International Journal on Recent and Innovation Trends in Computing and Communication ISSN: 2321-8169
Volume: 5 Issue: 7 502 – 506

Vigorous Module Based Data Management

M. P. Bidve
Department of Computer Science and Engineering
M.S.Bidve Engg. Collage
Latur, Maharastra, India
manishabidve2@gmail.com

Prof. N. J. Pathan
Department of Computer Science and Engineering
M.S.Bidve Engg. Collage
Latur, Maharastra, India
pathan_nj@rediffmail.com

***Abstract*—** Data is important in today's life and it must be saved using less amount of memory. Data is important in day to day life for many purposes, like Government activities, any organization needs their own database, hospitals, schools etc. It is necessary to save data into database as per the user's query generation with less memory conjunction. One of the novel techniques we have developed for saving data into database by using file similarity algorithm. This technique is used to split the text file into number of paragraphs and save these paragraphs using appropriate reference number. These reference numbers are stored in database, whenever same paragraph will appeared in another text file it will check database and then save the other references of that file which are new for that file. This technique requires less memory and data can be stored in appropriate manner.

.

Keywords-*component: OWL, DL-Lite, File-Similarity*

*****

I. INTRODUCTION

The Web Ontology Language is a family of knowledge representation languages for authoring ontologies. Ontologies are a formal way to describe taxonomies and classification networks, essentially defining the structure of knowledge for various domains: the nouns representing classes of objects and the verbs representing relations between the objects. Ontologies resemble class hierarchies in object-oriented programming but there are several critical differences. Class hierarchies are meant to represent structures used in source code that evolve fairly slowly (typically monthly revisions) whereas ontologies are meant to represent information on the Internet and are expected to be evolving almost constantly. Similarly, ontologies are typically far more flexible as they are meant to represent information on the Internet coming from all sorts of heterogeneous data sources. Class hierarchies on the other hand are meant to be fairly static and rely on far less diverse and more structured sources of data such as corporate databases[1]. The history of artificial intelligence shows that knowledge is critical for intelligent systems. In many cases, better knowledge can be more important for solving a task than better algorithms. To have truly intelligent systems, knowledge needs to be captured, processed, reused, and communicated. Ontologies support all these tasks. The term "ontology" can be defined as an explicit specification of conceptualization. Ontologies capture the structure of the domain, i.e. conceptualization. This includes the model of the domain with possible restrictions. The conceptualization describes knowledge about the domain, not about the particular state of affairs in the domain. In other words, the conceptualization is not changing, or is changing very rarely. Ontology is then specification of this conceptualization - the conceptualization is specified by using particular modeling language and particular terms. Formal specification is required in order to be able to process ontologies and operate on ontologies automatically.

Ontology describes a domain, while a knowledge base (based on an ontology) describes particular state of affairs[2]. Each knowledge based system or agent has its own knowledge base, and only what can be expressed using ontology can be stored and used in the knowledge base. When an agent wants to communicate to another agent, he uses the constructs from some ontology. In order to understand in communication, ontologies must be shared between agents[2][3]. The OWL languages are characterized by formal semantics. They are built upon a W3C XML standard for objects called the Resource Description Framework (RDF). OWL and RDF have attracted significant academic, medical and commercial interest[1].

In many application domains (e.g., medicine or biology), comprehensive schemas resulting from collaborative initiatives are made available. For instance, Systematized Nomenclature of Medicine (SNOMED) is an ontological schema containing more than 400,000 concept names covering various areas such as anatomy, diseases, medication, and even geographic locations. Such well-established schemas are often associated with reliable data that have been carefully collected, cleansed, and verified, thus providing reference ontology-based data management systems (DMSs) in different application domains. A good practice is therefore to build on the efforts made to design reference DMSs whenever we have to develop our own DMS with[5].

II. DL-LITE

As usual in DLs, *DL-Lite* allows for denoting binary relations between objects. *DL-Lite* concepts are defined as follows:

$$B ::= A \mid \exists R \mid \exists R^{-}$$
$$C ::= B \mid \neg B \mid C1 \sqcap C2$$

where *A* denotes an atomic concept and *R* denotes an (atomic) role; B denotes a *basic concept* that can be either an atomic concept, a concept of the form $\exists R$, specifically, the standard DL construct of unqualified existential quantification on roles, or a concept of the form $\exists R^{-}$, which involves an *inverse role*. *C* (possibly with subscript) denotes a (common) concept. Note that thwy uses reversal of basic concepts only, and we do not allow for disjunction.

A *DL-Lite* knowledge base (KB) is constituted by two apparatus: a TBox used to represent intensional knowledge, and an ABox, used to represent extensional information. *DL-Lite TBox* assertions are of the form:

$B \in C$ *inclusion assertions*
(funct R), (funct R^{-}) *functionality assertions*

An inclusion declaration expresses that a basic concept is subsumed by a common concept, while a functionality assertion expresses the (inclusive) functionality of a role, or of the inverse of a role.

As for the ABox, *DL-Lite* allows for assertions of the form:
B(*a*), R(*a*, *b*) *membership assertions*

where a and b are constants. These assertions utter respectively that the object denoted by *a* is an instance of the basic concept *B*, and that the pair of objects denoted by (a, b) is an instance of the role R.

Although *DL-Lite* is fairly simple from the language point of vision, it allows for querying the extensional knowledge of a KB in a much extra authoritative way than common DLs, in which only membership to a concept or to a role can be asked. *E*xpressly, *DL-Lite* allows for using conjunctive queries of random complexity. A conjunctive query (CQ) *q* over a knowledge base *K* is an look of the form:

$$q(\vec{x}) \leftarrow \exists \vec{y}.conj(\vec{x}, \vec{y})$$

where $\vec{x}$ are the so-called *well-known variables*, $\vec{y}$ are existentially quantified variables called the *non- well-known* variables, and *conj*($\vec{x}$, $\vec{y}$) is a conjunction of atoms of the form *B*(*z*), or *R*(*z*1, *z*2), where *B* and *R* are respectively a basic concept and a role in *K*, and *z*, *z*1, *z*2 are constants in *K* or variables in $\vec{x}$ or $\vec{y}$. Sometimes, for simplifying details, we will use the Datalog sentence structure, and write queries of the above form as *q*($\vec{x}$) ←*body*($\vec{x}$, $\vec{y}$) where the existential quantification $\exists \vec{y}$ has been ready inherent, and the symbol "," is used for conjunction in *body*($\vec{x}$, $\vec{y}$).

The semantics of *DL-Lite* is specified in terms of interpretations over a permanent endless *domain* Δ. They assume to have one constant for each object, denoting accurately that object. In other terms, they have *standard names* [15], and they will not differentiate among the alphabet of constants and Δ.

An *interpretation* $I = (\Delta, .^I)$ consists of a initial order arrangement over Δ with an *interpretation function* $.^I$ such that:

$A^I \sqsubseteq \Delta$ $R^I \sqsubseteq \Delta \times \Delta$
$(\neg B)^I = \Delta \setminus B^I$ $(\exists R)^I = \{c | \exists c'. (c, c') \in R^I\}$
$(C_1 \sqcap C_2)^I = C_1^I \cap C_2^I$ $(\exists R^{-})^I = \{c | \exists c'. (c, c') \in R^I\}$

An interpretation *I* is a *model* of an inclusion assertion $B \sqsubseteq C$ if and only if $B^I \sqsubseteq C^I$; *I* is a model of a functionality assertion (funct *R*) if $(c, c') \in R^I \wedge (c, c'') \in R \supset c' = c''$, similarly for (funct R^{-}); *I* is a form of a membership assertion *B*(*a*) (resp. *R*(a, *b*)) if $a \in BI$ (resp. (a, *b*) $\in R^I$). A *model of a KB K* is an interpretation *I* that is a model of all the assertions in *K*. A KB is *satisfiable* if it has at least one model. A KB *K sensibly implies* an assertion α if all the models of *K* are also models of . A query *q*($\vec{x}$)$\exists \leftarrow \vec{y}$,*conj* ($\vec{x}$, $\vec{y}$) is interpreted in an interpretation *I* as the set q^I of tuples $\sim c \in \Delta \times \ldots \times \Delta$ such that when replace with the variables $\vec{x}$ with the constants ~*c*, the method $\exists \vec{y}$.*conj* ($\vec{x}$, $\vec{y}$) evaluates to true in *I*.

Ever since *DL-Lite* deals with conjunctive queries, the vital logic services that are of interest are:

- *query answering*: known a query *q* with illustrious variables $\vec{x}$ and a KB *K*, return the set *ans*(*q*;*K*) of tuples ~*c* of constants of *K* such that in each model *I* of *K* we have $\sim c \in q^I$. Note that this job generalizes *instance checking* in DLs, i.e., inspection whether a given object is an example of a specified concept in each model of the knowledge base.
- *query containment*: specified two queries q_1 and q_2 and a KB *K*, validate whether in every model *I* of *K* $q_1^I \sqsubseteq q_2^I$. Note That this job generalizes *logical implication* of inclusion assertions in DLs.
- *KB satisfiability*: verify whether a KB is satisfiable.

Example 1 Let the infinitesimal concepts *Professor* and *Student*, the roles *TeachesTo* and *HasTutor*, and the following *DL-Lite* TBox *T* :

Professor $\sqsubseteq \exists$*TeachesTo* *Student* $\sqsubseteq \exists$*HasTutor*
$\exists$*TeachesTo*$^{-} \sqsubseteq$ *Student* $\exists$*HasTutor*$^{-} \sqsubseteq$ *Professor*
Professor $\sqsubseteq \neg$*Student* (funct *HasTutor*).

Suppose that the ABox *A* contains just the assertion (John,Mary). At last, think the query *q*(*x*) ←*TeachesTo*(*x*,*y*), *HasTutor*(*y*, *z*), asking for professors that teach to students that have a tutor.

Even though prepared with higher logic services, at initial sight *DL-Lite* might seem rather feeble in modeling intensional knowledge, and therefore of partial use in practice. Although the ease of its language and the specific form of inclusion assertions acceptable, *DL-Lite* is capable to capture the major notions (though not all, obviously) of both ontologies, and of intangible modeling formalisms used in databases and software engineering. In particular, *DL-Lite* assertions allow to specify *ISA*, e.g., stating that concept *A*1 is subsumed by concept *A*2, using $A1 \sqsubseteq A2$; *disjointness*, e.g., between concepts *A*1 and *A*2, using $A1 \sqsubseteq \neg A2$; *role-typing*, e.g., stating that the first (resp., second) component of the

relation R is an instance of $A1$ (resp., $A2$), using $\exists R \sqsubseteq A1$ (resp., $\exists R^{-} \sqsubseteq A2$); *participation constraints*, e.g., stating that all instances of concept A participate to the relation R as the first (resp., second) component, using $A \sqsubseteq \exists R$ (resp., $A \sqsubseteq \exists R^{-}$); *non-participation constraints*, using $A \sqsubseteq \neg \exists R$ and $A \sqsubseteq \neg \exists R^{-}$; *functionality restrictions* on relations, using (funct R) and (funct R^{-}). Notice that DL-Lite is a firm subset of OWL Lite, the fewer expressive sublanguage of OWL, which presents various constructs (e.g., some kinds of role limits) that are non expressible in DL-Lite, and that make logic in OWL Lite non- well-mannered in general.[14]

III. MODEL DEVELOPMENT

The current trend for building an ontology-based data management system is to capitalize on efforts made to design a preexisting well established DMS. The method amount's to extracting from the reference DMS a piece of schema relevant to the new application needs a module, possibly personalizing it with extra-constraints with respect to the application under construction, and then managing a dataset using the resulting schema.

3.1 Proposed System

Here, we extend the existing definitions of modules and we introduce novel properties of robustness that provide means for checking easily that a robust module-based DMS evolves safely with respect to both the schema and the data of the reference DMS We carry out our investigations in the splitting of documents into paragraphs instead of ontological language, like RDFS, OWL, and OWL2 from W3C. Notably, we focus on the splitting of paragraphs, and the extensions of file for comparison purpose for efficiently managing large datasets.

3.2Advantages

1. This is very useful to maintain data.
2. Redundancy is avoided.
3. For execution it requires less time so increases system efficiency.

3.3 Scope

Our main Aim is to restore the database for user satisfaction Data handling we introduce novel properties of robustness that provide means for checking easily that a robust module-based DMS evolves safely with respect to both the schema and the data of the reference DMS. We carry out our investigations in the splitting of document into paragraphs and store the references of each text file instead of signature logics which underlie modern ontology languages, like RDFS, OWL, and OWL2 from W3C.

3.4 Data Flow Diagram

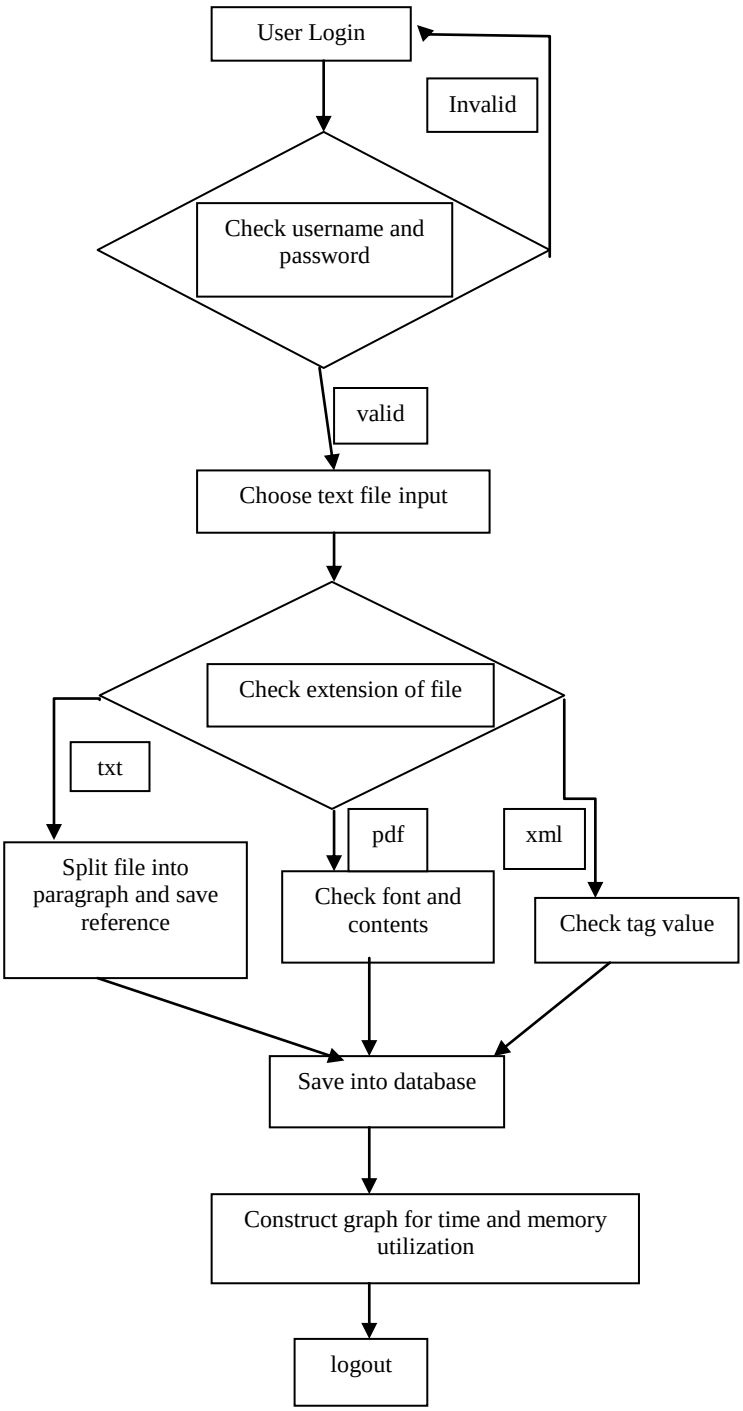

FIGURE 3.1 Data Flow Diagram

3.2.3 File Similarity

Algorithm for file similarity is as following:

1. Select the particular file which you want to save.
2. Check the extension of the File
3.If extension is txt:
i. Accept the file path.

ii. Divide the file paragraph wise.
iii. Compare each paragraph with the database.
iv. If paragraph already present then return the reference instead of saving it
v. If no similar paragraph saved in folder save that paragraph with reference no.
vi. Finally return all reference and save in database.

3. If extension is pdf:
i. Accept the file path.
ii. Read the file.
iii. Compare the file with database file.
iv. Check for each and every contents and font.
v. If equal return the reference.
vi. Else save in database.
vii. Finally return all reference and save in database.

3. If extension is XML:
i. Accept the file path.
ii. Read the file.
iii. Compare the file with database file.
iv. Check with file whether it is similar or identical
v. If equal return the reference.
vi. Else save in database.
vii. Finally return all reference and save

IV. RESULT

Proposed system is tested on text, pdf and xml dataset which is kept in a folder. By using this technique on the text document it is split into number of paragraphs and stored into a folder with unique reference number. If same paragraph will appear in other text file then it is not stored into a file and it will just provide the reference number of that paragraph into database.

FIGURE 4.1 Graph of Memory Utilization for Module Extractor and File Similarity.

FIGURE 4.2 Graph of Time Needed for Execution for Module Extractor and File Similarity.

CONCLUSIONS

The modules introduced in existing paper generalize both the modules obtained by extracting a subset of a schema with respect to selected relations or by forgetting about relations. In addition, in contrast with existing work, we have considered the problem of safe personalization of modules built from an existing reference DMS. This raises new issues to check easily that a module-based DMS evolves independently but coherently with respect to the reference DMS from which it has been built. We have developed file similarity technique where text data is split into number of paragraphs these paragraphs are stored in a folder with unique reference number. Whenever next time same paragraph will occur in the file it will just provide the same reference number to that paragraph which is already stored in folder. An extensive analysis has been carried out on the performance of our technique on dataset used for existing system. Our technique offers better performance with less time even when the module extractor need first OWL file which is generated manually by using protégé software which take more time to execution.

REFERENCES

[1] https://en.wikipedia.org/wiki/Web_Ontology_Language
[2] http://www.obitko.com/tutorials/ontologies-semantic-web/ontologies.html
[3] Shweta B.Barshe, D.K.Chitre “Agricuture System based on OntologyAgroSearch”, IJETAE, vol. 2, no. 8, 2012
[4] A. Bonnaccorsi, “On the Relationship between Firm Size and Export Intensity,” Journal of International Business Studies, XXIII (4), 1992, pp. 605-635.
[5] Francois Goasdou, and Marie-Christine Rousset, “Robust Module Based Data Management”, Institute of Electricals and Electronic Engineering , vol. 25, no. 3, 2013.
[6] R. Caves, Multinational Enterprise and Economic Analysis, Cambridge University Press, Cambridge, 1982.
[7] M. Clerc, “The Swarm and the Queen: Towards a Deterministic and Adaptive Particle Swarm Optimization,” In Proceedings of the IEEE Congress on Evolutionary Computation (CEC), 1999, pp. 1951-1957.
[8] H.H. Crokell, “Specialization and International Competitiveness,” in Managing the Multinational Subsidiary,

H. Etemad and L. S, Sulude (eds.), Croom-Helm, London, 1986.

[9] K. Deb, S. Agrawal, A. Pratab, T. Meyarivan, "A Fast Elitist Non-dominated Sorting Genetic Algorithms for Multiobjective Optimization: NSGA II," KanGAL report 200001, Indian Institute of Technology, Kanpur, India, 2000.

[10] J. Geralds, "Sega Ends Production of Dreamcast," vnunet.com, para. 2, Jan. 31, 2001.

[11] S. Abiteboul and O. Duschka. "Complexity of answering queries using materialized views". In *Proc. of PODS'98*, pages 254–265.

[12] S. Abiteboul, R. Hull, and V. Vianu. "*Foundations of Databases"*. Addison Wesley Publ. Co., 1995.

[13] M. Arenas, L. E. Bertossi, and J. Chomicki. "Consistent query answers in inconsistent databases". In *Proc. of PODS'99*, pages 68–79.

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

[15] A. Borgida and R. J. Brachman. "Conceptual modeling with description logics. In Baader et al". chapter 10, pages 349–372.

[16] A. Borgida, R. J. Brachman, D. L. McGuinness, and L. A. Resnick. "CLASSIC: A structural data model for objects". In *Proc. of ACM SIGMOD*, 1989, pages 59–67.

[17] L. Bravo and L. Bertossi. "Logic programming for consistently querying data integration systems". In *Proc. of IJCAI 2003*, pages 10–15.

[18] A. Cal`ı, D. Lembo, and R. Rosati, "On the decidability and complexity of query answering over inconsistent and incomplete databases". In *Proc. of PODS 2003*, pages 260–271.

[19] D. Calvanese. Reasoning with inclusion axioms in description logics: Algorithms and complexity. In *Proc. of ECAI'96*, pages 303–307. John Wiley & Sons.

[20] D. Calvanese, G. De Giacomo, D. Lembo, M. Lenzerini, and R. Rosati. "What to ask to a peer: Ontology-based query reformulation". In *Proc. of KR 2004*, pages 469–478.

[21] D. Calvanese, G. De Giacomo, and M. Lenzerini. "Answering queries using views over description logics knowledge bases". In *Proc. of AAAI 2000*, pages 386–391.