Representing Product Designs Using a Description Graph Extension to OWL 2

Henson Graves

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

Abstract. *Product development requires the ability to check design consistency, to verify design properties, and to answer questions about a design's possible implementations. These tasks require inference based on design knowledge. The inference depends on having a representation of the design that is sufficiently precise to capture the design's intended meaning. Description Logics are languages with a formal logical semantics that have been engineered for conceptual modeling. OWL 2 is a W3C standard based on Description Logic. For algorithms that use inference to be guaranteed to produce an answer to a question the design knowledge base must generate a decidable theory. While OWL 2 has a decidable formal semantics it does not provide the expressiveness needed to constrain the possible interpretations of a design to be the intuitively valid physical implementations. A Description Graph Extension to Description Logic can be used to enhance expressiveness and eliminate design models that do not reflect the intended meaning. In a Description Graph extension a product design is represented as a Graph-Extended Knowledge Base (GEKB). A GEKB contains an embedded a description graph, such as a SysML block diagram, which represents the structure of the design. To achieve a decidable theory a product design KB is restricted to represent a Product Line. A Product Line is a top level design with specific variants that refine the top level design. A Product Line KB is decidable and its models have the structure intended for implementations. GEKBs provide a method to integrate SysML into a DL language tailored for product development that has a formal semantics.*

Keywords: Conceptual Model, Description Logic, Graph-Extended DL, Ontology, OWL, Product Structure, SysML.

1 Introduction

Product development requires the ability to check design consistency, to verify design properties, and to answer questions about the design's possible implementations. These tasks require reasoning about a design and its possible implementations. For example, a designer may want to know what changes to a product implementation would be needed for the implementation to comply with some other design

specification. To answer this kind of question requires identifying the commonality and differences of the implementations for both designs.

In a logic-based language with a formal semantics the meaning of a design is captured with axioms. The set of logical consequences of axioms are referred to as the theory generated by the axioms. Among logic-based languages are Description Logics; they have a formal logical semantics and have been engineered expressly for conceptual modeling. Description Logics are descendents of Knowledge Representation languages developed in the 1980s. They have translations into First Order Logic (FOL). Most Description Logics restrict the full expressiveness of FOL to ensure that theories are decidable [1]. Both OWL 1.1 and the proposed OWL 2 are based on Description Logic [2]. As a Description Logic (DL) OWL has a formal semantics and a language for expressing axioms that enable making meaning precise. In OWL the axioms are referred to as a Knowledge Base. OWL is an emerging W3C standard with wide tool support. OWL is being used in a number of application areas. OWL provides a rich collection of class constructions including constructions for Subclass and class Equivalence. While there is general capability for representing structure within OWL [3,4,5,6], OWL is unable to constrain the meaning of a design to correspond to informal notions of valid implementations. Similar limitations were recognized in earlier efforts to use DLs in configuration systems [10]. There are more expressive conceptual modeling languages than OWL, defined for example by adding rule constructions which translate into arbitrary FOL axioms. However, the hesitancy to add rules to DL is that rules often have the effect of making the resulting theories undecidable which reduces their utility for question answering. Question answering depends on having a decidable theory with algorithms optimized for question answering.

System modeling languages, such as SysML [7] represent structure and behavior and design constraints. SysML is an OMG standard wide tool support. SysML has a formal graphical syntax, but does not have a formal semantics. A design is represented in SysML as a block diagram. A block diagram is a directed graph whose vertices are classes and whose edges are relations. The result is that the interpretation of a block diagram depends on domain subject matter experts interpreting the diagrams correctly. SysML design tools do not provide the capability to detect design inconsistency or provide question answering with logical justification. The lack of mechanisms to represent the meaning of a product design independently of subject matter expert (SME) understanding is a cause of cost, schedule overruns and lack of product integrity. When design inconsistency is not discovered early the cost of repair may be considerable. The syntax is sufficiently expressive to constrain the interpretations of the diagram, provided a formal semantics can be given to the diagrams. The SysML block diagram conventions provide the expressiveness needed to rule out unintended interpretations. These well worked out conventions are used to embed SysML within a DL extension.

A Description Graph Extension to Description Logic [8] can be used to enhance the expressiveness of DL and to eliminate design models that do not reflect the intended meaning. A Description Graph Extension of DL uses a Graph-Extended Knowledge Base (GEKB) that contains an embedded a description graph which represents the structure of the design. The description graphs used to describe the structure of a product design, called structure diagrams. Structure diagrams satisfy a

role restriction property which implies that they are description graphs. The design is represented as a GEKB. Additional constraints, such as the sum of weights of parts is constrained by the total weight, are needed to make a design sufficiently precise can be addressed with this extension, but will be addressed elsewhere. The Description Graph Extension to DL provides a method to integrate SysML with OWL into a modeling language with a formal semantics that is tailored for product development.

Not all structure diagrams make good product designs. A product line is a top level design with multiple variants that refine the top level design. A Product Line KB is decidable and its models correspond to the intended implementations. An illustrative design example is worked out to show how to represent a product line design with a specific collection of variants as a GEKB.

1.1 Representing a design and its implementations

We start by reviewing the informal connection between the concept of the meaning of a design with the concept of a design's valid implementations. This discussion allows us to frame the kind of solution that is needed for a modeling language. A conceptual model, such as a product design is represented in both SysML and OWL using a finite collection of class, relations, and individuals. Relations are called roles in OWL and associations in SysML.

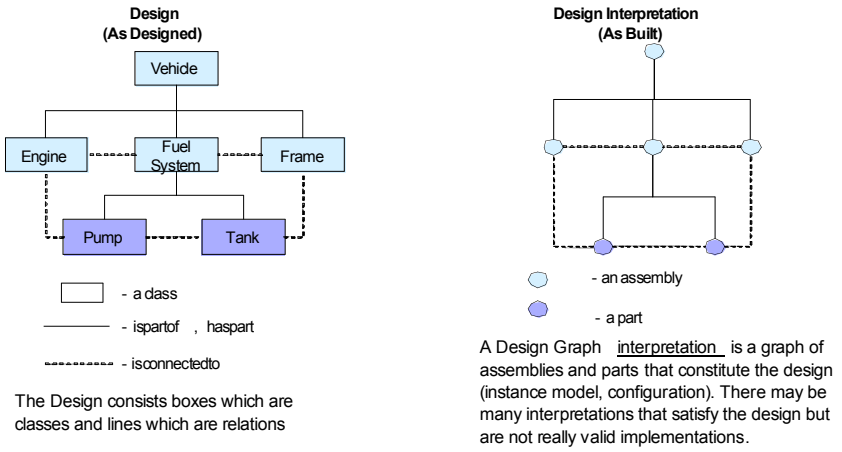


Fig. 1. The left side of the figure illustrates a top level design for a vehicle using a diagram (a SysML block diagram). The boxes represent classes, the lines relations. For the example, the classes in the diagram whose color is Light Blue are systems and the classes whose color is Dark Blue are physical objects. The light blue classes have as instances systems and the dark blue classes have as instances physical components. A "ispartof" relation is used with inverse relation "haspart". The dotted line represents an is connected to relation. The right side of figure 1 illustrates an implementation of the design. The circles are instances of the classes in the design diagram.

In Figure 1 the gray classes are subclasses of a class System and dark blue boxes are subclasses of Part. These two classes have data valued properties that specify

physical, material, geometric, and other properties for a part or system. For the most part the design representation will not deal with specific parts. A design diagram, such as is in Figure 1, describes many things, i.e., has many interpretations. Does this design specify that there must be an engine, or only possibly have an engine? Can an implementation have more than one engine? Unless further conditions more precisely specify the meaning of the diagram we do not know the answers to the questions. An interpretation of a design is a collection of instances for each class and relationship instances for each relation in the design graph. There may be many interpretations that satisfy the design but are not really valid implementations. Additional syntax will be needed to constrain the possible interpretations.

Intuitively an implementable design is one that can be built using parts and assemblies, and can only be built in one way, up to using equivalent parts. An implementation of a design is an interpretation of the design, but not all interpretations are valid implementations. The laws of physics constrain possible implementations and product requirements may prescribe additional constraints that a valid implementation must satisfy. In an implementable design each class corresponds to an assembly or component part. In product development a graph of parts and assemblies is referred to as a design configuration. Each part in an implementation is a member of one of the classes in the design model diagram. A product design may have many or no implementations. A design has sufficient detail when we know how to build it if we have a list parts from a catalog and sufficiently detailed descriptions of assemblies that must be constructed.

1.2 Answering Question from design knowledge

Both designers and implementers use product design representations and auxiliary information to answer questions about a design and its implementations. A designer may want to know:

- What are all of the known specializations of a design?
- Does any specialization have some specific capability?
- What modifications to a configuration are needed to convert it to a new configuration?
- What constraints on subsystems are implied by constraints on system?

An implementer may want to know:

- What are the variant designs, what is a valid parts configuration for a variant?
- Which design variants have a configuration?
- Does a specific list parts constitute a configuration parts list?
- Can a specific tank part be used on two variants?

A design representation can be used to answer questions such as those above correctly provided the design representation captures the intended meaning. The ability of a knowledge base management system to answer questions depends also on having algorithms that can make use of the representation to answer the questions in a timely fashion.

The objective is to find a representation of a structure diagram within a DL KB which is sufficiently expressive to capture the intended meaning of the design.

1.3 Ruling out Unintended Interpretations

In order to successfully answer questions about design implementations requires finding a design representation that rules out unintended interpretations. Then the questions can be answered by constructing the possible interpretations and using them to answer the questions. The design of a product will likely want to rule out having the same instance of a part in two distinct assemblies. One may need to distinguish whether an engine powers the front wheels or the rear wheels [9]. For the design of a hand we may want to rule out that there can be implementations with an arbitrary number of fingers. One may also want to rule out that an implementation can be disconnected; if you chop off a finger then it is not part of the hand. You may not want two fingers to share a joint. SysML has syntax to limit the interpretations of the diagrams. The informal meaning of the SysML constructions will be represented within a DL extension. The examples are presented in an informal graphical syntax rather than formal SysML.

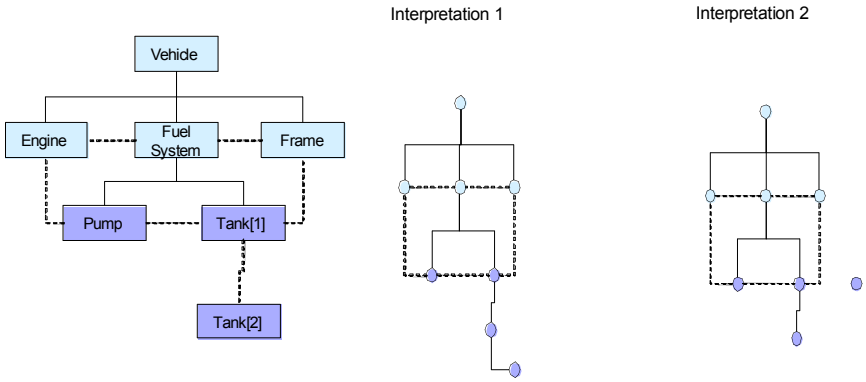


Fig. 2. The top level vehicle design, in the form of a structure diagram, is displayed on the left side of the figure. In the diagram the boxes labeled Tank[1] and Tank[2] are both the same kind of tank. The right side illustrates some possible unintended interpretations of the design diagram. The first interpretation has three tanks; the second had a tank that is disconnected from the rest of the system.

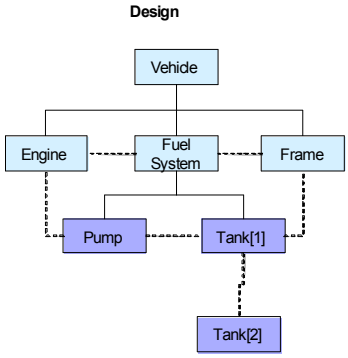
2 A Description Logic Design Knowledge Base

The standard way to capture meaning for a structure, such as a design, within a logic-based language is with axioms. We not only need to represent the graph, but to specify such things as whether exactly one or more fuel tanks are permitted. While we may not know for sure if axioms capture the intended meaning, we can use logic techniques to recognize that a collection of axioms are insufficient. If the axioms have interpretations that do not correspond the intuitively valid notion of an implementation, then the axioms under determine the design. In a Description Logic a collection of axioms is referred to as a Knowledge Base. The theory generated as the

closure of the KB under the language constructions is called an ontology. In DL axioms that involve classes and roles are referred to as the terminological part of the knowledge base, and is called the TBox. A TBox is used to represent usage (axioms) regarding classes and roles. The axioms that involve individuals are referred to as facts and are called the ABox. An Abox is used to represent knowledge regarding the existence of particular parts with specific properties. We use standard DL class constructions together with inverse roles and transitive roles. Role names such as ancestor, has-part, part-of may be declared to be transitive.

2.1 Initial KB construction

We start the construction of a design KB from a structure diagram. Embedding the diagram in a KB allows us to infer inconsistencies, calculate implicit subclass relations and use this information to answer questions about the design. To construct the KB we start with the vocabulary of atomic classes and relations present within the diagram: Fuel System, Tank, Pump, haspart, is-connected-to. SysML classes and relations become DL classes and roles. Our first attempt to axiomatize the Structure Diagram as a KB is below. We will see that the KB is only partially captures the intended meaning of the design and will need enhancements.



Atomic classes:
Vehicle, Engine, Frame, FuelSystem ,
Tank, Pump, …

Atomic relations:
haspart , is -connected -to

Axioms:
Vehicle SubClassof haspart some Engine
Engine is -connected -to FuelSystem
Engine is -connected -to Tank

Fig. 3. The Structure Diagram on the left specifies that the vehicle has two tanks and that they are connected. Axioms on the right capture some of the meaning in terms of the diagram. For example, the axiom "Vehicle SubClassof haspart some Engine" says that Vehicle class is a subclass of things that have an engine.

From the axioms we know that a pump is a part of a vehicle. We do not know how many parts are in an implementation or whether a fuel system has only one or can have multiple pumps.

2.2 Embedding a Structure Diagram in a KB

The structure diagram in Figure 2 reflects a precision not found in the KB representation we have developed so far; for example, it makes it clear that there is only one pump in the system, and that it connects the tank to the engine. To embed the structure diagram directly within the KB while retaining its semantics requires us to make a few extensions to the traditional DL KR to capture the expressiveness of the design graph. In the KB we distinguish the structure class and role symbols from the other KB terms. The table below contains some of the structure diagram class and role symbols. The structure class and roles are designated as being in the structure part of the KB.

Structure Class symbols:

FuelSystem, FuelTank, Pump, Tank[1], Tank[2]

Structure Relations – defined as restrictions:

FuelSystem-haspart-Tank, FuelSystem-connectedto-Pump

The vertices within the structure diagram are unique. A class such as Tank can be used more than once, but we want to identify the usages of each class. Within a Design Diagram each class which occurs more than once is labeled with an index. In figure 4 we have Tank[1] and Tank[2]. If for instance, Tank were itself a Structure Diagram with a class Part then in the expanded Design Diagram Part class would be labeled as Part[1,1] and Part[2,1]. Thus all of the nodes would have distinct labels.

We will use the relation restriction construction to make the relations in the KB corresponding to the edges of the structure diagram unique. The role restriction construction is defined by "domain" and a "range" classes. For a role R and classes A and B, the role construction A-R-B is defined by

for any a, b, A-R-B(a,b) IFF R(a,b) AND A(a) AND B(b)

For any structure relation symbol R, there are structure classes A, B with R = A-R-B. If R = A1-R-B1 then A1 SubClass A and B1 SubClass B. However, A and B are atomic and so A1 = A and B1 = B. Thus R has a unique source and target. The Structure Relations can be viewed as atomic with axioms for the restriction property. The edges of a directed graph have unique source and target vertices. The directed graph property can be checked by a graphical editing system. By defining the structure relations in terms of the restriction construction, the structure relation symbols have a unique source and target class symbol. Note that structure diagram is embedded isomorphically within the KB.

We need further properties to guarantee that the KB models have the intended graph structure. A relation is said to be edge like if there is only one pair (a,b) that is a member of the relation.

for any a,b R(a1,b1) and R(a2,b2) implies a1 = a2 and b1 = b2

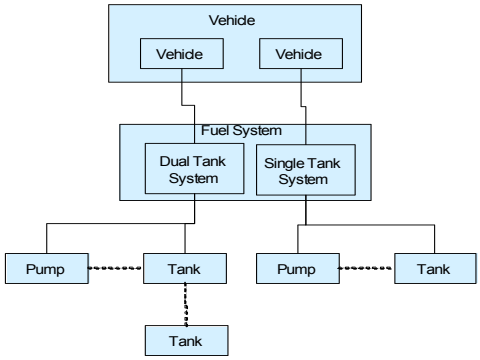
We assume that the relations in the structure diagram have the edge-like property. For a structure diagram embedded within a KB any model of the KB preserves a graph isomorphism of the Structure Diagram.

An embedded structure diagram is a Description Graph in the sense of [8]. The classes in the graph are names and the roles are uniquely labeled. The KB constructed from the structure diagram is a GEKB. We call the Graph-Extended KB constructed from a Design Graph a Product Design KB. A Product Design ontology is the theory generated by a Product Design KB. A model of a Product Design KB is a DL model in which the interpretation of the Structure Diagram is a graph isomorphism. An interpretation of a structure diagram in a DL model corresponds structurally to an implementation. Not all design graphs are the basis for a good design model. A design graph, if it contains loops, may make the ontology undecidable. Ideally one would want a KB whose models have the structure of the structure diagram. However, this is not always the case and we construct KBs models which have this property.

3 Product Line Designs

The objective of the design process is to refine a design so that it is implementable and satisfies its requirements and any design constraints. Many designs are constructed by starting with a top level design and refining it into a finite collection of variants. For example, a vehicle design may contain a Fuel System that is exclusively a Dual Fuel Tank or a Single Fuel Tank. The Fuel System is an abstract class that is not instantiated other than through its partition into Single and Double tanks, and so does not occur in the design graph. The result is a description graph with two disjoint subgraphs corresponding to the one and two tank design variants. In a product line KB the branching conditions for variant alternatives are expressed using OWL axioms. The combination of axioms that describes the design alternatives and the embedded structure diagram ensures that the interpretations of the GEKB have exactly the diagram structure.

We elaborate the structure diagram in Figure 1. Fuel System is refined to specify two alternatives, one with a single fuel tank and one with a dual tank. Dual Tank Fuel System and the Single Tank Fuel System are subclasses of Fuel System. The diagram represents alternatives, but can introduce disconnected interpretations.



DualTankSystem
Subclassof 2 haspart
AND haspart some Tank1
AND haspart some Tank2

FuelSsystem
SubClassof
DualTankSystem DISJOINTUNION
SingleTankSystem

DuelTankSystem
SubClassof 2 haspart AND haspart
Tank1 AND haspartTank2

Fig. 4. The left side shows an elaboration of the Fuel System design from Figure 2. The right side shows an interpretation that consists of the instance graph on the right. The class construction Disjoint-OR is used to express the axiom that the fuel system has alternatives, DTS and STS. This description ensures that the diagram only connected interpretations. Note as a result Fuel System is an abstract class. Any instances of Fuel System, only of Dual Tank System and Single Tank System

A Design Variant is a refinement of the top level design with no alternatives. Every time a variant is introduced classes are partitioned using a disjoint-or. The graph is a disjoint union of the variant sub-graphs. Call this a product line design. The models of a Product Line Design satisfy the key, disjoint, start, and layout conditions. The models of a Product Line Design KB are a collection of instances of the variant designs. Each structure diagram instance is connected and isomorphic to the structure diagram itself. Thus the models correspond to implementations and we can answer questions by model-theoretic reasoning.

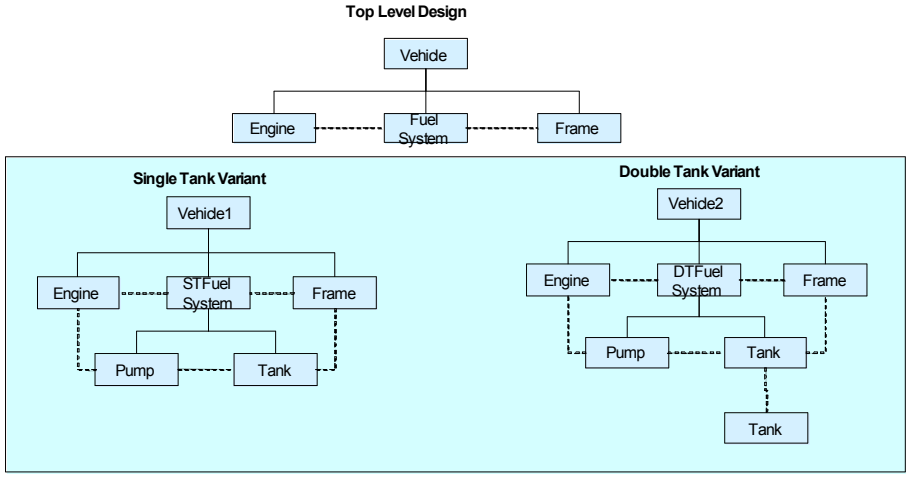


Fig. 5. The models of this Product Line Design KB prototype each variant. Each model of KB

contains directed graphs that prototype each variant. The variant instance graphs are disjoint and are connected as specified by the Design Graph of Figure 3.

Since the structure diagram and its KB models all have the same finite graph-theoretic structure, we can use rules to express constraints and answer questions. For example, we can express a constraint such as the weight of a vehicle is less than the max of the weight of its parts. We can define a rule that expresses that the weight of a vehicle is the sum of the weights of its parts and use this rule to calculate the vehicle weight from the weights of parts.

4 Conclusion

While formal results are not presented here, the concept of a structure diagram and the construction used to embed the graph into an extended KB are similar to [8]. A structure diagram has the properties of a Description Graph which allows us to use the results on Description Graph extensions to DL. The structure diagrams are designed to ensure that the KB models preserve exactly the graph structure. Design Variant graphs have the properties described in [8] which ensure that the extended KB is decidable. The next step is to give a complete detailed proof that SysML block diagrams are Description Graphs and that there is a formal embedding of that part of SysML into a DL.

References

1. Horrocks, I, *Ontologies and the Semantic Web*. Needham Lecture, 2005. Available at http://www.epsg.org.uk/pub/needham2005/Horrocks_needham2005.pdf
2. Patel-Schneider, P., Hayes, P., and Horrocks, I., *OWL Web Ontology Language semantics and abstract syntax*. W3C Recommendation, 10 February 2004. Available at http://www.w3.org/TR/owl-semantics/
3. Price, D., Bodington, R., *OWL, description logic and systems requirements satisfaction*, Proceedings of 8th NASA-ESA Workshop on Product Data Exchange, 2006.
4. Graves, H. *Design refinement and requirements satisfaction*, Proceedings of 9th NASA-ESA Workshop on Product Data Exchange, 2007. Available at. step.nasa.gov/pde2007/Design_Refinement_and_Requirements_Satisfaction_LMCO-HGraves.pdf
5. Graves, H., Ontology engineering for product development, Proceedings of the Third OWL Experiences and Directions Workshop, 2007. Available at www.webont.org/owled/2007/PapersPDF/submission_3.pdf
6. Graves, H., Horrocks, I., *Application of OWL 1.1 to Systems Engineering*, OWL Experiences and Directions April Workshop, 2008.
7. OMG Systems Modeling Language (OMG SysML™), V1.0, http://doc.omg.org/formal/07-09-01.
8. Motik, B., Grau, B., Horrocks, I., Sattler, U., *Representing Structured Objects using Description Graphs*, AAAI, 2008.
9. Bock, C., *UML Composition Model*, Journal of Object Technology, vol. 3, no. 10, November-December 2004, pp 47-73.

10. Deborah L. McGuinness, Jon R. Wright: Conceptual modelling for configuration: A description logic-based approach. AI EDAM 12(4): 333-344 (1998).