Towards a Language for Representing and Managing the Semantics of Big Data

Ermelinda Oro[1,2], Massimo Ruffolo[1,2], Pietro Gentile[1] and Giuseppe Bartone[2]
[1]*Institute of High Performance Computing and Networking, CNR, Via P. Bucci 41/C, 87036 Rende (CS), Italy*
[2]*ALTILIA srl, Piazza Vermicelli, snc, 87036 Rende (CS), Italy*

Keywords: Knowledge Representation and Reasoning, Big Data, Smart Data, Unstructured Data, Data Integration, Extract Transform and Load (ETL), Smart ETL, Database, NoSQL.

Abstract: The amount of data in our world has been exploding. Integrating, managing and analyzing large amounts of data – i.e. *Big Data* - will become a key issue for businesses for better operating and competing in today's markets. Data are only useful if used in a smart way. We introduce the concept of *Smart Data* that is web and enterprise structured and unstructured big data with explicit and implicit semantics that leverages context to understand intent for better driving business processes and for better and more informed decisions making. This paper proposes a language able to give a representation of *Big Data* based on ontologies and a system that implements an approach capable to satisfy the increasing need for efficiency and scalability in semantic data management. The proposed *MANTRA Language* allows for: (i) representing the semantics of data by knowledge representation constructs; (ii) acquiring data from disparate heterogeneous sources (e.g. data bases, documents); (iii) integrating and managing data; (iv) reasoning and querying with *Big Data*. The syntax of the proposed language is partially derived from logic programming, but the semantic is completely revised. The novelty of the language we propose is that a class can be thought of as a flexible collection of structurally heterogeneous individuals that have different properties *(schema-less)*. The language also allows executing efficient querying and reasoning for revealing implicit knowledge. These have been achieved by using a triple-based data persistency model and a scalable No-SQL storage system.

1 INTRODUCTION

One of the key challenges in making use of *Big Data* lies in finding ways of dealing with heterogeneity, diversity, and complexity of the data. *Big Data* volume, variety and velocity forbid the adoption of data integration and management solutions available for smaller datasets based on traditional *Extract Transformation and Load* (ETL) or manual data manipulation approaches.

Big Data, locked in the web and enterprise sources, can include both structured and unstructured data. If smartly utilized it can yield unprecedented insights into solving tough business issues and improving customer relations. We introduce the concept of *Smart Data* that is data with explicit semantics combined with implicit semantics that leverages context for better understanding intent, optimizing business processes, and supporting smarter decision-making. The challenge is using *Smart Data*, rather than just *Big Data*, to unlock the value held in data.

Implicit semantics can be obtained, for instance, by means of annotation, entity and metadata extraction, and natural language processing. Inference and machine learning techniques can discover implicit semantics. Ontologies are considered a key technology enabling semantic interoperability and integration of data, processes and querying (Motta, 1999)(Staab et al. 2001) for both structured data and unstructured data (Haase et al., 2008)(Cimiano et al., 2007).

Associating *Big Data* to an ontology description that allows for reasoning on large amounts of data is becoming an issue assuming increasingly importance. In order to recognize and join information contained in a system to concepts in the web or outside of your system, it is necessary a language in which represent, organize and reason about entities. This is the goal of the *Semantic Web* of the last 15 years (Berners-Lee et al., 2001). However, *semantic web systems* generally generate

Oro E., Ruffolo M., Gentile P. and Bartone G..
Towards a Language for Representing and Managing the Semantics of Big Data.
DOI: 10.5220/0004916906510656
In *Proceedings of the 6th International Conference on Agents and Artificial Intelligence* (ICAART-2014), pages 651-656
ISBN: 978-989-758-015-4

metadata and identify entities manually or serializing database values. Actually, the tagging process is very hard. An important issue is how to store ontologies and how to reason with them, without losing out of sight the need for scalability. In fact, the effective use of ontologies requires not only a well-designed and well-defined semantic language, but also adeguate support to operations. The *Linked Data* paradigm (Bizer et al., 2009) is one approach to cope with *Big Data*. *Linked Data* represents semantically well-structured, interconnected, syntactically interoperable datasets. Numerous commercial and non-commercial organisations have started to utilize *Linked Data* for purposes like acquisition, enrichment, or integration of information. But, only a small part of the web of documents is represented as rich data.

In this work we provide an initial contribution on open issues related to the semantic management of *Big Data*. The objective of this paper is twofold. First, we describe initial work on a new language, named *MANTRA Language* (ML), which allows for: (i) representing the semantics of data by knowledge representation constructs; (ii) acquiring *Big Data* from disparate heterogeneous sources (e.g. databases, web documents, social networks); (iii) integrating and managing data; (iv) reasoning and querying. Second, we present a triple-based data persistency model, which enables efficient storage and querying of *Smart Data,* and the implemented system supporting the ML. This has been achieved by using a triple-based data persistency model and a scalable storage system that allows to store *Big Data* in the form of triples, like RDF (W3C RDF) for the *Semantic Web*, and to execute efficient querying and reasoning operations in a distributed way. Roughly speaking, the ML and its supporting system enable to deal with *Big Data* from a knowledge representation perspective. They enable to extract data from heterogeneous data sources in order to integrate them and execute efficient reasoning and querying operations to reveal implicit knowledge.

The paper is organized as follows: Section 2 presents the *MANTRA Language* and Section 3 presents the main architecture of the system supporting the language.

2 MANTRA LANGUAGE

Businesses are increasingly looking for *semantic tools* that enable to model and manage complex domain-knowledge and to solve real-world problems (Dao, 2011) (Blomqvist, 2012). It is necessary a language in which represent, organize and reason about entities.

This section presents the *MANTRA Language* (ML). The ML introduces *ontological constructs* and *database* and *linguistic descriptors*, which enable to extract and integrate data available in heterogeneous data sources. The syntax is based on the intuitive logic programming. In particular, ontological constructs are partially derived from *OntoDLP* (Calimeri et al., 2003) (Ricca and Leone, 2007), whereas acquisition formalism are based on the *XOnto* language (Oro and Ruffolo, 2008) (Oro et Al, 2009). *OntoDLP* introduces many interesting features, including complex types, e.g. sets or lists, and intentional relations, which are used in ML. *XOnto* describes a simple way to equip ontological element by a set of rules that describe how recognize and extract objects contained into documents.

2.1 Ontology Constructs

Constructs that enable to define the structure of an ontology (*light schema*) and its instances are presented in the following.

Classes. A class can be thought of as a flexible collection of structurally heterogeneous individuals that may have different properties. Such collections can be defined by using the keyword *class* followed by its name. Class attributes can be specified by means of pairs (*attribute-name:attribute-type*), where *attribute-name* is the name of the property and *attribute-type* is the class the attribute belongs to. Class attributes model *canonical properties* present in class instances and admit null and multiple values by exploiting the triple-based data persistency model. Unlike OntoDLP, the ML allows for storing objects which properties do not match the declared class schema and objects that have different set of attributes. The syntax for declaring a class is shown below:

```
class_person(name:string,  age:integer,
father:person).
```

Class Instances. Class domains contain individuals, which are called *objects* or *instances*. Each individual in the ML belongs to a class and is uniquely identified by a constant called object identifier (*oid*). Objects are declared by asserting a special kind of logic facts (asserting that a given instance belongs to a class). However, as shown in following paragraphs, the most common way to define class instances in the ML is to use *descriptors*. The syntax that allows defining objects is the following:

```
oid:class_name(att_name_1:value_1,…,
att_name_n:value_n).

p1: person(name:'Jhon', age:21,
father:p2).

P2: person(name:'Sam', age:50,
father:p3, income:3000).
```

Tanomies. Ontology concepts are usually organized in taxonomies by using the *specialization/generalization* mechanism. This is usually done when it is possible to identify subsets of individuals having different sets of attributes. In particular, individuals that have at least one attribute more than a given set of objects can be considered a sub-class (specialization) of such objects. In the ML are allowed specializations that represent not exhaustive decompositions of the instances of a class in which objects may have different sets of attributes. The syntax that allows defining subclass is the following:

```
subclass_name(new_att_name_1:value_1,…,
att_name_n:new_value_n)
    subclassof superclass_name().
```

Relations. Another important feature of an ontology language is the ability to model n-ary relationships among individuals. Relations are declared like classes: the keyword *relation* precedes a list of *canonical attributes*. The set of attributes of a relation is called *relation schema* and can be flexible as for classes. The cardinality of the schema, also called *arity*, can vary for different relation instances. The ML allows for organizing relations in taxonomies. Relation instances are equipped with an oid when the number of attributes is above two (reification).

2.2 Data Integration Constructs

Database and linguistic descriptors, which enable to extract and integrate data available in heterogeneous data sources, are presented in the following.

Descriptors are founded on the basic idea described in (Oro and Ruffolo, 2008) and (Oro et Al, 2009) which describe a system for Information Extraction (IE) from PDF documents. It represents an approach founded on the idea that objects and classes of ontologies can be equipped by a set of rules that describe how recognize and extract objects contained into an external source. We use descriptors in order to acquire *Big Data* from disparate heterogeneous sources. In particular, we present two different kind of descriptors: for database and for documents in natural language.

A descriptor is a rule with the form $h \leftarrow b$ in which b (*descriptor body*) constitutes a pattern of ontology objects that allows recognizing a (set of) specific object defined in the left-side h (*descriptor head*). Database and Linguistic descriptors are described separately as follows.

Database descriptors. These descriptors allow for describing how to recognize and extract objects and relations instances starting from tuples stored in a relational database. Database descriptors can be defined by using the keyword *descriptor* followed by related class (or relation) name in the head. While in the body of the rule, there are: a built-in that enables database connection and the list of atoms that perform query operations on the knowledge base. For each concept in the ontology, can be defined a database descriptor able to connect the database, perform queries and assign the query results to variables declared in the head.

For example, suppose to have the following two relations:

holdsCar(*owner:Person, car:Car*)
hasCar(owner,car)

where the relation *holdsCar* assign for each *Person*, the owned *Car*. The table *hasCar* connects the owners to their cars, stored on disk.
If we want to extract information about the owned cars of the known persons from a database and populate the ontology, we have to state as follows:

```
Descriptor holdsCar(O,C) ←
#query("mysql:port:user:password",SELEC
T * FROM hasCar",O,C),O:Person().
```

The assignment of the value to the variables in the head can be done in the *#query* construct directly or in the following part of the descriptor body. If the view returned from the query has more variables than those in the list, the schema of the class (relation) in the head is increased with new attributes of generic type named with the column name. So a class can be seen as a container of objects potentially belonging to multiple subclasses. This implicit knowledge can be made explicit by reasoning operations and specific algorithms.

Linguistic descriptors. These descriptors extract concepts from an input text or document. For example, given the input string "John owns a Bentley Continental GT", the following descriptors enable to extract the person, the car and the relation between them.

```
person(X) <- X:#entity("Person").

car(X) <- X:#dictionary("Car").
```

```
carOwner(X,Y)       <<-       (sentence)
X:#entity(Person)        Z:#lemma("own")
Y:#dictionary("Car").
```

In this example, we search the concept *carOwner* as a weak sequence of concepts: a person, a lemma of the verb "to own" and a car. In order to identify persons we used the construct `#entity` that enables to use sophisticated algorithms for recognizing a specific type of entities, whereas for recognizing cars we used a simple dictionary by exploiting the construct `#dictionary`. In the example, a car is recognized only if the term in the input text matches to a term contained into the dictionary named `"Car"`. In order to recognize a lemma of the verb "to own" we used the construct `#lemma`. The constraints `(sentence)` ensure that these items are contained in the same sentence.

2.3 Querying and Reasoning

The language enables to define reasoning tasks that reason on the knowledge base and infer classes and relations. As said before, the ML gives the possibility to define classes as collections of instances belonging to different, potentially unknown, subclasses. So, in order to explicit subclasses membership of instances, reasoning modules are required to create new subclasses based on certain properties.

In ML, it is possible to derive new knowledge not only by reasoning on class and relation instances, but also by meta-reasoning on the structure of classes and relations. Currently, meta-reasoning is possible by using the following built-in constructs:

- *#class (name:string)*: contains a fact *class("c")* for each class with name "c";
- *#relation (name:string)*: contains a fact *relation("r")* for each relation with name *"r"*;
- *#subclass(sub:string,super:string)*: contains a fact for each class *"sub"* that is subclass of *"super"* (a built-in with the same meaning exists for the relationships *#subrelation*);
- *#type(class:string, instance:object)*: for each instance of class "c" contains a fact *#type(class:"c",instance:oid)*;
- *#attribute(name:string,attr:string)*: contains a fact *#attribute(name:"c",attr:"a")* for each class/relation "*c*" that has an attribute named "*a*".

The language can be used not only to derive new knowledge but also simply to query in order to extract knowledge contained in the ontology also not directly expressed.

For example, suppose you want to find all the *Persons* from Rome who have at least two cars. To express the query we have to use the following expression:

```
holds2Cars(N) <- person(name:N,
city:"Rome"), holdsCar(N,C1),
holdsCar(N,C2),C1<>C2.
```

Querying and meta-querying constructs can be merged in order to obtain also the schema information about objects.

3 SYSTEM IMPLEMENTATION

This section shortly describes the system implementing ML. The system represents a complete framework that allows for acquiring and integrating data coming from heterogeneous sources, for transforming data into *Smart Data* and for querying and reasoning on them.

Main components of the system implementing the ML are shown in Figure 1 and are described in the following.

- The *Parser Module* reads code listing and builds, in main memory, an image of the ML constructs it recognizes. Eventually, it throws error messages if syntax errors are found.
- The *Type Checker* module checks for inconsistencies in the ontology. It verifies the defined ontology contains some admissibility problem (i.e. a class is declared twice). It also ensures the compatibility of the defined and assigned types. Enabling schema-less representation consistency checking is a more robust problem that is better to not address here due to lack of space.
- The *Descriptor Evaluator Module* executes descriptors on data sources. In particular, for *database descriptors*, this module executes connection and queries to the particular database that contains data to extract. for *linguistic descriptors*, this module evaluates linguistic patterns on a given document and extract object instances when patterns are recognized.
- The functionalities of *Reasoning* and *Querying Modules* execute reasoning and querying tasks. It is noteworthy that the novelty of our approach consists in the triple-based data model with distributed persistency. This allows pushing integration algorithms directly within data storage environments exploiting map reduce approaches (Dean and Ghemawat, 2008). In particular the use of triple model enable to use map reduce approach

presented in (Sun and Jin, 2010) to querying distributed data store.

- *Persistency Manager* module deals with the persistency of schema and instances of the ontology.

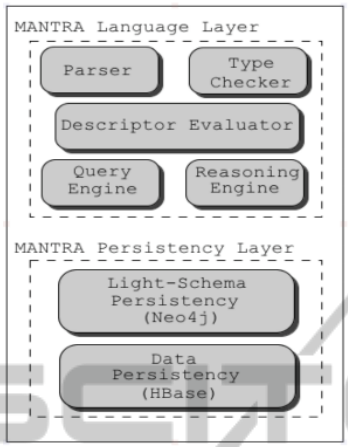

Figure 1: System Architecture.

To perform querying and reasoning operations is necessary to define a storage model able to make efficient operations. In fact, although OWL (Smith and Welty, 2003), based on *Descriptive Logics* (Baader et al., 2003), has been designed for the semantic Web, data in OWL is not easy to manipulate or query when they grow up. In order to overcome this problem, we need to study a tailored storage model for our ML. We believe that a specific physical representation of schemas and instances and an ad hoc index structures are necessary. In this work, we decided to store the ontology structure (T-Box in OWL) separate from the data (A-Box in OWL). The proposal is to obtain scalability by storing light schemas of ontologies and instances using *NoSQL* technologies. In particular, light schemas can be expressed by graphs in which nodes represent classes and reified relations of the ontology, while edges represent semantic relationships between classes (e.g. subclass, equivalence) and roles. Therefore, the use of graph database seems to be the most natural solution to store light schemas. In particular, we decided to store ontology structure in Neo4j (Neo4j). It is a full ACID-transaction compliant graph database written in Java. Although the data can also be viewed as a graph, it is not possible to adopt the same storage pattern seen for the schema above. We have to consider the different nature of the operations to be performed and the larger size of the data respect to the schema. We believe that querying operations respect some well-defined patterns. The most common queries consist in finding an object with a certain value for a specific attribute, all objects having a specific property, etc. Each query involves, like in SPARQL for RDF, three different entity: subject (S), predicate (P) and object (O). This led us to choose triple-based storage model for ontology instances. We decided to adopt the well-known open source project, *Hbase*, for distributed storage of triples. A data row in *HBase* is composed of a sortable row key and an arbitrary number of columns, which are further grouped into column families. Regarding the storage model, we adopt the idea of (Sun and Jin, 2010) which uses six tables to persist triples, one for each combination of subject, predicate and object: S_PO, P_SO, O_SP, PS_O, SO_P and PO_S. In these tables data are stored in row keys and column names (values are left empty). Each of these has only one column family and all columns belong to this. As we can see in (Sun and Jin, 2010), this storage model enable to execute efficient querying operations on distributed data store using map reduce strategy. The disadvantage of this approach is that it requires more storage space. There are several copies of data which locate in different tables. But storage space can be considered as infinite in a distributed environment.

4 CONCLUSIONS

In this work we addressed open issues related to the semantic management of *Big Data* emerging in several application areas. We described initial work on the *MANTRA Language* (ML) and the supporting system, which allows for representing the semantics of big data, acquiring big data from heterogeneous sources and enabling efficient reasoning and querying on them. Even though the *MANTRA Language* has a logic programming based syntax, it differs from logic programming becouse the schema of classes or relations is not fixed. This is possible because of the underlying triple-based persistence model, an one of the key feature of our framework. As future work we will deeply study and define formal syntax and semantic of the language and related coplexity issues. We will define semantic integration algorithms that can be pushed directly within data storage environments exploiting map reduce approaches (Dean and Ghemawat, 2008). Moreover, real examples of applications will be provided.

REFERENCES

Baader, F., Calvanese, D., McGuinness, D. L., Nardi, D., Patel-Schneider, P. F., 2003. The Description Logic Handbook: Theory, Implementation, and Applications, *Cambridge University Press,* Cambridge.

Berners-Lee, T., Hendler, J., Lassila, O., 2001. The Semantic Web. *Scientific American, 279 (5): p.34-43*.

Bizer, C., Heath, T., Berners-Lee, T., 2009. Linked Data - The Story So Far. *International Journal on Semantic Web and Information Systems*, Volume 5.

Blomqvist, E., 2012. The use of Semantic Web Technologies for Decision Support – A Survey, *Semantic Web*.

Calimeri, F., Galizia, S., Ruffolo, M., Rullo, P., 2003. OntoDLP: a Logic Formalism for Knowledge Representation. *Anwer Set Programming.*

Cimiano, P., Haase, P., Herold, M., Mantel, M., Buitelaar, P., 2007. LexOnto: A Model for Ontology Lexicons for Ontology-based NLP. *In Proceedings OntoLex (Workshop ISWC)*.

Dao, F., 2011. Semantic technologies for enterprises. *Technical report, SAP AG.*

Dean, J., Ghemawat, S., 2008. MapReduce: simplified data processing on large clusters. *Commun. ACM* 51.

Haase, P., Lewen, H., Studer, R., Erdmann, M., Gmbh, O., 2008. The NeOn Ontology Engineering Toolkit. *In 17th International World Wide Web Conference.*

Motta, E., 1999. Reusable Components for Knowledge Modelling, *IOS Press.* Amsterdam, The Netherlands.

Neo4j Graph Database, http://www.neo4j.org/. Retrieved 12/2013.

Oro, E., Ruffolo, M., 2008. XONTO: An Ontology-Based System for Semantic Information Extraction from PDF Documents. *ICTAI.*

Oro E., Ruffolo, M., Saccà, D., 2009. Ontology-Based Information Extraction from PDF Documents with Xonto. *International Journal on Artificial Intelligence Tools 18(5): 673-695.*

Ricca, F., Leone, N., 2007. Disjunctive logic programming with types and objects: The DLV+ system, *Journal of Applied Logic, Volume 5, Issue 3, 545-573.*

Smith, M. K., Welty, C., McGuinnes, A.L., 2003. OWL web ontology language guide. *World Wide Web Consortium.*

Staab, S., Erdmann, M., Studer, R., Sure, Y., Schnurr, H.-P., 2001. *Knowledge Processes and Ontologies*. IEEE Intelligent Systems, 16(1): p.26-34.

Sun, J., Jin, Q., 2010. Scalable RDF Store Based on HBase and MapReduce. *ICACTE*.

W3C. RDF. The Resource Description Framework. http://www.w3.org/RDF/. Retrieved 12/2013.