Integrating Reasoning with SysML

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

Abstract. Most engineering tasks require in-depth reasoning. For some tasks, automated reasoning is feasible and can provide high leverage for solving difficult practical problems. Engineering questions can be represented as questions about a model, provided the model contains sufficient domain knowledge. The widely used UML family languages have been embedded within logics. 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. By using this embedding engineers can use languages and tools that they are familiar with, have their models transparently translated into logic, automated inference applied, and the results be returned to their development environment. The logic embedding results of SysML, one of the UML family, is used to illustrate how engineering problems translate into logic problems and how the logic solutions translate back to engineering solutions. An engineering use case for air system capability analysis is used to show how a SysML model can be translated into an axiom set and the engineering question translates into a logic question and illustrates how formal reasoning about an axiom set can be used to answer the engineering question.

Introduction

Engineering models in languages such as SysML (OMG SysML 2008) are used to represent, specify, and analyze systems. The integration of modeling with automated reasoning provides the potential to use logic to answer engineering questions. The direct way to integrate reasoning with SysML is to embed a model as an axiom set within a logic. 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 derived from logic embedding results 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 is not new (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 system. Even though the axiom set and the model have the same information, the graphical notation of SysML is an efficient way of constructing an axiom set and opens a practical path to integrate reasoning with engineering development.

SysML is a general purpose graphical modeling language that can be used to represent system structure, behavior, parametric aspects, and requirement relationships. SysML has widespread use, and fragments of the language have been embedded within logic. SysML has become the defacto standard in many engineering domains. Engineers can use it, it scales for aerospace development, and retrofitting it with a formal foundation is feasible. This is not to say that SysML is as developed as other languages and tools for specialized areas, or that the language doesn't need improvement. OMG the keeper of the UML family of language specifications recognizes that the standards need improvement and are beginning to recognize that the languages need a formal semantics. The embedding of SysML into logic provides a practical way to integrate reasoning with product development. There do not appear to be other engineering languages that meet these success criteria for integration with reasoning.

This paper uses embedding results of SysML into logic to illustrate how reasoning can be used to answer 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. Results on embedding SysML models into a logical system (Berardi, et al. 2005) (Graves and Bijan 2011) are used to illustrate how engineering questions translate into questions about axiom sets which are amenable to automated inference. The reasoning example is simplified a static problem by averaging behavioral change. Examples where the SysML models involve behavior will be treated in future work.

Using reasoning results based on SysML models is dangerous unless precaution has been taken to ensure that the models correctly represent the application domain. In addition to being incorrect models may over or under specify the subject being modeled. A model underspecifies a problem domain when there are interpretations in addition to the intended interpretation. For underspecified problems the conclusions that can be made from the model are often insufficient to solve the engineering problem. Over specification results in a model that doesn't accurately describe the intended specification. Model validation, the term commonly used by engineers use for model correctness has an extensive practice, particularly for product test and evaluation. This paper is concerned with how an engineer's problem as represented within a SysML model translates into a reasoning problem and how the reasoning problem answers the engineering question. The answer is contingent on the validity of the assumption in the model.

Models and Interpretations

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 identify a target? The aircraft may be an existing aircraft, a proposed design, or a design modification. Simulation is the obvious method for understanding requirements and validating a design. However, simulation in itself does not verify that a design solution satisfies the capability. 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 target recognition 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 its operating context; the interpretation corresponds the model elements to elements in a 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 video of a sensor display. A model may have multiple interpretations, where an interpretation is anything that satisfies the relationships within the model. Building a system description model does not itself guarantee that any interpretations of the model exist. A model that cannot possibly have any valid interpretations is called inconsistent. For SysML, an inconsistent model is one in which all of the blocks of the model have an empty interpretation in any domain.

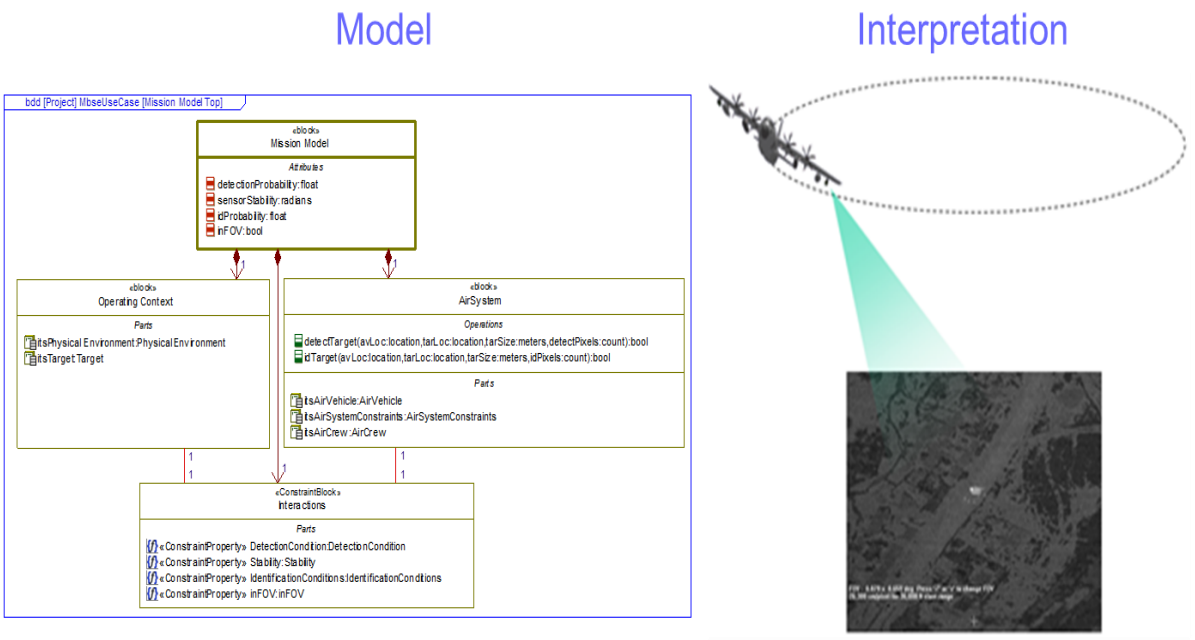

Figure 1. Relationship between Model and Interpretation

In logic, the use of the term "model" is reversed from its use in engineering: In logic a model is an interpretation of an axiom set within a logical system. In engineering, a model is a representation or description of its possible interpretations. However, the concepts of interpretation in engineering and model in logic are similar. Logical systems typically have an inference semantics and a reference semantics. In a logical system, axioms are assertions which are assumed to be true. 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 the 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) subdomains of a domain which correspond the SysML types. The properties and inclusion relationships in the model are preserved in the domain. Any inclusion relationships between the elements of a SysML

model are satisfied in the domain by the interpretation. The correctness of inference depends on the soundness and completeness for the logical system. When a model is embedded as an axiom set within a logic, questions about model translate to questions about the axiom set. For example the question of model consistency becomes the question of axiom set consistency. The question of whether 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.

Logic Embedding Overview

SysML has been embedded into several varieties of logic, however when the embedding is done properly the different embeddings are equivalent. Some of the logics used are noted in the Venn diagrams in **Figure 2**. In each of these cases, a SysML model is encoded as an axiom set within the language of a logic. The axioms take different forms in OWL and in First Order Logic (FOL). In OWL the axioms are class inclusion assertions. In FOL the axioms are quantified logical formulas using 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 FOL, and correspondences between the two different forms of logic. When these correspondences are made precise the embeddings are equivalent.

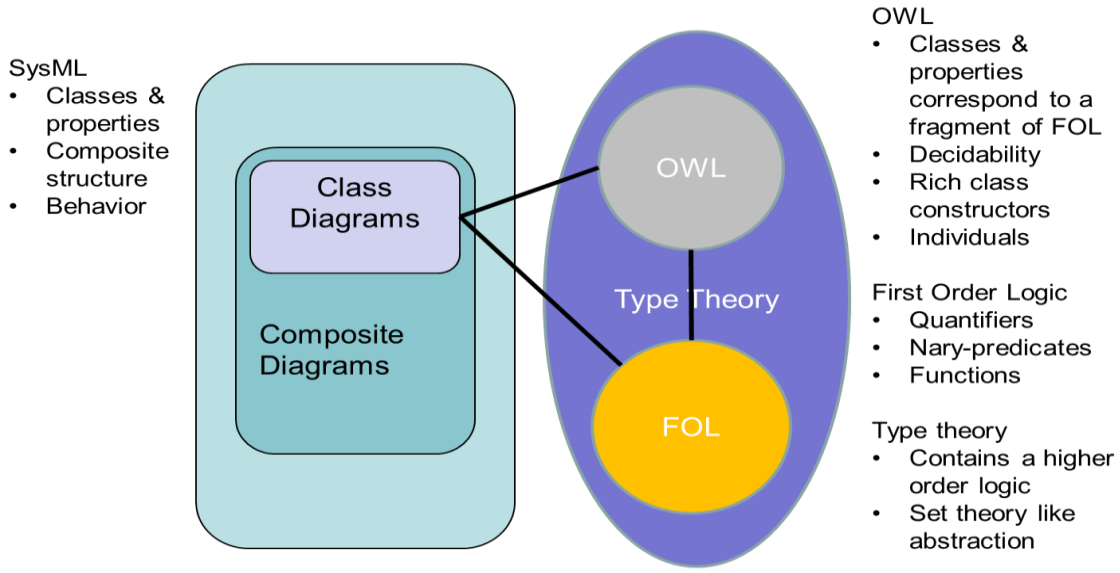

Figure 2. Relationship of Candidate Logics

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. The OWL 2, the most recent and precise version of OWL, uses classes, properties, and individuals. OWL 2 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. In conceptual modeling an axiom set is also called a Knowledge Base (KB). Where SysML models have been embedded within OWL2 they are embedded as axiom sets.

The formal foundation for OWL 2 (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 OWL 2, model questions become axiom set questions. A fragment of UML, called class diagrams has been embedded within the Description Logic (DL) SROIQ (Berardi, et al. 2005). Class diagrams are models that use only blocks and binary associations. The DL is known to correspond to a fragment of First Order Logic. This correspondence provides the equivalence of the OWL 2 embedding of a class model with the FOL embedding. The results hold as well for the corresponding fragment of SysML. This embedding provides the integration of a fragment of SysML with OWL reasoning. In OWL consistency of an axiom set and logical implication are decidable. OWL2 reasoners are well-developed and robust. They are able to reason about very large class diagrams.

Many language constructions essential to SysML's success are not covered by the class diagram embedding, for example structure diagrams. A structure diagram describes concrete structures which are assembles of parts and connections between the parts. Structure diagrams are common in engineering and science. An automobile model that contains a part decomposition with connections between the parts is a structure diagram. Finding the kinds of restrictions on axioms where the logical services are decidable has presented a challenge. Description Logic (DL) is a natural candidate for structural modeling. The ability to represent graph structures such as can be constructed with SysML Internal Block Diagrams (IBDs) is beyond the capability of OWL 2. A SysML model of a structure can be translated into the language of SROIQ (Horrocks, et al. 2006). However, the resulting axiom set contains property subtype assertions that do not satisfy the constraints of SROIQ axiom sets. OWL 2 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. An approach to the characterization of a SysML IBD as axiom set within an extended 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 (Graves. 2012).

Other language constructions used in SysML which are not class diagrams are operations with arguments that are declared within a block. Dynamic behavior and time are also not included. The air system target identification example is 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 results embedding a class diagram in a DL have been extended using a logical system called type theory (Lambek and Scott, 1986). Type theory contains both classes and formulae and a built-in correspondence between classes and properties with respectively unary and binary predicates within the type theory as well as with types in the type theory. The static part of SysML has been embedded into type theory (Graves and Bijan, 2012). 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 OWL 2 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. Logical implication problems such as the loitering question can be represented within type theory. Type theory extends OWL 2, and the embedding of SysML into type theory is consistent with the class diagram embedding of SysML models within OWL 2.

The correspondence between classes and formulas can be used to characterize exactly when an axiom set is consistent. 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. A class diagram is consistent if it admits an instantiation, i.e., if its classes can be populated without violating any of the assertions of 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*. An arbitrary axiom set may not be decidable and the sense that there is an algorithm which will terminate after a finite number of steps with a true or false answer. However, for some cases correspondence between classes and formulas can be used to reduce subclass assertions to be in a Description Logic which is decidable.

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. By embedding the model as an axiom set within logic the engineering question translates into a logic question. The approach to answering the logic question coincides with the way that engineers go about answering the question. The embedding of the model into logic will be treated informally. The SysML model is referred to as the *Mission Model*. The design analysis incrementally produces a composite model made up of many component models. The example is adapted from (Graves and Bijan, 2012). More detail can be found there.

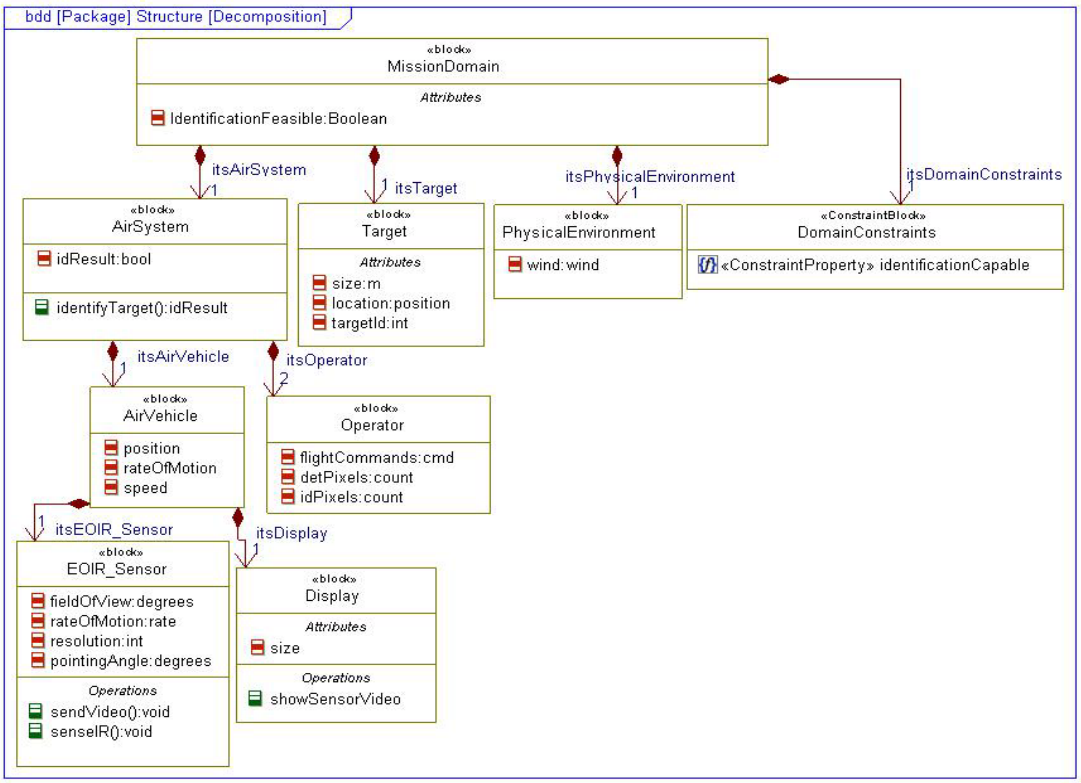

Figure 3. A Block Diagram for the Mission Model

Figure 1 represents schematically the relationship between a model, called the *Mission Model* and its interpretation. **Figure 3** illustrates a more detailed diagram of the *Mission Model*. 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 in constraint blocks that are part of the *MissionDomain* block. The diagram in **Figure 3** describes what components occur in an interpretation but do not show any constraint information. The complete *Mission Model* contains much more detail than is shown in this diagram.

The engineering question is whether a specifically equipped model of aircraft can perform target identification under specified operation conditions. A question such as the target identification question concerns the air system interaction with its operation environment. By constructing a SysML model that contains a model of the air system and the target, the physical environment specified by the operating conditions enables the assumptions to be explicit. 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 target identification 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. As analysis proceeds the *DomainModel* is refined to reflect more precise understanding.To answer questions based on a model requires more content than might ordinarily be produced in the development process. However, even if automated inference is not used the additional content is needed to justify engineering answers. SysML constraint constructions can be used to specify operating conditions, flight dynamic models, electro-optical models of sensors, and other factors that influence the answer. To answer the target identification question blocks corresponding to assumptions and constraints of component models are added to the *Mission Model*.

Composite structure

The diagram in **Figure 3** has the form of a tree whose root is the block *MissionDomain*. The nodes are the blocks, and the edges are part properties such as *itsAirSystem*, *itsEnvironment*, and *itsAirVehicle*. An interpretation of Mission Model contains a mission domain. The part property *itsAirSystem* is interpreted as a binary relation which assigns an air system to the mission domain. While SysML does not have instances the embedding logic has instances and one can write *md:MissinDomain* for an instance of *MissionDomain*. Then *md.itsAirSytem* is the air system that belongs to that domain.

A block such as *AirVehicle* may have a compartment labeled *attributes* which contains attributes; AirVehicle has attributes *position*, *rateOfmotion*, and *speed*. A block may also contain a compartment for operations. For example, the *EOIR* block of the Air Vehicle contains the components *sendvideo():Void* and *senseIR():Void*. These components are SysML operations. Each component of the *AirSystem* may have components with the same name. Qualified names are used to distinguish the names of operations of the *EOIR_Sensor* if there are two of them. For if there are two air systems a1 and a2 then is *a1.itsEOIT_Sensor.sendvideo()* distinct from *a1.itsEOIT_Sensor.sendvideo()*. The operation *Sensor.sendvideo()* has no arguments as indicated by the empty parenthesis and the Void value type. However, when activated they can read other air system attributes and update the value of air system attributes. Of course these operations have to be defined in terms of operations that are assumed to be available. While these two operations are extreme cases in that they do not have arguments or return values they illustrate the issue of the assumptions

on what they are allowed to read and update. The assumption is that they have visibility only to those variables which are components of the same owner that the operation has.

State spaces

The attributes translate into variables in the logic embedding. However, one has to use their qualified name to distinguish all of the unique variables. For example one variable is *itsAirsystem.position* and another is *itsEnvironment.itsTarget.position*. These qualified variables represent the state space of the problem. The change in values of these variables represent state change. In everyday mathematics these variables define a Cartesian product type of the types of the variables. SysML does not have a product type construction, but SysML can be embedded within logic that does have product types. For each variable occurring in the Mission model, let *xi : Xi* be a variable with its corresponding type. The notation

Domain = (x1 : X1, . . . , xn : Xn)

is used for the product of the types X1, . . . , Xn with named variables x1,…,xn. The notation <a1,…an> is used for a tuple of values. An easy way to understand the state space is to consider how one constructs a simulation for the Mission Model. Each variable in the state space will correspond to a variable in a software simulation of the Mission Model. In a simulation each variable does vary over time. For this example properties involving time have been averaged out to make the model static. As we will see not all tuples in *Domain* are valid for a simulation, only those which satisfy the constraint properties of the model.

Constraints define subtypes

The SysML *Mission Model* contains a lot of information not shown in **Figures 3**. 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.

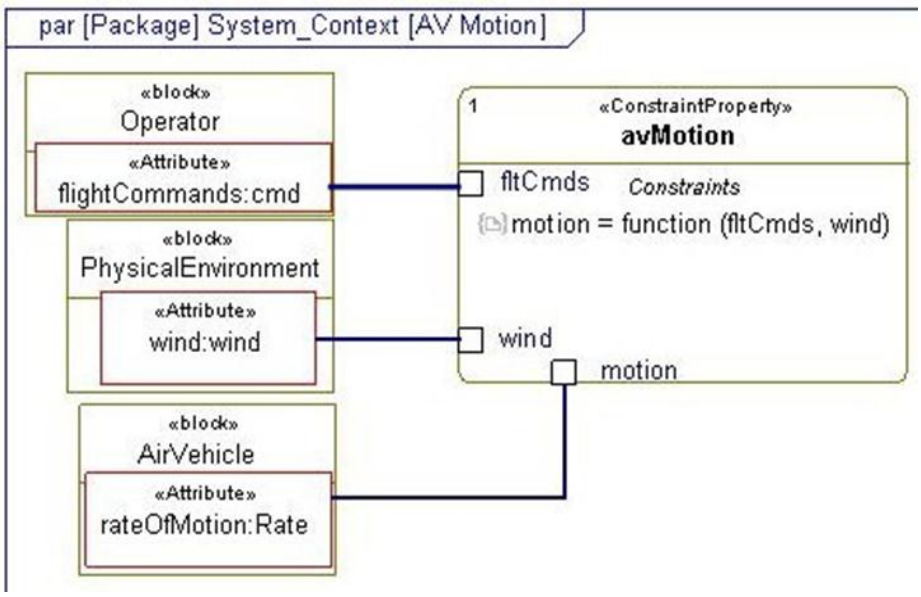

Figure 4. Using a Parametric Diagram to express application constraints

The diagram in **Figure 4** is a SysML parametric diagram is a component of the *DomainModel*. The block named *avMotion* is a reusable dynamic model for 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 are value properties. The lines in **Figure 4** connecting the parameters and the attributes are binding operations. The attributes are bound to the values of the parameters as computed by the equations in the parametric diagram. For example the diagram has the effect of enforcing the relationship between that attributes to which the parametric variables are bound. In *avMotion* small blocks *fltCmds*, *wind*, and *motion* are called parameters and are represented in logic as variables. The function:

rateOfMotion = function(wind,flightCommands)

gives the name *function* to the mathematical function which the rate of motion is a function of the wind and the flight commands. The parameters do not introduce new variables into the state space; they are a mechanism to enforce constraints between attributes in the state space. If we introduce that name *AirVehicleMotion* for product type

AirVehicleMotion = (flightCommand:cmd, wind:Wind, rateOfMotion:Rate)

then the functional relation defines a subtype of the product type of *AirVehicleMotion* using a type abstraction construction this is written in the logic as:

{ <flightCommand, wind, rateOfMotion> : motion = function(fltcmds,wind) }
$\sqsubseteq$ *AirVehicleMotion*

Conversely, for an abstraction subtype such as *AirVehicleMotion* there is a Boolean valued operation in the logic which is the characteristic operation for the subtype.

SysML constraint constructions are used to specify operating conditions, flight dynamic models, electro-optical models of sensors, and other factors that influence the answer. For this example the constraints can be represented as Boolean valued operations. Let

*allConstraints(*x1 : *X*1, . . . , *xn* : *Xn)* : *Bool*

be the conjunction of all constraints on the state variables defined by the parametric diagrams. Then *MissionDomain* is the subtype of *Domain* defined as:

MissionDomain = {< x1, . . . , xn > : *allConstraints(*x1, . . . , *xn) = true*}.

For notational simplification let *x* = < *x1, . . . , xn*> then

MissionDomain = {x : allConstraints(x) = true}.

An interpretation of the *Mission Model* assigns values to each of the state variables in *Mission Model* and which preserves all of the inclusion assertions. The state variables can be generalized to be functions of time and time based simulations are interpretations. For this example all of the time varying values of the attributes are averaged to give point solutions.

Formalizing Requirements

The requirements for target identification have the form that if specified constraints regarding target size and distance from the air system are satisfied then the air system has an operation that can correctly identify the target. To answer the target identification question the assumptions and constraints on component models must be added to the Mission Model. If a design solution is found then design verification shows that the target identification operation gives the correct result when the requirement constraints are met. The states that satisfy the requirements and physics constraints can be formalized as a subtype assertion within SysML extended with abstraction types. The requirement constraints are represented as a subtype of *Domain* type which satisfied the requirements constraints. We will write this as

{ x : idFeasible(x) = true }

For the requirements to have a solution all of the constraints that follow from the laws of physics as they apply to the mission domain must be satisfied as well. Thus,

$$\{ x : idFeasible(x) = true \} \cap \{ x : allConstraints(x) = true \} \neq Nothing$$

Otherwise no requirements solution is possible and the requirements constraints are inconsistent with the model constraints. The formalized requirements statement can be expressed as there exists an operator *f:Operator(AirSystem):()idResult* with

$$IdCapable \sqsubseteq \{ x : f(x) = itsEnvironment.itsTarget.id \}$$

The statement *f:Operator(AirSystem):()idResult* says that the domain of *f* is *AirSystem.* The operation *f* can access attributes such as *itsOperator.view and itsAirVehicle.itsSensor.display,* but can only access variables that occur as attributes within *AirSystem.*

Defining the Target Identification Operation

The requirements will be verified for an air system target identification operation named *identifytarget*. The operation will be defined in terms of operations of the aircraft's components. These components are assumed to exist and satisfy given specifications. The air system components include the display of the sensor, and also include an operation of the human operator. The air system and its subsystems determine its location, airspeed, and any other attributes needed by its subsystems to navigate the air vehicle and direct the sensors. The pilot operator is assumed to keep the rate of motion of the aircraft within the bounds required for the sensor to perform successfully, provided this is within the range of motion permissible by the air vehicle. An operator views the sensor display to make the actual identification. Assumptions about the ability of the operator to identify a target provided the sensor resolution is sufficient are part of *Mission Model*.

The operation *identifyTarget():idResult* is shown as a component of the *AirSystem* block in **Figure 3**. The graphical syntax shows *identifyTarget()idResult* as a component of a compartment in the block *AirSystem* is written in as linear syntax as:

identifyTarget : *Operations(AirSystem): ()idResult*

This syntax makes explicit that *identifyTarget* is a function of an air system argument. The operation is a declaration for the *AirSystem* block. If there were two air systems in the Mission Model then each would have its own operation and the two operations would not

necessarily give the same result. For simplicity *identifytarget* will be defined as a composition of operations. The attributes of components of the air system are available for use in the definition of *identifytarget*. These values do not have to be passed explicitly as argments.

The Mission Model models the real world by using constraint blocks to enforce the relationship between what the sensor sees and the environment attributes, as well as what the human operator identifies as a result of viewing the sensor display. The sensor's attributes are connected to the environment attributes via a parametric diagram. The EOIR_sensor has a *showSensorVideo* operation and the *Operator* has a *view* operation. The declarations in a linear syntax are:

showSensorVideo :Operations(Display) : ()VideoStream

view: Operations(Operator) :()idResult

The air system's target identification operation is the composition of the operator's view operation which view the sensor display. The sensor displays the result of the sensor's target tracking operation. The image display that the operator uses to identify the target depends on the sensor image and on degradation of video feed, screens resolution, refresh rate, and screen size. The operation *identifyTarget* is defined as

identifyTarget(AirSystem) =
itsOperator.view(itsSensor.itsDisplay.showSensorVideo())

The definition applies to any air system instance *as:AirSystem.* The syntactic correctness of the definition depends on the fact that *AirSystem* has as components an operator and an air vehicle. This assertion is represented within the embedding logic using a Description Logic type construction as:

$AirSystem \sqsubseteq \exists itsAirVehicle[1].AirVehicle \cap \exists itsOperator[1].Operator$

where *itsAirVehicle* and *itsOperator* are part properties used to define existentially restrictied types. The states in *MissionDomain* for which the targeting operation is correct can be written as:

IdentificationCorrect = { x : itsAirSystem.identifytarget() = itsEnvironmnet.target.id }

If we can show that

$identificationCapable \sqsubseteq IdentificationCorrect$

then the correctness of *identifyTarget* is verified within the context of the Mission Model. Within *Mission Model* the air system is a component of *MissionDomain* which is assumed to have an air system component, an environment component, and constraints which relate the attributes of the air system to those of the target component of the environment.

itsAirSystem.itsOperator.view(itsSensor.itsDisplay.showSensorVideo())
= itsTarget.id

Answering Questions

The engineering problem is a logical implication problem: do the mission model axioms imply the type inclusion

$$IdCapable \sqsubseteq \{ x : f(x) = itsEnvironment.itsTarget.id \}?$$

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

$$A \sqsubseteq \{ x : identifytarget() = itsEnvironment.itsTarget.id \}$$

to attempt to show

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

to obtain the desired conclusion.

What engineers do in practice to verify a system capability is to decompose the hoped for conclusion into a conjunction of statements. The decomposition of the conclusion is derived from the definition of the operation which purports to provide the capability. Then the task is to find assumptions about product components and the environment that they interact. These assumptions are used to validate or falsify the conclusion. We do not yet have sufficient assumptions to verify that the operation identifies targets under the conditions of the requirements.

Operator Identification Assumptions

For this problem we use assumptions about the ability of human operators to identify objects of certain sizes from a video display. The assumptions are derived from empirical studies of how operators perform. The operator assumptions give conditions on a sensor display image for which an operator can make a correct identification. The conditions are defined in terms of functions defined on the display. The motion on a sensor display which can be tolerated by the operator trying to identify a target is assumed to be less than MAXMOTION. MAXMOTION was determined by experimentation with operators. We assume that if the sensor display is sufficiently stable and other distance and size conditions are met then an operator viewing the display can make the correct identification. Sufficiently stable is defined by the type:

$$\begin{aligned} OperatorConstraint = \{ x : & \\ & display.image\_stability < MAXMOTION \text{ and} \\ & display.image\_resolution < MINRESOLUTION \text{ and} \\ & display.pixels > MINPIXELS \} \end{aligned}$$

The assumption made about operators can be expressed as:

$$OperatorConstraint \sqsubseteq IdentificationCorrect$$

The question becomes can the sensor can produce the required stability results under the feasibility conditions.

Sensor assumptions

The design solution is contingent on finding a sensor for which an operator can correctly identify the target. The assumptions for the sensor are based on electro-magnetic laws of physics applied in their engineering form. These laws are expressed as a constraint. In the form applied to air systems they are represented by a function *image_stability&resolution* which is a function of the field of view of the sensor, which is itself a function of the air vehicle and target locations as well as environmental conditions. The air vehicle motion is also an argument. The air vehicle motion is a function of the flight commands and the wind in the environment. Each of *image_stability&resolution,* i*nFOV,* and *avModtion* are represented using parametric diagrams. The composition

image_stability&resolution(inFOV(avloc,tloc,sensor.pA,sensor.fov),
avMotion(fltcmd,wind) < k

is represented by binding parameters to attributes. We assume that *avMotion (fltcmd,wind)* is less than some constant for which the value of the *image_stability&resolution* function has an acceptable value. With this assumption what has to be shown is that the sensor satisfies the stability and distance conditions when the *identificationFeasible* conditions are satisfied. The sensor is supplied with a specification which can be represented as a parametric diagram of the form:

sensorspec:Constraints(MissionDomain) =
Equation(image_stability&resolution(inFOV(avloc,tloc,sensor.pA,sensor.fov),
avMotion(fltcmd,wind)), MAXMOTION)

Thus is we define the abstraction type *SensorSpec* corresponding to the parametric diagram we have:

$SensorSpec \sqsubseteq OperatorConstraint$

This says that the sensor specification provides sufficient stability for the operator to make a correct identification.

Putting it together

Putting the inclusions together and recognizing that the final result is contingent on the operator assumptions and the sensor assumptions we have:

$IdentificationFeasible \cap SensorSpec \sqsubseteq OperatorConstraint \sqsubseteq$
IdentificationCorrect

and so

$IdentificationFeasible \sqsubseteq IdentificationCorrect$

which gives the desired result. This capability analysis scenario which has been translated into a verification scenario has represented the problem in a schematic form. The details can be filled in. Most of the work involves engineers using domain knowledge to find or solve equations to satisfy the assumptions. The verification scenario provides not only the plan for solving the problem but a justification of the results.

Conclusion

The use of reasoning to answer engineering questions has been illustrated with a capability question; can an air system can perform target identification correctly under specified conditions. The proof of correctness of the air system identification operation is outlined and it corresponds directly to standard design analysis and verification techniques. This example shows how reasoning can be integrated with SysML and provides evidence that formal methods can be integrated with everyday engineering activity. The integration is based on the fact that widely used engineering languages such as SyML can be retrofitted with a formal semantics. The reasoning in the example is justified by a semantic embedding of a fragment of SysML into type theory logic. One lesson implicit in the example is that one can and should include assumptions and constraints in models even if automated reasoning is not used. The verification is contingent on the validity of the context model with respect to the real world domain.

The embedding of the SysML *Mission Model* into type theory uses several principles:

1. SysML blocks, properties, and operations are mapped directly to corresponding type theory constructions. Attributes (value properties) are mapped to type theory projection maps (variables). Value properties of components of the model correspond to variables in the logic.

2. Product types within type theory corresponding to the types of the attributes contained within the blocks of the model are introduced to represent the state space of a model.

3. Parametric diagrams correspond to truth-valued operations defined for their variables. For each parametric diagram a subtype of the product which are the tuples of values for which the formulas are true is introduced

4. SysML contains composite constructions for parts, attributes, operations, and behaviour. These constructions embed as type constructions within type theory.

While there are other languages with a formal semantics such as the B-language (Abrial. 1996) that have some of the features needed to represent this example, there are no engineering languages with widespread use and good tool support that appear to have the traction of SysML.

Analysis of the examples suggests that a number of additions to the SysML language would be useful. These include adding: the abstraction subtype construction, DL class and property constructions, individuals, “Function call” to block diagrams. 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. Type theory provides the language extensions suggested by the examples with a formal semantics well adapted for use with inference engines. To extend the use of reasoning for SysML for behavior constructions additional axioms are needed for mechanical inference. Again type theory is well suited for representing behavior. The next step in integration of reasoning with SysML is the translation of the SysML model into type theory axioms and the integration of a reasoned to operate on the translation.

References

Abrial, J. R. *The B-Book*. Cambridge University Press, 1996.

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.