Application of OWL 1.1 to Systems Engineering

Henson Graves[1] and Ian Horrocks[2]

[1]Lockheed Martin Aeronautics Company
Fort Worth, Texas, USA
henson.graves@lmco.com

[2]Oxford University Computing Laboratory
Parks Road, Oxford, UK
Ian.Horrocks@comlab.ox.ac.uk

Abstract. Current systems engineering languages, standards, and tools are restricted in certain aspects of their expressiveness and do not provide formal semantics. While there is a long history of attempts to use formal methods for engineering, up to now the formalisms have generally proved too hard to use, and the tools do not scale for large complex system development. A semantic integration framework that can integrate representations from multiple system engineering languages and tools could, however, have a significant impact on cost, schedule, and product integrity. We are exploring the potential for OWL 1.1 to provide such a semantic integration for the air system engineering domain. To determine the potential to use OWL 1.1 in this setting we are developing a prototypical air system ontology in OWL 1.1 and evaluating the use of the language for developing and reasoning about systems engineering concepts such as requirements and product structure.

1 Introduction

Current systems engineering languages, standards,[1] and tools are restricted in certain aspects of their expressiveness and do not provide formal semantics. While there is a long history of attempts to use formal methods for engineering, up to now the formalisms have generally proved too hard to use, and the tools do not scale for large complex system development. A semantic integration framework that can integrate representations from multiple system engineering languages and tools could, however, have a significant impact on cost, schedule, and product integrity. We are exploring the potential for OWL 1.1[2] to provide such a semantic integration for the air system engineering domain [1]. The idea is that a product development ontology can capture the meaning of concepts in a form independent of interpretation by subject matter experts, something that is not possible with current engineering languages and tools. Automated

[1] See, e.g., http://www.sysml.org/
[2] See http://www.w3.org/2007/OWL/

reasoning could then be used to check design properties such as consistency and conformance with specification, and the ontology could also be used to integrate information from the large number of systems used in the design of an air vehicle.

To determine the potential to use OWL 1.1 in this setting we are developing a prototypical air system ontology in OWL 1.1 and evaluating the use of the language for developing and reasoning about systems engineering concepts such as requirements and product structure. The ontology has been developed within Protégé 4.0 using the Fact++ reasoner [2]. We have verified that a simplified air vehicle design specification satisfies requirements verification criteria under specific assumptions. Preliminary analysis indicates that OWL 1.1 has the potential to have a significant impact on systems engineering tools and processes. We have, however, also discovered some shortcomings of OWL 1.1, at least with respect to this application.

2 The use of knowledge representation and reasoning in product development

Systems engineering is a sufficiently complex domain that its vocabulary needs formal definition. The meaning of concepts in engineering depends on consensus of interpretation by domain experts. Sufficient precision regarding the meaning of concepts is notoriously difficult to obtain. The multiplicity of languages and data formats makes information integration a difficult and labour intensive task. This is a common problem associated with other domains such as life sciences. Large and complex ontologies are being developed and maintained in these domains. Explicit representation of concepts in an ontology and reasoning based on the ontology has been shown to provide essential support for the development of ontologies and for their application in semantic integration. We believe that the potential for reasoning within systems engineering includes, but goes beyond information integration.

Many systems engineering tasks directly involve reasoning. For example, requirements for a product to be developed need to be checked for consistency: it is not unheard of that requirements developed by multiple individuals and groups are inconsistent. Further, at any given stage of a complex concurrent development process, design artifacts are being produced using other design artifacts as input. Even with rigid controls, immature or superseded inputs may be used to produce some new design artifact, leading to inconsistency. Design artifacts can easily be inconsistent with each other at a given time. Automated consistency checking is therefore of great importance.

The real test of reasoning in product development comes, however, when a product which implements a design has been produced. The challenge is then to verify that the product satisfies its requirements. Of course, this verification task involves going outside any reasoning framework to perform analysis and make measurements. However, reasoning is still needed to check that the accumulated test results support the conclusion that the product which was tested actually satisfies the requirements. Since these requirements are pervasive in engineering,

the potential payoff for using ontologies and reasoning in terms of saved cost and schedule and elimination of rework is very high.

When decisions which are based on complex technical information have significant financial and legal implications, the correctness of conclusions and credibility of argument has high significance. The accepted norm for correct reasoning is to use a well understood logical framework consisting of a language and an inference procedure, for example First Order Logic and resolution. From statements in the language which are assumed to be true (axioms), conclusions are derived using the inference procedure. The application of logic is often referred to as formal methods [3]. The potential for use of logic in engineering goes beyond providing a foundation for correctness of argument. The concept of proof in logic is, again, the norm for representing the evidence for conclusions. A proof provides an explanation for a logically derived consequence, which could be highly relevant in engineering. For example, argument may be required to demonstrate that satisfying a specific collection of test conditions is logically equivalent to satisfying a requirement.

For at least 30 years [4] formal methods have been put forward as the best means to develop safe, reliable products that perform as expected. The reality is that formal methods are in regular use in only a limited number of specialized areas such as safety of flight control software [5] and hardware design [6]. One reason for this state of affairs is that tools with wide usage must satisfy a range of basic requirements for consideration in large scale complex product engineering efforts. Other reasons have to do with finding a good logical framework and engineering the framework to produce usable tools. A good logical framework balances questions of expressiveness with the complexity of reasoning procedures.

3 Languages and tools for product development

General requirements apply to all computer based languages and tools used in large scale product development engineering. Application languages must be somewhat intuitive to individuals with domain expertise, must be standards-based and must have good tool support. Tools must sufficiently simple to be used by systems engineering practitioners with only basic training, and must scale for large efforts. They must also have well defined interfaces for data exchange with other tools. Most commercial tools used for system engineering satisfy these requirements, or they do not receive much use.

Current systems engineering languages and tools, however, lack a formal semantics. This lack impedes the ability to integrate information from multiple languages and tools. Further, there are no logical mechanisms to check if classes are consistent, if one class is contained in another class, or if an individual is an instance of a class.

3.1 Why use OWL and why OWL 1.1?

Advances in the development of knowledge representation languages and methodologies for engineering knowledge bases, the provision of these languages with

a formal semantics, the development of efficient reasoning algorithms, and the adoption of OWL as a World Wide Web Consortium language standard, all contribute the potential to realize the promise of formal methods within engineering. In particular:

- OWL represents the culmination of more than 20 years of research aimed at developing formal languages for conceptual modeling. As a result of this research, the formal properties of the language are extremely well understood.
- OWL is a logical system with a formal semantics, and it is a fragment of First Order Logics. In particular, the language has been designed so as to be decidable (arbitrary queries can be answered) and to allow for sound and complete implementations (implementations where positive and negative answers can be given equal weight).
- OWL is a W3C standard language for Web ontologies, and has rapidly become a de facto standard for ontology engineering in fields as diverse as biology [7], medicine [8], geography [9], astronomy [10], agriculture [11], and defence [12].
- The standardization of OWL, along with the number and size of OWL applications, has fuelled the development of OWL tools and infrastructure, both academic and commercial, including editors such as Protégé 4.0 (http://protege.stanford.edu/) and Swoop [13], reasoners such as Pellet [14], FaCT++ [2], RACER [15], CEL [16], and KAON2 [17], APIs such as the OWL API [18] and foundational ontologies such as DOLCE [19]. These are now sufficiently robust and scalable to be used in large-scale applications.
- We can draw on extensive experience of using OWL in the above-mentioned applications, and in particular on experience using OWL and other Description Logic based ontology languages to model complex physically structured systems such as human anatomy, automobiles (Ford and Daimler Chrysler [20]) and even aircraft systems (Boeing [21]).

The use of OWL 1.1 is not intended to replace system engineering languages such as UML and SysM, or the use of 3D geometry languages, but to augment these languages and serve as an integration framework. A Knowledge Base (equivalently an OWL 1.1 ontology) can be used to represent product design and requirements specifications as well as facts regarding product implementation, testing, verification, and deployment.

3.2 Why use a foundational ontology and why use DOLCE?

In the development of our air system ontology a lot of effort has been saved through the use of a foundational ontology. The entities of interest in the system engineering domain are very diverse. Air system engineering entities include physical objects and their qualities, as well as events, plans, descriptions, and test results. An engineering ontology requires the representation of structural aspects and properties of products. In addition, the ontology will need to represent artifacts such as specifications for products and tests to be performed

on products. The usefulness of a foundational ontology is also documented in application areas such as life sciences [7, 8].

In our case we are making extensive use of the DOLCE Ultra Lite (DUL) foundational ontology. Our air system ontology is constructed by specializing classes and roles in DUL. The structure of and constraints on DUL classes and roles have been carefully worked out, and DUL can be readily loaded in Protégé 4.0. For example, we use and build on partOf and hasPart properties for specifying product structure. We use Quality to represent attributes and characteristics of product components. There are some specific concepts in other foundational ontologies that would be worth adopting and we plan to make use of these as we proceed. DOLCE entities not only include objects, both physical and social, events, but also descriptions of objects and events. These classes such as Description and Situation are useful to classify descriptions of test setups and occurrences of test processes with their observed/measured results.

4 Benefits and Shortcomings of OWL 1.1

Using OWL and DUL classes works well for representation of product structure and (static) properties. Class constructors allowed us to define classes of physical objects with structural properties (using partOf relations) and specify properties (using Quality with measured or computed values in Region).

For requirements that consist of structural properties and characteristics, we have been able to define classes and verify that the (requirements) class is consistent. We have been able to define classes characterized by generic instances (e.g., a design class) and construct a generic instance (in the ABox). We can then go outside the logic and measure, analyze and simulate in order to conclude (and then assert in the ABox) some additional properties of the instance. Finally, we can use the reasoner to check if the extended generic instance is consistent with the requirements and/or provably satisfies them.

Some of the new features of OWL 1.1 were useful or even crucial in our application. In representing the physical structure of an air vehicle, for example, we encounter many of the same problems that arise in medicine when describing anatomy. In this part of the model various kinds of part-whole properties play an important role, and the complex role inclusions of OWL 1.1 are of crucial importance in allowing us to transfer properties and constraints from parts to wholes and vice versa.

The extended support for datatypes in OWL 1.1 is critical in engineering applications. Numerous design constraints relate to concrete values such as weight, speed, temperature, distance etc. Some also relate to more complex datatypes, for example datatypes representing shapes or complex performance measures. N-ary datatype predicates might be useful for comparing values, but more powerful mechanisms, such as aggregation, would be needed in general. For example, the weight of a physical object can be calculated in multiple ways. Weight can be estimated, measured (using a scale) or calculated from the sum of the weights of components. A rule stating that the weight of a product is the sum of the

weights of its components is not easily expressed in OWL 1.1, although DL-safe rules might be used if their restricted applicability (i.e., to ABox individuals) proved to be acceptable [22]. Verification that a product satisfies some weight requirement, using the weight rule, must therefore be done externally to OWL.

When huge amounts of money, and even lives, may depend on the behaviour of an information system, tracking the provenance of both explicit and implicit information is of great importance. The improved annotations capabilities of OWL 1.1 are therefore of great importance—one needs to know where information came from and when. Explanations and/or proofs that show how implicit information has been derived will also be important, perhaps even essential.

Another observation is that representing dynamic properties of products, such as the change in fuel level during flight operation of an air vehicle, presents problems for OWL 1.1. Dynamic properties are properties that change with respect to time or some operational context. DUL provides concepts needed to represent behavior, but a more expressive language (such as FOL) seems to be required in order to fully model behavioral requirements and test results relating to behaviors.

With respect to OWL tools, improved interfaces to other computation systems would be very useful. For example, it would be very useful to be able to interface to an external database containing a parts list. On the one hand, it may be useful for the DB system to query the semantic framework about the configuration description to determine properties of the parts in the DB. On the other hand, where a database contains a parts list corresponding to a potential air vehicle configuration one would like the OWL tool to create in the ontology a product instance corresponding to the configuration; the reasoner could then be used to verify that the instance is consistent with and/or provably satisfies the design requirements.

5 Achievements and lessons learned

OWL 1.1 is not a replacement for Systems Engineering languages (e.g., SysML) and tools, but OWL 1.1 is a good candidate for defining Classes and relationships that can be used to describe products, and for semantic integration. The formal semantics provides a precise notion of meaning. Reasoning can be used to aid testing and verification. The serialisation of OWL 1.1 using XML schemas may facilitate the exchange of data between systems, but issues need to be resolve regarding interface to external systems. Given that a DL based system is unlikely to meet all the requirements in this large and complex domain, an architecture is needed for integration the of DL based systems with not-DL logics and other computational systems, including, but not limited to, database systems.

In spite of the difficulties mentioned in Section 4, we have been successful in using FaCT++ for simple reasoning experiments to show consistency or inconsistency of product descriptions. This is very promising as current systems engineering tools are not able to perform this kind of check. We are still experimenting with how best to divide the modeling and reasoning tasks between

the TBox and the ABox: the former allows for gradual refinement and varying granularity, but the latter allows for the more precise description of complex relational structures. Recent work on representing structured objects in OWL might be of great interest in this respect [23].

We are also investigating the use of the ontology to optimise testing. A typical issue for product verification is how to use the requirements decomposition tree constructed as part of the design process to determine test conditions with expected results that are used as input to the test planning process. Each test description describes a collection of input and output parameters to some operational process such as detecting and identifying objects within some area of interest. For example, a test may be to measure the accuracy of an air vehicle radar to detect and identify objects on the ground. The input parameters to the detection and identifying process describe aspects of the air vehicle operating environment. A specific test supplies values for the input parameters and obtains values for the output parameters. Representing features of tests could possibly be used to try to minimize redundancy by recognizing when, say, a conjunction of tests subsumes some other test.

Our ontology development effort is still in its beginning stages. However, we can identify areas whether further language and tool development would be of great use.

Ontology Management Development of ontologies for large complex systems involves many subdomains and disciplines. Each domain or discipline may have its own ontology. The overall process of product development will at any given time be using a unification of some of these ontologies. Changes will inevitably need to be made to individual ontologies and so mechanisms are needed to manage multiple ontologies by including or excluding them and tracking which ontologies were in force when specific conclusions were reached. Recent work on modularity will clearly be of great interest in this regard [24].

Explanation and Annotation As mentioned above, improved annotations are important as we go forward—one needs to know where information came from and when. Also explanations and/or proofs will be important in order to understand the provenance of implicit information. With the current methods being used in OWL theorem provers, it might seem unlikely that a machine constructed proof would be understandable. However, there is research which correlates model theoretic proof results with natural deduction results. This correspondence could perhaps be used to construct proofs to be used for evidence purposes.

Need for modularization Another potentially relevant strand of research for OWL application is modularization. It would be useful to check if different components and features that are to be tested are completely independent of each other and so can be tested completely independently. If you can show that

tests A and B are independent, then costs can be saved by reducing the number of different combinations of tests that need to be carried out. Elimination of test costs is of great concern in product development.

References

1. H. Graves. Ontology engineering for product development. In *Proc. of the Third OWL Experiences and Directions Workshop*, 2007.
2. Dmitry Tsarkov and Ian Horrocks. FaCT++ description logic reasoner: System description. In *Proc. of the Int. Joint Conf. on Automated Reasoning (IJCAR 2006)*, volume 4130 of *Lecture Notes in Artificial Intelligence*, pages 292–297. Springer, 2006.
3. C. Holloway. Why engineers should consider formal methods. In *Proc. of 16th Digital Avionics Systems Conference*, October 1997.
4. C. Hoare. An axiomatic basis for computer programming. *Communications of the ACM*, 12:576–580, 1969.
5. M. Blackburn. Using models for test generation and analysis. In *Proc. of 17th Digital Avionics Systems Conference*, October 1998.
6. C. Kern and M. Greenstreet. Formal verification in hardware design: A survey. *ACM Trans. on Design Automation of Electronic Systems*, 4(2), 1999.
7. Amandeep Sidhu, Tharam Dillon, Elisabeth Chang, and Baldev Singh Sidhu. Protein ontology development using OWL. In *Proc. of the First OWL Experiences and Directions Workshop*, volume 188 of *CEUR Workshop Proceedings*. CEUR (`http://ceur-ws.org/`), 2005.
8. Christine Golbreich, Songmao Zhang, and Olivier Bodenreider. The foundational model of anatomy in OWL: Experience and perspectives. *J. of Web Semantics*, 4(3), 2006.
9. John Goodwin. Experiences of using OWL at the ordnance survey. In *Proc. of the First OWL Experiences and Directions Workshop*, volume 188 of *CEUR Workshop Proceedings*. CEUR (`http://ceur-ws.org/`), 2005.
10. S. Derriere, A. Richard, and A. Preite-Martinez. An ontology of astronomical object types for the virtual observatory. *Proc. of Special Session 3 of the 26th meeting of the IAU: Virtual Observatory in Action: New Science, New Technology, and Next Generation Facilities*, 2006.
11. Dagobert Soergel, Boris Lauser, Anita Liang, Frehiwot Fisseha, Johannes Keizer, and Stephen Katz. Reengineering thesauri for new applications: The AGROVOC example. *J. of Digital Information*, 4(4), 2004.
12. Lee Lacy, Gabriel Aviles, Karen Fraser, William Gerber, Alice Mulvehill, and Robert Gaskill. Experiences using OWL in military applications. In *Proc. of the First OWL Experiences and Directions Workshop*, volume 188 of *CEUR Workshop Proceedings*. CEUR (`http://ceur-ws.org/`), 2005.
13. Aditya Kalyanpur, Bijan Parsia, Evren Sirin, Bernardo Cuenca-Grau, and James Hendler. SWOOP: a web ontology editing browser. *J. of Web Semantics*, 4(2), 2005.
14. E. Sirin, B. Parsia, B. Cuenca Grau, A. Kalyanpur, and Y. Katz. Pellet: A practical OWL-DL reasoner. *J. of Web Semantics*, 5(2):51–53, 2007.
15. Volker Haarslev and Ralf Möller. RACER system description. In *Proc. of the Int. Joint Conf. on Automated Reasoning (IJCAR 2001)*, volume 2083 of *Lecture Notes in Artificial Intelligence*, pages 701–705. Springer, 2001.

16. Franz Baader, Carsten Lutz, and B. Suntisrivaraporn. CEL—a polynomial-time reasoner for life science ontologies. In U. Furbach and N. Shankar, editors, *Proc. of the Int. Joint Conf. on Automated Reasoning (IJCAR 2006)*, volume 4130 of *Lecture Notes in Artificial Intelligence*, pages 287–291. SV, 2006.
17. Boris Motik. *Reasoning in Description Logics using Resolution and Deductive Databases.* PhD thesis, Univesität Karlsruhe (TH), Karlsruhe, Germany, January 2006.
18. Sean Bechhofer, Raphael Volz, and Phillip Lord. Cooking the semantic web with the OWL API. In Dieter Fensel, Katia Sycara, and John Mylopoulos, editors, *Proc. of the 2nd International Semantic Web Conference (ISWC 2003)*, volume 2870 of *Lecture Notes in Computer Science*. Springer, 2003.
19. Aldo Gangemi, Nicola Guarino, Claudio Masolo, Alessandro Oltramari, and Luc Schneider. Sweetening ontologies with DOLCE. In *Proc. of EKAW 2002*, pages 166–181, 2002.
20. Nestor Rychtyckyj. DLMS: An evaluation of KL-ONE in the automobile industry. In *Proc. of the 5th Int. Conf. on the Principles of Knowledge Representation and Reasoning (KR'96)*, pages 588–596, 1996.
21. Michael Uschold, Peter Clark, Fred Dickey, Casey Fung, Sonia Smith, Stephen Uczekaj Michael Wilke, Sean Bechhofer, and Ian Horrocks. A semantic infosphere. In Dieter Fensel, Katia Sycara, and John Mylopoulos, editors, *Proc. of the 2nd International Semantic Web Conference (ISWC 2003)*, volume 2870 of *Lecture Notes in Computer Science*, pages 882–896. Springer, 2003. Presentation available from `http://www.cs.man.ac.uk/~horrocks/Slides/ISWC-Presentation-SemanticFiltering.pdf`.
22. Boris Motik, Ulrike Sattler, and Rudi Studer. Query answering for owl-dl with rules. *J. of Web Semantics*, 3(1):41–60, 2005.
23. Boris Motik, Bernardo Cuenca Grau, and Uli Sattler. Structured objects in owl: Representation and reasoning. In *Proc. of the Seventeenth International World Wide Web Conference (WWW 2008)*. ACM Press, 2008.
24. Bernardo Cuenca Grau, Ian Horrocks, Yevgeny Kazakov, and Ulrike Sattler. Modular reuse of ontologies: Theory and practice. *J. of Artificial Intelligence Research*, 31:273–318, 2008.