Transforming Healthcare with the Internet of Things
J. Hofdijk et al. (Eds.)

doi:10.3233/978-1-61499-633-0-64

Ontology-Oriented Programming for Biomedical Informatics

Jean-Baptiste LAMY[a,1]
[a] *LIMICS, Université Paris 13, Sorbonne Paris Cité, 93017 Bobigny, France, INSERM UMRS 1142, UPMC Université Paris 6, Sorbonne Universités, Paris*

Abstract. Ontologies are now widely used in the biomedical domain. However, it is difficult to manipulate ontologies in a computer program and, consequently, it is not easy to integrate ontologies with databases or websites. Two main approaches have been proposed for accessing ontologies in a computer program: traditional API (Application Programming Interface) and ontology-oriented programming, either static or dynamic. In this paper, we will review these approaches and discuss their appropriateness for biomedical ontologies. We will also present an experience feedback about the integration of an ontology in a computer software during the VIIIP research project. Finally, we will present OwlReady, the solution we developed.

Keywords. Knowledge representation (computer), Computer programs and programming, Ontology, Ontology-oriented programming, Biomedical knowledge, Open-world assumption

Introduction

The biomedical domain is one of the most complex domain of Human knowledge today. It is necessary to structure and formalize this knowledge adequately. The field of knowledge representation leads to the development of ontologies, which can link knowledge together and produce inference using a reasoner. Ontologies can represent universal statements (true for all individuals of a given class, e.g. medical knowledge about disorders or drugs), terminological statements (related to terms in a given natural language, e.g. synonyms for disorder names) and assertional statements (related to a given individual, e.g. medical data of a given patient) [1].

Many methods and tools have been proposed for the design, maintenance, alignment or evaluation of biomedical ontologies [2]. However, fewer options are available for ontology programming interface, another problem identified by A. Rector et al. [3]: how to access and manipulate an ontology in a computer program, for example to connect the ontology to a database or to generate a website from the inferences produced by the ontology?

In this paper, we will first describe some particularities of ontologies in the biomedical domain. Then, we will review the various approaches proposed for accessing ontologies in a computer program, and we will discuss which approach is the most appropriated for biomedical ontologies. Finally, we will give an experience

[1] Corresponding Author.

feedback about the integration of an ontology in computer software during a research project, and we will present OwlReady, the solution we developed.

1. Methods

Ontology in biomedical informatics. An ontology is the specification of the concepts, their attributes and relationships, in a given domain of discourse, for instance using the Ontology Web Language (OWL). Ontologies can be used for performing logical inferences and linking knowledge together in the semantic web. Ontologies rely on the open-world assumption, i.e. any fact is considered as possible until it has been explicitly stated that the fact is impossible.

The inherent complexity of the biomedical domain leads to some particularities when designing ontologies. (a) The open-world assumption is not always appropriate for medical reasoning. It is desirable for patient-related knowledge, for example, it allows reasoners to make hypotheses about unknown patient's disorders, e.g. in a diagnostic decision support system. On the contrary, it is not appropriate for drug- or disorder-related knowledge, for example, when considering drug adverse effects, we usually consider that all adverse effects are known (even if this may not be completely true) and that any adverse effect that is not known cannot occur. Thus a decision support system is expected to reason only on known effects and not to make hypotheses about unknown potential adverse effects. This would lead to stupid alert message like "the patient has hyperkalaemia; the drug you are prescribing does not cause hyperkalaemia but it might have an unknown adverse effect of hyperkalaemia".

(b) Many medical concepts cannot be represented by individuals but need classes to represent them. Disorders can be expressed at various levels of granularity (e.g. inflammatory bowel disorders, Crohn disease, severe Crohn disease with skin manifestation...) with inheritance (is a) relations between them. The same problem occurs for Human-created artefact such as drug treatments or medical acts. Drugs can be described by chemical or therapeutic classes, by active principles, by brand names, or even with a dose or an indication (e.g. aspirin in the antiplatelet indication). Consequently, both disorders and treatments should be classes rather than individuals in medical ontologies.

1.1. Traditional API for OWL vs ontology-oriented programming.

Two approaches have been proposed for accessing an ontology in a computer program. (a) Traditional API (Application Programming Interface) for OWL, such as OWL API [5] in Java, provide functions and classes for manipulating OWL constructs, e.g. an OWL class is an instance of OWLClass from the programmer's point of view. (b) Ontology-oriented programming tries to unify the ontology with the object model of the programming language, e.g. an OWL class is a class from the programmer's point of view. This approach exploits the similarities between ontologies and the object-oriented programming paradigm [6]: classes, properties and individuals in ontologies correspond to classes, attributes and instances in object models.

Ontology-oriented programming leads to shorter and more readable source codes, as shown by W3C [4] (Figure 1), and it is well-known in software development that a shorter code means fewer bugs. This is particularly interesting in biomedical

informatics, because we have seen that treatments and disorders are classes, and classes are even more complex to manipulate than instances.

Traditional API with Java:

```
public static float getOrderCost(OWLIndividual order) {
   OWLModel model = order.getOWLModel();
   OWLProperty drugProperty  = model.getOWLObjectProperty("drug");
   OWLProperty priceProperty = model.getOWLDadatypeProperty("price");
   float cost = 0.0;
   Iterator drugs = order.listPropertyValues(drugProperty);
   while(purchases.hasNext()) {
      OWLIndividual drug = (OWLIndividual) drugs.next();
      Float price = (Float) drug.getPropertyValue(priceProperty);
      cost = cost + price.floatValue();
   }
   return cost;
}
```

Ontology-oriented programming with Python and OwlReady:

```
class Order(Thing):
   def get_cost(order):
      cost = 0.0
      for drug in order.drugs: cost = cost + drug.price
      return cost
```

Figure 1. Two examples of ontology integration in a computer program. Both examples compute the total cost of a drug order (considering one box of each drug).

1.2. Static vs dynamic ontology-oriented programming

Two approaches exist for ontology-oriented programming. (a) The static approach consists of software that generates the source code for classes, from an ontology described in OWL. Modules have been implemented for Java [7] and C# [8]. They allow access to the ontology and to verify typing at compile time, but due to their static nature, they do not allow inference, classification or dynamic class creation. More recently, a semi-dynamic approach in Java [9] allows inference on individuals but not on classes. (b) The dynamic approach uses dynamic programming languages to generates classes and instances from the ontology at run time. In this approach, the same class is considered from an ontological point of view (following the open-world assumption) and from an object-model point of view (following the closed-world assumption, i.e. any fact is considered as impossible until it has been explicitly stated that the fact is true). A first module was proposed in Common Lisp [6] and a limited prototype in Python [10].

We have seen previously that, in the biomedical domain, both open-world and closed-world assumption are desirable, and that manipulating and classifying classes is a requirement. Consequently, the dynamic approach seems the right one for biomedical informatics.

2. Results

Experience feedback and contribution. The VIIIP (Integrated Visualization of Information about Pharmaceutical Innovation) research project aims at presenting information about new drugs to physicians. To guarantee an independent information,

this information is produced automatically by comparing the new drug to the older ones, using criteria such as efficacy in clinical trials, contraindications and known adverse effects.

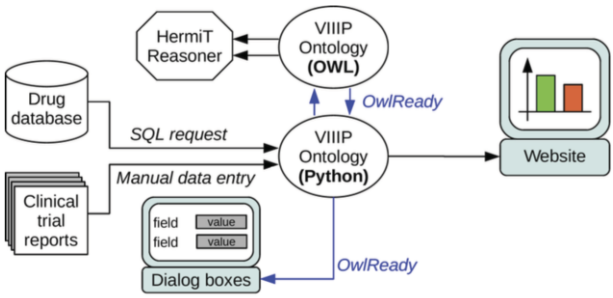

Figure 2. General structure of the VIIIP platform.

The automatic comparison of drug properties like contraindications is not easy because contraindications are often expressed at different levels of granularity in drug databases, e.g. rhythm disorder vs risk of torsades de pointes. To perform the comparison, we designed an ontology of contraindications. The ontology is populated from the French drug database Thériaque (http://theriaque.org), then a reasoner computes inferences, and finally the resulting inferences are presented in a website.

The general structure of the project was clear, but we experienced difficulties for connecting the ontology to the database and the website, and more generally to manipulate the ontology in the computer program. These difficulties lead us to a reflection about methods for accessing ontologies, and to the development of OwlReady (https://pypi.python.org/pypi/Owlready), a Python 3 module for ontology-oriented programming with full class-support, including dynamic class creation and classification of classes at run time using the HermiT reasoner [11]. OwlReady supports OWL 2.0. An experimental feature allows to automatically generate dialog boxes for editing individuals and classes in the ontology.

We successfully used OwlReady for implementing the VIIIP platform (Figure 2). Populating the ontology from the results of SQL requests, calling the reasoner, and generating HTML pages from the ontology were easy, from a computer-science point of view (but we encountered other problems related to the quality of data, out of the scope of this paper). The automatic generation of dialog boxes was very convenient, it allowed to modify the ontology without having to manually update the editing interface. The computation time of ontology-oriented programming with OwlReady and Python was higher than a traditional API in Java. However, the difference is insignificant compared to the time consumed by the HermiT reasoner, or the time we gained during software development (shorter source code implies faster development).

3. Discussion

We have shown that, in the biomedical domain, dynamic ontology-oriented programming is an interesting approach for integrating ontologies in computer software. In facts, it leads to simpler and shorter source code, while computer programs manipulating biomedical ontologies tend to be complex since disorders and treatments are represented by classes and not instances. It also allows a “mix” of open- and closed-

world assumption. We tested this approach in a research project and we proposed OwlReady, a module for dynamic ontology-oriented programming in Python.

In its current development stage, OwlReady is really practical for using ontology for a reasoning purpose, typically when one needs to dynamically create concepts, perform some reasoning on them, and then present the result of the reasoning. However, due to the lack of support of ontology file-formats (currently limited to a fair subset of OWL / XML), it is not yet well-suited for linking knowledge together.

Future works on OwlReady will focus on the use of classes. For example, some class restrictions could be exposed as if they were properties of the class, in a similar way to individual properties (e.g. the "Class property_x value 1" OWL restriction would be equivalent to "Class.property_x = 1" in Python). Class-class relations (i.e. all individuals of a class are related to all individuals of another, for example "all drugs of the anti-vitamin K pharmacological class interact with all drugs of the NSAI (Non-Steroid Anti-Inflammatory) pharmacological class") are often problematic in OWL because they cannot be represented directly. This is another area of possible improvement.

Acknowledgment

This work was partly funded by the French drug agency (ANSM, Agence Nationale de Sécurité du Médicament et des produits de santé) via the VIIIP project (AAP-2012-013).

References

[1] Schulz S, Jansen L. Formal ontologies in biomedical knowledge representation. Yearb Med Inform. 2013;8:132–46.
[2] Yu AC. Methods in biomedical ontology. J Biomed Inform. 2006;39(3):252–266.
[3] Rector A, Horridge M, Iannone L, Drummond N. Use Cases for Building OWL Ontologies as Modules: Localizing, Ontology and Programming Interfaces & Extensions. In: 4th Int Workshop on Semantic Web enabled software engineering (SWESE-08); 2008.
[4] Knublauch H, Oberle D, Tetlow P, Wallace E. A Semantic Web Primer for Object-Oriented Software Developers. W3C Working Group Note. 2006.
[5] Horridge M, Bechhofer S. The OWL API: A Java API for OWL ontologies. Semantic Web 2. 2011;p. 11–21.
[6] Koide S, Aasman J, Haflich S. OWL vs. Object Oriented Programming. In: the 4th International Semantic Web Conference (ISWC 2005), Workshop on Semantic Web Enabled Software Engineering (SWESE); 2005. p. 1–15.
[7] Kalyanpur A, Pastor D, Battle S, Padget J. Automatic mapping of OWL ontologies into Java. In: Proceedings of the Sixteenth International Conference on Software Engineering & Knowledge Engineering (SEKE'2004); 2004. p. 98–103.
[8] Goldman NM. Ontology-oriented programming: static typing for the inconsistent programmer. In: Lecture notes in computer science: the SemanticWeb, ISWC. vol. 2870; 2003. p. 850–865.
[9] Stevenson G, Dobson S. Sapphire: Generating Java Runtime Artefacts from OWL Ontologies. In: Lecture Notes in Business Information Processing, Advanced Information Systems Engineering Workshops. vol. 83; 2011. p. 425–436.
[10] Babik M, Hluchy L. Deep Integration of Python with Web Ontology Language. In: Proceedings of the 2nd workshop on scripting for the semantic web. Budva, Montenegro; 2006. p. 1–5.
[11] Motik B, Shearer R, Horrocks I. Hypertableau reasoning for description logics. Journal of Artificial Intelligence Research. 2009;36:165–228.