A Data-Driven Approach toward Building Dynamic Ontology

Dhomas Hatta Fudholi[1,2], Wenny Rahayu[1], Eric Pardede[1], and Hendrik[2]

[1] Department of Computer Science and Computer Engineering, La Trobe University, Australia
dfudholi@students.latrobe.edu.au,
{w.rahayu,e.pardede}@latrobe.edu.au
[2] Department of Informatics, Universitas Islam Indonesia, Indonesia
{hatta.fudholi,hendrik}@fti.uii.ac.id

Abstract. Ontology has been emerged as a powerful way to share common understanding, due to its ability to chain limitless amount of knowledge. In most cases, groups of domain expert design and standardize ontology model. Unfortunately, in some cases, domain experts are not yet available to develop an ontology. In this paper, we extend the possibilities of creating a shareable knowledge conceptualization terminology in uncommon domain knowledge where a standardized ontology developed by groups of experts is not yet available.

Our aim is to capture knowledge and behaviour which is represented by data. We propose a model of automatic data-driven dynamic ontology creation. The created ontology model can be used as a standard to create the whole populated ontology in different remote locations in order to perform data exchange more seamlessly. The dynamic ontology has a feature of a real-time propagation from the change in the data source structure. A novel *delta* script is developed as the base of propagation. In order to complete the model, we also present an information of application support in the form of Jena API mapping for propagation implementation.

Keywords : data-driven, dynamic ontology, propagation.

1 Introduction

Ontology has been used as a mechanism to share common knowledge and understanding [1]. Groups of domain experts have used ontology to represent certain knowledge into semantic structure of information, for instance, in medical health domain. However, there are a large amount of domain knowledge is still untouched by domain experts.

To save time, reduce manual work and facilitate communities who may not have the technical understanding in constructing an ontology, a few researchers have proposed some approaches to develop ontology from underlying data. Garcia et al. and Bohring et al. have done similar research in creating the concept of XML (eXtensible Markup Language) to OWL (Web Ontology Language) mapping, which can be found in [2] and [3]. Both research implement XSD (XML Schema) as the source of creating terminological ontology model. The XSD could be extracted from XML data. XSLT (XML Stylesheet Language Transformation) is used as the tool to translate XML-based information into ontology knowledge representation. Bohring et al. also use XSLT to populate the terminological ontology model. Zhou et al. [4] had research in automatic ontology creation from relational database (RDB). They create seven rules to map the database structure into the terminological conceptualization in ontology, and then populate the records as the ontology instance.

Data source knowledge can change very often. A method to propagate the ontology can be used to keep the ontology dynamic and up-to-date. Sari et al. in [5] propose a propagation model to update sub-ontology of SNOMED CT. This methodology propagates sub-ontology extracted from the main SNOMED CT ontology based on the change log in the SNOMED CT ontology.

Collective knowledge from communities can be extracted to form a formal standard of representation. When it becomes standard, any following knowledge representation could adopt the same terminology. It enables seamless knowledge sharing. The main aim of this research is to create a model for dynamic ontology, derived from a dynamic data source. The dynamic ontology is maintained through a systematic propagation method triggered by changes in the data source structure. The propagation method uses a *delta* script that contains the difference of the previous and the current data structure. When the remote propagation is needed, the use of *delta* script can save the resource rather than sending the whole new data source or the whole new ontology. The novel concept of *delta* script is also proposed in this paper.

The paper is organized as follows. Section 1 is the introduction, capturing the backgrounds, motivations and aims of the research. Section 2 states the related and supporting works for the research. Section 3 elaborates the whole concept model of the data-driven dynamic ontology. Section 4 focuses on the propagation features, starts from the different types of data changes, the *delta* script construction and the propagation process. Section 5 covers the application support for the propagation process in term of *delta* script and programming framework mapping. It also elaborates the case study as to show the implementation.

2 Related Work

The automation of data-driven ontology creation can be very useful for community to share their knowledge in the form of ontology. This also addresses the limitation of technical capability in ontology building. In general, there are two kinds of data sources that are used widely as a data repository, a structured database and semi-structured XML. A number of researchers have explored the techniques to support ontology creation from these two data sources.

Garcia et al. proposes the XSD2OWL. XSD2OWL contains packages based on an XSL (XML Stylesheet Language) that performs a partial mapping from XML Schema to OWL [2]. Even though it consists only of partial mapping that transform XML Schema to OWL, XSD2OWL covers most of ontology semantic structure. The full mapping table of XSD2OWL is described in [2]. To perform a complete XML-based ontology creation, XSD2OWL cannot be used as a single tool. It needs to be collaborated with XSD extraction tools to extract XSD from its XML data source, e.g. Trang [6] and oXygen XML Editor [7].

Bohring et al. [3] creates a similar mapping concept to translate extracted XSD from XML into OWL ontology. This work explicitly states the way to populate the ontology using the XSLT and the way to perform a mapping of domain and range in ontology properties.

An approach of semi-automatic ontology creation from RDB schema is introduced by Zhou et al. in [4]. The concept is originally created to overcome time consuming and tedious work in creating hand-built ontology. Zhou et al. give an extension in their concept using WordNet to handle similarities in word term. Zhou et al. made seven rules to map the RDB into ontology. All rules can be seen in [4].

Table 1 summarizes works in ontology mapping from XML and RDB. For instance, class or concept in ontology is generated from a *complexType* element in XML and from table or fixed instance value in RDB.

Table 1. General ontology mapping from XML and Relational Database based on Garcia et al. [2], Bohring et al. [3] and Zhou et al. [4] works

Ontology	XML	Relational Database
Class/Concept	*complexType* element	table, fixed instance value
ObjectProperty	*complexType* element	table relation
DatatypeProperty	*simpleType* element, attribute	column
Cardinality (Max, Min)	occurrence (maxOccurs, minOccurs)	constrain (NOT NULL, primary key)
Property Domain and Range	element, XSD datatype	table relation, column, column data type

3 System Design and Concept

The whole system scenario for a data-driven dynamic terminological ontology development can be seen in Fig. 1. The data source is dynamic, shown by the dashed arrow from the old data to the new data. The data source could be an XML data source or RDB data source. The data source consists of data records and data schema. Basically, there are two main parts of the whole concept scenario, the initial ontology creation process and dynamic ontology propagation process.

The initial ontology creation process is depicted using a solid line in Fig. 1. The aim of this process is to create a base dynamic ontology as the very first ontology model to be shared. Schema to ontology translator performs the creation by mapping the data schema inside the data source into ontology. The dynamic ontology propagation process handles the update of dynamic shared ontology and it will be updated directly when there is a change in the data source.

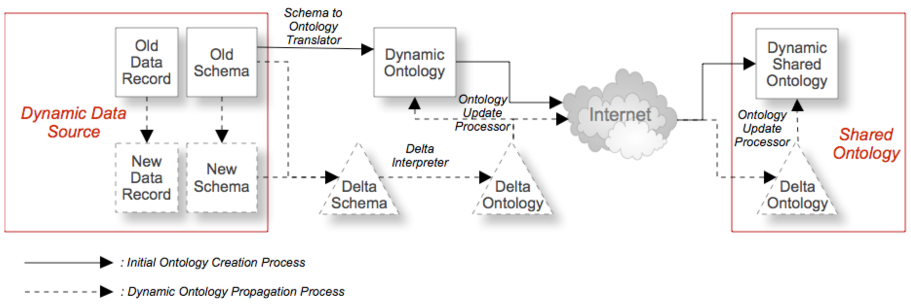


Fig. 1. Conceptual model for creating data-driven dynamic shared ontology to share community knowledge

The dashed lines in Fig. 1. represents the propagation process. The difference of the current and old data schema is stored as *delta schema* script. Since there would be different representation in XML-based schema and RDB schema representation, *delta ontology* is derived from the *delta schema* to create a common representation, which maps directly to the ontology changes. There are two ways of retrieving the current terminological ontology from the shared ontology: (i) by requesting directly the current ontology or (ii) by requesting the *delta ontology* script followed by propagating the ontology locally using propagation application. In addition, there will be only one application needed to use the *delta ontology* script when updating the ontology, since it will be data source type independent. The dynamic ontology propagation process and related tools will be elaborated in Section 4.

4 Dynamic Ontology Propagation

4.1 Changes in Data

Propagation is proposed as the solution to update the common terminological ontology based on the dynamic changes in the data source structure. The structures changes basically consist of *delete*, *insert, rename* and *move*. For XML-based data, all of changes operation could happen in every element and attribute. The *move* change operation of element is the changes in tree structure position of parent and child. RDB's table and column could also have the same change operation; however the *move* operation might be happening only in table column.

4.2 *Delta* Script

The differences of data source structure are gathered in a *delta* script. The purpose of the *delta* script is to patch or upgrade the dynamic ontology. The use of *delta* script can be useful when the source or the original version is not present in the same location and the ontology needs to be updated without sending the original file, which can be very big. Cobena et al. in [8] give four main benefits for using DIFF (difference) method as change detection: version and querying the past, learning about changes, monitoring changes, and indexing. In addition, Cobena et.al. in [9] proposes about a set of important criterias for a good *delta* script. Those aspects are *Completeness*, *Minimality*, *Performance and Complexity*, *"Move" Operation*, and *Semantics*. All aspects mentioned are considered and applied in the proposed *delta* script.

4.3 *Delta Schema* Script

Delta schema script consists of the difference between the current data schema with the previous version of the schema. It lists all of the difference structure from edit operations. The list includes *delete*, *insert*, *rename*, and *move* list.

Definition DS-1. $\Delta S \equiv \langle D, I, R, M \rangle$. *Delta schema* script comprise of 4 set of list, which are *delete (D)*, *insert (I)*, *rename (R)* and *move (M)*.

The list is proposed to keep up with the *completeness* and *minimality* of the operation. Even though command DELETE and INSERT (we use the all capital words to describe the programming command and to differentiate them from the *delta* script's list and their general common usage words) are the primitive operation and the *delete* and *insert* list could be used to represent *rename* and *move*, but the list will keep the *minimality* aspect and can be directly performed to some programming framework. Therefore, that list can potentially reduce the complexity and yet it is *complete*. The sequence of listed difference in the *delta schema* script should be as mentioned in the **Definition DS-1** to avoid the possible name duplication of the new inserted data and the need to state all inserted and renamed component in order to be the target of moved component. Therefore the sequence should be as follow:

Delete(D) → Insert(I) → Rename(R) → Move(M)

To maintain the semantic information of the data, the following rules need to be applied in *delta schema* script's list:

– **DS-Rule 1** - For all type list: There should be an initial sign to differentiate *complexType* element, *simpleType* element and attribute name in XML, also table and column name in RDB. In XML, sign "(c)" can be used to indicate *complexType* element. As for the attribute, sign "@" can be used as the initial. In RDB, sign "(t)" could be used to indicate table. To simplify the representation of each component, as an example, it could be written as follows:

  <initial><s><component name>

  where <s> is separator sign.

- **DS-Rule 2** - For *insert* list: The information of data type, constrains and location/path (if becomes the child or the part of other component) of the inserted component should be stated clearly. As an example, it could be written as follows:

  <initial><s><component name><s><datatype><s><constrain>
  <initial><s><component's parent>/<initial><s><new component name>

  Since the RDB mapping has some additional information to add when there exist relations in two tables as foreign key. Those relations will create an inverse object property between two concepts. Additional information in the insert list of RDB should be added, such as:

  <initial><s><component name> ←→ *<initial><s><new component name>*

- **DS-Rule 3** - For *rename* list: The list should contain the path or location of the renamed component along with the new component name. As an example, it could be written as follows:

  <initial><s><component's parent>/<initial><s><component name> →
  <initial><s><component's parent>/<initial><s><new component name>

- **DS-Rule 4** - For *move* list: The list should contain the path or location of the moved component along with the new component's parent name. For the new location path, information about data type and constrains need to be incuded to maintain the whole semantic information. As an example, it could be written as follows:

  <initial><s><component's parent>/<initial><s><component name> →
  <initial><s><component's new parent>/<initial><s><component name><s><datatype><s><constrain>

4.4 *Delta Ontology* Script

Delta ontology script consists of the list of ontology structure change, which is derived from the *delta schema* script based on the mapping in Table 1. There are three types of list in *delta ontology* script; *delete, insert* and *rename* list respectively. The move operation of column in RDB will affect in changing domain and range of property in ontology. Since there is no move operation for domain and range in the ontology, it will trigger the insert and delete operation instead. The move element operation in XML will affect in the ontology restriction. It will not move the restriction to other ontology class but it will trigger a delete and insert operation of the restriction. These two conditions are some reasons why the *move* list is absence in *delta ontology*. The following is the proposed syntax in writing *delta ontology* list:

- Delete List :

- For Class/Concept → *c(name)*
- For ObjectProperty → *op(name)*
- For DatatypeProperty → *dp(name)*

– Insert List :

- For Class/Concept → *c(name, superClass name)*
- For ObjectProperty → *op(name, domain, range, minC**, maxC**)*
- For DatatypeProperty → *dp(name, domain, range, minC**, maxC**)*
- For ObjectProperty domain and range change → *opdr(property name, domain, range, minC**, maxC**)*
- For DataTypeProperty domain and range change → *dpdr(property name, domain, range, minC**, maxC**)*

***minC* and *maxC* is an optional minimum cardinality and maximum cardinality information.

– Rename List :

- For Class/Concept → *c(previous name, current name)*
- For ObjectProperty → *op(previous name, current name)*
- For DatatypeProperty → *dp(previous name, current name)*

When transformed to *delta ontology,* a first character "C", "op" and "dp" is used to stated Class, ObjectProperty and DatatypeProperty respectively. The following example is about the translation process in RDB. The sample from XML is stated along the case study in Section 5.

Example. RDB. There is an additional column created named "author" in "book" table. The data type of "author" is string. The "author" column has NOT NULL constrain. This change could be listed in *delta schema* and *delta ontology* as follows:

```
DELTA SCHEMA                          DELTA ONTOLOGY

INSERT                                INSERT
  author | string | min-1               dp(dpauthor, Cbook, string, 1)
  (t)book/author
```

5 Application Support

To demonstrate the application support, especially in the dynamic ontology propagation concept using *delta* script, the OWL propagation mapping into Apache Jena™ [10] API within a Semantic Web application is developed. The application is used to update OWL ontology from changed data structure. The pattern to apply the Jena API in the application is shown in Fig. 2.

There are three main parts of the Jena programming command block that is applied. **(1)**. Call/open the base OWL ontology model. First, `createOntologyModel()` method is used to create a new ontology model which will be processed in-memory and it is expressed in the default ontology language (OWL). Then `read()` method will call/open OWL file path. **(2)**. Apply and execute Jena API for the propagation. This part can be filled with any method needed to do the propagation. The types of method are shown in Table 2. **(3)**. Write output of the updated OWL ontology. The `write()` method is used to perform this operation.

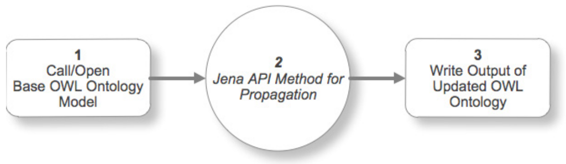


Fig. 2. Jena API Pattern for Propagation Implementation

Table 2. Delta ontology mapping to Jena API

Process	Delta Ontology	Jena API
DELETE	- Class/Concept - DatatypeProperty - ObjectProperty - MinCardinality - MaxCardinality	- `getOntClass()` then `remove()` - `getDatatypeProperty()` then `remove()` - `getObjectProperty()` then `remove()` - `listRestrictions()` then `remove()` - `listRestrictions()` then `remove()`
INSERT	- Class/Concept - DatatypeProperty - ObjectProperty - Set Property Domain - Set Property Range - MinCardinality - MaxCardinality	- `createClass()` - `createDatatypeProperty()` - `createObjectProperty()` - `setDomain()` - `setRange()` - `createMinCardinalityRestriction()` - `createMaxCardinalityRestriction()`
RENAME	Class/Concept or DatatypeProperty or ObjectProperty	`renameResource()`

As a study case, a section of the version 2012 of PubMed/MEDLINE [11] citation XML sample[1] is used. The PubMed/MEDLINE citation XML for the case study and the sample of change in the data can be seen in Table 3. Afterwards, the *delta schema* and *delta ontology* script could be generated as mentioned in Table 4. Fig. 3 depicts a JSP page for the ontology propagation built using Jena API. The input ontology model path, the propagation process step and the output file path can be seen in Fig. 3. Finally, Fig. 4. depicts the Protégé visualization for the propagated ontology based on the change stated in Table 3. It consists of the Class, DatatypeProperty, ObjectProperty and Restriction in the ontology. Due to the limitation of the page, it shows the Restriction for "PubDate" only. From the result, it can be said that the *delta* is complete and holds enough semantic information.

[1] Downloaded from
`http://www.nlm.nih.gov/databases/dtd/medsamp2012.xml`

Table 3. Data sample changes

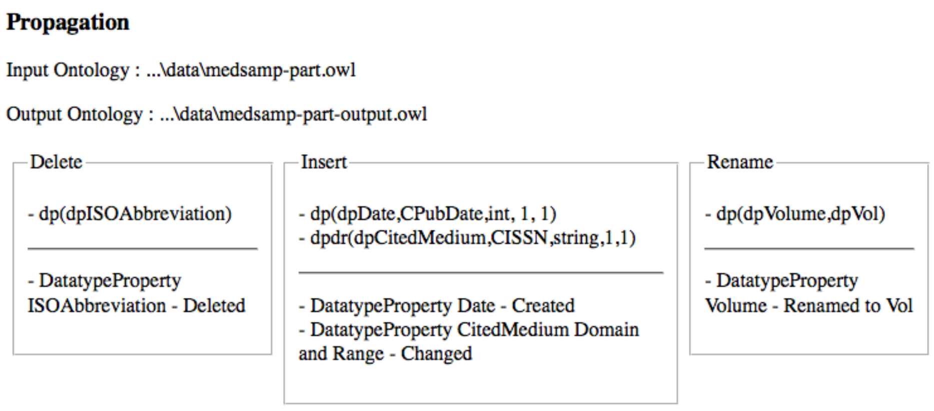
Previous Data	Current Data

`<Journal>`
`<ISSN IssnType="Print">0950-382X</ISSN>`
` <JournalIssue CitedMedium="Print">`
`  <Volume>34</Volume>`
`  <Issue>1</Issue>`
`  <PubDate>`
`   <Year>1999</Year>`
`   <Month>Oct</Month>`
`  </PubDate>`
` </JournalIssue>`
` <Title>Molecular microbiology</Title>`
` <ISOAbbreviation>M.M./ISOAbbreviation>`
`</Journal>`
	`<Journal>`
	`<ISSN IssnType="Print"`
	`CitedMedium="Print"``>0950-382X</ISSN>`
	` <JournalIssue>`
	`  `**`<Vol>34</Vol>`**
	`  <Issue>1</Issue>`
	`  <PubDate>`
	`   <Year>1999</Year>`
	`   <Month>Oct</Month>`
	`   `**`<Date>4</Date>`**
	`  </PubDate>`
	` </JournalIssue>`
	` <Title>Molecular microbiology</Title>`
	`</Journal>`

Table 4. *Delta* script from sample data changes

Delta Schema	*Delta Ontology*

`DELETE`
`  ISOAbbreviation`
`INSERT`
`  Date │ int │ min-1 │ max-1`
`  (c)PubDate/Date`
`RENAME`
`  (c)JournalIssue/Volume → (c)JournalIssue/Vol`
`MOVE`
`  (c)JournalIssue/@CitedMedium →`
`(c)ISSN/@CitedMedium │ string │ min-1 │ max-1`
	`DELETE`
	`  dp(dpISOAbbreviation)`
	`INSERT`
	`  dp(dpDate,CPubDate,int, 1, 1)`
	`dpdr(dpCitedMedium,CISSN,string,`
	`1,1)`
	`RENAME`
	`  dp(dpVolume,dpVol)`



Propagation

Input Ontology : ...\data\medsamp-part.owl

Output Ontology : ...\data\medsamp-part-output.owl

Delete

- dp(dpISOAbbreviation)

- DatatypeProperty ISOAbbreviation - Deleted

Insert

- dp(dpDate,CPubDate,int, 1, 1)
- dpdr(dpCitedMedium,CISSN,string,1,1)

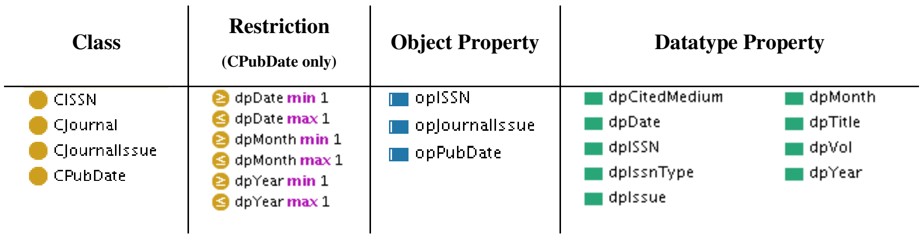
- DatatypeProperty Date - Created
- DatatypeProperty CitedMedium Domain and Range - Changed

Rename

- dp(dpVolume,dpVol)

- DatatypeProperty Volume - Renamed to Vol

Fig. 3. JSP page for ontology propagation built using Jena API



Class	Restriction (CPubDate only)	Object Property	Datatype Property	
● CISSN	≥ dpDate **min** 1	▮ opISSN	🟩 dpCitedMedium	🟩 dpMonth
● CJournal	≤ dpDate **max** 1	▮ opJournalIssue	🟩 dpDate	🟩 dpTitle
● CJournalIssue	≥ dpMonth **min** 1	▮ opPubDate	🟩 dpISSN	🟩 dpVol
● CPubDate	≤ dpMonth **max** 1		🟩 dpIssnType	🟩 dpYear
	≥ dpYear **min** 1		🟩 dpIssue	
	≤ dpYear **max** 1			

Fig. 4. Protégé visualization of the updated ontology

6 Conclusion

The need to create a common conceptualization from dynamic knowledge has motivated us to create a model for a data-driven dynamic ontology with propagation support. The propagation process updates the base ontology based on the underlying data structure changes. The use of *delta* script gives an advantage in updating remote ontology by sending the minimum source that can provide complete updates. A simple, minimized yet complete *delta* script is designed, and the mapping of the *delta* script list into a Jena API method within a Semantic Web application is demonstrated.

References

1. Calegari, S., Ciucci, D.: Integrating Fuzzy Logic In Ontologies. In: ICEIS (2006)
2. García, R.: A Semantic Web Approach to Digital Rights Management. PhD Thesis. Universitat Pompeu Fabra, Barcelona, Spain (2006)
3. Bohring, H., Auer, S.: Mapping XML to OWL Ontologies. In: Leipziger Informatik Tage. LNI, vol. 72 (2005)
4. Zhou, X., Xu, G., Liu, L.: An Approach for Ontology Construction Based on Relational Database. International Journal of Research and Reviews in Artificial Intelligence 1(1) (2011)
5. Sari, A.K., Rahayu, W., Bhatt, M.: An Approach For Sub-Ontology Evolution In A Distributed Health Care Enterprise. Information Systems Journal (2012)
6. Thai Open Source Software Center Ltd: Trang, Multi-format schema converter based on RELAX NG, `http://www.thaiopensource.com/relaxng/trang.html` (accessed November 20, 2012)
7. SyncRO Soft SRL, `http://oxygenxml.com` (accessed November 20, 2012)
8. Cobéna, G., Abiteboul, S., Marian, A.: Detecting Changes in XML Documents. In: Proceedings of the 18th International Conference of Data Engineering (2002)
9. Cobéna, G., Abdessalem, T., Hinnach, Y.: A comparative study of XML diff tools (2004), `http://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.91.5366`
10. The Apache Software Foundation: Apache Jena, `http://jena.apache.org/` (accessed November 20, 2012)
11. U.S. National Library of Medicine: MEDLINE®/PubMed® Resources Guide, `http://www.nlm.nih.gov/bsd/pmresources.html` (accessed November 20, 2012)