Chapter 11
Querying Conflicting Web Data Sources

Gilles Nachouki, Mohamed Quafafou, Omar Boucelma,
and François-Marie Colonna

Abstract. Over the last twenty years, information integration has received considerable efforts from both industry and academia. Approaches to information integration developed so far can be categorized as follows: (1) first-generation approaches, that require the definition of a global schema and a semantic integration which should be performed upfront (before query execution); (2) second-generation approaches, well illustrated by the *dataspace management* concept, which promote a *pay-as-you-go* data integration. The first category has led to well known mediation approaches such as GAV (Global as View), LAV (Local as View), GLAV (Generalized Local As View), BAV (Both As View), and BGLAV (BYU Global-Local-as-View). Approaches pertaining to the second category are geared towards the development of dataspace management systems and are currently gaining a lot of attention. In this chapter we are interested in exploiting both types of approaches in querying conflicting data spread over multiple web sources. To this aim, first we show how an XML-based BGLAV approach can handle these conflicting data sources, then we describe how the same problem can be addressed by using the Multi Fusion Approach (MFA), an approach pertaining to second-generation techniques. Both BGLAV and MFA are illustrated in using genomic data sources accessible through the Web.

Gilles Nachouki
LINA-UMR CNRS 6241, Nantes University, France
e-mail: `Gilles.Nachouki@univ-nantes.fr`

Mohamed Quafafou · Omar Boucelma
LSIS-UMR CNRS 6168, Aix-Marseille University, France
e-mail: `{mohamed.quafafou,omar.boucelma}@lsis.org`

François-Marie Colonna
Institut Supérieur de l'Electronique et du Numérique, France
e-mail: `francois-marie.colonna@isen.fr`

11.1 Introduction

Over the past two decades, the database communities (both industry and academia), have intensively addressed data integration problems. A first-generation category of approaches and systems has been developed, and significant contributions has been made in various subtopics [16] such as: source descriptions, schema mappings, query reformulation, and incomplete information modeling. Approaches pertaining to this first category require upfront semantic integration, that is a global/mediation schema needs to be supplied beforehand. As an example, in mediation approaches such as GAV (Global as View) [12], LAV(Local as View) [23], GLAV(Generalized Local As View) [11] or BAV(Both As View) [5], the *mediator* provides the user with a global schema and allows him/her to access heterogeneous data sources providing the illusion to access a single local database. The *wrapper*, another component of mediation systems, plays the role of an interface between the mediator and the data sources: it receives queries from a mediator and uses its own knowledge (source descriptions and mapping rules) in order to access the data sources.

As an example, in LAV the content of each data source is expressed in terms of a view over the global schema. Mapping rules associate a query over the global schema to each element of the local sources. On the other hand, in GAV each element of the global schema is expressed in terms of a view over the data sources. Each mapping rule associates a query over a local source to each element in the global schema.

The main component of first-generation data integration systems is the query rewriting module; it explores a set of mappings in order to rewrite queries, expressed upon the global schema, in terms of local sources' schemas. The complexity of the query rewriting phase depends on how the global schema is defined. For example, GAV query rewriting is very simple since the elements in the global schema are defined in terms of the source schemas [22]. In this case, query rewriting simply consists in unfolding the definitions of the elements in the global schema. However, in this case, adding a new source to the data integration system is not trivial. The new source may indeed have an impact on the definition of various elements of the global schema, whose associated views need to be redefined. On the other hand, LAV query rewriting is not straightforward (exponential time complexity) and many rewriting algorithms have been developed; among them, we recall the Bucket, Inverse Rules and MiniCon algorithms [14], to cite a few. At the same time, however, the LAV approach favors the extensibility of the system: adding a new source simply means enriching the mapping with a new assertion, without other changes.

In general, query rewriting works well assuming that the schema of the local source is known a-priori and static. Unfortunately, such assumptions are not satisfied by data managed in several new processing environments, where data sources to be integrated are selected and combined on-demand. New second-generation data integration approaches have therefore been proposed. Dataspaces and Dataspace Management Systems (DSMSs), described in Chapter 12, represent a significant example of second-generation integration approaches. A dataspace has the following features [10]:

1. It can handle data with different formats accessible through different interfaces (e.g., database system, data file etc.). A dataspace is designed to support all kinds of data (e.g., structured, semi-structured or unstructured data etc.).
2. It provides an interface to search, retrieve, update and manage a dataspace, through a DSMS. Unlike a Database Management System (DBMS), a DSMS does not completely control its data; but it offers various levels of services in order to return the best answer.
3. A dataspace provides all software in order to improve data integration.

The requirements and the architecture of a DSMS is presented in detail in [15]. DSMSs promote a *pay-as-you-go* integration system where "the system starts with very few (or inaccurate) semantic mappings and these mappings are improved over time as deemed necessary" [33, 19].

In this chapter, we exploit both first-generation and second-generation approaches in describing how to query multiple heterogeneous conflicting web data sources. To this aim, we assume that web data sources are represented in XML; a concrete example drawn from Genomics will be used to illustrate the proposed concepts. Two different data integration techniques will be considered. The first relies on a mediation approach [8, 7] based on BGLAV [36],[1] first defined for relational data and then extended to deal with XML data sources [7]. BGLAV has been proposed to overcome both GAV and LAV limitations. BGLAV improves both GAV and LAV because the global schema remains unchanged when a data source is added or updated. BGLAV, as a first-generation data integration approach, is characterized by the presence of a global schema and the need to specify a set of predefined, hard-coded correspondence queries (mappings), which specify how to solve conflicts among local sources. Mappings have to be specified before submitting queries to the mediator, which is in charge of the translation process, that leads to the generation of the sub-queries posed over local schemas.

In the web context, the definition of such a global schema and the maintenance of mappings become cumbersome. As an alternative approach to BGLAV we therefore consider the Multi-Fusion Approach (MFA), a data fusion method developed by Nachouki et al [26]. MFA, although not being directly inspired by the dataspace management systems concepts, relates to the second-generation of data integration approaches. MFA does not assume the definition of a global schema beforehand; rather, it relies on the Multi-data source Fusion Language (MFL) for the definition of a multi-data source schema (a kind of dataspace) and the retrieval of data issued from conflicting sources. With MFA, conflicts (i.e., assertions or mappings) between data sources can be specified by a (skilled) user who has some domain knowledge. Users have the possibility to refine these conflicts later in order to increase gradually the accuracy of their queries. In MFA, unlike BGLAV, retrieving data from conflicting data sources is established using semantic queries, which may include conflicting elements in their bodies. MFA query processing consists in resolving

[1] BYU Global-Local-as-View, where BYU stands for Brigham Young University.

conflicts and then in decomposing the query into a set of sub-queries to be sent to data sources for execution.

The contributions of this chapter are therefore twofold: (1) it fully illustrates the various phases of the data integration process, namely, schema integration, data reconciliation (fusion), query rewriting, etc., in taking into account both a traditional and a more advanced integration approach; (2) it demonstrates the application of the presented concepts and algorithms in using a real data set, drawn from the Genomics domain. The chapter is organized as follows. Section 11.2 describes conflicts between (biological) data sources and provides a taxonomy of conflicts in a general context. Section 11.3 illustrates how to process mediated queries in BGLAV. Section 11.4 is devoted to MFA. An overall example taken from the biological domain is provided in Section 11.5. Finally, Section 11.6 presents some discussion and conclusions and outlines future work.

11.2 Conflicting Web Data Sources

In the following, we first briefly classify conflicts that may arise from multiple data sources to be integrated. Then, we discuss various types of conflicts arising from the life science domain through an example and we present some assumptions about conflict representation upon which the results presented in the chapter rely.

11.2.1 Overview of Conflict Types

In most data integration examples covered in the literature, data sources present various types of heterogeneity, concerning differences in names, data structures, types, scale, just to cite a few. This is due to the fact that several perceptions of the same real world lead to different data modeling of the same entity. In order to integrate a set of data sources, all such conflicts have first to be solved. Conflicts can be classified as follows [35]:

- *Data conflicts*, referring to differences among definitions, such as attribute types, formats, or precision.
- *Structural conflicts*, arising from the description of the same concept in different ways and in different data sources. For example, a concept can be defined as an attribute in a relational schema `Sch1` and as a relation in another schema `Sch2`.
- *Descriptive conflicts*, including the usage of different names for the same entity (e.g., homonymous, synonymous), identity conflicts (e.g., a person is identified by a number in `Sch1` and social-security number in `Sch2`), scale conflicts (e.g., salaries are given in Dollar and Euro respectively in `Sch1` and `Sch2`).
- *Abstraction conflicts*, concerning the presence of generalization/specialization concepts (e.g., the concept of employee in `Sch1` generalizes the concept of

teacher in `Sch2`) and aggregation (e.g., date of birth in `Sch1` is a string while it is composed of three fields month, day and year in `Sch2`).

- *Semantic conflicts*, referring to differences and similarities in the meaning of concepts in the data sources.

Many works have investigated semantic conflicts in the literature. In [6], authors propose an algorithm which takes two schemas as input and returns the mappings that identify corresponding concepts in the two schemas, namely the concepts with the same or the closest meaning. In [32], authors provide a survey of different approaches to automatic schema matching.

11.2.2 Conflicting Data in Life Sciences

Many data management applications require the integration of data from multiple sources [9], often available on the Web as XML documents. For instance, in the field of biology, the number of data sources and tools available in the Web has grown in recent years. This huge augmentation of data sources has led to a deep heterogeneity between data sources and to a variety of interfaces. Until today, the reconciliation between data sources is performed manually by biologists. Scientific investigations on *Genes* or *Proteins* -for annotations or predictions- or information retrieval from scientific publications (journals, conferences, etc.) often lead researchers to submit queries to several (yet heterogeneous) data sources that are available on the Web. As an example, Mootha et al. [25] discovered one of the genes responsible of *Leigh* syndrome by integrating both expression, genomic and sub-cellular localization data.

In the biological domain, the same/identical information may be stored using distinct formats or structures such as ASN 1.0 [2] or Fasta [30], HTML or XML, leading to some data conflicts. As an example, Figure 11.1 shows a description of *ILB12*, a gene that encodes a subunit of *interleukin 12*, which is one of the regulatory proteins that are released by cells of the immune system. As illustrated, the same entity, *ILB12*, is described by means of several heterogeneous schemas.

Semantic conflicts are also quite common in the life sciences domain. For example, in [34] two definitions of the concept of *gene* have been compared: in GDB [18], a gene is a DNA fragment that can be transcribed and translated into a protein; for Genbank [3] and GSDB [21], a gene is a "*DNA region of biological interest with a name and that carries a genetic trait or phenotype*", which includes nonstructural coding DNA regions like intron, promoter and enhancer. There is a clear semantic distinction between those two notions of gene but both are still being used, hence adding another level of complexity into the data integration process. Another term that comes with multiple meanings is *protein function*, that could be defined either as a biochemical function (e.g., enzyme catalysis), a genetic function (e.g., transcription repressor), a cellular function (e.g. scaffold), or as a physiological function (e.g., signal transducer).

```
<!--                              HTML                                 -->
<!-------------------------------------------------------------------->
<table border="0" width="100" cellpadding="1" cellspacing="1">
  <tr>
    <td nowrap="nowrap">Entry name</td>
    <td width="100">
      <b>IL12B\_HUMAN</b>
    </td>
  </tr>
  <tr>
    <td nowrap="nowrap">Primary accession number</td>
    <td>
      <b>P29460</b>
    </td>
  </tr>
  <tr>
    <td nowrap="nowrap">Integrated into Swiss-Prot on</td>
    <td>April 1, 1993</td>
  </tr>
</table>
```

```
<!--                              ASN 1.0                              -->
<!-------------------------------------------------------------------->
Seq-entry ::= set {
  descr {title "Interleukin-12 subunit beta" ,
         update-date std {year 1991 ,month 5 ,day 17} ,
         source {org {taxname "Homo sapiens" , common "human" ,
                 db {db "taxon" , tag id 9606}
                }
         }
}
```

```
<!--                              FASTA                                -->
<!-------------------------------------------------------------------->
>IL12B|chr5|-|158674369|158690059
GATTACAAAGAAGAGTTTTTATTAGTTCAGCCTCAGAATGCAAAAATAAA
%TAAATAAATAAACAAACAGGAAACAAATGTAATCACTTTACAGAGCGCAC
ATACATTACTTAAAAGTAGCACCTTCATGGAGCCATATTTTCTGGTCATA
..................................................
```

```
<!--                              XML                                  -->
<!-------------------------------------------------------------------->
<SNPPER-RPC SOURCE="*RPCSERV-NAME*" VERSION="$Revision: 1.38$"     >
                                          GENOME="hg17" DBSNP="123">
  <GENEINFO>
    <GENEID>16348</GENEID>
    <NAME>IL12B</NAME>
    <CHROM>chr5</CHROM>
    <STRAND>-</STRAND>
    <TRANSCRIPT>
      <START>158674369</START>
      <END>158690059</END>
    </TRANSCRIPT>
  </GENEINFO>
</SNPPER-RPC>
```

Fig. 11.1 Structural conflicts between Genomics data sources.

11.2.3 Assumptions about Conflict Representation

In this chapter, we will consider the most important types of conflicts detailed above, i.e., data conflicts, structural conflicts, descriptive conflicts, and semantic conflicts. In order to deal with conflicts between data sources, we assume that all data sources schemas are represented according to a common data model (e.g., XML schema, DTD, relational model, etc.). In this chapter, we consider XML documents; we assume that schema information is represented as a DTD document. To simplify the discussion, we rely on a tree-based representation of both XML documents and schema information. Under this assumption, semantic conflicts between elements are specified using the concept of contexts of elements. The *context* of an element `E` is the set of elements connected to `E` by a parent-child or ancestor-descendant relationship. The context of an element is therefore the set of elements semantically depending on it. In other words, if a node `E2` is a child of a node `E1`, the element `E2` has to be interpreted in the scope of `E1`'s meaning. As a consequence, different occurrences of the same label do not have the same meaning: for example, label `Name` may appear several times in the same tree under different contexts, thus representing different semantic entities.

11.3 Mediating Biological Conflicting Data with BGLAV

In this section, we illustrate the BGLAV approach for querying conflicting web data sources.

11.3.1 BGLAV Overview

BGLAV was proposed initially by Li Xu et al. in [36] in the context of relational databases. We adapted this approach for mediating web data sources, represented as XML documents and optionally conflicting, queried through XQuery [7]. BGLAV [36] alleviates GAV and LAV drawbacks in defining source-to-target mappings based on a predefined conceptual target schema (global schema), which is specified independently of any of the sources. More precisely, in a GAV approach, changes in information sources or adding a new information source require revisions of the global schema and mappings between the global schema and source schemas. In a LAV approach, automating query reformulation is hard (i.e., it has exponential time complexity with respect to query and source schema definitions).

To resolve these problems, BGLAV offers an alternative point of view in defining source-to-target mappings based on a predefined conceptual global schema. In particular, the global schema in BGLAV is ontologically specified, independently of any of the sources. BGLAV keeps the advantages of the two approaches GAV and LAV: GAVs simple query reformulation and LAVs scalability. Additionally, it is characterized by the following features:

- Each concept in a target schema (global schema) is predefined and independent of any source schema. In contrast, under GAV, Data Base Administrators (DBAs) revise the global schema to include all concepts represented inside any source; under LAV, DBAs adjust the source schemas such that they contain only source relations that can be described by views over the global schema.
- A set of source-to-target mappings maps a source schema to a target schema. Source and target schemas can use different structures and vocabularies.
- When a new local source becomes available (i.e., a change occurs), a source-to-target mapping must be created (or adjusted).

11.3.2 Query Processing in BGLAV

In this section, we highlight BGLAV query processing. First we provide some necessary background, then we illustrate the query rewriting steps. We start by introducing the concept of *correspondence query* between a source schema and a global schema. The idea is that of defining a correspondence, i.e., a mapping, between (a part of) the source schema and (a part of) the global schema, taking into account existing conflicts.

Definition 1 (Correspondence query). Let S_l be a source schema and G be a global schema. Let T_{S_l} and T_G be two sub-trees belonging to S_l and G respectively. A correspondence query (or mapping query) is defined as a set of transformation operations which are applied to T_{S_l} and produce a new tree denoted by $T_{S_{l'}}$ whose elements are in direct correspondence with those of T_G. We assume that transformation operators are specified using XQuery.

Example 1. Consider the data sources S_1 and S_2 and the global schema G, represented in Figure 11.2. Let T_{S_l} and T_G be the sub-trees showed in the figure. What follows is an example of a correspondence query between S_1 and G:

```
<length>
  for $x in document(S1)/strands/dna/strand/length
  return $x/3
</length>
```

This query illustrates a scale conflict between the two elements *length* in G and S_1. The following is an example of a correspondence query between S_2 and G:

```
<date_seq>
  for $x in document(S2)/genes_list/gene/strand/date_seq
  return concat($x/day,'/', $x/month, '/', $x/year)
</date_seq>
```

This second query illustrates a structural conflict due to the different representations of element *date* in G and S_2, respectively.

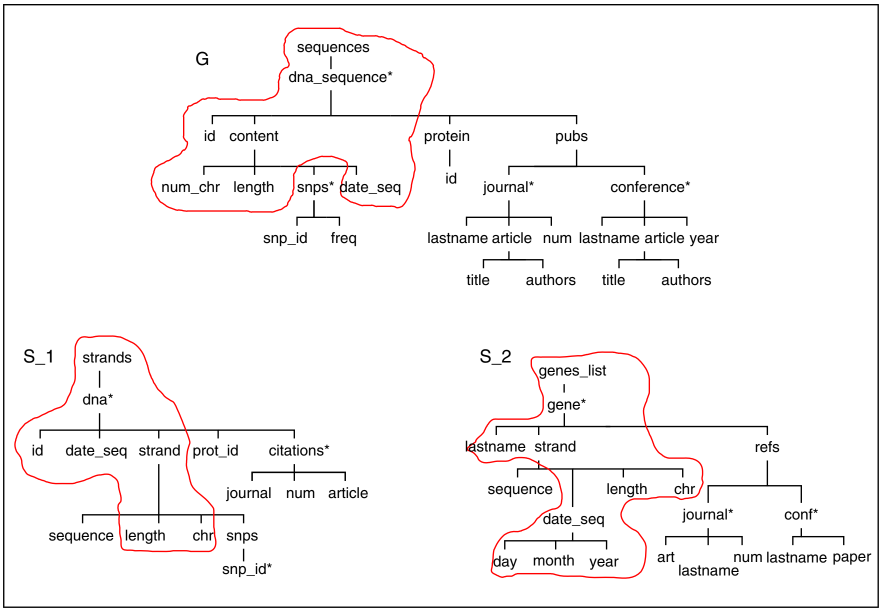


Fig. 11.2 Examples of queries in BGLAV.

Given a set of correspondence queries, from each local source it is possible to generate a derived schema, which is not (or at least partially) conflicting with G.

Definition 2 (Derived schema). Let S_l be a source schema and M_l be a set of correspondence queries that associate S_l to the global schema G. The schema V_l obtained by applying the mapping queries to S_l is called *derived schema*. The transformed elements in V_l are in direct correspondence with those of G.

Derived schemas satisfy the following property (inclusion dependency) with respect to the global schema.

Proposition 1 (Inclusion dependency). *Let $V = \{V_i | i \in [1,n]\}$ be a set of derived schemas and G be the global schema. For each sub-tree T_G of G, there exists a subset of derived schemas $\overline{V} = \{V_{i_1}, ..., V_{i_k}\}$ such that T_G corresponds to a set of sub-trees $\{T_{j,h}\}$ of $V_j \in \overline{V}$, $j \in \{i_1, ..., i_k\}$. The instances of T_G and $T_{j,k}$ satisfy an inclusion dependency defined as follows: $I(T_{j,k}) \subseteq I(T_G)$, where $I(T)$ denotes the set of instances (i.e., XML documents) of the sub-tree T.*

User queries can now be defined as follows.

Definition 3 (User query). Let G be the global schema, represented by a tree T_G. A query Q over G is expressed in XQuery over T_G and can be defined by a logical rule as Q(T_G):- T_1, T_2,..., T_p, C_Q, where:

- T_i, $i \in [1,p]$, is a sub-tree of T_G, specified through an XQuery expression, in accordance with the following sentence: $\nexists i \in [1,p]$, $\nexists j \in [1,p]$, $i \neq j$, and T_i is a sub-tree of T_j;
- C_Q is a set of conditions upon trees T_1, T_2,..., T_p, specified according to XQuery.

Depending on the relationship existing between the query tree and derived schemas, three distinct types of queries can be devised, as defined below. Different query types will lead to different choices during the query rewriting step.

Definition 4 (Completely and partially covered query). Let G be the global schema, represented by a tree T_G. Let $V = \{V_i | i \in [1,n]\}$ be a set of derived schema. Let Q(T_G):- T_1, T_2,..., T_p, C_Q be a query over T_G. Sub-trees T_i, $i \in [1;p]$, appearing in Q can be classified according to the three cases described below:

1. *Complete coverage*: T_i is completely covered by a sub-tree in all derived schemas in V.
2. *Partial coverage on some derived schemas*: T_i is partially covered by some derived schemas in V and completely covered by some others derived schemas in V;
3. *Partial coverage on all derived schemas*: T_i is partially covered by all derived schemas in V.

In case of complete coverage, the answer is the union of the results coming from local sources. In case of partial coverage on some derived schemas, from some sources only a partial result can be retrieved. In case of partial coverage on all derived schemas, partial results should be joined (through XQuery) by means of the key elements they share in order to get query answers.

Figure 11.3 illustrates a case of partial coverage, which is quite common in the web context. The partial answers returned by the two data sources are joined by means of the common values of key elements (e.g., *id_gene* or *gene_id*).

Query processing is performed in mainly two steps [8, 7]:

1. The first step of the algorithm consists in identifying the correspondence queries that should be taken into account in processing the user query Q specified over the global schema. These correspondence queries will be used to define the queries to be executed against the local sources. The overall idea is that of trying to completely cover the query trees by joining local results obtained from correspondence queries together. Correspondence queries generating deriving schemas which are either totally or partially covering trees T_i in Q are taken into account. When the query is decomposed into sub-queries, each tree T_i in Q is then replaced by such correspondence queries, according to the local sources that should be accessed.
2. The second step consists in generating the query plan, which contains the set of sub-queries that access data sources in order to extract the results. Then, the results obtained from each data source are merged together (i.e., they are fused) and returned to the user. For detailed description of the algorithm that performs this step, we refer the reader to [7].

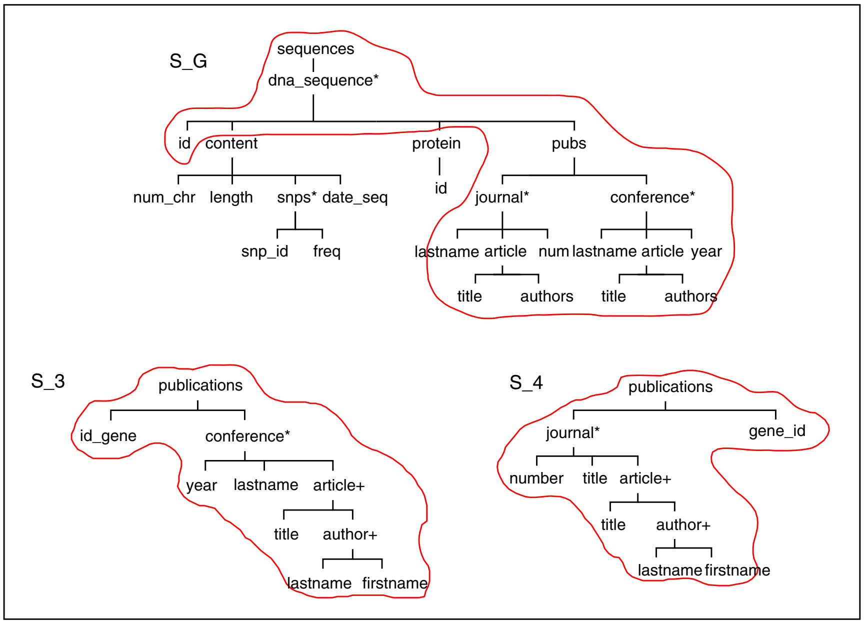


Fig. 11.3 Queries over partially covered global schema.

11.4 MFA - Multi-source Fusion Approach

In this section, we describe a second-generation of data integration approach and we show its use in querying conflicting web data sources.

11.4.1 MFA Overview

A lot of data sources are freely accessible on the Web and users often need to integrate them quickly without any help. Classical approaches based on mediator's reasoning do not facilitate the user's task since it is hard to unify data sources in a dynamic way. Rather, they assume a global mediated schema to model data sources. As such, these approaches often require an administrator to control the mediated schema. MFA differs from traditional approaches, like LAV, GAV, GLAV and BGLAV, in the two following aspects: (i) it does *not require a global schema* or ontology; (ii) mappings are established *only between data sources* (unlike other approaches). The motivation behind this approach is that, in some context, making calls to an administrator to check the mediated schema is not always required and it may be too onerous and restrictive. For example, in the web context, a user which wishes to integrate biological data is not forced to call an administrator to control the mediated schema. MFA thus provides an inexpensive solution to a hard problem of data integration. By cons, this approach requires a minimal user knowledge about these sources and of their conflicts.

Under MFA, the user just locates its data sources (e.g., web sites), builds the multi-data source schema, and submits the queries. Submitting queries may be accomplished before resolving all the conflicts between data sources: users have the possibility to add (or refine) conflicts later in order to increase gradually the accuracy of their queries. This approach facilitates the integration of new data sources or deletion of an existing source. It also provides some languages that permit to define and retrieve data from multiple data sources while taking into account conflicts between sources.

A data integration system that follows the MFA approach is defined as a triple $\langle MS, S_i, M\rangle$, where:

- *MS* is the multi-data source schema;
- S_i is the set of data sources' schemas;
- *M* is the set of source-to-source mappings expressed as functionals *f:* $T_{S_1} \to T_{S_2}$; f maps elements s_1 appearing in the tree T_{S_1} corresponding to the source schema S_1 into elements $s_1^{'}$ appearing in the tree T_{S_2} corresponding to the source schema S_2.

MFA is based on the Multi-data source Fusion Language (MFL). MFL allows the definition and the retrieval of data originating from conflicting data sources, through the concept of *multi-data source* as a set of local sources. MFL is a simple and powerful language. It facilitates queries over conflicting data sources and controls the semantics expressed in user queries. For each query posed over a multi-data source schema, MFL will search for conflicts in the query body. If no conflict is detected, the query is validated and executed; otherwise three cases may arise (for more details, we refer the reader to [26, 27, 28]):

1. conflicts can be solved at query execution time by using the available source-to-source mappings: in this case, the query is validated and executed;
2. conflicts cannot be solved (e.g., in the case of homonymies): the query is rejected;
3. only a subset of the conflicts can be solved: in this case, only a part of the query, related to the solved conflicts, will be executed and the results will be returned to the user with a warning message informing him/her of the detected conflict nature.

MFL provides two sub-languages [26]: MDL, the *Multi-data source Definition Language* for the definition of a multi-data source, and MRL, the *Multi-data source Retrieval Language* for data retrieval from a multi-data source. Defining a multi-data source in MDL is quite simple and intuitive: a collective name is assigned to a group of data sources. A collective name simplifies query expression; users specify inter-source conflicts between elements composing the multi-data source and store them into an additional specific data source. MRL extends XQuery in order to access multiple conflicting data sources through a single query. With MRL, it is easy to smooth out all semantic data differences which often exist in autonomous data sources.

In MDL a multi-data source schema is a collection of data source or multi-source schemas. It can be defined as an XML document, according to the Document Type Definition (DTD) presented in Figure 11.4. In such DTD, $< multisources >$ and $< source >$ elements refer to a specific multi-data source or a data source, respectively. Attribute $< name >$ denotes either the name of a source, of a multi-data source, or of a property in a data source. Attribute $< url >$ describes the path to reach a data source. Element $< feature >$ represents a property of a data source.

Conflicts between data sources are represented in a specific data source (thus, an XML document) called *Conflicts.xml*. Such data source represents the set M of mappings among data sources, specified to deal with conflicts. The structure of *Conflicts.xml* is detailed in Figure 11.5. Three distinct types of semantic and data conflict information can be represented for elements of a multi-data source: similarity and dissimilarity (elements *Similar* and *Dissimilar*) and scale conflicts (element *Scale*). In all the three cases, the children elements *Node* contain information concerning the elements involved in the declaration. Thus, any two elements children of element *Similar* are considered as semantically equivalent or synonym; any two elements children of element *Dissimilar* are considered as semantically different; any two elements children of element *Scale* are considered semantically similar with conflict of type *scale* (e.g., currency type).

Scale conflicts are resolved through a service. Element *Services* specifies the services (e.g., functions) devoted to resolve a conflict of a specific type (e.g., currency type) available in the multi-data source. A service is selected (during a query's treatment) following the type of conflict that occurs. For example, to resolve a conflict between two nodes N_1 and N_2 under a tag *Scale* with Currency type, a specific service corresponding to this type of conflicts may take node N_1 as input and returns a value that conforms with the currency of the second node N_2.

Example 2. Figure 11.7 shows the schema of the *Genome* MDS. It is composed of two conflicting data sources SL_1 and SL_2. They are conflicting since they contain elements with the same name. Conflicts are illustrated in Figure 11.6. File *Conflict.xml* specifies that features (i.e., element) id_1 and *Description* (in SL_1) are respectively similar to id_2 and *Description* (in SL_2).

```
<!ELEMENT multisources (source|multisources)+) >
<!ATTLIST multisources name CDATA #REQUIRED >
<!ELEMENT source (feature)+ >
<!ATTLIST source name CDATA #REQUIRED >
<!ATTLIST source url CDATA #REQUIRED >
<!ELEMENT feature (#PCDATA) >
<!ATTLIST feature name CDATA #REQUIRED >
```

Fig. 11.4 Multi-source DTD.

Element	Children(s)	Attribute(s)	Description
Conflicts	*Similar*∗, *Dissimilar*∗, *Scale*∗, *Services*∗	-	Root element
Similar	*Node*∗	Id	Similar elements. Each similar element has an identifier
Node	Path, Elt, Unit?	-	Description of a Node (i.e., name, path, etc.)
Dissimilar	Path, Elt,*Path*	Id	Dissimilar elements. Each dissimilar element has an identifier
Scale	*Node*	Type	Scales conflicts and their types (e.g., currency)
Services	*Service*∗	Type	Available services
Service	*Name*,*Path*,*Convert*	-	Description of a service
Convert	Unit1, Unit2	-	Information used in conversion services
Name	PCDATA	-	Text to be parsed
Unit1	PCDATA	-	Text to be parsed
Unit2	PCDATA	-	Text to be parsed
Path	PCDATA	-	Text to be parsed
Elt	PCDATA	-	Text to be parsed
Unit	PCDATA	-	Text to be parsed

Fig. 11.5 Structure of data source conflicts; labels in italic identify element names.

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE CONFLICTS SYSTEM "conflicts.dtd">
<CONFLICTS>
  <SIMILAR id="s1">
    <Node>
      <PATH>Bio/Adn/SL1/liste_genes_X/gene</PATH>
      <ELT>id1</ELT>
    </Node>
    <Node>
      <PATH>Bio/Adn/SL2/liste/EnsembleGene_ID</PATH>
      <ELT>id2</ELT>
    </Node>
  </SIMILAR>
  <SIMILAR id="s2">
    <Node>
      <PATH>Bio/Adn/SL1/liste_genes_X/gene</PATH>
      <ELT>description</ELT>
    </Node>
    <Node>
      <PATH>Bio/Adn/SL2/liste/EnsembleGene_ID</PATH>
      <ELT>description</ELT>
    </Node>
  </SIMILAR>
  ...
</CONFLICTS>
```

Fig. 11.6 Conflicts between SL_1 and SL_2.

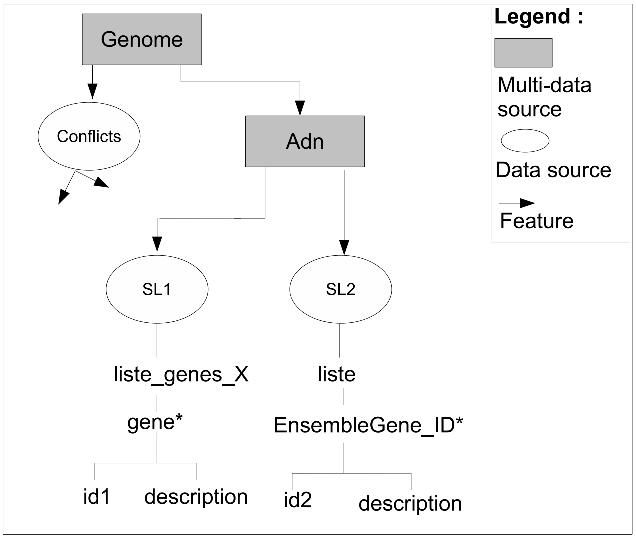


Fig. 11.7 Genome MDS.

11.4.2 Methodology for Semantic Reconciliation

Differently from traditional data integration approaches, like BGLAV, MFA does not require full semantic integration of data sources in order to execute queries on the multi-data source schema. Rather, mappings could be partial or approximated. Hence, MFA offers tools for semantic reconciliation between data sources, i.e., for identifying and solving conflicts between data sources. To this aim, ontologies can be used as part of the integration approach. Alternative approaches should be used whenever no ontology is available for the domain at hand. Examples of alternative methods are: similarity functions between data source elements; methods that infer mappings from answers of queries executed over data sources. The discovered semantic mappings are stored in the data source *Conflicts.xml*, as described in Section 11.4.1, and used later by the query rewriting module in order to answer queries. A user which is unsatisfied of the answers has the opportunity to add (or modify) mappings and submit once again its query. In doing so, the data source *Conflicts.xml* is gradually enriched and the quality of responses becomes increasingly accurate.

Figure 11.8 illustrates a methodology of reconciliation between data sources. The method relies on an ontology for defining concepts, properties, and relationships between these concepts. The usage of an ontology allows the user, at one side, to clearly specify the interest domain and, at the other side, improves the user's knowledge about the data source by clarifying the meaning of all its elements. For example, a user interested in a Biological domain must specify an ontology conformed to this domain. The user can then assign to each element of the data source schema the equivalent semantic element in the specified ontology. This step involves a

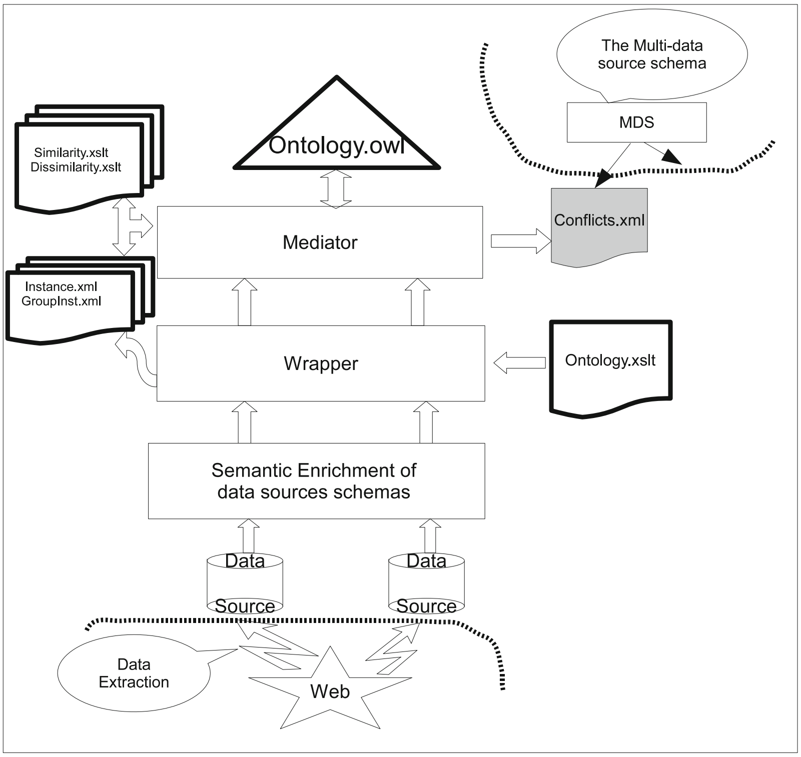


Fig. 11.8 Approach for semantic reconciliation between data sources.

semantic enrichment of the data source schema; such enrichment can be represented as an XML document conformed to the DTD presented in Figure 11.9, called *SemEnr.dtd*.

```
<!ELEMENT source (ontology+) >
<!ATTLIST source name CDATA #REQUIRED >
<!ATTLIST source url CDATA #REQUIRED >
<!ELEMENT ontology (feature) >
<!ATTLIST ontology name CDATA #REQUIRED >
<!ELEMENT feature (ontology?) >
<!ATTLIST feature name CDATA #REQUIRED >
```

Fig. 11.9 Semantic Enrichment DTD.

The next step is to represent data source instances according to the chosen ontology language and the semantically enriched data source schemas generated in the previous step. This conversion is performed by the *wrapper* component, using a defined template (e.g., Biological.xslt).

The final step consists in generating the *Conflicts.xml* data source. This activity is performed by the *mediator* component, which has two main tasks:

1. For each data type property, all the instances sharing that property are grouped together. Such step can be accomplished by querying directly the ontology. Assuming the ontology is represented as an OWL/RDF document, ontology queries can be considered at three different levels [17, 4]: syntactic level by using the XQuery language; structure level and semantic level by using an RDF query language such as RQL [20] and SPARQL [31, 17], respectively. A survey and more comparative analysis of different query languages has been published in [13]. The result is stored in an XML document named *GroupInst.xml*.
2. Generate information about similarities (dissimilarities) between data source instances by transforming the document *GroupInst.xml*, using an appropriate template (e.g., *Similarity.xslt* or *Dissimilarity.xslt*). The result of this step is stored in the document *Conflicts.xml*. In this document, a node with a tag *Similar* represents a semantic link between two elements of equivalence or synonym types. A node with a tag *Dissimilar* represents a semantic link between two elements of homonym or disjoint types. Each node, with a tag *Similar* or *Dissimilar*, is associated with an identifier $s_i(d_i)$.

11.4.3 Query Processing in MFA

In this section, we highlight MFA query processing. First we provide some necessary background on the type of the supported queries, then we illustrate the query rewriting steps.

11.4.3.1 Type of Queries in MRL

In MRL, a query is defined as follows:

Use (multi-)datasource$_1$ *name*$_1$ *[,(multi-)datasource*$_j$ *name*$_j$*]**
Allow $\$ < semantic - variables >$
(E)XQuery query
Close name$_1$ *[,name*$_j$*]**

Clauses *Use*, *(E)XQuery* and *Close* are mandatory whereas clause *Allow* is optional. Clause *Use* delimits the scope of the query and connects to (multi-)data sources for processing whilst *Close* disconnects from such data sources. $name_j$ is a given alias for either a data source or a multi-data source; clause *Allow* is used for the declaration of semantic variables. Through these variables, the user declares his/her intention to access data, in a given query, semantically similar and differently named. An (E)XQuery expression can be formulated as an XQuery query [1] or as an EXQuery query, as defined in [29]. In this last case, active data sources representing processing unit (e.g., web services) can also be invoked.

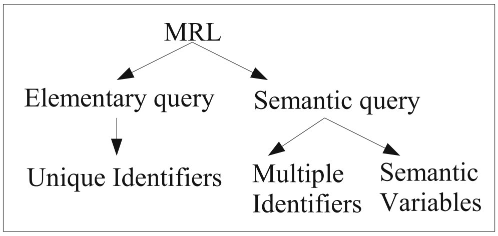


Fig. 11.10 Classification of MRL queries.

MRL queries can be classified depending on the type of paths (also called *designators*) appearing inside the (E)XQuery query component. Indeed, it is important to distinguish queries which access elements whose tag belongs to a single data source (elementary queries) from queries which access elements whose tag belongs to several data sources (*semantic queries*). The overall query classification is presented in Figure 11.10. Elementary and semantic queries can therefore be defined depending on the type of *identifiers* they contain. Elementary queries only contain *unique identifiers*, as defined below.

Definition 5 (Unique identifier). A unique identifier is a designator that univocally identifies an element in the scope of the query.

Semantic queries are used when various data sources represent the same universe in possibly different ways. Semantic queries are also called broadcast queries [24] because a user may have to broadcast the same query to several data sources. In its current form, XQuery does not easily capture such situations: indeed, with XQuery, the user needs to formulate as many queries as there are data sources. In contrast, semantic queries allow to broadcast the user intention in a single query. This is a major simplification, especially for a larger scope. Syntactically, semantic queries are formulated as elementary queries but rely on the usage of *multiple identifiers* and *semantic variables*.

Definition 6 (Multiple identifiers). A multiple identifier is a designator that identifies more than one element in the scope of the query.

Definition 7 (Semantic variables). A semantic variable is a variable whose domain is a set of elements that are semantically similar.

The aim of semantic variables is to enable the user to broadcast his/her intention over different elements which are related by similarity relationships inside the *Conflicts.xml* data source. A semantic query with semantic variables is considered as the set of pertinent elementary sub-queries resulting from all possible substitutions of semantic variables and multiple identifiers by unique identifiers.

Semantic variables can be declared inside the *Allow* clause of a MRL query according to the following syntax:

Allow $\$ < semantic - variable > = < designator > [, < designator >]^+$
$< semantic - variable > ::= < simple - variable > \mid < composed - variable >$
$< composed - variable > ::= < simple - variable > [. < simple - variable >]^+$
$< simple - variable > ::= < string >$
$< designator > ::= < string > [. < string >]^+$

We notice that, in MFA, unlike in BGLAV, we are faced with a single type of queries where the tree of the user's query is completely covered by the schemas of data sources. This is due to the fact that while in BGLAV an element E in the global schema is mapped into one or several elements (E_1, E_2, E_i) in the data sources, in MFA each element E in the global schema is mapped onto itself in the corresponding data source.

Example 3. Consider the MDS presented in Example 2. The query Q_1 below extracts the description associated with a gene identifier, posed over the *Genome* MDS.

Q_1:

```
Use Adn ad
Allow $a = id1.id2
For $x in document('mds')/Genome/ad
where $x/*/$a='ENSG000001018941' or $x/*/$a='ENSG00000146950' return
  <Result>
    $x/*/$a, $x/*/Description
  </Result>
```

In query Q_1, variable a is a semantic variable whose domain is $\{id_1, id_2\}$. Feature *Description* is a multiple identifier since it designates the *Description* feature in both data sources SL_1 and SL_2.

Example 4. Figure 11.11 describes a multi-data source named *DNA*, composed of two static data sources *Fragments* and *List_genes*. *Concat* and *Convert* are two active data sources that compose a multi-data source called *Services*. *Conflicts* describes the conflicts between *Fragments* and *List_genes* (not detailed in this figure). Services, Conflicts and DNA constitute a multi-data source called *Biology* which, in this case, is the root of the overall multi-data source. An MRL query is expressed directly over the *Biology* MDS; the answer is the union of answers returned by each component data source, namely S_1 and S_2. The only problem that arises here is how to solve conflicts between elements belonging to the user'query: Subsection 11.4.3.2 details the solution.

11.4.3.2 Query Rewriting in MFA

In this section, we detail step by step the algorithm for Rewriting Semantic Queries (RSQ) [28]. The overall process involves five steps: (i) query analysis; (ii) creation of the query tree; (iii) searching for semantic ambiguous elements; (iv) generation of sub-queries; (v) query execution. In the following, each step will be discussed in details.

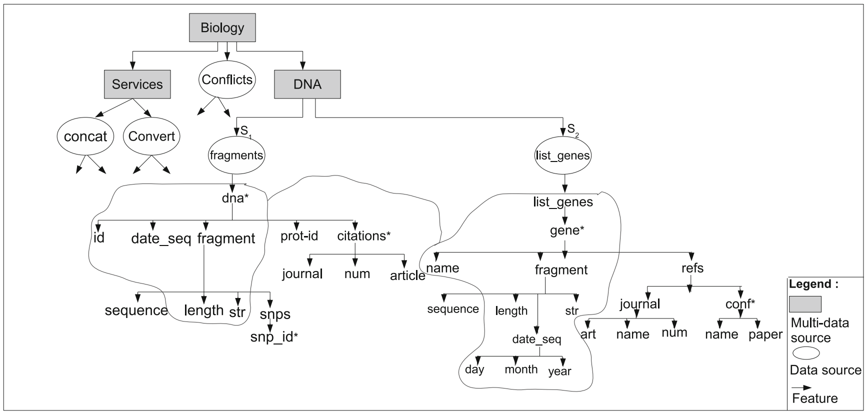


Fig. 11.11 Examples of queries in MFA.

Step 1: *Query analysis* (see Algorithm 1). This step consists in analyzing clauses *Use*, *Allow*, *For* (or *Let*) and *Return* (the last three appearing in the (E)XQuery query) of an input MRL query. It returns the following tables:

- table *MsoVarTab*, containing the names of data sources or multi-data sources with their related aliases;
- table *SemVarTab*, containing the semantic variables that have been specified in the Allow clause and their corresponding definition;
- *ConVarTab*, containing the different context variables specified in clauses For or Let and their respective values;
- *ResVarTab*, containing the (sub-)set of context variables specified in the clause Return.

The outcome of Step 1 for the query presented in Example 3 is illustrated in Figure 11.12.

Step 2: *Creation of the query tree (QTree)* (see Algorithm 2). Information gathered in Step 1 (*MsoVarTab*, *SemVarTab*, *ConVarTab*, *ResVarTab*) is used to build

Adn	ad

MsoVarTab

Semantic variables	Values
a	{id1, id2}

SemVarTab

$x	true

ResVarTab

Context variables	Values
$x	document("mds")/Genome/ad

ConVarTab

Fig. 11.12 Analysis of the query's clauses.

Algorithm 1. ***AnalyseQuery(Q_{MRL})***: Analysis of the user query.

```
Require: Input: $Q_{MRL}$ MRL query which is syntactically correct
    Output: MsoVarTab, SemVarTab, ConVarTab and ResVarTab:
    MsoVarTab contains the names of multi-data sources and their alias present in the clause Use
    MsoVarTab contains the semantic variables present in the clause Allow
    ConVarTab contains the variables declared in the body of the query
    ResVarTab contains the variables present in the clause Return
1: Initially, these variables are Empty.
2: MsoVarTab ⇐ AnalyseClauseUse($Use$)
3: SemVarTab ⇐ AnalyseClauseAllow($Allow$)
4: ConVarTab ⇐ AnalyseVarOfQuery($Body$)
5: ResVarTab ⇐ AnalyseVarOfQuery($Return$)
6: Return MsoVarTab, SemVarTab, ConVarTab and ResVarTab
```

the query tree, denoted by QTree. QTree describes the *context* of each element used in the query. The context of an element E is defined as the path that connects the root of the MDS to E. Each node in the QTree is labeled either with the path characterizing a given data source or with the path characterizing an element inside a data source. Paths are generated from those appearing in the query by replacing each variable with the corresponding values, according to the content of the input tables. Each leaf node in QTree is decorated with the two following information: (1) the set of conflict identifiers in *Conflicts.xml* where the element associated with the leaf node appears; (2) the name of the variables in clause Return (e.g., $a) in which the element associated with the leaf node appears. These information are used later in order to generate semantically coherent sub-queries. The result of this step is illustrated in Figure 11.13.

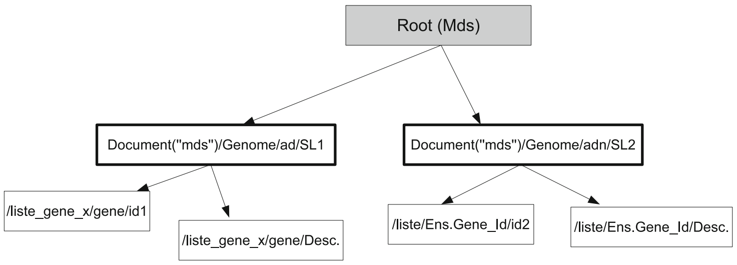


Fig. 11.13 *QTree* Creation.

Step 3: *Searching for Semantically Ambiguous Elements* (see Algorithm 3). For the sake of simplicity, we suppose that the query tree is represented through a table called QTab. QTab contains a column for each data source involved in the query and a row for each variable specified in the *Return* clause of the query. The table cell corresponding to a given data source S and a given variable x contains a set of

Algorithm 2. ***BuildQueryTree****(MsoVarTab, SemVarTab, ConVarTabandResVarTab)*: Construction of the tree of the query.

```
Require: Input: MsoVarTab, SemVarTab, ConVarTab, ResVarTab
    Output: QTree
 1: QTree ⇐ ∅ { Tree of the query}
 2: QTree ⇐ BuildTree(MsoVarTab, SemVarTab, ConVarTab, ResVarTab)
 3: QTree ⇐ EnrichedTree(QTree,Conflicts.xml)
 4: Return QTree
```

elements assigned to x in S. Each element can be associated with either a unique or multiple identifier in the query. The set is empty if no element in S is associated with x in the query.

Algorithm 3 checks each element in QTree as follows: if an element is a multiple identifier for a data source (i.e., it is associated with at least two contexts inside a data source), then this element is considered semantically ambiguous and consequently the QTree is ambiguous. In the example illustrated in Fig 11.14, QTree is not ambiguous and we move to Step 5.

Algorithm 3. ***Check(QTree)***: Search ambiguous elements in QTree.

```
Require: Input: QTree
    Output: Boolean
 1: Boolean ⇐ CheckQTree(QTree) { Checks if any element in QTree is semantically am-
    biguous}
 2: Return True or False
```

	SL1	SL2
$x	/liste_gene_x/gene/id1 {s1}	/liste/Ens.Gene_Id/id2 {s1}
$x	/liste_gene_x/gene/Desc. {s2}	/liste/Ens.Gene_Id/Desc. {s2}

Fig. 11.14 Checking semantic conflicts in QTab.

Step 4: *SubTrees Generation* (see Algorithm 4). This step is invoked only if QTree has been considered ambiguous in the previous step. Recall that QTree contains all semantic information about the user's query. This tree allows to check all semantic equivalences between elements and therefore all query conflicts. A *SubTree* is a semantic tree with the same structure of QTree but with the following restriction: each return variable and each data source in a SubTree is associated with elements (if any) defined within a single context (i.e., unique identifiers). Each SubTree leads to a set of pertinent sub-queries, which are semantically coherent.

When QTree is considered ambiguous, it means that one identifier is associated with an element with at least two contexts. in a data source. In this case at least two semantic trees (SubTrees) are generated. The overall idea is hat of separating each context in a sub-tree. Thus, each sub-tree contains only elements of the query which are semantically coherent.

From there, the generation of SubTrees requires the control of conflicts between elements of the QTree. Controlling conflicts between elements consists in checking, for each variable of a user's query (e.g., \$a, given in a row) and for each data source involved in the query (given in column), the elements that have similar identifiers (i.e., the elements designated with the same number s_i means that these elements are semantically similar). The result is stored in the corresponding SubTree. This task is repeated until all cases are processed and the set of SubTrees is generated.

Algorithm 4. ***GenerateSubTrees(QTree)***: Generate SubTrees which are semantically coherent.

```
Require: Input: QTree
    Output: at least two trees (SubTree) are generated
 1: SetOfSubTrees ⇐ ∅
    {while remains cases not treated:}
 2: while true do
 3:     SubTree ⇐ GenerateSubTree()
        {Generate empty SubTree having the same structure as QTree}
 4:     for all Variable (V) ∈ QTree do
 5:         SimilarElement ⇐ CheckConflicts(V,QTree)
            {for a variable V (e.g., $a) asked by the user in the clause RETURN of the query,
            check conflicts between elements through the set of data sources involved in this
            query, and returns similar elements}
 6:         SubTree ⇐ UpdateSubTree(V, SimilarElement)
            {Update the SubTree which is semantically consistent}
 7:     end for
 8:     SetOfSubTrees ⇐ SetOfSubTrees ∪ SubTree
 9: end while
10: return SetOfSubTrees
```

Step 5: *Generate pertinent sub-queries and Query Execution Plan* (see Algorithm 5). For each SubTree, a set of pertinent sub-queries is generated, which are semantically coherent with each others. Each generated sub-query is an (E)XQuery expression. This step generates ultimately the query plan for each local source.

Notice that, in a general settings, since the number of sub-queries can be very high, after this step the user has three choices:

1. to refine his/her query semantically;
2. to execute the sub-queries which require the maximal number of data sources (and/or a minimum number of missing elements in the sub-queries);

3. by default, to browse the whole set of possible responses. This last choice is not realistic since exploring this set is costly.

Algorithm 5. ***GenerateQEP(SetOfSubTrees)***: Generate sub-queries.

```
Require: Input: SetOfSubTrees
    Output: Query Execution Plan (QEP)
 1: for all SubTree ∈ SetOfSubTree do
 2:     subqueries ⇐ GenerateSubQueries(SubTree)
        {Generate semantically consistent subqueries}
 3:     Scheduling(P)
 4:     Save(P)
 5:     Return P
 6: end for
```

Example 5. Query Q_1 presented in Example 3, during Step 5, is decomposed into two sub-queries Q_{11} and Q_{12}, to be executed over data sources SL_1 and SL_2, respectively.

Q_{11}:

```
For $x in document('SL1')/liste_gene_X/gene
where $x/id1='ENSG000001018941' or $x/id1='ENSG00000146950'
return
  <Result>
    $x/id,$x/description
  </Result>
```

Q_{12}:

```
For $x in document('SL_2'})/liste/EnsemblGeneID
where $x/id2='ENSG000001018941' or $x/id2='ENSG00000146950'
return
  <Result>
    $x/id2, $x/description
  </Result>
```

11.5 Application

In this section, we consider an application in Genomics and we discuss query rewriting performed according to BGLAV and MFA approaches. In particular, we start by providing the description of the considered data sources, assuming that some restrictions exist concerning data access in each source. Then, for both BGLAV and MFA approaches, we first propose a global schema, i.e., a mediation schema for BGLAV and a multi-data source schema for MFA. We also provide information about conflict management, i.e., a set of correspondence queries for BGLAV and document *Conflicts.xml* for MFA. Finally, we present some queries and we show how they can be rewritten into queries over the data sources.

11.5.1 Data Source Description

Figure 11.15 illustrates three data sources that contain information about the DNA of the human X chromosome. Data are extracted from the well known `Ensembl` database,[2] then split into different files in order to simulate conflicts such as scale and name conflicts.

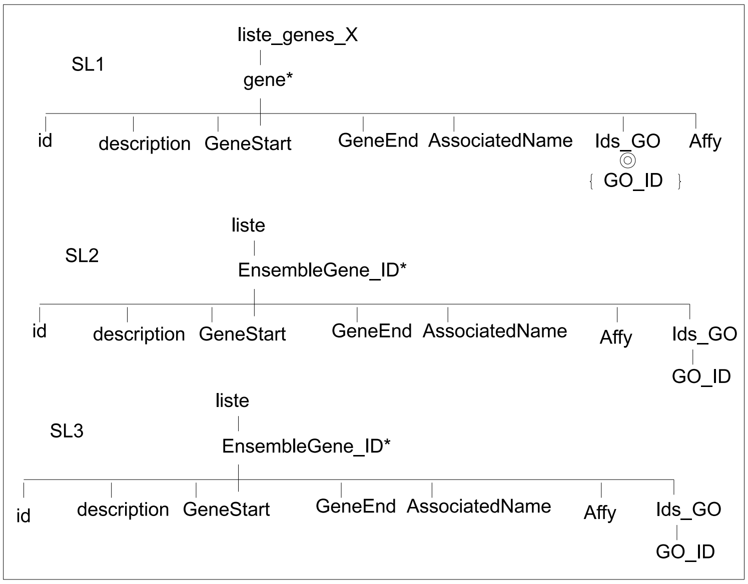


Fig. 11.15 Human chromosomes schemas.

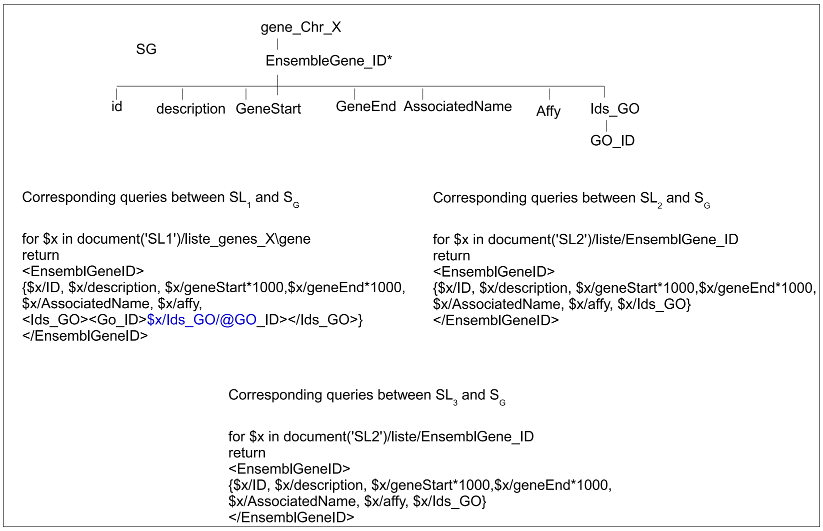
We suppose that there are access restrictions on data sources. In our example, we assume the following constraints:

1. for SL_1, the value of the feature [/liste_Genes_X/Gene/ID] must be specified in order to access its data;
2. for SL_2, the value of the feature [/liste/EnsemblGene_ID/ID] must be specified in order to access its data;
3. for SL_3 the value of the feature [/liste/EnsemblGeneID/GeneStart] must be specified in order to access its data.

11.5.2 BGLAV Illustrating Examples

Figure 11.16 shows the global schema and some correspondence queries between the schemas of data sources SL_1, SL_2 and SL_3 and this global schema.

[2] `www.ensembl.org`



Corresponding queries between SL_1 and S_G

```
for $x in document('SL1')/liste_genes_X\gene
return
<EnsemblGeneID>
{$x/ID, $x/description, $x/geneStart*1000,$x/geneEnd*1000,
$x/AssociatedName, $x/affy,
<Ids_GO><Go_ID>$x/Ids_GO/@GO_ID></Ids_GO>}
</EnsemblGeneID>
```

Corresponding queries between SL_2 and S_G

```
for $x in document('SL2')/liste/EnsemblGene_ID
return
<EnsemblGeneID>
{$x/ID, $x/description, $x/geneStart*1000,$x/geneEnd*1000,
$x/AssociatedName, $x/affy, $x/Ids_GO}
</EnsemblGeneID>
```

Corresponding queries between SL_3 and S_G

```
for $x in document('SL2')/liste/EnsemblGene_ID
return
<EnsemblGeneID>
{$x/ID, $x/description, $x/geneStart*1000,$x/geneEnd*1000,
$x/AssociatedName, $x/affy, $x/Ids_GO}
</EnsemblGeneID>
```

Fig. 11.16 Correspondence queries in BGLAV.

First, let us consider query (Q_2) that extracts values associated with the identifier of a given gene. This query illustrates the concept of query introduced in Definition 3.

Q_2:

```
For $x in document('S_G')/Gene_Chr_X/EnsemblGeneID
where $x/ID='ENSG000001018941' or $x/ID='ENSG00000146950'
return
  <Result>
    $x/ID,$x/Description,$x/GeneStart,$x/GeneEnd}
  </Result>
```

Query Q_2 is decomposed into two sub-queries Q_{21} and Q_{22} targeting respectively SL_1 and SL_2 and presented below. Data source SL_3 is not involved in this query since the restriction access to this source is not satisfied.

Q_{21}:

```
For $x in document('SL_1')/liste_genes_X/gene
where $x/ID='ENSG000001018941' or $x/ID='ENSG00000146950'
return
  <EnsemblGeneID>
    $x/ID, $x/Description, $x/GeneStart*1000, $x/GeneEnd*1000,
    $x/AssociatedName, $x/Aff
    <Ids_GO>
      <GO_ID> $x/Ids_GO/@GO_ID$ </GO_ID>
    </Ids_GO>
  </EnsemblGeneID>
```

Q_{22}:

```
For $x in document('SL_2')/liste/EnsemblGeneID
where $x/ID='ENSG000001018941' or $x/ID='ENSG00000146950'
return
  <EnsemblGeneID>
    $x/ID, $x/Description, $x/GeneStart*1000, $x/GeneEnd*1000,
    $x/AssociatedName, $x/Aff, $x/Ids_GO
  </EnsemblGeneID>
```

Now, lets us consider query Q_3 below which involves data source SL_3:

Q_3:

```
For $x in document('$S_{G}$')/Gene_Chr_X/EnsemblGeneID
where $x/GeneStart > '7770303' or $x/GeneEnd < 9092647
return
  <Result>
    $x/ID, $x/Description, $x/GeneStart, $x/GeneEnd,
    $x/AssociatedName, $x/Affy, $x/Ids_GO
  </Result>
```

Query Q_3 is translated into sub-query Q_{31} posed over data source SL_3 and whose expression is as follows: Notice that, due to access restrictions on SL_1 and SL_2, no sub-queries are generated.

Q_{31}:

```
For $x in document('SL_3')/liste/EnsemblGene_ID
where $x/GeneStart > 7770303 or $x/GeneEnd < 9092647
return
  <EnsemblGeneID>
    $x/ID, $x/Description, $x/GeneStart*1000, $x/GeneEnd*1000,
    $x/AssociatedName, $x/Affy, $x/Ids_GO
  </EnsemblGeneID>
```

11.5.3 MFA Illustrating Examples

Figure 11.17 shows the Genome MDS composed of the three data sources SL_1, SL_2 and SL_3 presented in Figure 11.15 while Figure 11.18 shows part of the conflicts. Notice that the node <SCALE> describes the elements which are semantically similar but in addition they also represent scale conflicts between them. For example, the two elements *GeneStart* in the two data sources *SL1* and *SL2* are similar and they represent a scale conflict of type *Measure*.

Consider now query (Q_4) that extracts the values associated with a gene identifier.

Q_4:

```
Use Adn ad
For $x in document('mds')/Genome/ad
where $x/*/ID='ENSG000001018941' or $x/*/ID='ENSG00000146950'
return
  <Result>
    $x/*/ID, $x/*/Description, $x/*/GeneStart, $x/*/GeneEnd
  </Result>
```

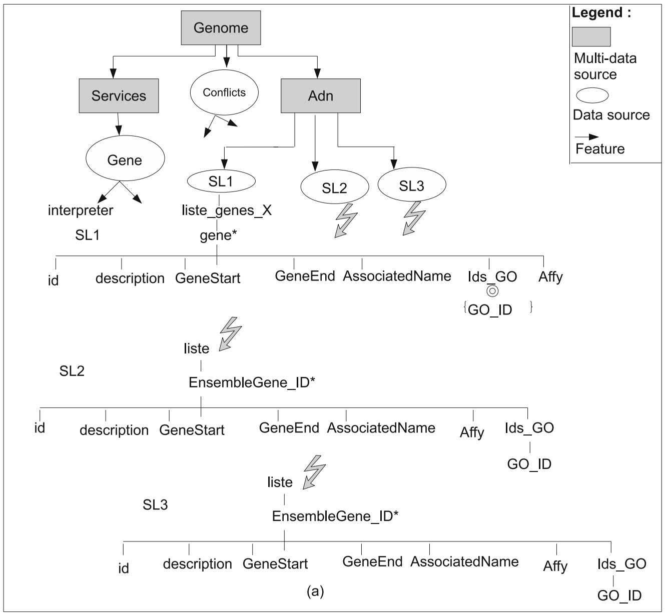


Fig. 11.17 The Genome MDS.

In query Q_4, *id* is a multiple identifier since it designates the feature *id* in the three data sources SL_1, SL_2 and SL_3. The same remark applies to features *Description* and *GeneEnd*. In addition, *id*, *Description*, and *GeneEnd* are similar both in the set $\{SL_1, SL_2, SL_3\}$ and the Conflicts data source (see Figure 11.18). Element *GeneStart* is a multiple identifier in the scope of the query and it represents a scale conflict between the two data sources SL_1 and SL_2.

Query Q_4 is decomposed into two sub-queries Q_{41} and Q_{42} targeting respectively SL_1 and SL_2. Data source SL_3 is not involved in this query since the restriction access to this source is not satisfied. Sub-query Q_{41} is an EXQuery expression that involves a static data source and an active one (i.e., the *Gene* service), while the second sub-query is an XQuery expression.

Q_{41}:

```
For $x in document('SL1')/liste_gene_X/gene
For $y in  document('Gene')/service
where $x/ID='ENSG000001018941' or $x/ID='ENSG00000146950'
return
  <Result>
    $x/ID, $x/Description, $x/GeneEnd, $x/AssociatedName, $x/Aff,
    service($y/interpreter,$x/GeneStrart), $x/Ids_GO/@GO_ID
  </Result>
```

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE CONFLICTS SYSTEM "conflicts.dtd">
<CONFLICTS>
  ...
<SIMILAR  id='s4'>
  <Node>
    <PATH>Bio/Adn/SL1/liste_genes_X/gene</PATH>
    <ELT>GeneStart</ELT>
  </Node>
  <Node>
    <PATH>Bio/Adn/SL2/liste/EnsembleGene_ID</PATH>
      <ELT>GeneStart</ELT>
  </Node>
  <Node>
    <PATH>Bio/Adn/SL3/liste/EnsembleGene_ID</PATH>
    <ELT>GeneStart</ELT>
  </Node>
</SIMILAR>
<SCALE type='Measure'>
  <Node>
    <PATH>Bio/Adn/SL1/liste_genes_X/gene</PATH>
    <ELT>GeneStart</ELT>
  </Node>
  <Node>
    <PATH>Bio/Adn/SL2/liste/EnsembleGene_ID</PATH>
    <ELT>GeneStart</ELT>
  </Node>
</SCALE>
  ...
</CONFLICTS>
```

Fig. 11.18 Excerpt of conflicts in the Genome MDS.

Q_{42}:

```
For $x in document('SL_2')/liste/EnsemblGeneID
where $x/ID='ENSG000001018941' or $x/ID='ENSG00000146950'
return
  <Result>
    $x/ID, $x/Description, $x/GeneStart, $x/GeneEnd,
    $x/AssociatedName, $x/Aff, $x/Ids_GO
  </Result>
```

Finally, let us consider query Q_5 below:

Q_5:

```
Use Adn ad
For $x in document('Multi-Data Source')/Genome/ad
where $x/*/GeneStart > '7770303' or $x/*/GeneEnd < '9092647'
return
  <Result>
    $x/*/ID, $x/*/Description, $x/*/GeneStart, $x/*/GeneEnd,
    $x/*/AssociatedName, $x/*/Affy, $x/*/Ids_GO
  </Result>
```

Query Q_5 is translated into sub-query Q_{51} below and targets data source SL_3:
Q_{51}:

```
For $x in document('$SL_{3}$')/liste/EnsemblGene_ID
where $x/GeneStart > '7770303' or $x/GeneEnd < '9092647'
return
  <Result>
    $x/ID, $x/Description, $x/GeneStart*1000, $x/GeneEnd*1000,
    $x/AssociatedName, $x/Affy, $x/Ids_GO
  </Result>
```

11.5.4 Evaluation of MFA Queries

In this section, we study the performance of RSQ algorithms. In particular, we focus on Step 2 of the algorithm, dealing with the generation of the query tree. Recall that this step substitutes each context variable (e.g., $a) in the input tables with their corresponding values and validates the correctness of their paths on the multi-data source schema. The processing of this step is compared with a baseline version that uses a Cartesian Product (Cart. Prod.) between the set of values taken by the semantic variables in order to find the valid paths on the multi-data source schema.

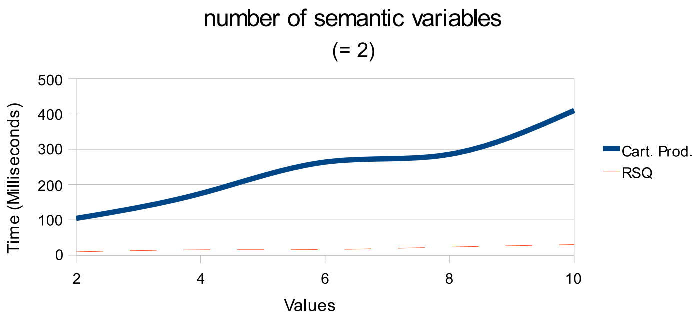


Fig. 11.19 Two semantic variables; each variable takes values varying from 2 to 10.

For each experiment, we considered 20 data sources and we assumed the values taken by each semantic variable vary from 2 to 10. Figure 11.19 shows the case of a query that uses in its body two semantic variables. In this experiment, the response time increases faster in the baseline method, based on 'Cartesian Product', with the increase of the number of values taken by the semantic variables. Figure 11.20 shows the case of a query that uses in its body five semantic variables. In this experiment, the response time in the baseline method increases dramatically, compared to RSQ algorithm, with the increase of the number of values taken by the semantic variables. From the reported experiments, we also observe that the response time is more sensitive to variations concerning the number of semantic variables in a query than to the number of values taken by these variables.

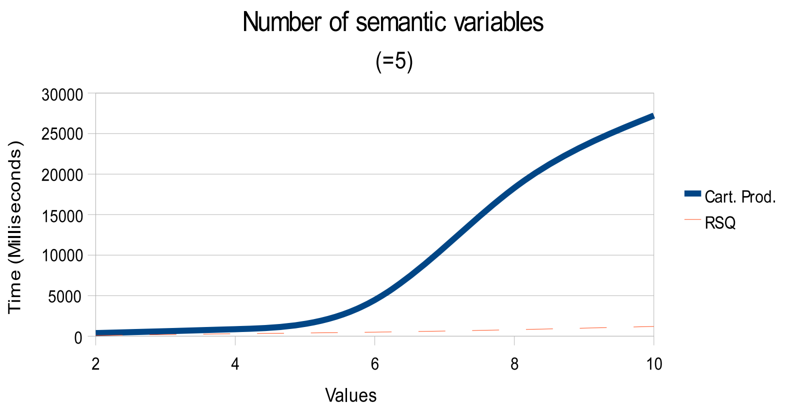


Fig. 11.20 Five semantic variables; each variable takes values varying from 2 to 10.

11.6 Conclusion and Open Issues

In this chapter, we described how to query distributed conflicting web data sources, by means of two data integration approaches. For illustration purposes, we used concrete data drawn from the Genomics domain in a real experimental settings. In order to tackle the data integration problem, we described two approaches: the first one is an XML adaptation of a the well know BGLAV, which pertain to the first-generation of data integration approaches. The second approach, MFA (for multi-source fusion approach) which does not rely on a preexisting mediation schema but rather on a multi-data source schema composed of various data sources, allows flexibility and bootstrapping. Although not being directly inspired by the *dataspace management systems* concepts, MFA relates to this second-generation of data integration approaches.

Because large scale data integration is still a challenge, for future work, we are planning to extend MFA by leveraging existing automated techniques such as schema matching and reference reconciliation: this will help in providing initial correspondences between data sources, hence auto-bootstrapping the system. Feedback from a (more or less) skilled user could be solicited in order to accommodate additional information and build an efficient pay-as-you-go integration system.

References

1. XQuery 1.0: An XML Query Language. `http://www.w3.org/TR/xquery/`
2. ASN.1: Abstract Syntax Notation One, `http://asn1.elibel.tm.fr/en/`
3. Benson, D., Boguski, M., Lipman, D., Ostell, J., GenBank., J.: Nucleic Acids Res. 1–6 (1997)

4. Bönström, V., Hinze, A., Schweppe, H.: Storing RDF as a Graph. In: Proc. of the First Conference on Latin American Web Congress. IEEE Computer Society (2003)
5. Brien, M., Poulovassilis, A.: Data Integration by Bi-Directional Schema Transformation Rules. In: ICDE, pp. 227–238 (2003)
6. Castano, S., Ferrara, A., Montanelli, S.: H-Match: An Algorithm for Dynamically Matching Ontologies in Peer-based Systems. In: Proc. of the 1st Int. Workshop on Semantic Web and Databases (SWDB) VLDB 2003, pp. 231–250 (2003)
7. Colonna, F.M.: Intégration de Données Hétérogènes et Distribuées sur le Web et Applications à la Biologie. Ph.D. thesis. University Paul Cézanne, Aix-Marseille 3 (2008)
8. Colonna, F.M., Sam, Y., Boucelma, O.: Database Integration for Predisposition Genes Discovery. In: Challenges and Opportunities of Healthgrids, Proc. of 4th HealthGrid Annual Conference. Studies in Health Technology and Informatics, vol. 120. IOS Press (2006)
9. Dong, X.L., Berti-Equille, L., Srivastava, D.: Integrating Conflicting Data: The Role of Source Dependence. In: Proceedings of VLDB 2009, pp. 562–573 (2009)
10. Franklin, M.J., Halevy, A.Y., Maier, D.: From Databases to Dataspaces: a New Abstraction for Information Management. SIGMOD Record 34(4), 27–33 (2005)
11. Friedman, M., Levy, A., Millstein, T.: Navigational Plans for Data Integration. In: Proc. of the National Conference on Artificial Intelligence (1999)
12. Garcia-Molina, H., Papakonstantinou, Y., Quass, D., Rajaraman, A., Sagiv, Y., Ullman, J., Vassalos, V., Widom, J.: The TSIMMIS Approach to Mediation: Data Models and Languages. Journal of Intelligent Information Systems 8, 17–132 (1997)
13. Haase, P., Broekstra, J., Eberhart, A., Volz, R.: A Comparison of RDF Query Languages. In: McIlraith, S.A., Plexousakis, D., van Harmelen, F. (eds.) ISWC 2004. LNCS, vol. 3298, pp. 502–517. Springer, Heidelberg (2004)
14. Halevy, A.: Answering Queries using Views: A Survey. Journal of the VLDB, 270–294 (2001)
15. Halevy, A., Franklin, M., Maier, D.: Principles of Dataspace Systems. In: Proc. of PODS, pp. 1–9. ACM Press (2006)
16. Halevy, A., Rajaraman, A., Ordille, J.: Data Integration: The Teenage Years. In: Proceedings of VLDB (2006)
17. Hertel, A., Broekstra, J., Stuckenschmidt, H.: RDF Storage and Retrieval System. In: Staab, S., Studer, R. (eds.) Handbook on Ontologies, pp. 489–508. Springer, Heidelberg (2009)
18. International, R.: The GDB Human Genome Database (2006), `http://www.gdb.org`
19. Jeffery, S., Franklin, M., Halevy, A.: Pay-as-you-go User Feedback for Dataspace Systems. In: Proc. of ACM SIGMOD, pp. 847–859. ACM Press (2008)
20. Karvounarakis, G., Alexaki, S., Christophides, V., Plexousakis, D., Scholl, M.: RQL: A Declarative Query Language for RDF. In: Proc. of the 11th International Conference on World Wide Web, pp. 592–603 (2002)
21. Keen, G., Burton, J., Crowley, G., Dickinson, E., Espinosa-Lujan, A., Franks, E., Harger, C., Manning, M., March, S., McLeod, M., O'Neill, J., Power, A., Pumilia, M., Reinert, R., Rider, D., Rohrlich, J., Schwertfeger, J., Smyth, L., Thayer, N., Troup, C., Fields, C.: The Genome Sequence DataBase (GSDB): Meeting the Challenge of Genomic Sequencing. Nucleic Acids Res. 24, 13–16 (1996)
22. Lenzerini, M.: Data Integration: A Theoretical Perspective. In: PODS, pp. 236–246 (2002)
23. Levy, A., Rajaraman, A., Ordille, J.: Query-Answering Algorithms for Information Agents. In: Proc. of the 13th National Conference on Artificial Intelligence (IAAI 1996), AAAI Press, MIT Press, pp. 40–47 (1996)

24. Lyngbaek, P., McLeod, D.: An Approach to Object Sharing in Distributed Database Systems. In: Proc. of the VLDB, pp. 364–375 (1983)
25. Mootha, V., Lepage, P., Miller, K., Bunkenborg, J., Reich, M., Hjerrild, M., Delmonte, T., Villeneuve, A., Sladek, R., Xu, F., Mitchell, G.A., Morin, C., Mann, M., Hudson, T., Robinson, B., Rioux, J., Lande, E.S.: Identification of a Gene Causing Human Cytochrome Oxidase Deficiency by Integrative Genomics. Proc. of the National Academy of Sciences, 605–610 (2003)
26. Nachouki, G., Quafafou, M.: Multi-Data Source Fusion. Information Fusion 9(4), 523–537 (2008)
27. Nachouki, G., Quafafou, M.: MashUp Web Data Sources and Services based on Semantic Queries. Special Issue: Semantic Integration of Data, Multimedia and Services 36(2), 151–173 (2011); ISSN 0306-4379
28. Nachouki, G., Quafafou, M.: Using Semantic equivalence for MRL Queries Rewriting in Multi-Data Source Fusion System. In: Jin, H. (ed.) Data Management in Semantic Web, pp. 345–382. Nova Science Publishers (2011)
29. Nachouki, G., Quafafou, M., Chastang, M.: A System Based on Multidatasource Approach for Data Integration. In: IEEE-International Conference on Web Intelligence (WI), pp. 438–441 (2005)
30. NCBI: Fasta format. (2006), `http://www.ncbi.nlm.nih.gov/blast/fasta.shtml`
31. Prud'hommeaux, E., Seaborne, A.: SPARQL Query Language for RDF, W3C Recommendation. (2008), `http://www.w3.org/TR/rdf-sparql-query/`
32. Rahm, E., Bernstein, P.: A Survey of Approaches to Automatic Schema Matching. Journal of the VLDB 10(4), 334–350 (2001)
33. Sarma, A.D., Dong, X., Halevy, A.: Bootstrapping Pay-As-You-Go Data Integration Systems. In: Proc. of ACM SIGMOD, pp. 663–674. ACM Press (2008)
34. Schulze-Kremer, S.: Ontologies for Molecular Biology. In: Proc. of the 3rd Pacific Symposium on Biocomputing, pp. 705–716 (1998)
35. Sheth, A., Larson, J.: Federated Database Systems for Managing Distributed, Heterogeneous, and Autonomous Databases. ACM Computing Surveys (CSUR), 183–236 (1990)
36. Xu, L., Embley, D.W.: Combining the Best of Global-as-View and Local-as-View for Data Integration. In: ISTA, pp. 123–136 (2004)