# Ontology-Driven Software Development in the Context of the Semantic Web: An Example Scenario with Protégé/OWL

Holger Knublauch
Stanford Medical Informatics, Stanford University, CA
holger@smi.stanford.edu

**Abstract**

Recent efforts towards the Semantic Web vision have lead to a number of standards such as OWL and Web Service languages. While these standards provide a technical infrastructure, software developers have little guidance on how to build real-world Semantic Web applications. Based on a realistic application scenario, we present some initial thoughts on a software architecture and a development methodology for Web services and agents for the Semantic Web. This architecture is driven by formal domain models (ontologies). The methodology applies best practices from agile development methodologies, including systematic tests, short feedback loops, and close involvement of domain experts. We illustrate how these techniques can be put into practice using the modern Semantic Web development tool Protégé, and indicate future possibilities.

## 1 Introduction

The goal of the *Semantic Web initiative* [3] is to provide an open infrastructure for intelligent agents and Web Services. This infrastructure is based on formal domain models (ontologies) that are linked to each other on the Web. These linked ontologies provide the applications with shared terminologies and understanding. The W3C has recently finalized the *Web Ontology Language* (OWL) [12] as the standard format in which ontologies are represented online. Similar standardization efforts such as SWRL [1] and SCL [2] are well underway.

While a lot of effort is being devoted to defining these languages and appropriate tool support, work on development methodologies for Semantic Web applications is still in its infancy. As an initial effort, W3C has recently started a working group to explore best practices and design patterns for OWL [3]. However, this group focuses on ontology construction and does not help with more general issues on software architecture, use of ontologies in applications, and software testing. Most non-trivial

[1] http://www.daml.org/2003/11/swrl
[2] http://cl.tamu.edu
[3] http://www.w3.org/2001/sw/BestPractices/

Semantic Web applications will consist of conventional components developed in languages such as Java, and comprehensive methodologies are needed that integrate these components in a Semantic Web context. Also, while the potential benefits of ontologies in general-purpose software technology have been widely discussed [5], ontologies have not achieved a major breakthrough yet.

In this paper we will explore some issues of developing software for the Semantic Web. Since this field is rather new, and few people have experience in the development of real-world systems, we believe it is important to collect example application scenarios that illustrate common problems and challenges. Such examples can lay out an application-oriented grounding for work on methodologies. We will present a realistic example scenario from the tourism domain (Section 2), and suggest a software architecture for applications of this kind (Section 3). From this architecture we will derive some development guidelines (Section 4), before we discuss lessons learned and future work (Section 5).

## 2 A Semantic Web Example Scenario

This section introduces an example Semantic Web scenario from the tourism domain. The basic idea is that providers of travel-related services such as holiday activities and accommodation advertise their services on the Semantic Web, so that intelligent agents can find them dynamically. These agents could then make suggestions on vacation planning and make travel arrangements in consideration of user preferences.

As illustrated in Figure 1, the Semantic Web infrastructure for these agents would be based on a few core ontologies. For example, a travel ontology could be defined by a standards body of the tourism industry, whereas a geography ontology could be provided by a government agency. Both ontologies would be published on fixed URI's as OWL files. The core tourism ontology would define concepts such as `ActivityProvider` to link an `Activity` with a `ContactAddress`. There could be a set of subtypes of activities such as `BungeeJumping` or `Caving`, and these could be categorized into types like `AdventureActivity`. Based on the rich expressiveness of OWL, it is furthermore possible to define classes by their logical characteristics. For example, a class `BackpackersDestination` could be defined as a destination that offers budget accommodation and some adventure activities. These defined classes allow reasoners to automatically classify existing domain objects into matching categories [9].

A base ontology like the travel ontology would allow providers to publish metadata about their services and contact information. Providers would instantiate the classes from the ontology and publish the resulting individuals as OWL files on their web sites. Then, a Semantic Web service specialized in vacation planning could send out a crawler agent to collect the available activities. If a user then asks for an exciting adventure destination, the agent could exploit the categorization of the ontology hierarchy to find suitable matches, and call auxiliary Web Services via the links into the geography ontology.

While some of this functionality could also be implemented with a traditional client-server web application, the main benefit of the Semantic Web is its open ar-

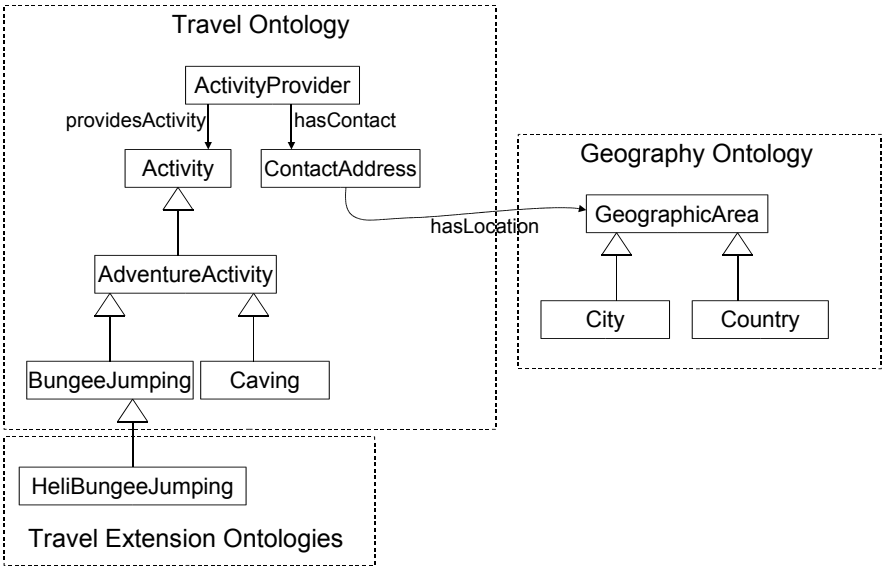


Figure 1: Ontologies from an example Semantic Web scenario.

chitecture and semantic richness. Providers of activities can not only publish their metadata dynamically, but they can also define their own specializations of the default classes. For example, an ontology module could define `HeliBungeeJumping` as a subclass of `BungeeJumping`, and put semantic restrictions on this class to describe its characteristics. Then, if an agent searches for bungee jumping facilities it would also find the instances of the subtypes, and also learn that jumps from a helicopter are traditionally more expensive than conventional jumps, that they involve aerial sightseeing, etc.

# 3 Architecture of a Semantic Web Application

Ontologies such as those described in the previous section can be exploited by different Semantic Web applications. Figure 2 illustrates the software architecture of an application that finds appropriate holiday destinations for a customer. The functionality of this application is made available to software agents through a Web Service interface, and to end-users through a conventional Web browser interface. Input to these services is in both cases a collection of data objects about a customer (e.g., age, personal preferences, hobbies, budget). The output is a list of suitable vacation destinations together with a list of suggested activities and corresponding contact addresses. These input and output data structures are formally represented in terms of OWL ontologies, so that external agents can correctly use the service.

Much of the application logic itself is implemented in a conventional object-oriented language such as Java. For example, the system must manage data bases, sessions, and the user interface. The application needs to represent the objects that are exchanged

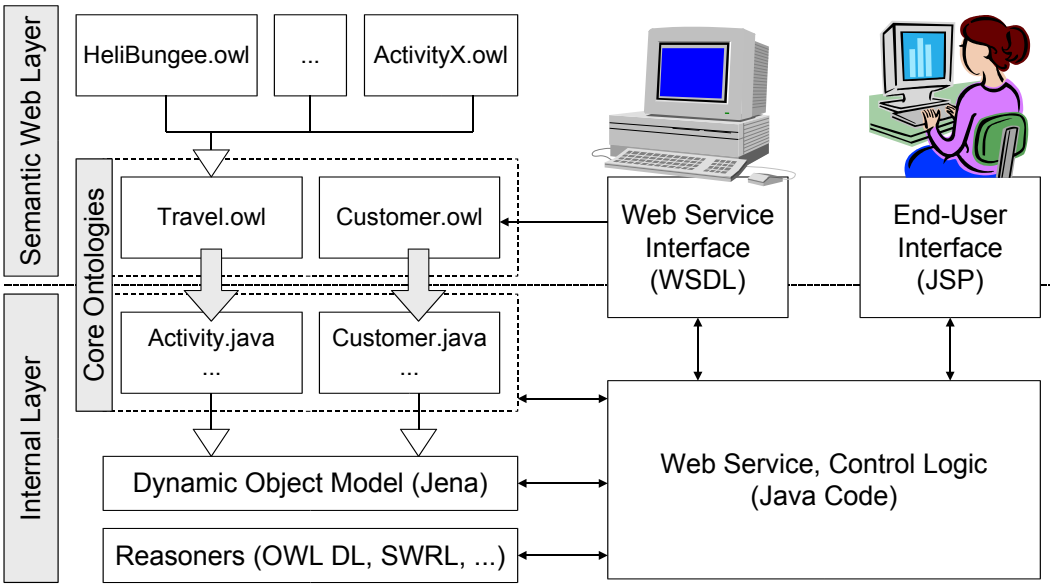


Figure 2: Architecture of an example Semantic Web Application.

between the application and other services or the user interface as Java objects. A typical implementation would employ an OWL parsing library such as Jena [4] for that purpose. Jena provides a dynamic object model in which OWL classes, properties and individuals are stored using generic Java classes like `OntClass` and `Individual`. While such an object model allows programs to operate on arbitrary OWL models, they are inconvenient to handle [6]. References to ontology objects are established only through names (i.e., Strings), making code hard to maintain and test. Also, as the ontology evolves at design time, existing code may run out of synch. If an ontology is known to the application beforehand, it is better to reflect the ontology concepts with custom-tailored Java classes, so that the ontological structure is exploited at compile-time. This also allows programmers to attach methods to these classes, leading to cleaner object-oriented design patterns. Some approaches on how to map OWL ontologies into object-oriented languages are available [7, 6].

In addition to the rather simple input/output data structures, ontologies are also used to represent the background knowledge that is needed by the application to fulfill its task. There are some core ontologies that define the basic structure of this knowledge by means of base classes. These base classes can be extended and instantiated arbitrarily by external ontology providers on the Semantic Web. While the base classes can and must be hard-wired into the executable system, the knowledge encoded in the external ontologies can only be used by generic reasoning engines such as Description Logic classifiers [1] or rule execution engines. We argue that most Semantic Web applications will have a similar architecture around core ontologies,

[4] `http://jena.sourceforge.net`

external ontologies, control components, and (user) interfaces.

## 4 Ontology-Driven Software Development

An important conclusion from the architecture from the previous section is that Semantic Web applications consist of two separate but linked layers: The *Semantic Web Layer* makes ontologies and interfaces available to the public, whereas the *Internal Layer* consists of the control and reasoning mechanisms. While the latter components can reside inside a "black box", the artifacts in the Semantic Web Layer are shared with other applications, and must therefore meet higher quality standards than the internal components. Also, the models in the Semantic Web Layer are used to control the internal behavior, in particular the outcome of reasoning algorithms. As a result, the code inside the internal components may be of relatively small size, and has to meet lower quality standards than the externally visible modules. Much of the internal components can be generated from the higher-level models and consists of generic libraries that are simply used out of the box. All this suggests that agile development approaches such as eXtreme Programming [2] are perfectly sufficient for the development of the application's internal parts.

Much more effort has to be spent with the system's ontologies. These ontologies must be consistent, generally useful and formally correct. Also, they must be extensible and may become very large. In order to ensure a high quality of these ontologies, developers can exploit the formal foundation of OWL on Description Logics [1] at design time to find inconsistencies, to maintain complex subclass relationships along multiple axes, and to reveal hidden relationships [11].

Modern ontology development tools such as Protégé with the OWL Plugin [10] allow users to exploit these services conveniently, and provide intelligent guidance to find mistakes similar to a debugger in a programming environment. As illustrated in Figure 3, the user only needs to press a "classify" button to get the correct classification of classes and to detect inconsistencies. Furthermore, Protégé serves as a rapid prototyping environment in which ontology designers can instantly create individuals of their ontology and experiment with semantic restrictions. The system is able to generate user interfaces that can be further customized for knowledge acquisition in a particular domain [8].

In addition to supporting ontology designers, Protégé also has an open architecture that allows programmers to insert arbitrary components into the tool. This can be exploited during the development of Semantic Web applications: developers can package the implementation of the application as a Protégé plugin and immediately test how the system behaves in response to ontology changes. The Protégé OWL Plugin even provides an open testing framework in which code similar to JUnit test cases can be executed at any time. Another feature of Protégé is code generation: the system takes an OWL ontology and creates corresponding Java classes from it.

To summarize, the development of Semantic Web applications with a tool such as Protégé is driven by ontologies. Ontologies can be developed by domain experts, and these experts have direct control over the behavior of the executing system and

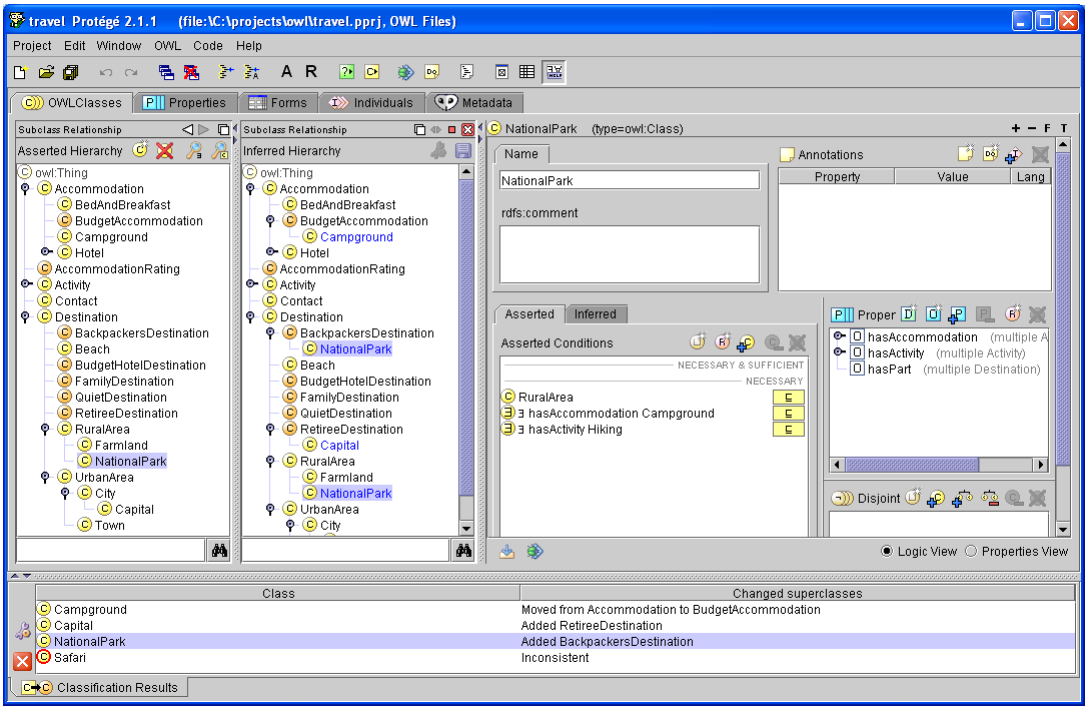


Figure 3: A screen shot of the Protégé ontology development environment.

some aspects of the implementation. Similar to agile methodologies like eXtreme Programming, feedback is available frequently since the domain models rapidly lead to executable systems. Best practices from agile methodologies can be employed to build high-quality domain models. In particular, ontologies can be developed in pairs, so that they are instantly peer-reviewed. To ensure that domain models lead to efficient implementations, domain experts may pair with programmers. Another important aspect of agile approaches is systematic testing. The formal backing of languages like OWL can be employed for testing both at design-time and at run-time. For example, the logical definitions from an OWL class can be exploited to verify invariants on objects similar to Design-by-Contract at run-time.

## 5 Discussion and Future Directions

Our example scenario highlights the importance of high-level domain models in software development in a Semantic Web context. This approach is remarkably similar to recent developments in mainstream software technology. The *Model-Driven Architecture* (MDA) [4] movement initiated by the OMG explores ways on how to better integrate high-level domain models into the development cycles of conventional software. A central idea of MDA is to employ domain models in languages like UML, and to have code generated from them for specific applications and platforms. Ontology-Driven Software Development as described above follows a very similar approach, but

applies these ideas in a much more extreme way: domain models are not only used for code generation, but they are used as executable artifacts at run-time. Therefore, it is natural to assume that progress in the field of developing Semantic Web applications could be applied to MDA as well. Also, progress in MDA technology and tools may be fruitful for the Semantic Web community.

So what can the ontology community learn from MDA? A major achievement of MDA is the grounding of the various separate modeling language standards under the umbrella of a single meta-metamodel called MOF [4]. MOF will support tool inter-operability and standardize model translations. The OMG's recent efforts in defining a mapping between OWL and MOF/UML [5] can become a cornerstone in bringing the fields and tools much closer together. Also, MDA advocates suggest that no single modeling formalism (such as UML class diagrams) is sufficient for domain modeling. Instead, domain-specific languages are selected depending on the task and the expertise of the domain modelers. The ontology community has made significant advances with the standardization of OWL, but OWL may not be the best modeling language for everyone. Instead, domain experts may prefer domain-specific languages or "intermediate representations" that are then translated into the details of OWL or SRWL or any other relevant language. Parts of these translations can be taken over by editing tools, by providing simplified views of lower-level modeling constructs. A common grounding of metamodels would help to formalize these views.

And what can the Software Engineering community learn from the Semantic Web? Domain models should not be regarded as intermediate artifacts that are only used for generating code and then put safely inside the drawers of a company's archive. Instead, domain models can be made executable on their own, and they can be shared between applications in an open-world setting such as the Semantic Web. This encourages reuse and inter-operability. Also, in order to build high-quality models, formal editing support such as reasoners are essential. Maintaining large quality ontologies without support by a reasoner quickly becomes impossible [11]. Therefore, MDA languages should strengthen their formal foundation, and they should become better executable.

As a consequence of these observations, a goal in future work should be to further leverage the role of declarative domain models in executable systems. Revisiting the architecture diagram from Figure 2, the goal would be to reduce the size of the "Java" boxes, and to increase the importance of the transformation arrows. With the availability of more expressive rule languages, it should be possible to encode a much bigger part of the application logic in formal, declarative models. At least, rules could be used for the generation of executable code and test cases. Furthermore, user interfaces such as input forms for customer data could be automatically generated from ontological structures. This has been shown by systems like Protégé, where flexible user interface are generated from class definitions.

[5] `http://www.omg.org/ontology`

## References


[1] Franz Baader, Diego Calvanese, Deborah McGuineness, Daniele Nardi, and Peter Patel-Schneider, editors. *The Description Logic Handbook*. Cambridge University Press, 2003.

[2] Kent Beck. *Extreme Programming Explained: Embrace Change*. Addison-Wesley, Reading, MA, 1999.

[3] Tim Berners-Lee, James Hendler, and Ora Lassila. The Semantic Web. *Scientific American*, 284(5):34–43, 2001.

[4] Grady Booch, Alan Brown, Sridhar Iyengar, James Rumbaugh, and Bran Selic. An MDA Manifesto. MDA Journal, May `http://www.bptrends.com/search.cfm?keyword=MDA+Journal&gogo=1`, 2004.

[5] Vladan Devedzić. Understanding ontological engineering. *Communications of the ACM*, 45(4):136–144, 2002.

[6] Neil M. Goldman. Ontology-oriented programming: Static typing for the inconsistent programmer. In *2nd International Semantic Web Conference (ISWC 2003)*, Sanibel Island, FL, 2003.

[7] Aditya Kalyanpur, Daniel Jiménez Pastor, Steve Battle, and Julian Padget. Automatic mapping of OWL ontologies into Java. In *16th International Conference on Software Engineering and Knowledge Engineering (SEKE)*, Banff, Canada, 2004.

[8] Holger Knublauch. An AI tool for the real world: Knowledge modeling with Protégé. *JavaWorld*, June 20, 2003.

[9] Holger Knublauch. Protégé OWL Plugin tutorial. 7th International Protégé Conference, Bethesda, MD, `http://protege.stanford.edu/plugins/owl/documentation.html`, 2004.

[10] Holger Knublauch, Ray W. Fergerson, Natalya F. Noy, and Mark A. Musen. The Protégé OWL Plugin: An open development environment for semantic web applications. In *3rd International Semantic Web Conference (ISWC 2004)*, Hiroshima, Japan, 2004.

[11] Alan Rector. Description logics in medical informatics. Chapter in [1].

[12] World Wide Web Consortium. OWL Web Ontology Language Reference. W3C Recommendation 10 Feb, 2004.