

Use Cases for Building OWL Ontologies as Modules: Localizing, Ontology and Programming Interfaces & Extensions

Alan Rector, Matthew Horridge, Luigi Iannone, Nick Drummond

School of Computer Science, University of Manchester Manchester M13 9PL, UK
[rector | mhorridge | iannone | ndrummy]@cs.manchester.ac.uk

Abstract: The notion of an Application Programming Interface (API) was a breakthrough in re-usable software development. OWL's import mechanism makes it possible to define analogous strategies for modular ontology development. This paper explores five use cases for such development: normalization, pluggable ontologies, extensions, localization, and ontology programming interfaces with applications.

1. Introduction

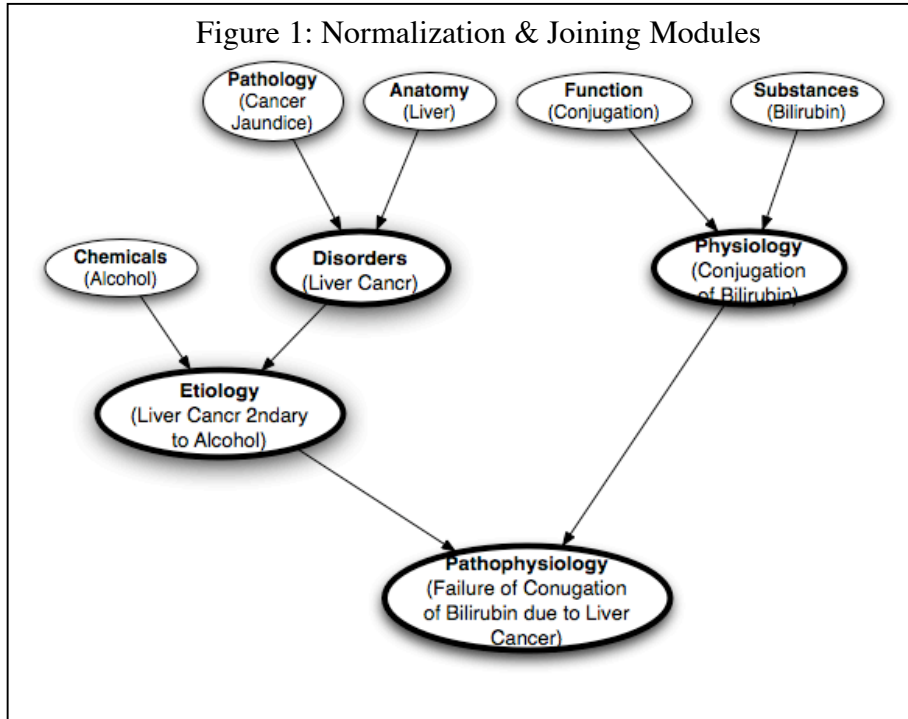
The notion of an Application Programming Interface (API) proved a breakthrough in software modularization and re-use. APIs allow developers to separate applications' public interfaces from their detailed internal structure and operation. They also help to focus developers' attention on providing clean sets of operations and methods to allow others to understand and re-use their code.

Currently, there is no similar widely accepted practice for ontologies, even those designed to be used in "ontology driven architectures." Most work has concentrated on extracting segments that preserve key properties from pre-existing ontology, *e.g.* [1], on the related notion of conservative extensions, *e.g.* [2] on the related issue of importing and re-using parts of pre-existing ontologies, *e.g.* [3] or on extracting modules that can be locked to allow concurrent ontology development, *e.g.* [4].

All these approaches involve *post hoc* segmentation of the ontology. By contrast, Bao et al [5] describe a mechanism for ontology packages as explicit extensions to DL languages. In this paper, we examine use cases, two of which are closely related to Bao's, and describe how they can be implemented within the framework of OWL-DL, or at least the proposed extension to OWL 1.1/2.

All the use cases proposed here are aimed at defining and managing dependencies amongst ontologies and between ontologies and ontology driven software as an intrinsic part of their design and development rather than retrofitting them to existing ontologies. Both strategies have their uses, but the authors believe that, for many applications, modular design provides advantages over a monolithic approach.

Until recently, many OWL ontology development environments made modular development mechanically difficult. The work reported here has been made possible



in large part because of the new Protégé-4¹ OWL development environment and the new OWL API [6]. Note that all examples are given using the Manchester OWL syntax [7] with the slight variation that for conciseness we abbreviate “subclassOf” to “→” and “equivalentClass” to “,↔”.

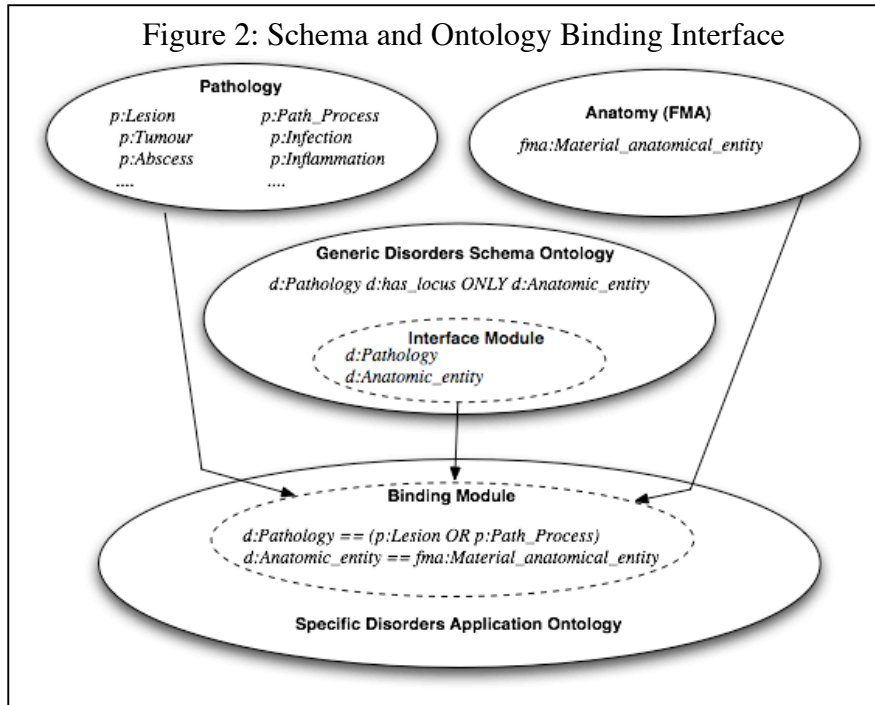
2. Use Cases

1.1 Use case 1: Ontology normalization and joining

In a previous paper [8], we introduced the notion of a normalized ontology formed by a set of strict mono-hierarchical trees of primitive entities joined by definitions and descriptions. The technique had been developed and well proven in other environments in the GALEN project [9] and has subsequently been used in ongoing work on biomedicine including, own work on clinical ontologies[10].

Originally, both the primitive trees and the joining axioms were implemented in a single module. Efficient means for modularization means that we can now implement each major tree – e.g. structure, function, disorder, causative agents, etc – in a separate module and then provide one or more “joining ontologies” that contain the axioms and definitions join them together. Figure 1 shows a cascade of such

¹ See <http://protégé.stanford.edu>



normalized and joining ontologies, with the joining ontologies shown in bold. The cascade allows the composition of complex notion out of careful factored individual ontologies, in this example that of a “Cancer (disorder) of Liver (structure) secondary to Alcohol (causative agents) that impairs conjugation (function) of Bilirubin (structure) causing Jaundice (disorder)”.

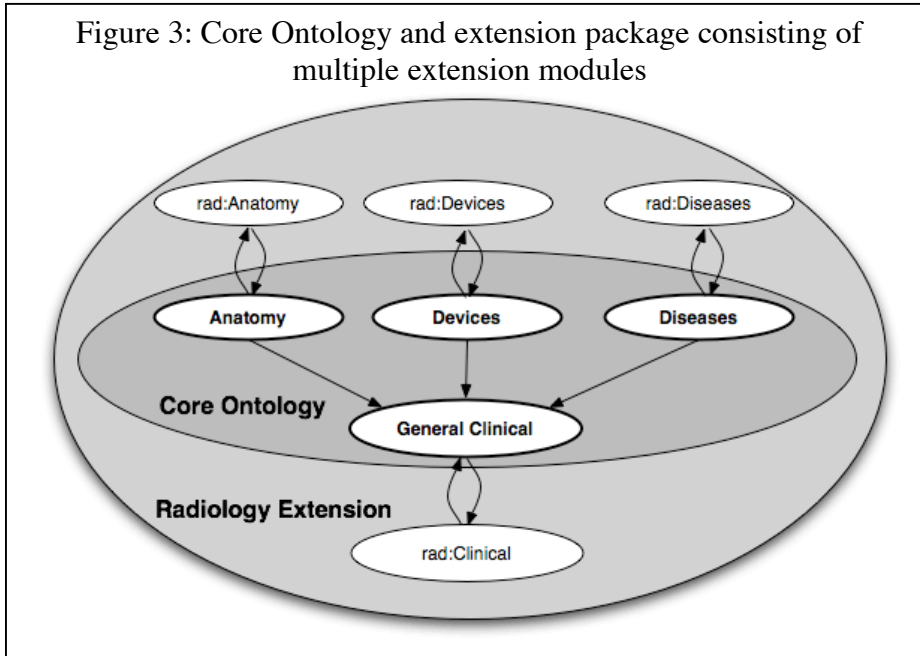
1.2 Use case 2: Pluggable modules: ontology Binding Interface and Placeholders

In many situations, a single core schema ontology is to be used with several alternative domain ontologies, *e.g.* a single ontology of the structure of clinical trials might be bound to a number of different disease and treatment ontologies, depending on the topic – *e.g.* cancer, infectious disease, congenital malformations, etc. In this case, the interface sub-ontology is the direct analogue of the API.

Consider the example in Figure 1. A single schema ontology of disorders might be used with several different ontologies of anatomy – one for surgical anatomy and an alternative one for developmental anatomy – and several alternative ontologies for pathology.

What is required in this case is for the schema ontology to be able to define the domains and ranges of relations at a generic level, and then allow these to be bound to the specific entities in appropriate subontologies for each subtopic.

Our strategy is for the generic schema ontology to identify key entities needed for its schema by “placeholders” classes. Equivalence and subclass axioms can then be



used to bind the placeholder classes to the specific classes in an application specific sub-ontologies as required.

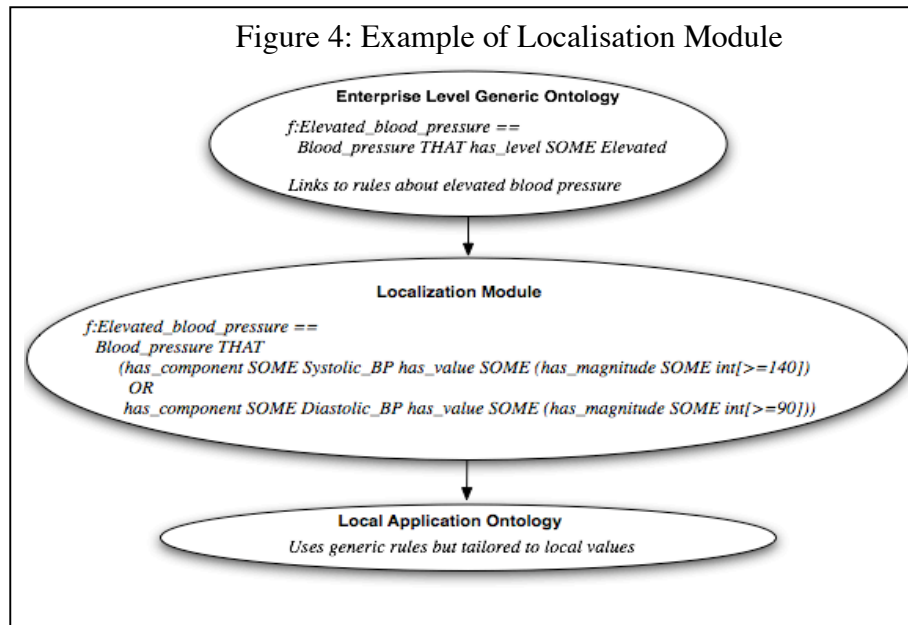
Figure 2 demonstrates the principle by showing the relations to the disorder module in Figure 1 in more detail. Prefixes are used for the namespaces for each ontology. The generic disorders schema ontology includes axioms defining the domain and range of properties that will link the imported Pathology and Anatomy pathology.² However, the disorder schema ontology makes minimal commitment to the nature of the anatomy or pathology ontologies or their contents.

To use the generic disorder schema, an application ontology must implement a binding ontology that defines the placeholders from the disorder schema – `d:Pathology` and `d:Anatomic_entity` in terms of the imported ontologies for anatomy and pathology. Usually the binding definitions are formulated as equivalence axioms, but this may be too strong in some cases. For example, the disjunction in Figure 2 could be weakened by replacing it with a pair of subclass axioms, thereby allowing the possibility that other classes, from other ontologies, might be kinds of pathology.

1.3 Use case 3: Ontology extensions and packages

Because imports in OWL simply add axioms to a monotonic system, and entities are assumed to exist when mentioned, there is no barrier to circular imports. This has proved a particular useful technique for developing extensions, especially during ontology development. An extension to a module both imports the module and is imported by it. This means that everything that is accessible in the base module is visible in the extension. This notion can be used either to extend a base ontology for

² The “Foundational Model of Anatomy” (FMA)[11] is assumed as a reference for anatomy.

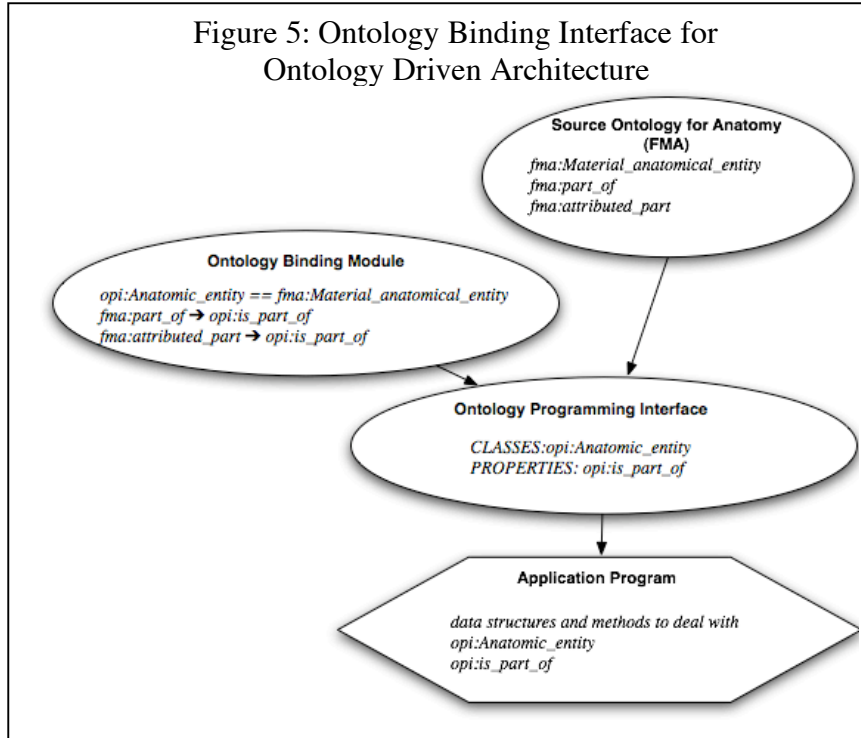


a particular subspecialty – a variant on the notion of plugable ontologies above – or as a development strategy to add new experimental information which can later be merged into the main ontologies. It has also proved fruitful to define testing extensions consisting of test cases with suitable annotations for “unit testing” to show that the classification is as intended and that intended constraints are actually enforced– *i.e.* that classe definitions that violate them are unsatisfiable. Placing these test cases in separate extensions makes it easy to remove them from the published ontology or to publish them separately.

Frequently sets of extensions form a package – *e.g.* a “radiology package” might require extending modules for anatomy, devices, and diseases (“pathology”). Currently this must be done by naming convention as shown in Figure 3. Plans are in hand to provide a more systematic mechanism so that all of the extensions that form a package can be managed as a unit and the naming conventions maintained automatically.

1.4 Use case 4: Localization

In many situations, there are general schemas and policies at the organisation or enterprise level that must be specialized at the local level. For example, there might be a general enterprise wide policy and generic rules for what to do in cases of “elevated blood pressure.” However, different departments’ or sites might have different criteria for when a blood pressure is to be considered elevated. Variations between sites in normal ranges and thresholds are common, particularly new biological assays and genetic tests, but occur even with relatively common measurements. Furthermore, these thresholds change with time. For example, in the UK, the policy for treating newly diagnosed type two diabetes has remained



unchanged for several years, but the threshold for diagnosis has been repeatedly lowered. Managing such local variations is a major task for many clinical systems.

Figure 4 shows a sketch of one solution to important parts of this problem. The generic axioms concerning elevated blood pressure reside in the *core* or *enterprise* ontology and associated with enterprise wide rules. The more specific axioms, *e.g.* the actual numeric thresholds, reside in a *localizing* ontology that is imported by applications along with the enterprise ontology.

1.5 Use case 5: Ontology Programming Interface

In Ontology Driven Architectures, analogous to Model Driven Architectures, application data structures and behavior are derived from an ontology, either statically or dynamically. This produces a tight coupling between the application and the ontology that can restrict both their development. Typically, there are a small number of key high level classes and properties in the ontology that are referenced directly by the application. Confining these to a separate *application interface* sub-ontology, which is agreed to remain stable, provide the necessary decoupling analogous to that provided by an API between program modules.

This pattern is very close to use case 2, except that in this case the “Interface Module” defines those classes that must be understood by the software as in some sense “special” rather than those that must be bound in another ontology. For example, the software might be aware of, and have special provision for, the fact that

anatomic entities have parts and perhaps even for the transitivity of parthood. However, the specific part-whole relations required for different classes of body parts would reside in the external anatomy ontology.

If an externally developed ontology is used, it is usually necessary to introduce an ontology binding module, as described in use case 3, to link the classes in the Ontology Programming Interface to the external ontology. An example of this combined strategy is shown in Figure 5, in which a standard anatomy ontology, the Foundational Model of Anatomy (FMA) is bound to an Ontology Programming Interface to be imported by an application that is written only in terms of generic notions of anatomical structures and part-whole relations. An extended case study and analysis of related methods is presented in [12]

3. Discussion

1.6 Issues in OWL for Modularization and Binding

Although OWL is often presented as if it were a collection of objects – classes, individuals, and properties – an OWL ontology actually consists simply of a collection of axioms about entities that do not have to be named before they can be referenced. Unlike a Java class, or a class in a typical frame system, there is no formal sense in which an OWL class “belongs” to a particular module. Any module can contain axioms about any class. This makes it easy to add additional information to “existing” classes in new modules, a feature that is critical to the strategies of using a binding ontology to add axioms to the description of the placeholder classes.

On the other hand, the disadvantage of OWL’s approach is that, for organisational and housekeeping purposes, it is helpful, perhaps even necessary, to identify a given class or property with the module in which it “originates.” At a minimum, it is necessary to identify the module, or modules, in which each axiom resides.

In OWL 1.0, axioms themselves have no identifiers and cannot themselves be annotated. Without this information, sensible editing policies are almost impossible. This problem is being addressed in OWL 1.1³ and the drafts for OWL 2 by allowing annotations to appear on individual axioms, which can be used to identify the ontology in which the axiom occurs. Protege4 is experimenting with overlaying this convention with the notion of the “originating ontology” for an entity – normally, the ontology that shares the base URI with the identifier for that entity – and the “original definition” – all the axioms about that entity in that ontology.

The larger problem is that to use such modular ontologies, it must be possible to treat sets of ontologies as units and to be able to copy and move such units easily without changing numerous absolute URIs in import statements. OWL 1.0 had no redirection mechanism although one has been being proposed for OWL 2.0. Protege4 is adopting conventions that include a notion of a set of ontologies residing in a single directory. It overloads the notion of “base URI” to act as an identifier for the ontology, independent of its physical location, so that the application simply looks for

³ <http://www.webont.org/owl/1.1/>

an ontology with the appropriate base URI in a sequence of locations: the local folder, a local library, a global library, and then the Web.

1.7 Conclusion

Re-use of ontologies is still in its infancy. Most work on ontology modularization has concentrated on extracting modules from existing large ontologies using notions such as conservative extensions rather than modular construction and re-use.

This paper looks at strategies for managing dependencies and encouraging re-use by establishing well defined interfaces between ontologies, analogous to APIs for programming languages. The an analogous strategy is proposed for controlling dependencies between ontologies and software using those ontologies. The strategy is also aimed to make ontologies “pluggable” so that alternative ontology modules may be used in conjunction with a single core ontology, *e.g.* different disease ontologies for different clinical domains in for a core ontology on clinical trials.

A modular approach is particularly important in very large domains where no single group is likely to be able to develop all the ontologies required. It makes it possible for specialized groups, such as those concerned with describing clinical trials, to focus on their specialty and link in a controlled and predefined way to external ontologies while minimizing their dependence on the internal details of those ontologies.

Experiments with the approach have included development of the CLEF Ontology and Chronicle system described in [12], the Ontology for Clinical Research (OCRe)⁴ and in ongoing efforts by the Ontogenesis consortium to refine the Gene Ontology⁵ as well as two commercial collaborations.

Acknowledgements

This work supported in part by the JISC and UK EPSRC projects CO-ODE and HyOntUse (GR/S44686/1) the EU funded Semantic Mining Network of Excellence and SemanticHealth Specific Support Action (IST-27328-SSA). The collaboration of the Ontologies for Clinical Research (OCRe) and Ontogenesis consortium is gratefully acknowledged.

References

1. Schlicht A, Stuckenschmidt, H; Towards structural criteria for ontology modularization. 2006; 1st International Workshop on Modular Ontologies, WoMO'06: Athens, Georgia:
2. Lutz C, Walther D, Wolter F; Conservative extensions of expressive description logics. 2007; International Joint Conference on Artificial intelligence (IJCAI 07): 453-458.
3. Pan JZ, Serafini L, Zhao Y; Semantic Import: An Approach for Partial Ontology Reuse. 2006; Workshop on Modular Ontologies (WoMO'06): Athens, Georgia:
4. Seidenberg J, Rector A; Web ontology segmentation: Analysis classification and use. 2006; WWW2006: Edinburgh, Scotland: W3C; <http://www2006.org/programme/item.php?id=4026>.
5. Bao J, Caragea D, Honavar V; Towards Collaborative Environments for Ontology Construction and Sharing. 0; 2006; International Symposium on Collaborative

⁴ <http://rctbank.ucsf.edu/home/ocre.html>

⁵ Robert Stevens, Personal Communication, 2008

- Technologies and Systems (CTS-06): Los Alamitos, CA, USA: IEEE Computer Society; 99-108.
6. Horridge M, Bechhofer S, Noppens O; Igniting the OWL 1.1 touch paper: The OWL API. 2007; OWL Experiences and Directions (OWLEd 2007): Innsbruck, Austria:
 7. Horridge M, Drummond N, Goodwin J, Rector A, Stevens R, Wang H; The Manchester OWL syntax. 2006; OWL: Experiences and Directions (OWLED 06): Athens, Georgia: CEUR; http://sunsite.informatik.rwth-aachen.de/Publications/CEUR-WS//Vol-216/submission_9.pdf.
 8. Rector A; Modularisation of domain ontologies Implemented in description logics and related formalisms including OWL. 2003; Knowledge Capture 2003: Sanibel Island, FL: ACM; 121-128.
 9. Rector AL, Zanstra PE, Solomon WD, et al. Reconciling users' needs and formal requirements: Issues in developing a re-usable ontology for medicine. *IEEE Transactions on Information Technology in BioMedicine*. 1999;2:229-242.
 10. Rector A; Patterns, properties and minimizing commitment: Reconstruction of the GALEN Upper Ontology in OWL). 2004; Workshop on Core Ontologies (CorOnt) in conjunction with EKAW-2004: Northampton, UK:
 11. Rosse C, Shapiro IG, Brinkley JF. The Digital Anatomist foundational model: Principles for defining and structuring its concept domain. *Journal of the American Medical Informatics Association*. 1998;820-824.
 12. Pulestin C, Parsia B, Cunningham J, Rector A; Building hybrid ontology-backed software models. 2008; International Conference on Semantic Web (ISWC-2008): Karlsruhe, De: (in press).