading model or technique that has conditioned the development of QVS approaches.

As for the query/view definition languages, they are all different among the surveyed approaches, and as in mapping-based ones, there is a reduced use of SQL (only PRISM). In this context, it is worth mentioning RDQL for dealing with ontologies (BQOC) and mappings in ToMAS.

As for schema change languages, we find a mix of languages used in operation- and mapping-based approaches. Also for managed changes, half of the surveyed approaches reflect the characteristics of the mapping-based ones, enabling one to manage all possible changes, and the remaining half enable the management of common schema change operations mentioned in operation-based approaches.

Concerning the semantic issues, we cannot say that hybrid approaches always exploit the advantages of both operation- and mapping-based approaches. In fact, although we find more complete automated support for synchronizations, there is only one approach (PRISM) supporting the management of information loss w.r.t. about 60% in operation-based and 50% in mapping-based approaches. The synchronization policies reflect more the mapping aspect, since four out of six use global synchronization policies. This is more evident for the support of the transparency of evolution, since no operation-based approach supports it, whereas like mapping-based ones, also in hybrid approaches we find five of them supporting it. However, they use completely different techniques.

7. DISCUSSION AND FINAL REMARKS

The QVS problem can be considered a critical one, since its resolution enables the possibility of continuing to use information systems when schemas evolve.

The analysis of the existing approaches reveals that a considerable research effort has been devoted to this area, which has led to the proliferation of many different approaches and tools. The interesting thing here is that each research activity has led to both the definition of a technique, tool, or system to solve the problem and a better comprehension of the characteristics of the problem itself.

One of the main goals of this work has been to provide a means to group and analyze QVS approaches and systems, enabling a critical comparison of them. To this end, a classification framework has been introduced, whose parameters have been used to characterize the approaches and the systems analyzed in the previous sections.

It allows us to perform an in-depth analysis of the surveyed approaches from different points of view. Nevertheless, it turns out to be impossible to precisely evaluate each single approach, given the many factors to be considered. Thus, although the proposed analysis does not allow us to determine which is the best approach, it enables us to perform both a general comparison based on relevant characteristics and to detect the most useful and suitable approach for a given application context. However, from the analysis made in this work, we can affirm that there is still a low number of implemented approaches that are effectively usable in practice. In fact, many approaches are either not implemented or implemented only at a prototype level. Thus, the QVS problem has been deeply analyzed in all of its aspects, but there is still no evidence about which tool(s) is (are) capable of completely solving the problem. This is mainly due to the fact that the implemented approaches are either too specific for a given application context (as, for example, the studies on XPath views and on ontologies, of Sections 5.5 and 6.5, respectively), are still immature or rudimentary and hence do not enable some types of modification operations, or handle the problem through different data models. In fact, although some of the surveyed approaches seem to be perfect from a theoretical and an architectural point of view, they still lack implementation accuracy and completeness.

The shortage of effectively implemented approaches also prevented us from making considerations and analyses concerning performances. In fact, only for few approaches are efficiency/performance aspects documented [Jalel 2007; Velegrakis et al. 2004b].

It is also important to notice that some parameters of the proposed framework allow us to highlight both the choices characterizing adopted solutions and strength/weak points of the approaches as a consequence of such choices. As an example, approaches with a global synchronization policy will provide the advantage of simplicity but will not be suitable for handling specific cases of the problem, as in the case of approaches keeping a local synchronization policy, which are able to refer to the single query/view. A similar consideration can be made for the parameter *automation level* of the synchronization. In fact, although manual approaches are not considered useful, there is a gap between the simplicity of use for automatic approaches and the completeness of semiautomatic ones. Moreover, although the VE and VEP parameters contribute to raise the automation level of the QVS process, the specification of such parameters is only possible when the DBA knows the schema semantics, which is not possible in big environments like the web. Finally, it would be desirable to develop more effective approaches for synchronizing queries/views in the presence of capacity-reducing changes. Currently, we are working on techniques exploiting approximate functional dependencies [Nambiar and Kambhampati 2004; Bohannon et al. 2007; Chang et al. 2007; Caruccio et al. 2016], which can be extracted from data instances and can provide clues on how to handle deleted data when these are involved in functional dependencies. In fact, such functional dependencies provide us information on data related to the deleted ones, which can help us better synchronize queries/views defined on them.

Given the current state of the art, the QVS research area can proceed in one of two directions. The first yields the definition of approaches and systems that are specific with respect to single application domains. The second yields the definition of approaches and systems that seek a global solution to the QVS problem, taking into consideration all the relevant aspects. These can be viewed as the basis from which to start the needed specialization for the application to specific contexts.

REFERENCES

Serge Abitebul, Richard Hull, and Victor Vianu. 1995. *Foundation of Databases*. Addison-Wesley.

Alanoly J. Andrews, Nematollaah Shiri, Laks V. S. Lakshmanan, and Iyer N. Subramanian. 1996. On implementing schemaLoga database programming language. In *Proceedings of the 5th International Conference on Information and Knowledge Management (CIKM'96)*. ACM, 309–316.

Jay Banerjee, Won Kim, Hyoung-Joo Kim, and Henry F. Korth. 1987. *Semantics and Implementation of Schema Evolution in Object-Oriented Databases*. Vol. 16. ACM Press. 311–322 pages.

Jonas Barklund, Pierangelo DellAcqua, Stefania Costantini, and Gaetano A. Lanzarone. 1997. Metareasoning agents for query-answering systems. In *Flexible Query Answering Systems*. Springer, 103–121.

Zohra Bellahsene. 2002. Schema evolution in data warehouses. *Knowledge and Information Systems* 4, 3 (2002), 283–304.

Philip A. Bernstein. 2001. Generic model management: A database infrastructure for schema manipulation. In *Cooperative Information Systems*. Springer, 1–6.

Philip A. Bernstein. 2003. Applying model management to classical meta data problems. In *Proceedings of the 1st Biennial Conference on Innovative Data Systems Research (CIDR'03)*.

Phillip A. Bernstein, Alon Y. Halevy, and Rachel A. Pottinger. 2000. A vision for management of complex models. *ACM Sigmod Record* 29, 4 (2000), 55–63.

Philip A. Bernstein and Sergey Melnik. 2007. Model management 2.0: Manipulating richer mappings. In *Proceedings of the ACM SIGMOD International Conference on Management of Data (COMAD'07)*. ACM, 1–12.

Philip A. Bernstein and Erhard Rahm. 2000. Data warehouse scenarios for model management. In *Conceptual Modeling (ER'00)*. Springer, 1–15.

Elisa Bertino. 1992. A view mechanism for object-oriented databases. In *Advances in Database Technology (EDBT'92)*. Springer, 136–151.

Philip Bohannon, Wenfei Fan, Floris Geerts, Xibei Jia, and Anastasios Kementsietsidis. 2007. Conditional functional dependencies for data cleaning. In *Proceedings of the IEEE 23rd International Conference on Data Engineering, 2007 (ICDE'07)*. IEEE, 746–755.

Mokrane Bouzeghoub, Bernadette Farias Lóscio, Zoubida Kedad, and Ana Carolina Salgado. 2003. Managing the evolution of mediation queries. In *On The Move to Meaningful Internet Systems 2003: CoopIS, DOA, and ODBASE*. Springer, 22–37.

Michael Boyd, Sasivimol Kittivoravitkul, Charalambos Lazanitis, Peter McBrien, and Nikos Rizopoulos. 2004. AutoMed: A BAV data integration system for heterogeneous data sources. In *Advanced Information Systems Engineering*. Springer, 82–97.

Peter McBrien and Alexandra Poulovassilis. 2001. A semantic approach to integrating XML and structured data sources. In *Advanced Information Systems Engineering*. Springer, 330–345.

Loredana Caruccio, Vincenzo Deufemia, and Giuseppe Polese. 2014. Visual data integration based on description logic reasoning. In *Proceedings of the 18th International Database Engineering & Applications Symposium*. ACM, 19–28.

Loredana Caruccio, Vincenzo Deufemia, and Giuseppe Polese. 2016. Relaxed functional dependencies - a survey of approaches. *IEEE Transactions on Knowledge and Data Engineering* 28, 1 (2016), 147–165.

Shi-Kuo Chang, Vincenzo Deufemia, Giuseppe Polese, and Mario Vacca. 2007. A normalization framework for multimedia databases. *IEEE Transactions on Knowledge and Data Engineering* 19, 12 (2007), 1666–1679.

Alcino Cunha and Joost Visser. 2007a. Strongly typed rewriting for coupled software transformation. *Electronic Notes in Theoretical Computer Science* 174, 1 (2007), 17–34.

Alcino Cunha and Joost Visser. 2007b. Transformation of structure-shy programs: Applied to XPath queries and strategic functions. In *Proceedings of the 2007 ACM SIGPLAN Symposium on Partial Evaluation and Semantics-Based Program Manipulation*. ACM, 11–21.

Frédéric Cuppens and Robert Demolombe. 1988. Cooperative answering: A methodology to provide intelligent access to databases. In *Proceedings of 2nd International Conference Expert Database Systems*. 621–643.

Carlo Curino, Hyun Jin Moon, Alin Deutsch, and Carlo Zaniolo. 2013. Automating the database schema evolution process. *VLDB Journal International Journal on Very Large Data Bases* 22, 1 (2013), 73–98.

Carlo Curino, Hyun Jin Moon, and Carlo Zaniolo. 2008a. Managing the history of metadata in support for DB archiving and schema evolution. In *Proceedings of Advances in Conceptual Modeling – Challenges and Opportunities*. Springer, 78–88.

Carlo Curino, Hyun J. Moon, and Carlo Zaniolo. 2009. Automating database schema evolution in information system upgrades. In *Proceedings of the 2nd International Workshop on Hot Topics in Software Upgrades*. ACM, 5.

Carlo A. Curino, Hyun Jin Moon, Alin Deutsch, and Carlo Zaniolo. 2010. Update rewriting and integrity constraint maintenance in a schema evolution support system: Prism++. *Proceedings of the VLDB Endowment* 4, 2 (2010), 117–128.

Carlo A. Curino, Hyun Jin Moon, MyungWon Ham, and Carlo Zaniolo. 2009. The PRISM workwench: Database schema evolution without tears. In *Proceedings of 25th International Conference on Data Engineering (ICDE'09)*. IEEE, 1523–1526.

Carlo A. Curino, Hyun J. Moon, and Carlo Zaniolo. 2008b. Graceful database schema evolution: The prism workbench. *Proceedings of the VLDB Endowment* 1, 1 (2008), 761–772.

Carlo A. Curino, Hyun J. Moon, and Carlo Zaniolo. 2008c. Managing the history of metadata in support for db archiving and schema evolution. In *Advances in Conceptual Modeling–Challenges and Opportunities*. Lecture Notes in Computer Science, Vol. 5232. Springer, 78–88.

Carlo A. Curino, Letizia Tanca, Hyun J. Moon, and Carlo Zaniolo. 2008. Schema evolution in wikipedia: Toward a web information system benchmark. In *Proceedings of the International Conference on Enterprise Information Systems (ICEIS'08)*. Citeseer, 323–332.

Krzysztof Czarnecki, J. Nathan Foster, Zhenjiang Hu, Ralf Lämmel, Andy Schürr, and James F. Terwilliger. 2009. Bidirectional transformations: A cross-discipline perspective. In *Theory and Practice of Model Transformations*. Springer, 260–283.

Cristina De Castro, Fabio Grandi, and Maria Rita Scalas. 1997. Schema versioning for multitemporal relational databases. *Information Systems* 22, 5 (1997), 249–290.

Denise de Vries, Sally Rice, and John F. Roddick. 2004. In support of mesodata in database management systems. In *Database and Expert Systems Applications*. Springer, 663–674.

Denise de Vries and John F. Roddick. 2004. Facilitating database attribute domain evolution using mesodata. In *Evolution and Change in Data Management (ECDM'04) - ER 2004 Workshops*, Vol. 3289.

Denise de Vries and John F. Roddick. 2007. The case for mesodata: An empirical investigation of an evolving database system. *Information and Software Technology* 49, 9 (2007), 1061–1072.

Vincenzo Deufemia, Massimiliano Giordano, Giuseppe Polese, and Genoveffa Tortora. 2014. A visual language-based system for extraction–transformation–loading development. *Software: Practice and Experience* 44, 12 (2014), 1417–1440.

Alin Deutsch and Val Tannen. 2003. MARS: A system for publishing XML from mixed and redundant storage. In *Proceedings of the 29th International Conference on Very Large Data Bases (VLDB'03)*, Vol. 29. VLDB Endowment, 201–212.

Ronald Fagin. 2006. Inverting schema mappings. In *Principles of Database Systems (PODS'06)*, Vansummeren (Ed.). 50–59.

Ronald Fagin, Phokion G. Kolaitis, Lucian Popa, and Wang-Chiew Tan. 2005. Composing schema mappings: Second-order dependencies to the rescue. *ACM Transactions on Database Systems (TODS)* 30, 4 (2005), 994–1055.

Hao Fan and Alexandra Poulovassilis. 2003. Using AutoMed metadata in data warehousing environments. In *Proceedings of the 6th ACM International Workshop on Data Warehousing and OLAP*. ACM, 86–93.

Fabrizio Ferrandina, Thorsten Meyer, and Roberto Zicari. 1994. Implementing lazy database updates for an object database system. In *Proceedings of the International Conference on Very Large Data Bases (VLDB'94)*. Citeseer, 261–261.

Enrico Franconi, Fabio Grandi, and Federica Mandreoli. 2000. A semantic approach for schema evolution and versioning in object-oriented databases. In *Computational Logic (CL'00)*. Springer, 1048–1062.

Enrico Franconi, Fabio Grandi, and Federica Mandreoli. 2001. Schema evolution and versioning: A logical and computational characterisation. *Database Schema Evolution and Meta-Modeling.* Springer, 85–99.

Frédéric Gingras, Laks V. S. Lakshmanan, Iyer N. Subramanian, Despina Papoulis, and Nematollaah Shiri. 1997. Languages for multi-database interoperability. In *ACM SIGMOD Record*, Vol. 26. ACM, 536–538.

Fabio Grandi and Federica Mandreoli. 2003. A formal model for temporal schema versioning in object-oriented databases. *Data & Knowledge Engineering* 46, 2 (2003), 123–167.

Peter Haase and Boris Motik. 2005. A mapping system for the integration of owl-dl ontologies. In *Proceedings of the 1st International Workshop on Interoperability of Heterogeneous Information Systems*. ACM, 9–16.

Alon Y. Halevy, Zachary G. Ives, Peter Mork, and Igor Tatarinov. 2003. Piazza: Data management infrastructure for semantic web applications. In *Proceedings of the 12th International Conference on World Wide Web*. ACM, 556–567.

Mark Hansen, Stuart Madnick, and Michael Siegel. 2003. *Data Integration Using Web Services*. Springer.

Jean-Marc Hick and Jean-Luc Hainaut. 2006. Database application evolution: A transformational approach. *Data & Knowledge Engineering* 59, 3 (2006), 534–558.

Jaakko Hintikka and James Bachman. 1991. *What If-? Toward Excellence in Reasoning*. Mayfield Publishing Company.

Richard Hull. 1986. Relative information capacity of simple relational database schemata. *SIAM Journal of Computing* 15, 3 (1986), 856–886.

Akaichi Jalel. 2007. E-learning data warehouse maintenance system for collaborative learning availability resources optimization. *International Journal of Education and Development Using ICT* 3, 3 (2007), 16–29.

Edgar Jasper. 2002. Global query processing in the AutoMed heterogeneous database environment. In *Proceedings of the 19th British National Conference on Databases (BNCOD'02)*. Springer, 46–49.

Ole Guttorm Jensen and Michael Böhlen. 2002. Current, legacy, and invalid tuples in conditionally evolving databases. In *Advances in Information Systems (ADVIS'02)*. Springer, 65–82.

Peter Sune Jørgensen and Michael Böhlen. 2007. Versioned relations: Support for conditional schema changes and schema versioning. In *Advances in Databases: Concepts, Systems and Applications*. Springer, 1058–1061.

Christian Kaas, Torben Bach Pedersen, and Bjørn Rasmussen. 2004. Schema evolution for stars and snowflakes. *ICEIS* (2004), 425–433.

Zoubida Kedad and Mokrane Bouzeghoub. 1999. Discovering view expressions from a multi-source information system. In *Proceedings of the International Conference on Cooperative Information Systems (IFCIS'99)*. IEEE, 57–68.

Andreas Koeller and Elke A. Rundensteiner. 2000. History-driven view synchronization. In *Data Warehousing and Knowledge Discovery*, Y. Kambayashi, M. K. Mohania, and A. M. Tjoia (Eds.). Lecture Notes in Computer Science, Vol. 1874. Springer, 168–177.

Andreas Koeller and Elke A. Rundensteiner. 2005. A history-driven approach at evolving views under meta data changes. *Knowledge and Information Systems* 8, 1 (2005), 34–67.

Andreas Koeller, Elke A. Rundensteiner, and Nabil Hachem. 1998. Integrating the rewriting and ranking phases of view synchronization. In *Proceedings of the 1st ACM International Workshop on Data Warehousing and OLAP (DOLAP'98)*. ACM, 60–65.

Phokion G. Kolaitis. 2005. Schema mappings, data exchange, and metadata management. In *Proceedings of the 24th ACM SIGMOD-SIGACT-SIGART Symposium on Principles of Database Systems (PODS'05)*, C. Li (Ed.). ACM, 61–75.

Laks V. S. Lakshmanan, Fereidoon Sadri, and Iyer N. Subramanian. 1993. On the logical foundations of schema integration and evolution in heterogeneous database systems. In *Deductive and Object-Oriented Databases (DOOD'93)*. Springer, 81–100.

Laks V. S. Lakshmanan, Fereidoon Sadri, and Iyer N. Subramanian. 1997. Logic and algebraic languages for interoperability in multidatabase systems. *Journal of Logic Programming* 33, 2 (1997), 101–149.

Ralf Lämmel. 2004. Coupled software transformations. In *Proceedings of the 1st International Workshop on Software Evolution Transformations*. 31–35.

Amy J. Lee, Andreas Koeller, Anisoara Nica, and Elke A. Rundensteiner. 1999a. Data warehouses evolution: Trade-offs between quality and cost of query rewritings. In *Proceedings of 15th International Conference on Data Engineering (ICDE'99)*. IEEE, 255.

Amy J. Lee, Andreas Koeller, Anisoara Nica, and Elke A. Rundensteiner. 1999b. Non-equivalent query rewritings. In *International Database Conference*. Citeseer.

Amy J. Lee, Anisoara Nica, and Elke A. Rundensteiner. 2002. The EVE approach: View synchronization in dynamic distributed environments. *IEEE Transactions on Knowledge and Data Engineering* 14, 5 (2002), 931–954.

Maurizio Lenzerini. 2002. Data integration: A theoretical perspective. In *Proceedings of the 21st ACM SIGMOD-SIGACT-SIGART Symposium on Principles of Database Systems (PODS'02)*, L. Popa (Ed.). ACM, 233–246.

Barbara Staudt Lerner. 2000. A model for compound type changes encountered in schema evolution. *ACM Transactions on Database Systems (TODS)* 25, 1 (2000), 83–127.

Xue Li. 1999. A survey of schema evolution in object-oriented databases. In *Proceedings of Technology of Object-Oriented Languages and Systems (TOOLS'99)*. IEEE, 362–371.

Yaozhong Liang, Harith Alani, and Nigel Shadbolt. 2006. Changing ontology breaks queries. In *The Semantic Web (ISWC'06)*, I. F. Cruz, S. Decker, D. Allemang, C. Preist, D. Schwabe, P. Mika, M. Uschold, and L. Aroyo (Eds.). Lecture Notes on Computer Science, Vol. 4275. Springer, 982–985.

Bernadette Farias Lóscio and Ana Carolina Salgado. 2004. Evolution of XML-based mediation queries in a data integration system. In *Conceptual Modeling for Advanced Application Domains*, S. Wang, D. Yang, K. Tanaka, F. Grandi, S. Zhou, E. E. Mangina, T. W. Ling, I.-Y. Song, J. Guan, and H. C. Mayr (Eds.). Lecture Notes in Computer Science, Vol. 3289. Springer, 402–414.

Jayant Madhavan, S. Jeffery, Shirley Cohen, X. Dong, David Ko, Cong Yu, and Alon Halevy. 2007. Web-scale data integration: You can only afford to pay as you go. In *Proceedings of the 3rd Biennial Conference on Innovative Data Systems Research (CIDR'07)*. 342–350.

Andy Maule, Wolfgang Emmerich, and David S. Rosenblum. 2008. Impact analysis of database schema changes. In *Proceedings of the 30th International Conference on Software Engineering (ICSE'08)*. ACM, 451–460.

Peter McBrien and Alexandra Poulovassilis. 1998. A formalisation of semantic schema integration. *Information Systems* 23, 5 (1998), 307–334.

Peter McBrien and Alexandra Poulovassilis. 1999a. Automatic migration and wrapping of database applications - a schema transformation approach. In *Proceedings of 18th International Conference on Conceptual Modeling*. Springer, 96–113.

Peter McBrien and Alexandra Poulovassilis. 1999b. A uniform approach to inter-model transformations. In *Advanced Information Systems Engineering*. Springer, 333–348.

Peter McBrien and Alexandra Poulovassilis. 2002. Schema evolution in heterogeneous database architectures, a schema transformation approach. In *Advanced Information Systems Engineering*. Springer, 484–499.

Peter McBrien and Alexandra Poulovassilis. 2003. Data integration by bi-directional schema transformation rules. In *Proceedings of 19th International Conference on Data Engineering*. IEEE, 227–238.

Sergey Melnik. 2004. *Generic Model Management: Concepts and Algorithms*. Vol. 2967. Springer-Verlag New York Incorporated.

Sergey Melnik. 2005. Model management: First steps and beyond. *BTW, LNI* 65 (2005), 455–464.

Sergey Melnik, Erhard Rahm, and Philip A. Bernstein. 2003a. Developing metadata-intensive applications with Rondo. *Web Semantics: Science, Services and Agents on the World Wide Web* 1, 1 (2003), 47–74.

Sergey Melnik, Erhard Rahm, and Philip A. Bernstein. 2003b. Rondo: A programming platform for generic model management. In *Proceedings of the 2003 ACM SIGMOD International Conference on Management of Data*, A. Y. Halevy, Z. G. Ives, and Doan A. (Eds.). ACM, 193–204.

Renée J. Miller, Mauricio A. Hernández, Laura M. Haas, Ling-Ling Yan, CT Howard Ho, Ronald Fagin, and Lucian Popa. 2001. The Clio project: Managing heterogeneity. *SIgMOD Record* 30, 1 (2001), 78–83.

Renée J. Miller, Yannis E. Ioannidis, and Raghu Ramakrishnan. 1993. The use of information capacity in schema integration and translation. In *Proceedings of the 19th International Conference on Very Large Data Bases (VLDB'93)*, Vol. 93. Citeseer, 120–133.

Hyun J. Moon, Carlo A. Curino, Alin Deutsch, Chien-Yi Hou, and Carlo Zaniolo. 2008. Managing and querying transaction-time databases under schema evolution. *Proceedings of the VLDB Endowment* 1, 1 (2008), 882–895.

Mirella M. Moro, Susan Malaika, and Lipyeow Lim. 2007. Preserving XML queries during schema evolution. In *Proceedings of the 16th International Conference on World Wide Web*, C. L. Williamson, M. E. Zurko, P. F. Patel-Schneider, and P. J. Shenoy (Eds.). ACM, 1341–1342.

Ullas Nambiar and Subbarao Kambhampati. 2004. Mining approximate functional dependencies and concept similarities to answer imprecise queries. In *Proceedings of the 7th International Workshop on the Web and Databases: Colocated with ACM SIGMOD/PODS 2004*. ACM, 73–78.

Anisoara Nica, Amy J. Lee, and Elke A. Rundensteiner. 1998. The CVS algorithm for view synchronization in evolvable large-scale information systems. In *Advances in Database Technology (EDBT)*, H.-J. Schek, F. Saltor, I. Ramos, and G. Alonso (Eds.). Lecture Notes in Computer Science, Vol. 1377. Springer, 357–373.

Anisoara Nica and Elke A. Rundensteiner. 1998. Using containment information for view evolution in dynamic distributed environments. In *Proceedings of the 9th International Workshop on Database and Expert Systems Applications (DEXA'98)*. IEEE, 212–217.

Natalya F. Noy and Mark A. Musen. 2002. PROMPTDIFF: A fixed-point algorithm for comparing ontology versions. In *Proceedings of the National Conference on Artificial Intelligence (AAAI/IAAI'02)*. 744–750.

George Papastefanatos, Fotini Anagnostou, Yannis Vassiliou, and Panos Vassiliadis. 2008. Hecataeus: A what-if analysis tool for database schema evolution. In *Proceedings of the 12th European Conference on Software Maintenance and Reengineering (CSMR'08)*. IEEE, 326–328.

George Papastefanatos, Panos Vassiliadis, Alkis Simitsis, and Yannis Vassiliou. 2007. What-if analysis for data warehouse evolution. In *Data Warehousing and Knowledge Discovery*. Springer, 23–33.

George Papastefanatos, Panos Vassiliadis, Alkis Simitsis, and Yannis Vassiliou. 2009. Policy-regulated management of ETL evolution. *Journal on Data Semantics XIII*. Springer, 147–177.

George Papastefanatos, Panos Vassiliadis, Alkis Simitsis, and Yannis Vassiliou. 2010. HECATAEUS: Regulating schema evolution. In *Proceedings of the IEEE 26th International Conference on Data Engineering (ICDE'10)*. IEEE, 1181–1184.

George Papastefanatos, Panos Vassiliadis, and Yannis Vassiliou. 2006. Adaptive query formulation to handle database evolution. In *Proceedings of the 18th International Conference on Advanced Information Systems Engineering (CAiSE'06) (CEUR Workshop Prooceedings)*, N. Boudjilida, D. Cheng, and N. Guelfi (Eds.), Vol. 231.

Despina Papoulis. 1994. *Realizing SchemaLog*. Technical Report. Department of Computer Science, Concordia University, Montreal, Canada.

Dennis Pedersen and Torben Bach Pedersen. 2004a. Synchronizing XPath views. In *International Database Engineering and Applications Symposium (IDEAS'04)*. IEEE, 149–160.

Dennis Pedersen and Torben Bach Pedersen. 2004b. *Synchronizing XPath Views*. Technical Report 7. DBTR.

Torben Bach Pedersen and Christian S. Jensen. 2001. Multidimensional database technology. *IEEE Computer* 34, 12 (2001), 40–46.

Torben Bach Pedersen, Dennis Pedersen, and Jesper Pedersen. 2008. Integrating XML data in the TARGIT OLAP system. *International Journal of Web Engineering and Technology (IJWET)* 4, 4 (2008), 495–533.

Randel J. Peters and M. Tamer Özsu. 1997. An axiomatic model of dynamic schema evolution in objectbase systems. *ACM Transactions on Database Systems (TODS)* 22, 1 (1997), 75–114.

Giuseppe Polese and Mario Vacca. 2009a. A dialogue-based model for the query synchronization problem. In *Proceedings of the IEEE 5th International Conference on Intelligent Computer Communication and Processing (ICCP'09)*.

Giuseppe Polese and Mario Vacca. 2009b. Notes on view synchronization using default logic. In *Proceedings of 17th Italian Symposium on Advanced Database Systems (SEBD'09)*. 253–260.

Alexandra Poulovassilis and Peter McBrien. 1998. A general formal framework for schema transformation. *Data & Knowledge Engineering* 28, 1 (1998), 47–71.

Erhard Rahm, Andreas Thor, David Aumueller, Hong-Hai Do, Nick Golovin, and Toralf Kirsten. 2005. iFuice-information fusion utilizing instance correspondences and peer mappings. In *Proceedings of 8th WebDB*. Citeseer, 7–12.

Sudha Ram and G. Shankaranarayanan. 2003. Research issues in database schema evolution: The road not taken. *Boston University School of Management, Department of Information Systems, Working Paper* 2003-15 (2003).

Raymond Reiter. 1980. A logic for default reasoning. *Artificial Intelligence* 13, 1 (1980), 81–132.

John F. Roddick, Noel G. Craske, and Thomas J. Richards. 1993. A taxonomy for schema versioning based on the relational and entity relationship models. In *Entity-Relationship Approach (ER'93)*. Springer, 137–148.

Elke A. Rundensteiner, A. Koeller, A. Lee, Y. Li, A. Nica, and X Zhang. 1998. Evolvable view environment (EVE) project: Synchronizing views over dynamic distributed information sources. In *Demo Session Proceedings of International Conference on Extending Database Technology (EDBT'98)*. Citeseer, 41–42.

Elke A. Rundensteiner, Andreas Koeller, and Xin Zhang. 2000. Maintaining data warehouses over changing information sources. *Communications of the ACM* 43, 6 (2000), 57–62.

Elke A. Rundensteiner, Andreas Koeller, Xin Zhang, Amy J. Lee, Anisoara Nica, A. Van Wyk, and Y. Lee. 1999. Evolvable view environment (EVE): Non-equivalent view maintenance under schema changes. In *Proceedings of the SIGMOD Conference*, A. Delis, C. Faloutsos, and S. Ghandeharizadeh (Eds.), Vol. 28. ACM Press, 553–555.

Elke A. Rundensteiner, Amy J. Lee, and Anisoara Nica. 1997. On preserving views in evolving environments. In *Knowledge Representation Meets DataBases (KRDB'97) (CEUR Workshop Proceedings)*, F. Baader, M. A. Jeusfeld, and W. Nutt (Eds.), Vol. 8. 13.11–13.11.

Len Seligman, Peter Mork, Alon Halevy, Ken Smith, Michael J. Carey, Kuang Chen, Chris Wolf, Jayant Madhavan, Akshay Kannan, and Doug Burdick. 2010. OpenII: An open source information integration toolkit. In *Proceedings of the 2010 ACM SIGMOD International Conference on Management of Data*. ACM, 1057–1060.

Ben Shneiderman and Glenn Thomas. 1982a. An architecture for automatic relational database system conversion. *ACM Transactions on Database Systems (TODS)* 7, 2 (1982), 235–257.

Ben Shneiderman and Glenn Thomas. 1982b. Automatic database system conversion: Schema revision, data translation, and source-to-source program transformation. In *Proceedings of the National Computer Conference*. ACM, 579–587.

Balder Ten Cate and Phokion G. Kolaitis. 2009. Structural characterizations of schema-mapping languages. In *Proceedings of the 12th International Conference on Database Theory (ICDT'09) (ACM International Conference Proceeding)*, R. Fagin (Ed.), Vol. 361. ACM, 63–72.

James F. Terwilliger, Anthony Cleve, and Carlo A. Curino. 2012. How clean is your sandbox? In *Theory and Practice of Model Transformations*. Springer, 1–23.

Andreas Thor, David Aumueller, and Erhard Rahm. 2007. Data integration support for mashups. In *Workshops at the 22nd AAAI Conference on Artificial Intelligence*.

Yannis Velegrakis, Renée J. Miller, and Lucian Popa. 2003a. *Adapting Mappings in Frequently Changing Environments*. CSRG 468. University of Toronto, Department of Computer Science.

Yannis Velegrakis, Renée J. Miller, and Lucian Popa. 2003b. Mapping adaptation under evolving schemas. In *Proceedings of the 29th International Conference on Very Large Data Bases (VLDB'03)*, Vol. 29. VLDB Endowment, 584–595.

Yannis Velegrakis, Renée J. Miller, and Lucian Popa. 2004a. Preserving mapping consistency under schema changes. *VLDB Journal* 13, 3 (2004), 274–293.

Yannis Velegrakis, Renée J. Miller, Lucian Popa, and John Mylopoulos. 2004b. Tomas: A system for adapting mappings while schemas evolve. In *Proceedings of the 20th International Conference on Data Engineering (ICDE'04)*. IEEE Computer Society, 862.

Joost Visser. 2008. Coupled transformation of schemas, documents, queries, and constraints. *Electronic Notes in Theoretical Computer Science* 200, 3 (2008), 3–23.

Michael Weiss and G. R. Gangadharan. 2010. Modeling the mashup ecosystem: Structure and growth. *R&D Management* 40, 1 (2010), 40–49.

Jin Yu, Boualem Benatallah, Fabio Casati, and Florian Daniel. 2008. Understanding mashup development. *IEEE Internet Computing* 12, 5 (2008), 44–52.

Roberto Zicari. 1991. A framework for schema updates in an object-oriented database system. In *Proceedings of the 7th International Conference on Data Engineering (ICDE'91)*. IEEE Computer Society, 2–13.

Received February 2014; revised July 2015; accepted December 2015