Integrating Reasoning with SysML

Henson Graves
Algos Associates
henson.graves@hotmail.com
2829 West Cantey Street, Fort Worth, Texas 76109

Abstract. Many engineering tasks employ reasoning. For some tasks automated reasoning is feasible and high leverage for solving difficult practical problems. Engineering questions can be represented as questions about a model provided the model contains sufficient domain knowledge. By embedding a model as an axiom set within a suitable logic, engineering questions translate into questions about axiom sets. Automated reasoning can potentially be used to answer these questions. Examples of engineering questions that can be represented as logic questions include verification of a system capability (or a requirement satisfaction) and verification of whether a design change invalidates design constraints. Results for embedding the class diagram fragment of SysML into OWL are extended to cover other SysML constructions. Examples are given to relate a variety of engineering questions to axiom set questions and illustrate how formal reasoning can be exploited to answer questions.

Introduction

Engineering models in languages such as SysML (OMG SysML 2008) are used to represent, specify, and analyze systems. The integration of modeling with reasoning provides the potential to use models to answer engineering questions. The molecular unit of SysML is a model. The direct way to integrate reasoning with SysML is to embed a model as an axiom set within a logical system. Of course the formal semantics of the logical system has to be in accord with the informal semantics of SysML. Otherwise reasoning may give incorrect results. The issue of representing engineering questions as logical questions is straightforward provided the model has been properly designed. For a properly designed model, an engineering question translates directly into a logic question. The principles for constructing models to be used to answer questions apply even when automated reasoning is not used.

The paradigm of representing engineering questions as questions about axiom sets has been around for a long time (Graves and Horrocks 2008). The direct approach of constructing an axiom set to represent an engineering problem has been adopted for specialized classes of problems, such as verifying safety critical software. However, directly constructing axiom sets to represent engineering problems is not in widespread use. Practically it has proven very difficult for engineers to develop an axiom set to describe a problem. Constructing a SysML model rather than an axiom set requires less specialized training than does learning to construct axioms within a logic. Even though the axiom set and the model have the same information, the language for modeling is an efficient way of constructing an axiom set and opens a practical path to integrating reasoning with engineering development.

This paper illustrates how reasoning can be used to answer a variety of engineering questions. Examples are given of how models are developed to answer engineering questions. These examples are used to show how reasoning about an axiom set answers the original question. Analysis of the axiom set resulting from a model provides useful information with regard to

the model's intended use. For example, if the model is intended as a detailed design, then one likely wants all of the valid implementations to have the same structure. Results on embedding SysML models into a logical system (Berardi, at al. 2005)(Graves and Bijan 2011)are used to illustrate how engineering questions translate into questions about axiom systems which are amenable to automated inference.

Engineering questions that can be represented as questions about models include questions of system capability and design solution verification. For example, can an aircraft under specified operating conditions loiter in an area for a specified time duration. The aircraft may be an existing aircraft, a proposed design, or a design modification. The engineering question is answerable in the context of the assumptions and constraints on the aircraft operation. If we build a SysML model which contains all of the domain assumptions, then the question is equivalent to whether the axiom set corresponding to the model logically implies the loitering conditions. While including domain assumptions and design constraints requires effort, a strong case can be made to include this information in the application model even if automated reasoning is not used. Overlooking these assumptions is where most design errors occur.

Figure 1 is a schematic diagram that shows the relationship between a SysML model and an interpretation. In this case the model represents an air system and environment; the interpretation corresponds the model to elements of some domain. The interpretation of the model shows an aircraft and a region of terrain as seen through a sensor display on the aircraft. The operator on the aircraft is attempting to identify a target from the sensor display.A model may have multiple interpretations, where an interpretation is anything that satisfies the model. Building a model for a product does not itself guarantee that any implementations of the model exist. In logic, the use of the term "model" is reversed from its use in engineering: In logic a model is an interpretation ofan axiom set. However, the concepts of interpretation in engineering and model in logic are similar.

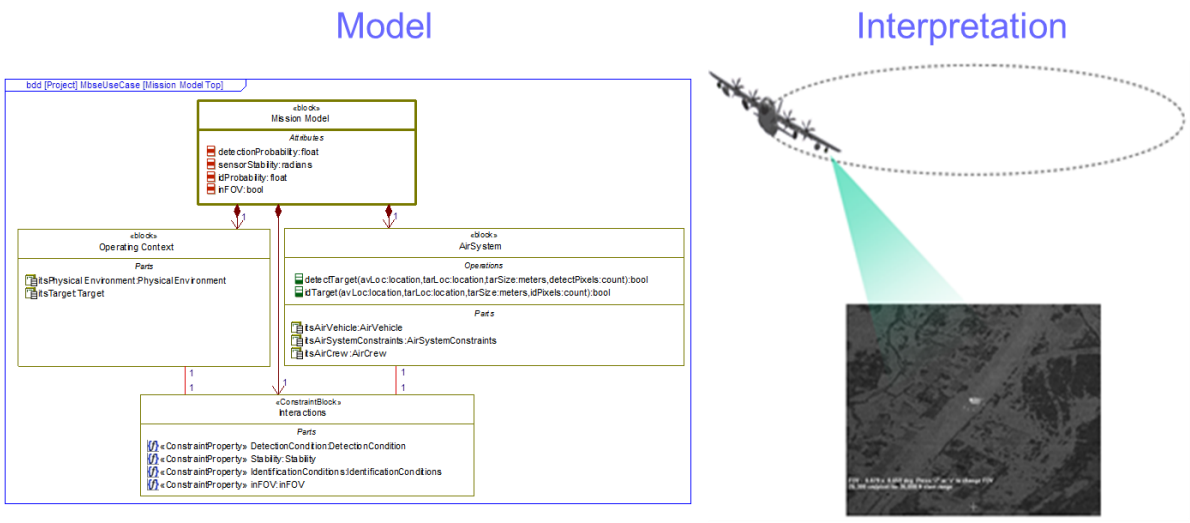

Figure 1. Relationship Between Model and Interpretation

Logical systems typically have an inference semantics and a reference semantics. An inference semantics is given by rules which can be used to derive conclusions from axioms, i.e., other true statements. The reference semantics for an axiom set describes a class of valid

interpretations. The axioms are by definition true in any valid interpretation. Of course an axiom set may not have any valid interpretations. The inference rules are sound if a statement derived from the axioms is true in any valid interpretation. The inference rules are complete when all statements true in the reference semantics can be derived from the axiom set. The reference semantics for a SysML model is an interpretation of the model. An interpretation is a collection of real or imaginary (simulation) domains corresponding the SysML types with relations between the domains corresponding to the properties in the model. Any inclusion relationships between the SysML block are satisfied by the interpretation in the domain of the blocks. The logic notion of a valid interpretation corresponds with the informal reference semantics of SysML. When a model is embedded as an axiom set within a logic questions about model translate to questions about the axiom set. For example is the model consistent becomes is the axiom set consistent and can a component or relationship be added to the model without making it inconsistent becomes the logical question of whether adding the statement corresponding to the model change make the axiom set consistent.

Some of the candidate logics used as targets for embedding SysML are noted in the Venn diagram in Figure 2. In each of these cases a SysML model is encoded as an axiom set within the language of the logic. The axioms take different forms in OWL and in FOL. In OWL the axioms are class inclusion assertions. In FOL the axioms are formulas defined in terms of binary and unary predicates, i.e., predicates with two or one argument. The diagram in Figure 2 uses lines to indicate the embedding of SysML class diagrams in both OWL (OWL 2 2008) and in First Order Logic (FOL) and correspondences between the two different forms of logic.

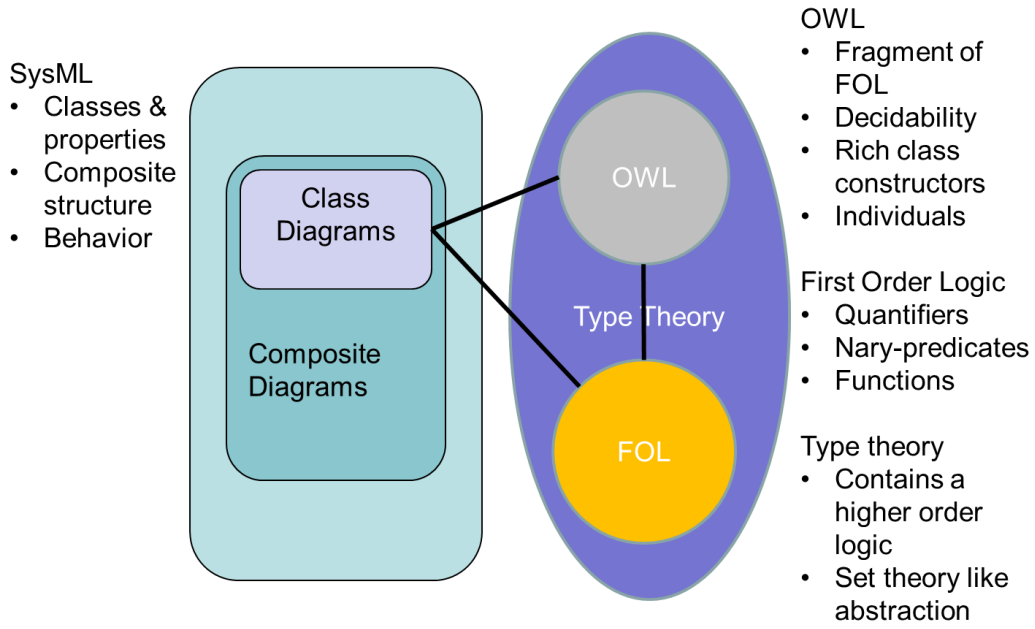

Figure 2. Relationship of Candidate Logics

Syntactically SysML and OWL languages have a lot of syntactic overlap. SysML uses a graphical syntax with elements for blocks, associations, part properties, and subclass relationships between classes. OWL2, the most recent and precise version of OWL, as a conceptual modeling language uses the paradigm of representing knowledge as an axiom set (model in engineering terms) and using reasoning on the axiom set to answer questions. The

OWL2 language uses classes and properties. The formal foundation for OWL2 (Horrocks, et al. 2006) is SROIQ which is a Description Logic (DL) (Baader at al. 2010). SysML blocks and associations correspond respectively to OWL2 classes and properties. When SysML is embedded in OWL2, model questions become axiom set questions. A fragment of UML, called class diagrams has been embedded within a Description Logic (DL). The results hold as well for the corresponding fragment of SysML. Class diagrams use only blocks and binary associations. The DL is known to correspond to a fragment of First Order Logic. Class diagrams, a fragment of SysML, can be embedded into DL, the foundation for OWL2. This embedding provides the integration of a fragment of SysML with OWL reasoning. In OWL this would be a decidable problem provided the axiom set were an OWL2 axiom set. Logical implication problems such as the loitering question can be handled within a logic that extends OWL, but is consistent with the class diagram embedded within OWL2.

Many language constructions essential to SysML's success are not covered by the embedding in [BER] as they are not class diagrams. One class of examples not covered by the class diagram embedding are structure diagrams. Structure diagrams are common in engineering and science. The Berardi results do not cover structure diagrams or more generally composite structure. Finding the kinds of restrictions on axioms in which to embed an IBD and for which logical services are decidable has presented a challenge. Description Logic (DL) is a natural candidate for structural modeling. However, the ability to represent graph structures such as can be constructed with SysML Internal Block Diagrams (IBDs) is beyond the capability of OWL2. SysML models of structures can be translated into the language of SROIQ (Horrocks, et al. 2006) However, the resulting axiom sets contain role equalities that do not satisfy the constraints of SROIQ axiom sets. OWL2 with a Description Graph (DG) extension has been proposed as a candidate for structural modeling. Analysis of application-use cases suggest that often the DL axioms, as well as the DG extension axioms, do not correctly capture the properties of the structures being modeled. A DL, called structural DL (SDL), is described for representing structures found in science and engineering. An approach to characterizing The characterization of a SysML IBD as axiom set within a Description Logic has been described in (Graves and Bijan 2011). A proof of the decidability of satisfaction for these axiom sets is in draft form.

Other language constructions used in engineering which are not class diagrams are operations with arguments that are declared within a block. Dynamic behavior and time are also not included. Some of the examples illustrated here are outside the scope of class and structure diagrams. The full potential for integration of reasoning with SysML requires finding a way to build on and extend these results to embed SysML models as axiom sets within a richer logic where automated reasoning is still feasible. The recognition that classes and properties correspond to unary and binary predicates plays a major role in logic-based foundation for these languages and in the development of reasoning engines. (Berardi et al. 2005) introduces a First Order Logic (FOL) embedding for class diagrams and use this embedding to justify the correctness of the embedding into DL. The correspondence between the DL and the FOL gives insight into how the informal semantics of SysML corresponds with the formal logic-based semantics. It follows from the embedding that decidable inference services are available for the class fragment of SysML.

The results embedding a class diagram in a DL have been extended using a logical system called type theory (Lambek and Scott, 1986) can be used as the target embedding for SysML. Type theory contains both classes and formula and a built-in correspondence between classes and properties with unary and binary predicates. The static part of SysML has been

embedded into type theory (Graves and Bijan, to be published). The extension of the embedding of class and structure diagram fragments of SysML is needed to answer questions such as the aircraft loitering question. The same kind of reasoning used in OWL2 can be used to determine whether the resulting axiom set is consistent or inconsistent, which gives an answer to the engineering question, provided that decidability results hold.

Representing Engineering Questions as Model Questions

This section introduces an engineering question and sketches how a SysML model can be developed to assist in answering the question. The example is taken from (Graves and Bijan, to be published).A general method for embedding a SysML model within a formal logical system is outlined and is used to show how consistency checking of the resulting axiom set can be used to answer the engineering question. More detail on how the embedding works is given in subsequent sections.

A realistic engineering question is whether a specifically equipped model of aircraft can perform target identification under specified operation conditions. For simplicity we assume that the answer is true or false. The approach outlined extends to probabilistic statements. In engineering practice the answer to a question, such as the loitering question, uses analysis based on engineering models and possibly simulation. The quality of the answer depends on the accuracy and detail of the engineering models used. A question such as the loitering question has to do with system interaction with its operation environment. By constructing a SysML model that contains a model of the aircraft and the target, the physical environment specified by the operating conditions enables the assumptions to be explicit. **Figure 3** illustrates a top level diagram for a model of air vehicle and its operating context. The model has a top level block, called *MissionDomain*. *MissionDomain* contains components for the aircraft, the physical environment, and the target. By using a single composite model there is less risk that inconsistent assumptions are used. Top level assumptions and constraints about how the subsystems interact with each other and the environment are represented with constraint blocks that are part of the *MissionDomain* block.

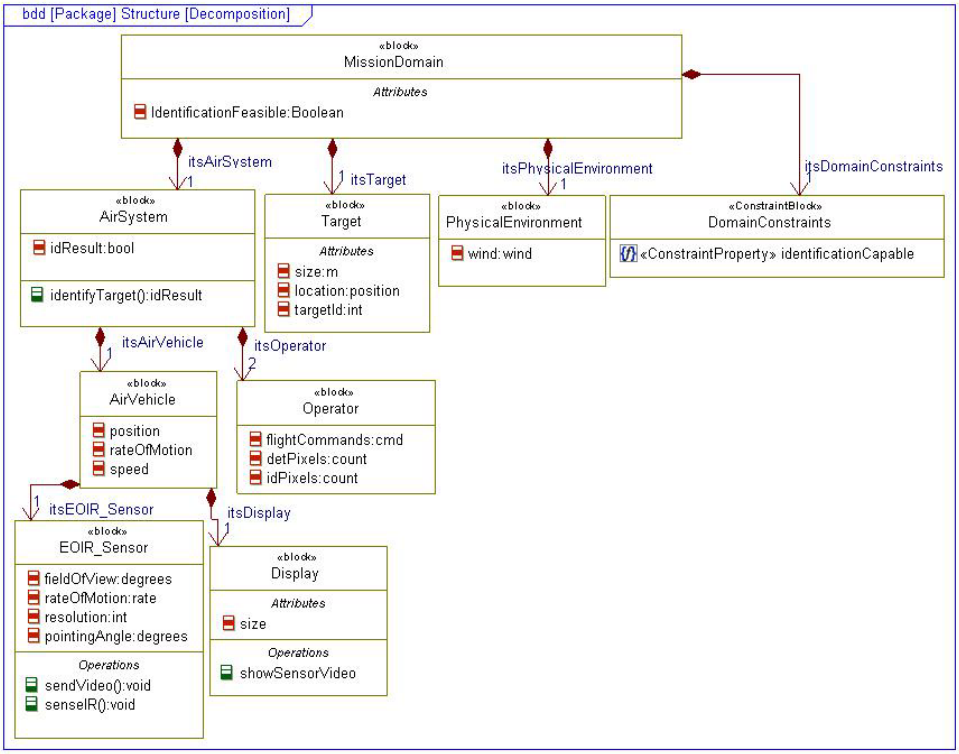

Figure 3. A Block Diagram for the Mission Model

By using SysML constraint constructions operating conditions, flight dynamic models, electro-optical models of sensors, and other factors that influence the answer can be represented as component models. To answer the target identification question new blocks (classes) corresponding to these diagrams need to be defined.

The diagram in Figure 4 is a SysML parametric diagram. The diagram uses a dynamic model for the aircraft motion. The motion is a functional relation of the flight commands and the wind in the operating context. The diagram contains three rectangles labeled *Operator*, *Physical Environment*, and *Air Vehicle*. These represent blocks. The small rectangles inside represent value properties. The rounded corner rectangle in the diagram represents a functional relation which specifies the air vehicle motion as a function of the operator commands and the wind.

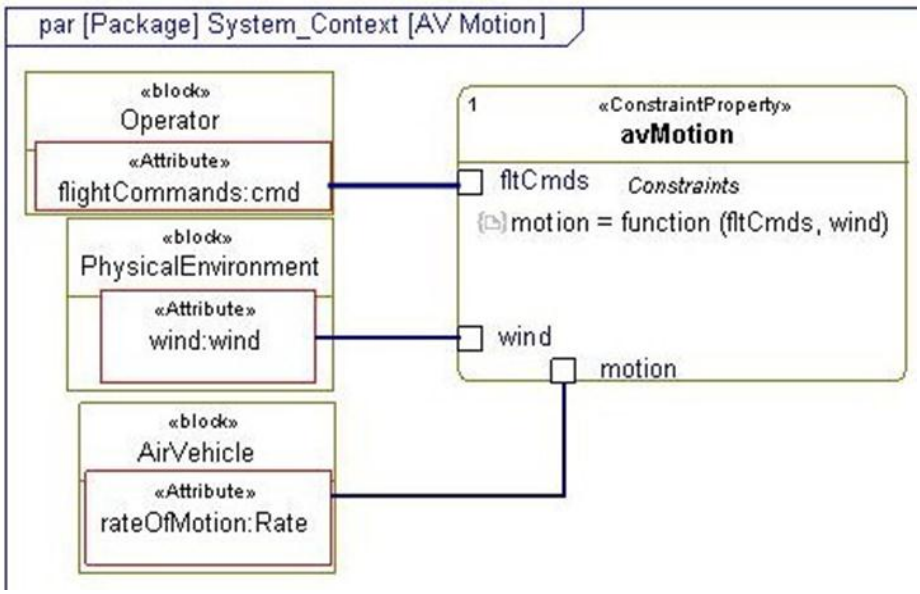

Figure 4.

SysML models contain variables in the form of block value properties. Conceptually the value properties are the state variables of the block. For example, the loitering conditions can be expressed as a Boolean valued operation whose arguments are variables related to the effects of altitude on operating performance and other constraints. SysML constraint properties express equations that can be bound to value properties. The variable values which satisfy a constraint equation define a subtype of the type of tuples that satisfy the equation. For example, in Figure 4 the value properties *flightCommands (fltCmds)*, *wind*, and *rateOfMotion* are variables with respective types, *cmd*, *wind*, and *Rate*. The parameters *fltCmds, wind,* and *motion* in the constraint property block are bound to the value properties. Informally the parametric diagram specifies a functional relation which defines a subtype of the product type of the three types, *cmd*, *wind*, and *Rate.* This type can be written as

$$\{ \langle x,y,z \rangle : z = fn(x,y) \}$$

where <x,y,z> is a 3-tuple and *x*, *y*, and *z* are variables with respective types *cmd*, *wind*, and *rate*, and *fn* is a function. This class is not definable within SysML but within the logic in which SysML is embedded. The class is defined using a set-like abstraction construction.

The SysML mission model contains a lot of information not shown in Figures 3 and 4. This information includes models of aerodynamic performance of the aircraft, the effect of aircraft motion on the stability of the sensors, and further detailed decomposition of the air system. The aircraft has a target identification operation defined in terms of operations of the aircraft's system components such as a sensor. For simplicity we assume that *identifytarget* is a composition of functions defined for the aircraft components.

From the mission model the totality of the value properties that occur in any of the blocks are variables in the logic. They represent the state space for the application. Schematically we let *<x1,…xn>* be a tuple of the variables that occur in the mission model. Then the operating conditions can be represented as

$$\{\langle x1,\ldots xn \rangle : idFeasible(x1,\ldots xn) = true \}.$$

For example, if *operatingConditions* is an equation defined for the variable types of the mission domain model, then using a set construction as a class construction, the constraint defines the class as

{ <x1,…xn> : operatingCondition(x1,…,xn) = true }.

Conversely a subtype *A* of *X* has a characteristic predicate (Boolean valued operation) *A^* which has the property that for any tuple of individual *ai:X*,

<a1…an>:A iff A^(a1,…,an) = true.

The correspondence between classes and formulas can be used to characterize exactly when an axiom set is consistent. A class diagram is consistent if it admits an instantiation, i.e., if its classes can be populated without violating any of the requirements imposed by the diagram. A class *C* is satisfiable in a class diagram if it is not equivalent to *Nothing*. When the diagram is not consistent, then *Nothing = Thing*. Of course for arbitrary type theory axiom sets, the decidability properties are lost. However, for some cases correspondence between classes and formulas can be used to reduce subclass assertions to be DL like. In the SysML to type theory embedding, axiom set consistency is equivalent to the satisfiablility of the formula that represents the conjunction of the axioms.

The formalized requirements are a subtype of *MissionDomain*

{x : idFeasible(x) = true }

consisting of those tuples which satisfy the further constraint *identificationCapable*. We introduce the name *MissionDomain{identificationCapable}.* If we assume requirements are consistent

{x : idFeasible(x) = true } $\neq$ NoThing.

Note that if

{x : idFeasible(x) = true } = NoThing,

nothing can satisfy them. The type

{ x : correct(identifytarget(x)) = true }

is defined where *correct(identifytarget(x))* is the result of evaluating the target identification operation. The question of whether the *identifytarget* operation gives the correct answer can be written as

{ x : idFeasible(x) = true } $\sqsubseteq$ *{ x : correct(identifytarget(x)) = true }.*

By using the subtype construction we have implicitly embedded the SysML model as an axiom set within type theory. Type theory has an abstraction construction which enables one to define subtypes from relations (formulas). These axioms can be equivalently expressed in FOL.

The problem is a logical implication problem: Do the mission model axioms imply the loitering conditional? If the augmented axiom set is inconsistent, then the loitering statement cannot possibly be true with respect to the mission model. Examples of whether a design change introduces inconsistency work similarly. The correspondence between classes and formulas can be used to characterize exactly when an axiom set is consistent. A class diagram is consistent if it admits an instantiation, i.e., if its classes can be populated without violating any of the requirements imposed by the diagram. The corresponding formula admits a model in which at least one class has a non-empty extension. When the diagram is not consistent, then *Nothing = Thing*.

To attempt to prove this assertion, one typically uses assumptions of the form

$$A \sqsubseteq \{ x : correct(identifytarget(x)) = true \}$$

to attempt to show

$$\{ x : idFeasible(x) = true \} \sqsubseteq A.$$

This gives the desired conclusion.

We have modelled an application, seen that the answer to the capability question is equivalent to whether a class is equal to Nothing, i.e., is empty. We have suggested what a proof looks like.

The embedding of the model into type theory that we have used so far uses several principles:

1. Value properties of components of the model correspond to variables in the logic.
2. The logic contains product types corresponding to the types of the value properties.
3. For each formula with variables *x1,…,xn*, we can define a subtype of the product which are the tuples of values for which the formulas are true.
4. Parametric diagrams correspond to truth-valued operations in their variables.

The correspondence that we have used between classes and unary predicates (and properties and binary predicates) holds between DL classes and FOL predicates. However, it holds in type theory which can be used to extend the ideas of DL reasoning to a much larger fragment of SysML. In the following we show how DL class constructions can be used to represent further kinds of assertions.

Class Diagrams. This subsection shows how specific SysML language constructions are embedded as both formulas and as DL assertions. Multiple authors have observed blocks, associations, and specialization correspond directly to OWL classes, properties, and subclass relationships. Berardi (Berardi at al 2005) uses the FOL representation to explain the semantics of class diagrams. Both the formula and the DL representations are equivalent in that from one the other can be derived. In the process we introduce new DL class constructions that are needed for embedding SysML models. These constructions will be used in the reasoning examples. By the correspondence principle between DL and FOL we use the symbol ^ for the predicate corresponding to concept or role.

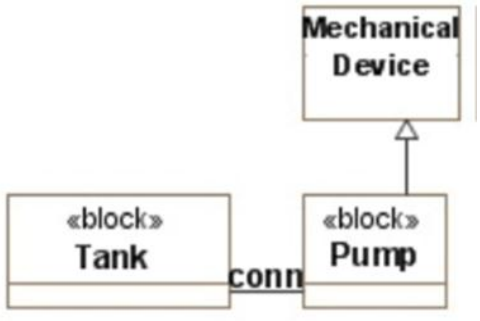

Figure 5. A Class Diagram

The diagram in Figure 5 is a simple kind of model called a class diagram. A class diagram is a SysML model which only uses blocks and binary associations, with subclass relations between blocks. The diagram has three blocks, *Pump*, *MechanicalDevice,* and *Tank*. The arrow with the triangle head indicates that Pump is a subtype of *MechanicalDevice*. The translation of the block inclusion embeds directly as the DL assertion

Pump ⊑ MechanicalDevice.

Informally, the diagram means that any pump is a mechanical device. In the correspondence between DL and FOL, classes are embedded as unary predicates, and roles are embedded as binary predicates. The diagram does not reference specific individuals, only concepts or classes of individuals. If one has an instance of the conn relation between a pump and another individual, then that individual is a tank. If one views the predicate *Pump^* and *conn^* as recognition procedures, this is what the FOL axioms say. Using the correspondence principle inclusion *Pump ⊑ MechanicalDevice* is translated as

∀x. Pump^(x) implies MechanicalDevice^(x).

Note that this formula is a statement that uses individual variables and universal quantification. In DL the correspondence principle is informal. In type theory the correspondence principle is part of the logic. For example, if a formula is defined for individuals of type X, which we write as *f(x:X):Bool*, then we can form the abstraction type as *{ x : f(x) = true }*. Conversely, for a type *Y* which is a subtype of *X*, we can form its characteristic Boolean valued operation, *X^* where *X^* is typed as *X^(x:Y):Bool* which as the property that *X = { x : x:Y and^(x)} = true }.*

The informal semantics of arrow in the diagram labeled *conn* from Pump to Tank is that when a pump is connected via conn, then the pump is connected to a tank. This translates as

∀x. Pump^(x) implies ∀y.conn^(x,y) implies Tank^(y),

which provides a formula semantics for the connection arrow. By using a DL class construction, called value restriction, the formula can be translated into DL as the equivalent class inclusion

Pump ⊑ ∀conn.Tank.

By using type theory the value restriction type can be defined as

$$\forall conn.Tank = \{ x : (\forall y.conn\hat{}(x,y) \text{ implies } Tank\hat{}(y)) = true \},$$

which makes clear the correspondence between the class $\forall conn.Tank$ and its corresponding formula

$$\forall y.conn\hat{}(x,y) \text{ implies } Tank\hat{}(y).$$

Structure Diagrams. The approach used for class and properties can be extended to other constructions found in SysML block diagrams. Figure 6 contains two diagrams which are views of the same model. A water molecule has three parts, an oxygen atom and two hydrogen atoms. The oxygen atom is connected to each of the hydrogen atoms with a bonding relation.

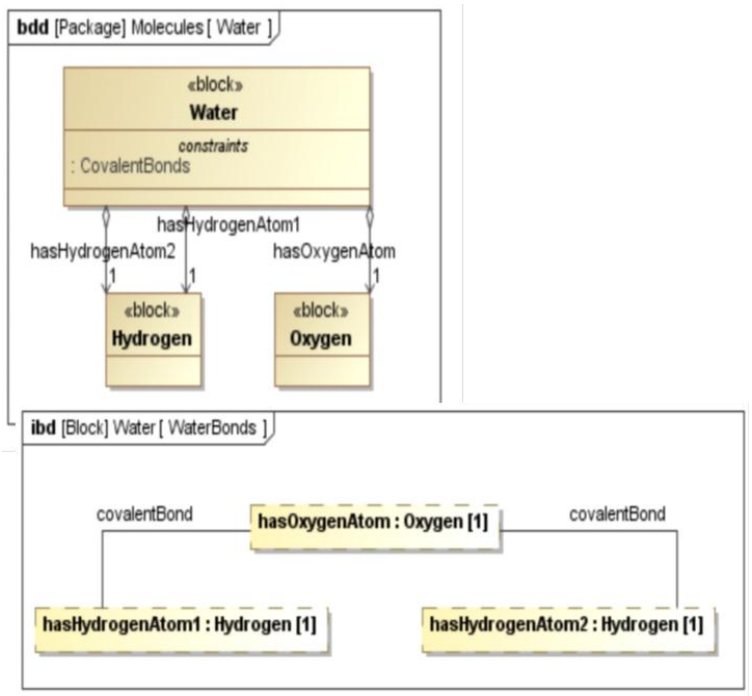

Figure 6. A Model for the Water Molecule

The BDD shows that the water molecule has three part properties. Each has a multiplicity of 1 which means that any water molecule instance has exactly 1 of each of the three parts. The IBD shows internal connections between the three part properties. In the IBD the rectangles represent part properties rather than blocks. For example, the *hasOxygenAtom* part is connected via the *covalentBond* property with the *hasHydrogen1* part property.

The formula representing the *hasOxygenPart* property relation with water is expressed with

$$\forall x \exists! x.Water\hat{}(x) \text{ implies } hasOxygenPart1\hat{}(x,y) \text{ and } Oxygen\hat{}(y).$$

This says that a water molecule necessarily has exactly one oxygen atom. The formula can be represented within DL using the existential type construction with

$$Water \sqsubseteq \exists Oxygen\hat{}(x.\ hasOxygenPart1\hat{}*).$$

The Water model also has two covalent bonds. The informal semantic for the arrow in the IBD between *hasOxygen:Oxygen* and *hasHydrogen1:Hydrogen[1]* is that for any water molecule its oxygen atom part is bonded to its hydrogen1 part. We can represent this as a formula. The easiest way is to make use of the fact that the part properties are functional and replace a part property *p* with a function *p**. For *x:Water* we write *x.p** for the value of the function. Using this replacement we have

$$\forall x.\ Water\hat{}(x) \text{ implies } x.hasOxygen\hat{}*.connectbond\hat{} = x.hasHydrogen1\hat{}*.$$

This formula can be translated into a DL if we make an extension to standard DL. The extension is to introduce a new class construction, called an equalizer. If we use this construction then the formula translates into the class inclusion

$$Water \sqsubseteq Water\{hasOxygen.connectbond1, hasHydrogen1\}$$

where the equalizer can be defined in type theory as

$$Water\{hasOxygen.connectbond1, hasHydrogen1\} = \{ x : (Water\hat{}(x) \text{ implies } x.hasOxygen\hat{}*.connectbond\hat{} = x.hasHydrogen1\hat{}*) = true \}.$$

Of course to expect to do reasoning with constructions like this, one must verify that for the axiom sets which use this construction, consistency is decidable. However, as will be shown elsewhere, this fact is true for the extended DL. These diagrams are called Structure Diagrams.

Using Reasoning

One place where reasoning can be used is to check design consistency as it is developed. For this to work the models have to contain the constraints as assumptions. Of course developers are not yet in the habit of including constraints as part of their models. However, design inconsistency resulting during design development is one of the major reasons for rework, additional time and money even though it is a bit hard to document. The way in which design models are typically developed is by adding components and making connections between components.

When adding a component to a system which has to be connected to another system such as a hydraulic or electrical system, there is the potential that the new connection may violate constraints regarding admissible connections to the hydraulic system, for instance. The violation of a connection constraint may not be apparent from simply making the connection to a component of the hydraulic system. Using DL constructions, connection constraints may be expressed within the model and the consistency of the connection constraints checked when changes are made. In the following example the constraint on the hydraulic system is that the block Source is constrained to have at most three connections. A connection is not just a direct connection of a block to Source, but any path (composition) of connections from a block to Source.

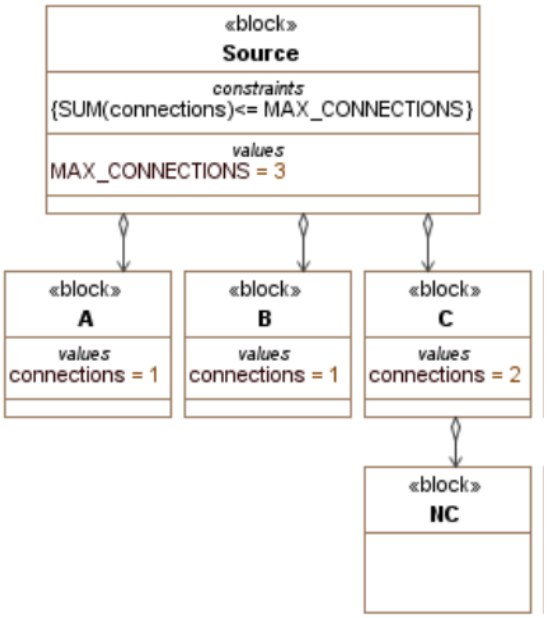

Figure 7.A Diagram Whose Connections Violate a Design Constraint

Figure 7 is a diagrammatic representation of a SysML Block Definition Diagram (BDD) that contains five blocks, each of which is a system component. The Source block has a constraint which specifies that the sum of all connections must be less than the maximum of 3. The connections from *Source* are not just the direct connections to *A*, *B*, and *C*, but any path connections formed by composition of connection properties.

A variant of the problem of adding a component that invalidates design constraints is adding a component that has its own constraints for its use, making it incompatible with its inclusion as a component of a larger model. For example, a model of a pump may have been developed for use as a component of a specific kind of assembly. When it is used in the assembly that it was developed for, it is always connected to a specific kind of valve. However, the pump model may be incorporated into other kinds of assemblies where the original assumption is not needed and may be inconsistent with the new usage.

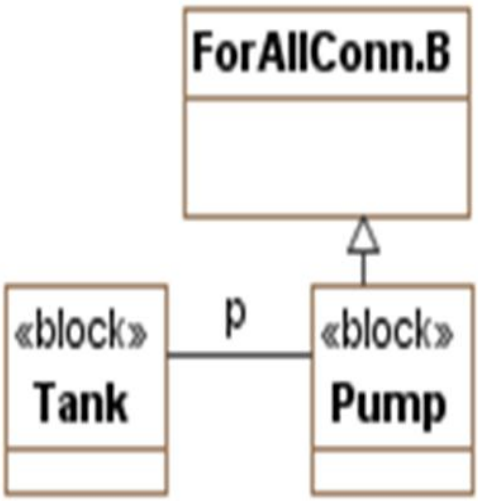

Figure 8. A Pump Model Which Contains Constraints on Connections

The pump model in Figure 8 expresses the constraint that it can only be connected to components of type *B* with the subclass relation

$$Pump \sqsubseteq \forall conn.B.$$

If this model is imported to be a component of a larger model, and one attempts to connect *Pump* to a block *A* where *A* and *B* are disjoint, then the connection violates the composite model as *A* and *B* are disjoint. To use the pump model the assumption in the pump model must be modified.

The kinds of reasoning used in the design consistency and the target identification problem are different. In the design consistency problem the abstraction construction was used to define subtypes of a product type of type *Number*. The arguments for inclusions such as occurred within the target identification verification required more numeric analysis and computation. In general, these kinds of problems are not in a fragment of SysML where the entire problem is algorithmically decidable. However, when the component inclusions are verified, which may take considerable mathematically oriented possibly non-automated reasoning, the final result can be verified from the component inclusions.

Conclusion

The use of reasoning to answer engineering questions has been illustrated with three examples. The first was a capability verification, a verification that an aircraft model can perform a specific function. The verification is contingent on the validity of the context model with respect to the real world domain. The second two examples are variants of the development problem of maintaining design consistency by verifying consistency of design change. The examples show how reasoning can be integrated with SysML. One lesson implicit in the discussion is that one can and should include assumptions and constraints in models even if automated reasoning is not used.

The reasoning in the examples is justified by a semantic embedding of a fragment of SysML into a type theory logic. While some SysML models can be embedded as axiom sets within OWL2, many important SysML language constructions, such as an IBD cannot be embedded within OWL. However, important fragments of SysML can be embedded within a DL extension in which deterministic reasoning is possible.

Analysis of the examples suggests that a number of additions to the SysML language would be useful. These include adding:

- DL class and property constructions
- Individuals
- “Function call” to block diagrams
- Abstraction.
-

A bolder step would be to use an engineered version of type theory as the foundation for SysML. The engineered type theory could be part of the SysML specification. To facilitate the use of reasoning with SysML, a formal semantics is needed as part of the SysML specification. Type theory provides the language extensions suggested by the examples with

a formal semantics well adapted for use with inference engines. One of the next steps to extend the use of reasoning for SysML is to find axioms for behavior that are amenable to mechanical inference. Again type theory is well suited for representing behavior.

References

Baader, F., Calvanese, D., McGuinness, D. L., and Nardi, D. 2010. *The Description Logic Handbook*. Cambridge University Press.

Berardi, D., Calvanese, D., and De Giacomoa, G. 2005. "Reasoning on UML class diagrams." Artificial Intelligence Volume 168, Issues 1-2.

Estefan, J.A., 2008. ``Survey of Model-Based Systems Engineering (MBSE) Methodologies," Rev. B, INCOSE Technical Publication, International Council on Systems Engineering.

Graves, H., Horrocks, I. 2008. "Application of OWL 1.1 to Systems Engineering", OWL Experiences and Directions April Workshop.

———, 2010. Ontological Foundations for SysML, Proceedings of 3rd International Conference on Model-Based Systems Engineering.

———2008. Representing Product Designs Using a Description Graph Extension to OWL 2. OWL Experiences and Directions October Workshop.

———, Bijan, Y. 2011. Modeling Structure in Description Logic, DL2011.

———, Bijan, Y. "Using Formal Methods with SysML in Aerospace Design and Engineering" to be published in: *Annals of Mathematics and Artificial Intelligence*.

Horrocks, I., Kutz, O., Sattler, U. 2006. The Even More Irresistible SROIQ, in: Proc. KR 2006, Lake District, UK.

Lambek, J., Scott, P. J., Introduction to higher-order categorical logic, Cambridge University Press, 1986.

OMG Systems Modeling Language (OMG SysML™), V1.1, November 2008.

OWL 2 Web Ontology Language, W3C Working Draft 11, June 2009.

Biography

Dr. Henson Graves is a Lockheed Martin Senior Technical Fellow Emeritus and a San Jose State University Emeritus Professor in Mathematics and Computer Science. He has a PhD in mathematics from McMaster University. Dr. Graves is the principle of Algos Associates, a technology consulting firm.