Integration Languages for Data-Driven Approaches to Ontology Population and Maintenance – Extended Abstract –

Eduardo Torres Schumann¹ and Klaus U. Schulz²

¹ SfS, University of Tübingen, Germany
² CIS, University of Munich (LMU), Germany
{torres,schulz}@cis.uni-muenchen.de

1 Motivation

A key application of ontologies is providing semantic interoperability between information sources in order to achieve an integrated access to heterogeneous data. Unfortunately, the process of ontology development requires considerable human effort and constitutes a bottleneck on the way to the semantic web. Moreover, ontologies have to be refined and maintained regularly in an iterative, data-driven process called the ontology learning cycle [1]. If large amounts of existent data are intended to be integrated through ontologies, approaches to ontology population and maintenance need to work in a data-driven, automatized way but at the same time enable efficient human supervision.

Here, we consider both extending and maintaining ontologies by using already existent data. Data could be e.g. a manually compiled list of names given as a table, or some pertinent results of an information extraction process given as an XML file. The core of our method consists in developing an extension of the representation language of the target ontology, a so-called integration language, used to define special templates. Such a template is used to generate expressions of the ontology language for the incoming data, specifying this way how data is mapped and integrated in the ontology. In order to cope with different data formats, the integration language includes various devices for referring to data parts addressed by the template. Generated entries are then aligned with the existing ontology in order to reveal semantic conflicts and identify redundant entries. For inspection, alignment results are presented in a special interface to the ontology maintainer, which can select some of the generated entries and load them to the ontology.

Human supervision of the ontology population process is facilitated by our approach in the following way. Using a template and aligning generated entries is looking into the ontology under a specific view and obtaining an immediate feedback about the state of the ontology in relationship to certain data. Such a focused view provides orientation to the maintainer even when she lacks an exhaustive knowledge of the ontology or the ontology is very large, and constitutes a good starting point for further interactive ontology editing operations.

We have developed an integration language for the EFGT net formalism [2], an ontology format specially designed for encoding encyclopedic and common purpose knowledge. A tool for integration of data into an existing EFGT net has been implemented and successfully applied for substantially extending a core ontology. By using already existent lists and data extracted specially for this purpose, we were able to incorporate to the net about $10^4$ new concepts representing common named entities, most of them geographical names and names of famous people.

2 EFGT Net - An Overview

The EFGT net formalism is presented in this section on an informal basis. We focus on its logical language, that will be extended to an integration language in the next
sections. A more formal definition of the data model and the motivations behind EFGT nets can be found in [2].

**Knowledge Representation and Inference with EFGT Nets.** Concepts are captured in a EFGT net by creating an entry consisting in an unique identifier for the concept, its ID string, and a concept name in one of the supported languages. The ID string determines the position of the concept in the net by means of some formal deduction mechanisms, see [3] for details. These ensure the resulting structure to be a directed acyclic graph (DAG), where nodes represent the concepts and edges binary relations between concepts.

There are two possibilities to create a new ID string for a new concept. The first way, a local introduction, is used when the new concept can be sufficiently characterized as a set or set element that narrows the meaning of a more general concept. For example, the concept “Cities” can be regarded as a specialization of “Locations” denoting a set of geographical locations whereas “Oslo” can be conceptualized as an element of the latter. This is represented in the formal language by marking the ID string of the refined concept with a special type corresponding to the kind of specialization (the kind of set or set element), as well as an index for enumeration. In the example, “Locations” could be coded as a refinement of the top node (), marking it with the type of sets of geographical entities and the index 1, i.e. \( G().1 \). Analogously, “Cities” could be coded as \( G(G().1).1 \) and “Oslo” as the (third) element of “Cities” by using the type \( g \) of geographical set elements, \( g(G(G().1).1).3 \). Available types can be arranged into E, F, G and T types and motivate the name EFGT net. Uppercase types (lowercase types) denote sets (set elements):

- **E,e** Type E denotes a set of Entities like composers whereas type e denotes a singleton entity like J. S. Bach.
- **F** Type F denotes a thematic Field like quantum physics. Since every thematic field can be regarded as a set of subfields, there is no type f.
- **G,g** As mentioned before, type G denotes Geographical sets like rivers whereas type g stands for singleton geographic instances like the Alps.
- **T,t** Finally, type T denotes a Temporal period like epochs in art, and type t denotes an individual temporal interval like September 11th.

The other way to create a new ID string is by combining two ID strings with the operator & to form a new one. This is called a concept intersection and has different interpretations depending on if the two ID strings are both sets, the interpretation then being an intersection of the sets, or a set and a set element, which is interpreted as the set “in the context of” or “from the point of view of” the singleton element. E.g. combining “Persons”, \( E().1 \), and “Science”, \( F(F().1).2 \), would yield “Persons in Science/Scientists”), \( (E().1)&(F(F().1).2) \). “European Countries” may result from joining the identifier for “Europe”, \( (g(...).1) \), and “Country”, \( G(...).2 \), to \( ((g(...).1)&(G(...).2)) \). Figure 1 summarizes the syntax of ID strings.

Additional information for each concept is recorded as attributes. Attributes are specified with a semantic type that specify which kind of information they hold (birth date of a person, the number of inhabitants of a city, a company’s webpage, etc.), a syntactic type that specifies how the information is conveyed (as a date, proper name, URL, etc.) and a language type for the language used. The list of semantic and syntactic types is open and can be extended to accommodate different requirements. Particularly, we record a linguistic representation of each concept as a list of synonyms, writing variants, flection forms, etc.
3 Integration Languages as Extensions of Ontology Representation Languages

Entry templates (expressions of the integration language) are intended to specify how data can be integrated in the ontology, i.e. how data can be semantically interpreted as concepts that relate to other concepts in the ontology. This can also be regarded as a transformation that maps data values appearing in some structural relationship to concepts standing in some ontological relationship. Ontology entries are the result of the instantiation of the template with incoming data, each entry providing a logical characterization and a name for a concept.

Extending the EFGT Net Language to an Integration Language. In the case of the EFGT net formalism, an entry template should specify how to construct ID strings that relate data to existing concepts in the net. As an example, consider the data about Switzerland's geography compiled in Fig. 2. Suppose you want to add each district in the table to a EFGT net already holding the cantons. One may want to create a set of districts for each canton and state that each district is an element of the set of districts of the corresponding canton. E.g., the concept “districts in canton Thurgau” would be a set of type \(G\) introduced under the concept “Thurgau”. In our integration language, the expression \((\text{districts } G)[\text{Thurgau}.n]\) constructs a suitable ID string for this set: it queries the net for the concept between the square brackets, takes its ID string and makes a new one as a local introduction with type \(G\) and a fresh index represented by \(n\) in the expression. The string ‘districts’ in tiny letters, a concept variable, marks the expression between the parentheses as the ID string for a new concept and is used to assign a name and possibly other attributes to this concept.

\[
\begin{array}{ccc}
\text{Canton} & \text{District} & \text{Capital} \\
\hline
\text{Thurgau} & \text{Bezirk Weinfelden} & \text{Weinfelden} \\
\text{Thurgau} & \text{Bezirk Bischofszell} & \text{Bischofszell} \\
\text{Wallis} & \text{Bezirk Brig} & \text{Brig-Glis} \\
\end{array}
\]

Fig. 2. Geographical data about Switzerland

\[
\begin{align*}
(\text{capital} & (\text{district} G[\text{districts } G[\text{Canton}.n].n] & [\text{capitals}]), \\
\text{districts.name.en.name} & = \text{“districts in canton Canton”} \\
\text{district.name.en.name} & = \text{“District”} \\
\text{capital.name.en.name} & = \text{“Capital”}
\end{align*}
\]

Fig. 3. Example of an entry template

\[
I' := ( ) | (I' \cdot N) | (I' & I') | [Q] | (\varphi I') | (\psi I' & I')
\]

\[
V := \text{Alphanumeric}^+ \\
Q := I \mid \text{Lit} \\
\text{Lit} := \text{String} \mid \text{R}
\]

Fig. 4. Extended ID string syntax.

The complete extended ID string syntax for template definition is shown in Fig. 4. The last three alternatives for \(V\) are the core of the extension, where the first of them represents a concept query and the other two are variants of the local introduction and concept intersection rules that introduce concept variables. Names for concept variables are simply alphanumeric strings, as specified by the rule for \(V\). Pos-
sible queries are the productions of $Q$, namely an ID string or a literal. A literal is an arbitrary string value or a reference (symbolized by $R$) delivering a value from data. References will be discussed in the next section.

The grammar of attribute assignments to concept variables is summarized in Fig. 5. For each variable, literals are assigned to attributes specified by their semantic, language and a syntactic type. Only templates introducing at least one concept variable with a corresponding attribute specification generate entries that can be aligned with the net. One template produces as many ontology entries as variables are introduced in the template.

4 Referring to Data for Entry Template Instantiation

Although the full paper will discuss on requirements arising from heterogeneity at the level of data values, here we only focus on methods for coping within an integration language with varying data structures. An approach would consist in normalizing the different structures to a standard one used as reference for templates. This would involve considering general data transformations, that e.g. in the case of XML can be performed by special-purpose languages like XSLT [4]. Instead of that, we examine how functional dependencies present in data can guide the development of pertinent referring mechanisms for accessing different data structures, which are used then to obtain instantiations of the templates.

Table Data. In an EFGT net entry template, the outermost concept variable is a descendant of and more specific than all the other concept variables and query concepts appearing in the template. Thus, the ordinary but not mandatory choice is to let the outermost variable range over a key column that functionally determines the values in the other columns. Then, if references to tables simply point to the different columns, each row of the table defines an instantiation of the entry template, generating a set of new entries that can be checked against the ontology.

Sometimes, one wants to skip some rows of the table or to apply different templates depending on the value of some fields. This is achieved by using if-then-else statements with conditions on references, also included in the integration language.

XML data. Functional dependencies are naturally encoded by different structural relationships in the XML data model, basically a tree with different kinds of nodes [5]. The most prominent cases are the relation of an element node with its attribute nodes, the relation with its parent and the relation with its textual content. This means that given some data, the same set of functional dependencies can be rendered in different XML representations. As a principle for defining references to XML files that capture functional dependencies, the outermost variable can be identified with some set of nodes in the XML tree acting as key node set. The references to the other, more general concepts appearing in the template are let to point to nodes that structurally depend on the key nodes, i.e. to element nodes or attribute nodes on the path from each key node to the document root. In our implementation (see Sect. 6), this is realized using XPath expressions [6]. The key node set is determined by means of an absolute XPath expression, while the other references are relative XPath expressions interpreted with respect to the paths starting at key nodes and ending at the document root. In XPath terms, relative paths are evaluated against nodes in the ancestor axis of each key node or the key node itself (ancestor-or-self axis).
5 Alignment of Generated Entries

In general, generated entries can be aligned with the ontology by examining first, whether there is another concept in the ontology with an equivalent logical characterization (logical existence), and second, whether the concept is already linguistically present in the ontology, i.e. by checking if there holds some lexical relation between the concept name and other concept names in the ontology. This is particularly relevant when there is a rich linguistic representation of the concepts in the ontology, e.g. a list of flection forms, synonyms, related adjectives, nouns, etc. conveying how the concept is expressed in natural language texts. In the case of the EFGT net language, it is enough to match the generated ID string against all ID strings in the net to decide the logical existence. Whether a concept is linguistically present in the ontology can be decided by performing a search over the attributes holding linguistic information. The cases in Fig. 6 can then be distinguished.

A generated entry can be considered a new concept to be added to the ontology when there is simply nothing indicating that it collides with another concept in the ontology. Logical clashes can be obliterated by merging the attribute representation of both concepts. This makes sense when the colliding concept name is just a variant not included in the linguistic representation of the existing concept. It may also be the case that the semantic analysis of two different concepts is too coarse to distinguish between them. A concept match is given when the generated entry is indistinguishable from another concept in the ontology. Name clashes also have two possible interpretations. It may occur that two semantically different concepts happen to be homonyms, in which case the constructed entity should be also created. The converse case is that an existing concept is modeled by the template in a different way than in the net. If one of the cases is expected to be more frequent than the other, implementing two different alignment modes for enabling or disabling homonym entries may be useful to handle conflicts of this type.

<table>
<thead>
<tr>
<th>Log. existent</th>
<th>Ling. present</th>
<th>Conflict type</th>
</tr>
</thead>
<tbody>
<tr>
<td>×</td>
<td>×</td>
<td>Potential new entry</td>
</tr>
<tr>
<td>√</td>
<td>×</td>
<td>Logical clash</td>
</tr>
<tr>
<td>√</td>
<td>√</td>
<td>Concept match</td>
</tr>
<tr>
<td>×</td>
<td>√</td>
<td>Name clash</td>
</tr>
</tbody>
</table>

Fig. 6. Alignment cases

6 Ontology Population and Inspection with the Upload Tool

We have developed a prototype called the upload tool that interprets EFGT net entry templates. It has a client-server architecture. The EFGT net resides in a RDBS backend queried by the web client, that interprets the template language and performs the alignment. Generated entries can be selected for uploading them to the database, where inference takes place and the structure of the net is rearranged for accommodating new concepts. Figure 7 shows a screenshot of the upload tool. The upper part contains the entry template, submitted as a file in a previous step. The entry template can be edited and evaluated online. The alignment results are showed in the lower part of the window as a list of entries for each template instantiation. Entries are colored red when there is a concept match and green in the other cases. Green entities can be selected in a check-box for uploading. Blue entities are entities that already have been considered in a previous template instantiation. In the case of conflicting logical representations, a warning appears together with a button for merging the attribute representation of the conflicting concepts. For handling with
name clashes, the user can decide to enable or disable the creation of homonym entries. A warning list of clashing concepts is displayed, as for “Wallendorf” in Fig. 7.

Once the net has been populated with data from a specific file, the same file can always be retried. If some entries have changed in the meanwhile, they will appear as name clashes when aligning them again with the original template. This is an easy way to track changes in the representation of data and to inspect the ontology thematically.

References