A Data Model-Independent Approach to Big Research Data Integration

Abstract: The paper discusses the data integration problem in the context of the scientific domain. The main characteristics of the big research data, that make the traditional approach of data integration unfeasible are presented. Two new emerging practices, i.e., an exploratory approach to data seeking and an empiricist epistemological approach to knowledge creation, that also contribute to making this approach not suited for this domain are discussed. Based on these considerations a new paradigm of data integration is proposed and discussed. An application ontology, i.e., the BDI ontology that supports this new paradigm is presented. The ontology is based on five types of events, concerning the creation of new databases or views, the update of the schema of a database or of the query defining a view, or the update of the data content of a database. Every event is extensionally modeled as an input/output operation on the involved data entity. The strong point of the ontology and of the whole approach to data integration, is that no assumption is made on the data models in which the databases or the views are expressed. This provides a level of generality that successfully deals with the heterogeneity of the domain, making our approach applicable in principle to every data integration context. Different approaches for implementing data integration based on the proposed ontology are also discussed, and an approach guaranteeing consistency while preserving efficiency is proposed. Some implementation issues are discussed and, as a proof of feasibility, the main algorithms for realizing the approach are also given in the Appendix.

Keywords: Data integration; Big Research Data; Ontology; Semantic Web

1 Introduction

Data Integration has the goal of enriching and completing the information available to the users by adding complementary information residing at diverse information sources. It aims at providing a more comprehensive information basis in order to better satisfy user information needs. This is achieved by combining data residing at diverse data sets and creating a unified view of these datasets. This view provides a single access point to these distributed, heterogeneous and autonomous data sets. Therefore, it frees the user from the necessity of interacting separately with each of these data sets.

We distinguish two types of data integration. The first type of data integration, *structural* data integration, refers to the ability to accommodate in a common data representation model distributed data sets represented in different data representation models and formats. In essence, in this type of integration the goal is to augment the dimensionality of an entity/object represented in different distributed data sets by collecting together all the attributes/features associated with this entity/object. The second type of data integration, *semantic* data integration, refers to the ability to combine distributed data sets on the basis of existing semantic relationships between them. In essence, in this type of integration the goal is to augment the relationality of an entity/object represented in a data set by linking it to entities/objects semantically closely related to it and represented in other distributed data sets.

Data integration very much depends on the characteristics of the data to be integrated as well as on the characteristics of the application context within which data integration is performed. In this paper, we study the process of integrating research data produced by researchers during their research activities, where data integration is instrumental in exploratory studies carried out by research teams.

It is argued that in such a context, the traditional approach to data integration is not only unfeasible from a technical point of view, but also not well suited to support the discovery process carried out by the researchers. A new approach to data integration will then be proposed, that takes into consideration the characteristics of the research data as well as some new emerging practices in the scientific domain in the era of big data.

The paper is organized as follows: In Section 2, an overview of the characteristics of the Scientific domain as well as those of the Big Research Data that heavily influence the data integration process is presented. In Section 2.1, a new paradigm of integrating Big Research Data is illustrated. In Section 3, a generic scientific application scenario for the purpose of better illustrating the new data integration paradigm is presented; some concepts underlying such scenario, that are relevant for the new paradigm, are introduced. In Section 4, an application ontology is developed to support the new paradigm in a data-model independent approach; some implementation considerations are also proposed. Section 5 offers some concluding remarks that summarize

the new paradigm. Finally, in the Appendix some fundamental algorithms are presented.

2 Characteristics of the Scientific Domain

Data integration in the scientific domain Council et al. (2010) is heavily influenced by the characteristics of this domain as well as by the characteristics of the big research data. These characteristics include:

- The large number of data sets to be integrated
- The huge volumes of data contained in these data sets
- The widely differing data qualities of the data contained in these data sets
- The extreme heterogeneity of the local data schemata, manifested by:
  - The different ways of conceptualizing the same scientific problem by different schools of thought
  - The different formats adopted by different research communities for the same type of data
  - The different ways of accounting for the uncertainty in the data followed by different research communities.
- The dynamic creation of local data schemata
- The lack of a priori knowledge of the local schemata
- The evolution of local data schemata over time; in fact, these schemata evolve as new insights are gained in a scientific domain; for example, certain concepts can be invalidated in the light of new discoveries
- The different concepts of what to include in the metadata.

Characteristics of the Big Research Data that influence the data integration process include:

- an ever-increasing production of new data types that augments the complexity of data sets;
- a worldwide distribution of research data sets;
- data sets with high dynamism, uncertainty, exhaustivity, and relationality.

In such an application environment, the traditional approach to data integration, based on the design of a unified view (global schema) is technically and economically unfeasible Ziegler and Dittrich (2004), Levy (1998). First, due to the extreme heterogeneity of the data sets to be integrated the design of a global schema is a very complex task. Second, given the fact that the local schemata are created dynamically and evolve over time the global schema and the mappings from the global to the local schemata should undergo continuous restructuring.

2.1 Big Research Data Integration: A New Paradigm

In the previous Section we have listed a number of data characteristics that make the traditional approach to data integration unfeasible in the scientific domain. In this Section we describe two emerging practices that make the concept of data integration not suited for this domain.

First, in many disciplines the amount of data contained in data repositories outgrows the capabilities of query processing technology. In this case, a new paradigm of data seeking has been proposed: "data exploration" Idreos (2013). Exploration-based systems guide users towards the path that their queries and the result lead to, in an incremental and adaptable way. Data exploration, therefore, is a new approach in data seeking that allows discovering connections and correlations between data.

Second, a new empiricist epistemological method for creating new knowledge is emerging Kitchin (2014). In the traditional scientific method, *i.e.*, *hypothesis driven* research, the data are analyzed with a specific question in mind, that is, a hypothesis. In essence, this scientific method adopts a deductive reasoning for discovering new insights from the data. In the empiricist method, *i.e.*, *data driven* research, the data are analyzed with no specific question in mind. In essence, huge volumes of data together with powerful analytic tools enable data to speak for themselves. Mining big data can reveal relationships and correlations that researchers did not even know to look for. In this method an inductive or abductive reasoning is adopted for discovering new insights from the data.

In order to support data exploration and data driven research a new paradigm of data integration is needed. The exploratory approach to data seeking suggests the possibility for researchers to start browsing in one data set and then navigating along links into related data sets, or to support data search engines to crawl a linked data space Bizer (2013) by following links between data sets.

The empiricist method entails the logic that must guide the creation of these links. In fact, in the hypothesis driven method a link between two data sets is established only when the current hypothesis dictates a semantic relationship between variables within these data sets. In contrast, in the data driven method a link between two data sets is established only when a significant relationship between variables within these data sets is found. The logic underlying this method enables researchers to discover new insights by analyzing data linked together on the basis of a inductive/abductive reasoning.

These two practices pave the way for the creation of linked data spaces. Such linked data spaces can be implemented by exploiting linking technologies that allow to connect/link semantically related data sets. For example, data sets produced worldwide and related to the same phenomenon could be linked together creating, thus, linked data spaces in the form of thematic graphs. Researchers, interested in discovering correlations and semantic relationships between data sets contained in linked data spaces, should go through these thematic graphs. In essence, now the researchers have to explore a linked research data space by navigating through it. Based on all the above considerations the data integration problem can be re-formulated as follows:

Given a number of distributed heterogeneous and time varying data sets, link them on the basis of existing semantic and temporal relationships among them.

In essence, the idea of the traditional concept of data integration, that is, to combine distributed heterogeneous data sets by defining on top of them a common view (global schema) that accommodates structural and semantic heterogeneities among them is replaced by the concept of connecting distributed heterogeneous data sets on the basis of existing semantic and temporal relationships among them.

Consequently, even the role of a data integration system is changing . In the traditional approach, a data integration system has to map a user query issued against the global schema into a number of queries issued against the local schemata. The role of a data integration system, in the new paradigm, is to make explicit hidden semantic relationships or correlations between data sets. Making explicit hidden semantic relationships/correlations implies the creation of links between data sets.

3 Concepts underlying a generic Scientific Data Integration Scenario

This section presents a conceptualization of a generic scientific application scenario that emerges from the new paradigm described in the previous Section. The conceptualization views data integration in a data-model independent way, and, most importantly, reduces schema-level operations to data-level operations on the underlying concepts. The conceptualization is formally specified in the *BDI* ontology, introduced in the next Section, along with the operations that reflect the evolution of the scientific data domain at hand.

Databases. In our hypothetical application environment, there are n databases distributed worldwide containing heterogeneous data represented in different formats and managed by different data management systems. These data are produced by different research teams, each following its own practices and protocols. In spite of this heterogeneity, these databases have a common trait that allows treating them in a uniform way for our purposes. In particular, each database has an intension and an extension:

- The *intension* of a database describes the structure of the database and is expressed as a schema of the particular data model that the database conforms to. For instance, the intension of a database conforming to the relational data model is a relational schema that defines the tables of the database and the structure of each table in terms of columns (attributes) and their domains; possibly, the schema also gives constraints such as keys, referential constraints and functional dependencies.

- The *extension* of a database is a set of data that are structured according to the database intension. For instance, the extension of a relational database is a set of relational tables, each conforming to the definition and constraints given in the schema of the database.

The scientific application environment is a changing environment due to the dynamic nature of the research activity and of the research data. Therefore, both the intension and the extension of a database are time-dependent. In particular,

- The intension of a database is defined at the database creation time, and may later be revised any number of times to reflect changes in the structure of real-world objects or in the user requirements, for instance as new insights are gained in the scientific domain.

- Changes in the extension of a database are the result of database operations such as the acquisition of more research data (through an INSERT command), or modification of existing data (through an UPDATE command) or the elimination of some data (through a DELETE command).

These changes entail a support for DB schema and DB extension versioning. According to Jensen et al. (1998) a database system supports schema versioning if it allows the accessing of the schema extension, both retrospectively and prospectively, through user definable version interfaces. Support for versioning requires that a history of changes be maintained to enable the retention of past DB intensions and extensions.

In order to support versioning, we consider events in our scenario. Events are generally defined as "changes of states in cultural, social or physical systems, regardless of scale, brought about by a series or group of coherent physical, cultural, technological or legal phenomena" Doerr (2003). For the present purposes, we consider three kinds of events related to databases:

- DB Creation: produces a database with an associated schema and an empty extension

- Schema Change: produces a new DB schema from an existing one
- Data Change: produces a new DB extension from an existing one.

We further assume that these events occur instantly. That is, when the schema of a database changes it is instantly replaced by the changed one, so that at each instant of the database lifetime exactly one schema is associated to the database. Likewise, we assume that any operation altering the extension of a database occurs instantly as a replacement of the current database extension by a new extension reflecting the effects of the operation. Technically, this assumption is enforced by any database management system, for instance by locking the database during any of the above operations, so that no other operation interferes with the current one. The net effect of this behavior is precisely the assumption that we have made.

Database Views. As already said, research databases contain huge amounts of data. Usually, researchers are interested only in some parts of a database. These parts of a database (called sub datasets) can be formally defined as database views. Database views can be considered as epistemological data abstractions Floridi and Sanders (2004). The epistemological approach to abstraction is concerned with the different levels of observation or interpretation at which a database can be studied. For example, a database can be observed and analyzed at different levels of abstraction, consisting of data related by time, place, instrument, or object of observation. Examples of epistemological levels of abstraction are spatial and temporal data abstractions. A database view can also be defined as a function Buneman et al. (2016) that when applied to a database produces a data subset of that database. We think that each large database should be endowed with a number of (possibly overlapping) views.

In our scenario there are also several database views, hereafter simply views, each defined on top of a database[1]. The heterogeneity of databases implies that of views. But also views have a common trait that allows treating them uniformly. In particular, each view has an intension and an extension:

- The intension of a View describes the semantics of the view and is expressed by a query in the query language of the database on which the view is defined. For example, the intension of a view defined over an RDF database will typically be a SPARQL SELECT/CONSTRUCT query. We note that the intension of a view embodies the schema of the view, by giving the structure of the query result.
- The extension of a view is the data subset defined by the intension of the view, that is, the result of running the view query against the (extension of the) database on which the view is defined.

Typically, a view has a set of metadata that describe the view from an application- or a system-specific point of view. These metadata are not further considered in our scenario because they are out of the present scope.

As a consequence of the fact that both the intension and the extension of a database evolve over time, also the intension and the extension of a view evolve over time. In particular,

- a change in the intension (*i.e.*, schema) of a database may cause a change in the intension of any view defined on that database, this intension being a query expressed against the database schema. For instance, the renaming of an attribute in a relational table requires that any reference to that attribute on a views intension (query) be renamed as well. Similarly for the suppression or the addition of attributes.
- a change in the extension of a database may induce a change in the extension of any view defined over that database, this extension being the result of a query evaluated against the database extension. For instance, the insertion of a row to a relational table will cause the addition of that row to the extension of any view whose query is satisfied by the inserted row.

In order to support versioning of views, we will follow a similar approach to that followed for databases, by introducing two more kinds of events in our scenario:

- View Creation: produces a view and associated intension and extension.
- View Intension Change: produces a new view intension from an existing one, and a corresponding new view extension from an existing one.

Note that we do not introduce a new type of event for the evolution of view extensions, because that evolution can only be implicitly caused either by data change events or by view intension change events¡ thus, no specific action is required by the user. As a consequence, it is expected that the evolution of view extensions be treated in a completely automatic way, transparently to the user. We only revise the definition of a Data Change event to include view extension evolution, as follows:

- Data Change: produces a new DB extension from an existing one, and a new view extension from an existing one.

4 The *BDI* Ontology

In order to formally represent the notions of the conceptualization introduced in the previous Section, we now present the *BDI* ontology. The *BDI* ontology has to be regarded as an application ontology Guarino

(1998), in the sense that its vocabulary offers terms that are related to a particular application (research data integration) and that are to be seen as specializations of more general terms drawn from a top-level ontology, for clarity and interoperability. In particular, we will indicate the terms from the CIDOC CRM Doerr (2003) top ontology. The *BDI* terms can be seen as specializations of the terms of CIDOC CRM. We choose the CIDOC CRM as reference top ontology because it is an ISO standard largely employed in Cultural Heritage.

The *BDI* ontology is presented as an UML class diagram, split into several diagrams for readability. Each diagram gives classes (boxes) and associations (arrows and their labels), and documents the domain and range of each association. Additional knowledge on the classes and the associations of the ontology are specified in a terse natural language that is close to formalization. Classes are named using the corresponding English words, while with a few exceptions associations are identified by numbers to simplify reading.

4.0.1 Database and view modeling

The diagram on Figure 1 presents the modeling of the basic entities, that is databases and their views.

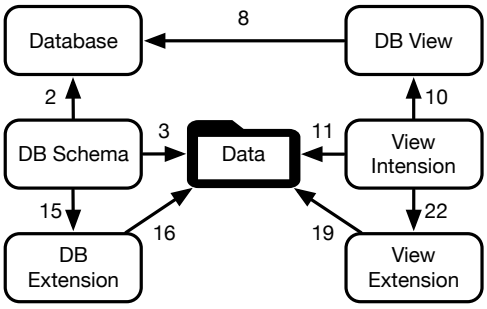

Figure 1 Basic data entities of *BDI*

The classes Database, DB Schema and DB Extension have as instances databases, database schemata and database extensions, respectively. The schema of a database is connected to its database by association 2 and to its expression via association 3; such expressions is a data object, instance of class Data, giving the actual schema in some machine-readable notation. The ontology does not give any detail about the class Data, which is depicted with a special symbol to signify that its instances are totally implementation-dependent. Possible examples of Data instances are: locally accessible XML documents; IRIs that lead to the data via de-referentiation; code (such as a web service call or a database query) that leads to the data via execution.

To capture the dependency of database extensions from schemata, the ontology directly associates the former to the latter via association 15. Database extensions include the empty database extension , which is used as extension of a newly created database. Each DB extension is linked to the actual content of the database (also instances of Data) via association 16. The empty database extension is linked by association 16 to a special instance of class Data corresponding to the empty data object.

The modeling of database views is very similar to that of databases. A view over a database is an instance of the class DB View. A view is associated to its database via association 8 and has one or more intensions, instances of class View Intension, each linked to its view via association 10. Each view intension is also associated (via association 11) with a data object, instance of class Data, giving the actual expression of the intension, *i.e.*, a query in some query language appropriate to the schema of the database. A view intension has one or more extensions, instances of class View Extension, connected to it via association 22. View extensions include the special object that represents the empty view extension. Each view extension is linked via association 19 to its content, which is a Data object obtained by applying the intensions query to the current DB Extension.

The following constraints apply to the associations introduced thus far:

- Associations 2 and 10 are total and functional, that is, every schema is the schema of exactly a database,
- Associations 3, 11, 16 and 19 are total and functional; that is, every DB schema, DB extension, view intension and view extension are associated to exactly one data object giving the relative data content.
- Association 8 is total and functional, that is, every view is defined over exactly one database.
- The inverse of associations 15 and 22 are total and functional; that is, every DB extension is associated to exactly one DB schema and every view extension is associated to exactly one view intension.

4.0.2 Event modeling

Events are represented in the *BDI* ontology by the classes shown in the UML class diagram in Figure 2, double-framed for perspicuity:

The most general such class is Event (specialization of the CRM class E5 Event), which has five sub-classes, each corresponding to a different kind of event illustrated in the previous Section. In the diagram, empty-headed arrows stand for sub-class associations.

The sub-classes of Event are pairwise disjoint, and their disjunction equals the class Event, *i.e.*, every event is an instance of exactly one of the sub-classes of Event.

The class Time (specialization of the CRM class E50 Date) has as instances time units, usually referred to as timestamps, at which changes occur. The association ET relates an event to a timestamp, and is inherited by each

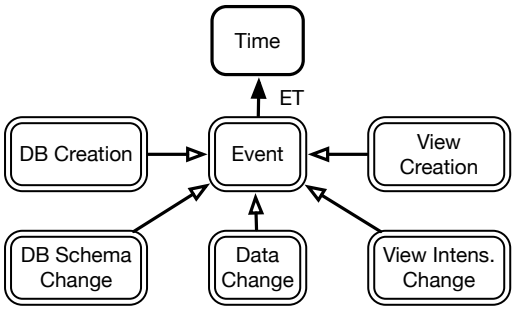

Figure 2 Event classes of *BDI*

sub-class of Event, so that any event of any kind has an associated timestamp.

- Association ET is total and functional, that is, every view is defined over exactly one database. Moreover, it is injective as no two events occur at the same time.

From the discussion presented in the previous Section, all events in our scenario produce some data entity, either anew or by transforming some existing data entity. As such, they can be modeled following one of two alternative approaches, the intensional or the extensional approach. In the intensional approach, each event is characterized structurally by the parameters of the operation that it performs. Such parameters depend on the model of the data entity that the operation produces. For instance, a DB Schema Change event on a relational schema has parameters describing which table definitions are changed, and how; the same event on an RDF Schema would be characterized by parameters describing which class and properties definitions are changed, and how. In sum, the intensional modeling of events would tie the *BDI* ontology to one or more data models. For the sake of simplicity and generality, the *BDI* ontology chooses to characterize events following the alternative approach, the extensional approach. In this approach, each event is characterized solely in term of the input (if any) and the output of the corresponding operation. For instance, a data change event transforms the extension of a database, and in the extensional approach it is characterized by the transformed database extension before the transformation (the input) and the database extension after the transformation (the output). The same applies for the other kinds of events.

We now illustrate how every kind of event in our scenario is structured in the *BDI* ontology.

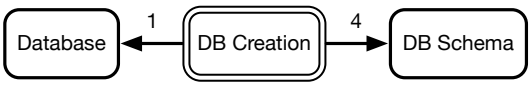

Figure 3 DB Creation Event

Creation events. According to the standard practice in data engineering, a database is created with a well-defined schema and an empty extension. Successively to its creation, the database is populated through a series of Data Change events: the first of such events replaces the empty extension with a non-empty extension, while the successive ones insert more data into the database extension. The same protocol is to be used to populate a DB extension after a DB schema change: upon creating the new schema, the schema is associated with the empty extension; to populate the modified schema with the data, one or more data change events have to be used. This practice is assumed by the ontology, which defines the DB Creation event class and endows it with two associations (see Figure 3): association 1 connecting a DB Creation event with the created database, and association 4 connecting a DB Creation event with the schema of the created database.

Similarly, a View creation event (see Figure 4) is connected to the created view (association 9) and to the intension of the created view via association 12.

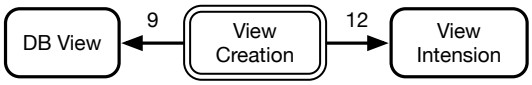

Figure 4 View Creation Event

The following constraints apply to the associations introduced thus far:

- Associations 1 and 9 are total, functional, injective and surjective, that is, a DB (respectively, view Creation) event creates exactly one DB (view), no two events create the same DB (view), and every DB (view) is the result of a creation event.
- Associations 4 and 12 are total, functional and injective like associations 1 and 9 but not surjective, as a DB schema or a view intension may result from a change event.

Change events. A DB Schema change event (see Figure 5) is modeled as a transformation from one DB Schema to a newly generated DB Schema. Associations 5 and 6 connect a Schema Change event to the input and the output DB schemata, respectively.

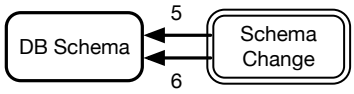

Figure 5 Database Schema Change Event

Notice that a DB Schema change may require changing the intension (*i.e.*, the query) of some (possibly all) of the views defined over the involved database, to make the corresponding queries conforming to the new

schema. However, this change requires the intervention of the user, who has to provide the new query(ies) through the apposite view intension change events. Therefore the ontology does not introduce any association reflecting the view intension change that may follow a DB schema change.

Similarly, a View Intension change event (see Figure 6) transforms one View Intension to a newly generated one. Associations 13 and 14 connect a View Intension Change event to the input and the newly generated View Intensions, respectively.

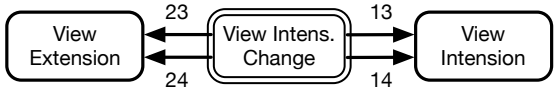

Figure 6 View Intension Change Event

Any View Intension change, in turn, causes a corresponding transformation of the extension of the same view, which can be handled in a completely automatic way because all is needed is the modified query provided by the user in the context of the view intension change.

Associations 23 and 24 connect a View Intension Change event to the extension of the input intension and to a newly generated View Extension, respectively.

Figure 7 Data Change Event

Finally, a Data Change event (see Figure 7) causes a DB extension to be replaced by a newly created DB extension, linked to the event by associations 17 and 18 respectively. Any Data change, in turn, causes a corresponding transformation of the current Extension of any view defined over the involved database into a new View Extension. The affected current view extensions are all connected to the event via associations 20, while those that replace them are connected to the same event via association 21.

The following constraints apply to the associations introduced thus far:

- All associations modeling changes, namely associations 5, 6, 13, 14, 23, 24, 17, 18, 20 and 21, are total, functional and injective, whether they model the entity before the change or the entity after the change. Totality and functionality ensure that every change has exactly one input and one output entity. Injectivity ensures that no two changes have the same input or the same output.

4.0.3 Semantic Linking

In our definition of data integration the logic that guides the discovery of semantic relationships between database views plays a key role (see Section 2.1). Several types of semantic relationships can exist between concepts defined in different database view schemata. Examples of semantic relationships include: the inclusion relationship that is the standard sub-type/super-type relationship; is-a and part-of relationships; member-collection relationship (association relationship); feature-event relationship; phase-activity relationship; place-area relationship; component-object relationship; antonyms/synonyms relationships, etc. Other types of semantic relationships can exist that are domain-specific.

Several kinds of logic can be adopted in order to discover relationships between database views. The adopted logic to discover a semantic relationship between two database views, as said in Section 2.1, depends on the scientific method followed (hypothesis driven/data driven method). In addition, we think that the choice of a suitable logic depends also on the scientific context within which the integration process is carried out. Different kinds of logic (conventional logic, modal logic, causal logic, temporal logic, *etc.*) can be explored. An additional important issue to be taken into consideration when designing discovery algorithms is their computational complexity.

A relationship between two views/extensions is materialized by a link that connects these two views/extensions. Instrumental in implementing an efficient linking process is the creation of a catalogue where all the different database views are published. By database view publication we mean a process that allows the research community to discover, understand and make assertions about the fitness of a view for integration purposes. This means that each view schema should be endowed with an ID, metadata and a timestamp.

In our ontology, the SemLink association connects semantically related view Extensions. As argued above, this connection is purely based on the semantics of the application, and it may involve any arbitrary pair of extensions. So no constraint is set on it.

4.0.4 Other constraints

This Section presents the constraints of the *BDI* ontology not stated so far, categorizing them in homogeneous groups.

Disjointness constraints.

- All classes in the *BDI* ontology are pairwise disjoint, except Event and its sub-classes.
- All associations in the *BDI* ontology are pairwise disjoint.

Inheritance constraints. These constraints concern inheritance of associations 2 and 10 via DB Schema or View Intension change events, respectively.

- A DB schema s′ resulting from a DB schema s via a DB Schema Change event, is associated (via association 2) to the same Database as s.
- A View Intension v′ resulting from a View Intension v via a View Intension Change event, is associated (via association 10) to the same View as v.

Closure constraints. These constraints express conditions on the extension of the classes whose instances results from events.

- Every database is the result of a DB creation event (via association 1)
- Every DB View is the result of a View creation event (via association 9)
- Every DB schema is either the result of a DB creation event (via association 4) or of a DB schema change event (via association 6), but not both.
- Every View Intension is either the result of a view creation event (via association 12) or of a View Intension change event (via association 14), but not both.
- Every View Extension is either the result of a Data Change event (via association 21) or of a View Intension change event (via association 24), but not both.

Pairing constraints. These constraints concern the change events. All such events have an input and an output object. Therefore each of them must be connected via the appropriate associations to one (or more than one) pairs of entities. We notice that pairing constraints involve solely total, functional and injective associations.

- Every schema change event is connected to exactly two DB schemata, one via associations 5 and the other via association 6.
- Every view intension change event is connected to exactly two view intensions, one via associations 13 and the other via association 14.
- Every view intension change event is connected to exactly two view extensions, one via associations 23 and the other via association 24.
- Every data change event is connected to exactly two DB extensions, one via associations 17 and the other via association 18.
- Every data change event is connected to exactly $2k$ view extensions, k via associations 20 and the other k via association 21.

Conformance constraints. These constraints involve conformance relations between intensions (schemata or queries) or between intensions and extensions. As such, they cannot be stated by simply posing existential or cardinality conditions on the extensions of the *BDI* classes and associations. Moreover they depend from external factors; in particular, the first two kinds of constraints depend on the input of the user, while the last kind depend on the query evaluation. For these reasons these constraints do not play any role on the correctness of the solution discussed below, they are just presented for the sake of completeness.

- Conformance between DB schemata and DB extensions: the data of a DB extension conform to the DB schema of the extension.
- Conformance between DB schemata and View intensions: At any time, the query of the intension of a view is a valid query against the schema of the database on which the view is defined at that time.
- Conformance between View intensions and View extensions: At any time, the extension of a view is the result of the query of the view intension against the extension of the schema of the database over which the view is defined at that time.

4.1 The Research Data Integration Problem

Based on the considerations about the characteristics of the scientific application scenario described in this section and the subsequent formalization of this scenario, we give two definitions of the research data integration problem. The first definition is an informal one; the second one is stated in terms of the concepts described in this section.

(Informal) Definition of Research Data Integration: Given a set of distributed, heterogeneous and time-dependent database views, by data integration we mean the ability of a researcher/software engine to link their extensions on the basis of existing semantic and/or temporal relationships among the intensions of the database views these extensions belong to.

(Formal) Definition of Research Data Integration: An instance of the problem is given by an instance of the ontology *BDI* , that is an information system S composed of

- a finite set of objects OS;
- for each class C in *BDI* , a subset CS of OS, called the extension of C in S;
- for each association A in *BDI* , a subset of (OS x OS), called the extension of A in S.

An instance is *consistent* if it satisfies all constraints.

Notice that an instance would include also an extension of the SemLink association, connecting extensions of database views between each other, on

the basis of semantic relationships among their database views.

The Data Integration Problem can be formulated as the development of an information system that, given a stream of events of the kinds considered in this study, is able to maintain a consistent instance of the *BDI* ontology including those events and the data and views that they involve, making databases, views, their extensions or intensions citable and versioned.

4.2 Implementation Considerations

The complete characterization of the *BDI* ontology provided in a previous part of this Section naturally suggests an ontology-based implementation strategy for the Research Data Integration Problem. Such strategy would be based on two pillars:

1. the ability to express the constraints of the *BDI* ontology and the instance of the Research Data Integration Problem at hand as axioms of an ontology of the most recent and powerful language of the Semantic Web family, namely the OWL 2 DL language Motik et al. (2012); and

2. the ability to implement the operations for creating, evolving and querying the instance by using a suitable OWL engine (such as OpenLink Virtuoso[2]) as basic technology, relying on the consistency checking functionality of the engine to maintain consistency.

Albeit appealing for its declarative approach, this strategy presents three main drawbacks from a system engineering point of view:

- Not all constraints can be expressed as OWL 2 axioms; aside from conformance constraints, which require checking domain specific relations, pairing constraints (which demand the existence of an individual in correspondence of the existence of a different individual) cannot be expressed in a description logic, due to the fact that such logic operates under the Open World Assumption; therefore, the lack of an individual is never interpreted as an inconsistency. Another problem arises from the fact that some constraints can be captured as complex role inclusion axioms; for instance the first one of the two inheritance constraints can be expressed (using the logical notation of Description Logics) as $(6^{-} \circ 5 \circ 2) \sqsubseteq 2$. As a result, association 2 is composite and cannot be declared to be functional, because doing so would violate a global restriction on the axioms of the ontology.

- Once consistency is broken, there is little support or no support at all to help the user understanding what went wrong and how the instance can be repaired.

- Efficiency may be an issue, as the KB consistency checking problem for the Description Logic SROIQ Horrocks et al. (2006), on which OWL 2 DL is based, is intractable. Furthermore, the size of any realistic instance of the problem may easily feature thousands of views, each with hundreds of versions, if not more; under the circumstances, even a polynomial algorithm where the polynomial has rank at least 2, is going to be inadequate to any expectation of the users concerning efficiency. As it will be shown, such expectations are perfectly legitimate in the present case.

We consider therefore the ontology-based implementation strategy appropriate only for a fast prototyping of the system, at best, because in those circumstances it is not vital to capture all constraints, while efficiency is not an issue since the prototype is likely to be tested on toy examples.

An implementation strategy that guarantees consistency while attaining efficiency, can be devised based on the idea of keeping the system under control by implementing the operations that create and modify an instance of the problem in a consistency-preserving way. That is, instead of giving the user the freedom to enter any item of information into the system and checking consistency *afterwards,* this approach *prevents* the insurgence of inconsistencies by strictly controlling the way the instance is created and modified by the user. This approach clearly limits the freedom of the users, by allowing them to perform only certain operations; however, the permitted operations can be chosen carefully enough to avoid any impact on the usability of the system. In particular, the permitted operation should strictly mirror the events by which the reality that the system models evolves, so as to be able to reflect any change of the reality into the system.

To prove the feasibility of this last implementation strategy, in the Appendix we present the fundamental algorithms for inserting knowledge into a system implementing in a straightforward way an instance of the problem. The implementation is straightforward in the sense that it reflect the definition of instance of the problem; thus, each class is implemented as a set of objects, and each associations is implemented as a set of object pairs. This storage strategy is sufficient for expressing the algorithms and showing that the consistency of the system can be efficiently implemented; clearly, it has to be replaced by a database oriented solution in any realistic setting.

5 Concluding Remarks

In this paper we have addressed the challenging problem of data integration in the context of an evolving research data scenario. Such scenario embodies the main characteristics of big data as well as those of a dynamic scientific environment. In particular, we have considered,

in addressing this problem, not only the high dimension of the data sets to be integrated but also the variability of their structure. To address the high volume of the data sets we have used the concept of database view that identifies data subsets of interest to be integrated instead of consider the whole (voluminous) data sets. The time dependence of both volume and structure of data sets and, therefore, the need to integrate them not only on the basis of semantic relationships but also temporal ones make the data integration problem very complex. To our best knowledge such a complex problem has not been addressed in the literature. In addressing this problem we have adopted a theoretical approach. In essence, we have, based on the *BDI* ontology, modeled all the events that modify structure and volume of the data sets as well as the associations among them. Such formalization, in conjunction with an appropriate integration logic that is domain-specific, enables to nicely address and solve a complex problem.

In addition, in Appendix, we describe a set of algorithms that implement the integration scenario described in this paper.

References

Bizer, C. (2013), 'Interlinking scientific data on a global scale', *Data Science Journal*, Vol. 12, pp.GRDI6–GRDI12.

Buneman, P., Davidson, S. and Frew, J. (2016), 'Why data citation is a computational problem', *Communications of the ACM*, Vol. 59, No. 9, pp.50–57.

Council, N. R. et al. (2010), *Steps toward large-scale data integration in the sciences: Summary of a workshop*, National Academies Press.

Doerr, M. (2003), 'The cidoc conceptual reference module: An ontological approach to semantic interoperability of metadata', *AI Mag.*, Vol. 24, No. 3, pp.75–92.

Floridi, L. and Sanders, J. (2004), 'Levellism and the method of abstraction, information ethics group'.

Guarino, N. (1998), 'Formal ontology in information systems', *Proceedings of FOIS 98*, IOS Press, Amsterdam, pp.3–15. Amended version.

Horrocks, I., Kutz, O. and Sattler, U. (2006), 'The even more irresistible sroiq', *Proc ́of the 10th Int ́Conf ́on Principles of Knowledge Representation and Reasoning (KR 2006)*, AAAI Press, pp.57–67.

Idreos, S. (2013), 'Big data exploration', *Big Data Computing*.

Jensen, C. S., Dyreson, C. E., Böhlen, M., Clifford, J., Elmasri, R., Gadia, S. K., Grandi, F., Hayes, P., Jajodia, S., Käfer, W. et al. (1998), 'The consensus glossary of temporal database conceptsfebruary 1998 version', *Temporal Databases: Research and Practice*, Springer, pp.367–405.

Kitchin, R. (2014), 'Big data, new epistemologies and paradigm shifts', *Big Data & Society*, Vol. 1, No. 1, pp.2053951714528481.

Levy, A. (1998), 'The information manifold approach to data integration', *IEEE Intelligent Systems*, Vol. 13, No. 5, pp.12–16.

Motik, B., Patel-Schneider, P. F. and Parsia, B. (2012), 'OWL 2 Web Ontology Language structural specification and functional-style syntax (second edition)', W3C recommendation, W3C. `http://www.w3.org/TR/2012/REC-owl2-syntax-20121211/`.

Ziegler, P. and Dittrich, K. R. (2004), 'Three decades of data intecrationall problems solved?', *Building the Information Society*, Springer, pp.3–12.

A A solution to the Data Integration Problem

In this appendix, we describe a solution to the Data Integration Problem, in the form of a set of algorithms that, given a consistent instance of the *BDI* ontology, implement: (1) the basic operations in the data integration domain so as to produce the minimal, consistent instance of the *BDI* ontology that reflects each operation, and (2) the retrieval of versions of view extensions. We also briefly discuss soundness and completeness of these algorithms.

A.1 Creating and evolving an instance of the BDI *ontology*

As pointed out in Section *BDI* , the ontology *BDI* offers five kinds of events to model the evolution of an application scenario:

- DB Creation: produces a database with an associated schema and an empty extension.
- Schema Change: produces a new DB schema from an existing one.
- Data Change: produces a new DB extension from an existing one, and a new view extension from an existing one.
- View Creation: produces a view and associated intension and extension.
- View Intension Change: produces a new view intension from an existing one, and a corresponding new view extension from an existing one.

Any implementation of the scenario must therefore provide an operation for each one of the five events, establishing the actions to be performed on the instance of the ontology in order to correctly reflect the corresponding event.

Below we provide an algorithm for each operation. They all take as input a consistent, possibly empty, information system S that is an instance of the ontology *BDI* and return S modified according to the semantics of the operation. For readability, these input and output will be tacitly understood.

A.1.1 Database Creation

Let us now see the algorithm for the creation of a database. The algorithm takes as input a data file with the schema of the database to be created. To simplify the description of the algorithm, in this and in all the following operations we assume that this input parameters are correct, thus avoiding checking them to detect anomalies.

The database creation algorithm first creates a DB creation event with the current time (step 1), then it creates a new database associated with the event (step 2), finally it creates a DB schema and associates it to the event, the database, the provided file and the empty extension (step 3). Technically:

1. an instance ev of DB Creation event is created and associated with the current time via association ET
2. an instance db of Database is created and associated with ev via association 1
3. an instance sch of DB schema is created and associated with:
   (a) ev via association 4
   (b) db via association 2
   (c) the data file that holds the schema of D via association 3
   (d) the empty DB extension via association 15.

Notice that the identifiers of the objects created by the algorithms (ev, db and sch) are assumed to be generated in an automatic way, so as to simplify the operation and avoid any danger of collision with existing identifiers. If desired, these identifiers can be provided by the user as input parameters.

A.1.2 View Creation

The algorithm for the creation of a view takes as inputs:

- the identifier D of an existing database over which the view to be created is defined, and
- a data file with the query giving the actual intension of the view to be created.

The algorithm first creates a View creation event with the current time (step 1), then it creates a new view associated with the event and the database D (step 2), finally it creates a View Intension and associates it to the event, the view, the provided file and the empty view extension (step 3). Technically:

1. an instance ev of the View Creation event is created and associated with the current time via association ET
2. an instance vw of DB View is created and associated with:
   (a) ev via association 9
   (b) the database D via association 8
3. an instance vin of View Intension is created and associated with:
   (a) ev via association 12
   (b) vw via association 10
   (c) the actual intension via association 11
   (d) the empty view extension via association 22.

A.1.3 DB Schema Change

The algorithm for the change to a DB schema takes as inputs:

- the identifier Sch of the existing DB schema that must be changed, and
- a data file with the new schema.

The algorithm first creates a DB Schema Change event associated with the current time and with the schema to be changed (step 1), then it creates a new DB Schema associated with the event, the same database as the the schema to be modified, the provided file with the schema expression and the empty DB extension (step 2). Technically:

1. an instance ev of the DB Schema Change event is created and associated with:
   (a) the current time via association ET
   (b) Sch via association 5
2. an instance sch of DB Schema is created and associated with:
   (a) ev via (the inverse of) association 6
   (b) the same database as Sch via association 2
   (c) the data file with the new schema via association 3
   (d) the empty DB extension via association 15

Notice that in this way, at any time every DB schema has exactly one database, at least one DB extension and exactly one content connected to it. To populate the extension of the new schema, a data change operation must be performed, exactly in the same way the schema of a newly created database is populated.

A.1.4 View Intension Change

The algorithm for the change to a View Intension takes as inputs:

- the identifier VI of the existing View Intension that must be changed, and
- a data file with the query of the new intension.

The algorithm takes into account the fact that a view intension change event produces not only a new view intension, but also a corresponding new view extension, which replaces the current extension of the intension to be changed. The content of that view extension are computed by applying the query of the new intension to the current extension of the involved database. For this reason, the algorithm is a bit more complex than the algorithm for DB Schema change, which is similar in scope.

The algorithm first creates a View Intension Change event associated with the current time, with the intension to be changed and with the current (*i.e.*, latest) extension of this intension (step 1). Then it creates a new View Intension associated with the event, the same view as the intension to be modified and the provided data file with the query of the new intension (step 2). Finally, it creates a new view extension associated with the event, the new intension and the actual content computed as described above (step 3). Technically:

1. an instance ev of the View Intension Change event is created and associated with:
   (a) the current time via association ET
   (b) VI via association 13
   (c) the current extension of VI, via association 23
2. an instance vi of View Intension is created and associated with:
   (a) ev via (the inverse of) association 14
   (b) the same view as VI, via association 10
   (c) the data file with the query of the new intension via association 11
3. an instance vext of View Extension is created and associated with:
   (a) ev via (the inverse of) association 24
   (b) vi via association 22
   (c) the actual view extension content via association 19. Such content is computed by evaluating the query provided as input against the current extension of the database over which VI is defined.

A.1.5 Data Change

The algorithm for the change to a DB Extension takes as inputs:

- the identifier D of the existing database whose extension must be changed, and
- a data file with the new extension.

Similarly to the View Intension change algorithm, this algorithm takes into account the fact that a database extension change event produces not only a new database extension, but also a corresponding set of new view extensions, one for each view defined over that database. Each such new view extension replaces the current extension of the current intension of the view. The content of the replacing view extension are computed by applying the query of the corresponding intension to the new extension of the involved database.

The algorithm first retrieves the current schema of the database to be changed, the current extension of that schema, and the set of the current extensions of each view defined over D (step 1). It then creates a Data Change event associated with the current time and with the DB

extension to be changed (step 2). Then it creates a new DB Extension associated with the event, with the current schema of D and with the provided data file (step 3). Finally, for each extension of a view defined over D, it associates the extension with the event and it creates a new view extension also associated with the event; the newly created extension has the same intension as the old one and a new content computed as described above (step 4). Technically:

1. let:
   (a) Sch be the current DB schema of D, retrieved via association 2
   (b) DExt be the current extension of Sch, retrieved via association 15
   (c) VExt_1, VExt_2, ..., VExt_n be the current extensions of the views defined over D, retrieved via association 8, 10 and 22
2. an instance ev of the Data Change event is created and associated with:
   (a) the current time via association ET
   (b) DExt via association 17
3. An instance Dext of DB Extension is created and associated with
   (a) ev via association 18
   (b) Sch via association 15
   (c) the input data file with the new extension via association 16
4. for each view extension VExt_k in VExt_1, VExt_2, ..., VExt_n
   (a) VExt_k is associated with ev via association 20
   (b) an instance VE_k of View Extension is created associated with
      i. ev via association 21
      ii. the same view Intension as VExt_k via association 22
      iii. the actual view extension content via association 19. Such content is computed by evaluating the query of the Intension of VExtk against the new extension of database D, resulting from the data change.

A.1.6 Correctness of the algorithms

We have claimed at the beginning of the Section that the algorithms introduced thus far are a solution to the Data Integration Problem. In this Section we provide a proof of the claim, by showing the following Proposition.

Proposition 1 *Given a consistent instance S_0 of the ontology* BDI *and a stream of events E_1, E_2, ..., E_n, ... each of one of the five kinds introduced thus far, the above operations produce a sequence S_1, S_2, ..., S_n, ... of instances of the ontology* BDI *such that:*

1. *each instance is consistent;*
2. *each instance S_i is the minimal instance of* BDI *that reflects the event E_i.*

Proof. *By induction. We first establish the base case by proving that S_1 resulting from the processing of event E_1 on S_0, satisfies both properties 1 and 2 for each of the five kinds of the events that E_0 may belong to. We then assume that the Proposition holds for the k-th event E_k and prove it for E_{k+1} for each of the five kinds of the events that E_{k+1} may belong to.*

A.2 Information extraction: Versioning

As already said, temporal relationships between extensions of database views are important in the scientific context. A temporal relationship denotes an ordering in time of events or states. Examples of temporal relationships include: antecedent-forerunner relationship; synchronicity relationship; asynchrony relationship; sequential relationship; etc. Again, other temporal relationships can exist that are domain-specific. Spatial relationships can also be of interest when integrating research data. Examples of spatial relationships include topological relationships; distance relationships; directional relationships; etc.

As a consequence of the extensional approach followed in event modeling, the *BDI* ontology allows keeping track of one set of temporal relationships, namely the versioning of the extensions and the intensions of view and databases. Overall, these are four associations, and they can all be derived from the associations of the *BDI* ontology introduced so far. To illustrate how, we now give an algorithm to derive the most complex of these four associations, the versioning of view extensions. The algorithms for deriving the other three associations are, mutatis mutandis, simplified versions of the algorithm presented below.

We further assume that S is consistent, that is it satisfies all the constraints given in *BDI* .

The algorithm takes as input S and the identifier V of a view in S. In return, the algorithm gives the chronologically ordered sequence V_1 V_2, ..., V_k of the extensions of V in S.

In order to obtain this result, the algorithm needs to consider the temporal ordering of the events that generate the extensions. There are two kinds of events that may produce extensions of a view V: view intension changes over V and data change events over the database D on which V is defined. In fact, every view intension change determines a new intension of V, which in time may have any number of extensions each produced by a change in the data of the extension of the database

D. The algorithm needs to retrieve these events from S, merge them, and sort the result chronologically based on the timestamp of each event, retrieved via the ET association. Once the ordered sequence of events is computed, it suffices to replace each event by the input and the output view extension to obtain the desired result.

It can be proved that this algorithm is sound and complete on any consistent instance S of the ontology *BDI* . It can also be quickly verified that the algorithm runs in a number of steps that is polynomial in the size of S.

The algorithm can be substantially simplified and improved from the performance point of view by associating each data entity in *BDI* whose versioning is desired, with the timestamp at which the entity is created. This can be done by propagating the timestamp from the events that create or modify each data entity to the data entity itself. With this information available, it suffices to retrieve all the entities of the desired kind or with the desired property and sort them based on the associated timestamp.

Conceptually, the modification can be introduced in the ontology by adding a new class Entity which generalizes all time-dependent classes, and by making this class the domain of the ET association (see Figure 8).

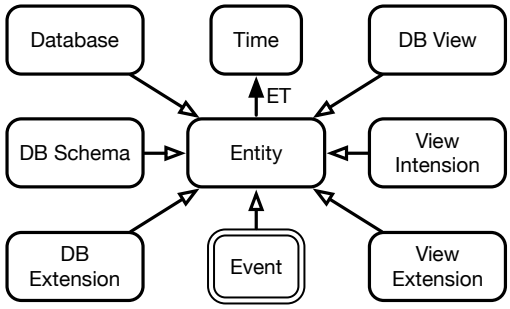

Figure 8 Class Taxonomy of a Refined Ontology

Any instance of a sub-class of Entity inherits the ET property and can therefore have an associated timestamp giving its creation time. Computing the ordered sequence of the extensions of a view V in any instance of this ontology can then be done as follows:

1. obtain the intensions $I_1, I_2, \ldots, I_n$ of V by walking backward association 10 from V

2. obtain the extensions of any such intension I_k by walking association 22 forward from I_k

3. sort these extensions based on the value of the ET association, *i.e.*, extension E precedes extension E′ if the timestamp of E precedes that of E′.

The correctness of this algorithm is established by the following Proposition:

Proposition 2 *Proposition. Given a consistent instance S_0 of the ontology* BDI *and an instance V of class View Extension, the above algorithm returns the sequence of all and only versions of V, in chronological order of creation.*

Proof. *Soundness: we prove that the sequence contains versions of V in the correct chronological order.*
Completeness: we prove that any version of V shows up in the sequence.