DESCRIPTION LOGIC BASED GENERATOR OF DATA CENTRIC APPLICATIONS

Pawel Kaplanski
Faculty of Electronics, Telecommunications and Informatics
Gdansk University of Technology
Gdansk, Poland
pawel.kaplanski@gmail.com

Abstract—The knowledge stored in Ontology Management Systems (OMS) that originally has the form of expressions, can be seen as a user application specification or as knowledge provided by an expert. The generator of applications discussed in this paper is defined as a program that automatically generates an application that meets a certain specification stored in OMS. It is shown that it is possible to build a user interface for data management with an algorithm that crawls over taxonomy inferred by the description logic reasoner. Finally, an example prototype of a medical application will be discussed to prove the feasibility of the proposed approach.

Keywords—description logic, software engineering, model view controller, design pattern, software architecture

I. INTRODUCTION

Description Logic (DL) can be used in a spectrum of tasks that appears in the development of data-centric software applications in natural and intuitive way. It allows to describe the application domain and application specification, database schema, database constraints as well as database contents using a common language. On the other hand, DL can be used as a language that allows domain experts to express domain-specific knowledge. This article describes a flexible application generator with self-descriptive attributes designed for domain user applications that is build on the DL framework. An example of a medical realization of a generator is presented to prove the concept.

II. DESCRIPTION LOGIC AS FOUNDATION FOR APPLICATIONS

Although the foundation of DL was primarily associated with modern web technologies only 0, it is nowadays becoming an important component for a wide range of different applications. Starting from a set of atomic symbols and then applying suitable constructors (see Fig. 1) one can build complex DL concepts/roles. They can be used later to construct the DL knowledge base by applying the knowledge specification mechanism (see Fig. 2).

<table>
<thead>
<tr>
<th>Construction</th>
<th>Meaning</th>
<th>Transcription</th>
</tr>
</thead>
<tbody>
<tr>
<td>C,D→A</td>
<td>atomic concept</td>
<td>&lt;noun&gt;</td>
</tr>
<tr>
<td>⊤</td>
<td>universal concept</td>
<td>something</td>
</tr>
<tr>
<td>⊥</td>
<td>bottom concept</td>
<td>nothing</td>
</tr>
<tr>
<td>¬A</td>
<td>atomic negation</td>
<td>not &lt;noun&gt;</td>
</tr>
<tr>
<td>C/D</td>
<td>intersection</td>
<td>… and</td>
</tr>
<tr>
<td>¬R,C</td>
<td>weak existential restriction</td>
<td>&lt;verb&gt; only &lt;noun&gt;</td>
</tr>
<tr>
<td>C ⊂ D</td>
<td>union</td>
<td>… or</td>
</tr>
<tr>
<td>∃R,C</td>
<td>existential restriction</td>
<td>&lt;verb&gt; &lt;noun&gt;</td>
</tr>
<tr>
<td>¬C</td>
<td>negation</td>
<td>not</td>
</tr>
<tr>
<td>∃R</td>
<td>at most</td>
<td>&lt;verb&gt; at most &lt;number&gt; things</td>
</tr>
<tr>
<td>≥R</td>
<td>at least</td>
<td>&lt;verb&gt; at least &lt;number&gt; things</td>
</tr>
<tr>
<td>(n1,n2,…,nM)</td>
<td>one of</td>
<td>either n1 or n2 … an</td>
</tr>
<tr>
<td>P,Q,R→R'</td>
<td>inverse relation</td>
<td>&lt;verb&gt; by</td>
</tr>
<tr>
<td>Q</td>
<td>quantified number restriction</td>
<td>&lt;verb&gt; at most &lt;number&gt; &lt;noun&gt;</td>
</tr>
<tr>
<td>≥R</td>
<td>number at least</td>
<td>&lt;verb&gt; at least &lt;number&gt; &lt;noun&gt;</td>
</tr>
<tr>
<td>P(Q,R)</td>
<td>complex role inclusion</td>
<td>if &lt;role expr&gt;</td>
</tr>
</tbody>
</table>

Figure 1. DL Constructors and their meaning

A DL knowledge clip that is focused on some specific subject is called an Ontology. Ontologies can be stored in an Ontology Management System (OMS) (like: OWLDB [http://owldb.sourceforge.net], Jena [http://jena.sourceforge.net]) and accessed via a standard API (e.g., OWLAPI [http://owlapi.sourceforge.net]). An OMS can be built on top of a standard (Database Management System (DBMS)) equipped with an Object Relational Mapper (ORM) or even on an Object Oriented DBMS (OODBMS).

<table>
<thead>
<tr>
<th>Construction</th>
<th>Meaning</th>
<th>Transcription</th>
</tr>
</thead>
<tbody>
<tr>
<td>C,D</td>
<td>concept</td>
<td>Every &lt;C&gt; is a &lt;D&gt;</td>
</tr>
<tr>
<td>C,D⊆R,D</td>
<td>specification</td>
<td>Every &lt;C&gt; &lt;R&gt; a &lt;D&gt;</td>
</tr>
<tr>
<td>C(L)</td>
<td>assertions</td>
<td>&lt;L&gt; is &lt;C&gt;</td>
</tr>
<tr>
<td>R(L,J)</td>
<td>assertions</td>
<td>&lt;L&gt; &lt;R&gt; &lt;J&gt;</td>
</tr>
</tbody>
</table>

Figure 2. DL Expressions and their meanings

The DL foundation is equipped also with a set of automated reasoning tasks (e.g., to form the taxonomy of all atomic concepts, to determine subconcepts or to check whether a
concept is satisfiable). These tasks are supported by specialized computable algorithms called Reasoners (e.g. Hermit [http://hermit-reasoner.com], Fact++ [http://owl.man.ac.uk/factplusplus], CEL [http://lat.inf.tu-dresden.de/systems/cel], Pellet [http://clarkparsia.com/pellet]).

There are two types of DL expressions: a concept terminology (historically called TBox) and assertions (ABox). Systems built on the DL foundation can be therefore approached from two different perspectives: an expert or a user one. While the expert perspective allows to enter domain-specific knowledge stored in a TBox, the user perspective has to be intuitive enough to let any user of the system to manage instances and their relations that are stored in an ABox. The expert view is equipped with advanced knowledge management tools such as Protege [4]. There are also research initiatives to use a Controlled Natural Language (CNL) - a subset of a natural language which grammar and dictionary is limited to reduce or eliminate both ambiguity and complexity - to enrich this perspective. It was shown [5] that it is possible to build a bidirectional mapper\footnote{There are some limitations in expressivity of CNL: DL concept descriptions can be made arbitrarily complex by using parentheses whereas in CNL it is not possible.} from the DL to CNL. A mapper based editor can then be used by a domain expert to express knowledge in intuitive and understandable (for himself) and formal (for a reasoner) way.

III. GENERATOR

Nowadays architectures of data-centric applications are well understood. The Model-View-Controller (MVC) [6][7] architectural pattern, together with a database is one of the most frequently used solutions in data-centric systems. It divides an application concerning the three roles:

1. A Model that contains the core functionality and provides an access to the database.
2. Views that display information to a user.
3. Controllers that handle a user input.

Views and Controllers together form a User Interface (UI). A change-propagation mechanism ensures consistency between the UI and the Model (see Fig. 3).

The generator of applications discussed here is defined as a program that automatically generates the MVC-application that fulfills a specification formed by a given ontology (see Fig. 5). The UI of the generated application should allow a user to browse and modify the ABox of its parental ontology that it is built on. The application is built on the knowledge base itself so the “programmers work” must be made within the part of the system responsible for the UI creation. During the interactive activities that results in the ABox modification, the generator might be needed again to adopt the current state of the ontology to the MVC View/Controller and update its Model. In other words, the application arise as a product of the generator and the generator continuously reconstructs the application with respect to the modified ontology.

Let such a system is named an Interactive Knowledge Browser (IKB). There are two main components of the IKB: a Generator and an Application. The Generator itself is made of tree main components: Reasoner, OMS and a MVC-Factor. The OMS Session is a component that can communicate with the OMS and, according to a given signature, maintain ontology of interests. The Reasoner manages the inferred knowledge and provides automated reasoning tasks. The heart of the system – the MVC Generator – uses the Reasoner to build and/or update MVC Applications according to the knowledge inferred by the Reasoner (see Fig. 4).

The way in which the MVC Factory determines the whole UI is dependent on the implementation and the UI framework, but a general algorithm can be described. The algorithm assumes that the UI framework supports the MVC by

\[\text{Figure 3. MVC architectural pattern}\]

\[\text{Figure 4. INTERACTIVE KNOWLEDGE BROWSER}\]
providing a set of UI controls that can be binded to its models. Suppose that the UI framework contains: panel control (control that can aggregate other UI controls), Combo Box control (control that allows selection), Text Box, Button, List etc. To build View/Controller MVC factory the algorithm goes recursively from the top of the inferred ontology and places a Panel control for each subsumption of concepts. Each concept can then be examined whether its instances (possibly unnamed) exist and these instances, if found, can be checked whether there are relationships to other instances. If such a relationship is discovered, then the algorithm can create a Combo Box to allow to select any instance of a Most Specific Concept that the related instance is an instance of. The controller part of such an interface inserts or modifies the ABox of the instance; both the concept-assertion and role-assertion (see Fig. 5). It is easy to note that if the MVC Application is allowed to change only the ABox part of an ontology, then only the Model part of MVC is modified by Generator during user interactions. The View and Controller in this scenario remain unchanged, but the TBox is accessible still from other application (i.e. the application for experts) and brings the need for the occasional full UI update. Applications for experts are not discussed in this article. However, it is worth noticing that information derived from the Reasoner, can be submitted back from the IKB to the expert for a review or analysis to let him fully understand the semantic implications of asserted knowledge.

IV. RELATED WORK

There are dozens known ontology-editors and a number of them is growing from day to day. The most famous is Protégé [4]. Most of them are difficult to understand and use by a typical domain user (they require knowledge about ontology engineering). Protégé allows to edit an ontology and inspect the inferred knowledge. Both Protégé and IKB allow to edit an ABox via an automatically generated UI. However, the main difference between the IKB and Protégé lies in the way of user’s interaction with the system. While Protégé generates a UI for an ABox basing on the asserted ontology, the IKB does it using the Reasoner. User interactions force the reasoner to classify new facts and inferred (and only inferred) knowledge is used to build a UI or display a message to a user. As the IKB application is used, DL reasoning services intensively make it unique in the field of ontology-editors.

V. TNM CALCULATOR FOR LUNG CANCER STAGEING

Clinical practice guidelines are systematically developed statements designed to assist medical practitioners and patients with decisions about appropriate health-care for specific clinical circumstances [8]. The automation of a decision support occurs when the computer can make use of patients’ clinical data, follow its own algorithm, and present the information relevant to the current clinical situation [9].

The Interactive Browser System presented in this article is to be implemented as a clinical practice guideline application that supports the diagnosis of a lung cancer stage (see Fig. 6).

The prototype has been fully implemented in the Java+MySQL technology and proved that the approach is feasible. The prototype is released under the GPL license as the major components come from an open-source community.

As the system is build on a DBMS it can be used in a distributed environment allowing simultaneous access by plenty of users and experts. The change propagation is realized via CORBA.

The expert medical-knowledge is accessed via Controlled English (see Fig.7). The Interactive Knowledge Browser (see Fig. 8) is to be used by practitioners to equip patients with adequate therapies.

VI. CONCLUSION AND FUTURE WORK

Interactive Knowledge Browser is a new idea of building data-centric applications. It allows for the automatic generation of application without the need for immediate implementation of a domain specific part. More research is needed in this area, and the full implementation of the example system, which is expected to be done soon. Such kind of solutions can give the

\[\text{Equation}\]

\[\text{Figure 5. Ontology and examples of interactive browser user interfaces}\]
common reference for medicians, letting them test their diagnoses against the expert knowledge in an intuitive manner.

The application itself will prove that reasoning services are important software components usable in real-life applications.

Everything that is a tumor-index or is a node-index or is a metastasis-index is a tum
index. The tumor-index is something that has the feature-of-persistence-of-primary-tumor
and has the observation-feature and has the gathering-feature and has the satellite-nodes.
The feature-of-persistence-of-primary-tumor is something that has Carcinoma-in-situ and
has the biggest-size-of-tumor. Every biggest-size-of-tumor is a size. The size is measured
by the centimeters. The observation-feature has the observation-feature-place. The
observation-feature has the observation-feature-damage. Everything that is None or is Partial
or is Total is an observation-feature-damage. Everything that is Atelectasis or is Pleurone
is an observation-feature-place. The gathering-feature is something that has the
pleural-cavity-gathering and has the main-bronchi-gathering and has Chest-wall-gathering
and has Thoracic-diaphragm-gathering and has Phrenic-nerve-gathering and has Sternum-
tumor-gathering and has Gland-tumor-of-the-pleura-gathering. Everything that is Pleural-
cavity-nerve-gathering or is Pleural-cavity-pericardial-gathering or is Pleural-cavity-
mediastinal-gathering is a plural-cavity-gathering. Everything that is None or is Carina-
trachea-occupation or is the distance-to-carina-trachea is a main-bronchi-gathering.
Every distance-to-carina-trachea is a distance. The distance is measured by the
centimeters. The satellite-nodes is something that has Satellite-nodes-on-the-same-lobe.

Figure 7. Example domain specific expert knowledge in CNL

Figure 8. Example interactive browser

REFERENCES
American Magazine. http://www.sciam.com/article.cfm?id=the-
P. The Description Logic Handbook: Theory, Implementation, and
Applications
Elements of Reusable Object-Oriented Software. Addison Wesley
Language. PhD thesis, Faculty of Mathematics and Computer Science,
University of Tartu, 2007.
Pattern-Oriented Software Architecture: A System of Patterns, vol. 1 of
Software Architecture: On Patterns and Pattern Languages, vol. 5 of
program," Institute of Medicine, Committee on Clinical Practice
Web-Based Decision Support System to Facilitate the Efficient and
Effective Use of Clinical Practice Guidelines.," Proceedings of the 37th
Annual Hawaii International Conference on System Sciences
(HICSS’04)

GENERATOR APLIKACJI
ZORIENTOWANYCH NA DANE
OPARTY NA LOGICE OPISOWEJ