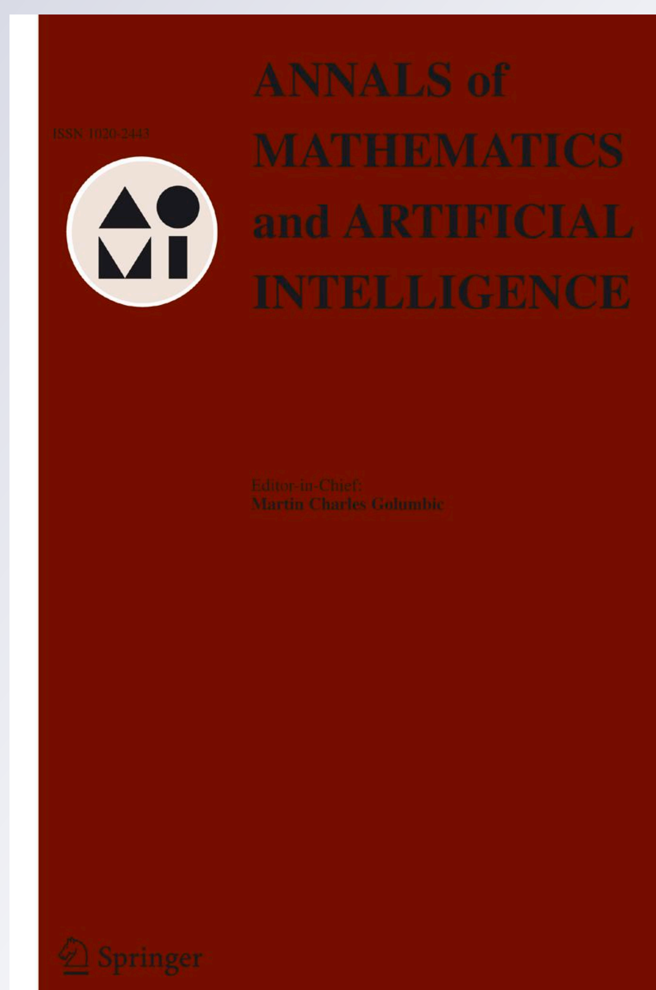
Using formal methods with SysML in aerospace design and engineering

Henson Graves & Yvonne Bijan

Annals of Mathematics and Artificial Intelligence

ISSN 1012-2443

Ann Math Artif Intell
DOI 10.1007/s10472-011-9267-5



Springer

Your article is protected by copyright and all rights are held exclusively by Springer Science+Business Media B.V.. This e-offprint is for personal use only and shall not be self-archived in electronic repositories. If you wish to self-archive your work, please use the accepted author's version for posting to your own website or your institution's repository. You may further deposit the accepted author's version on a funder's repository at a funder's request, provided it is not made publicly available until 12 months after publication.

Springer

Author's personal copy

Ann Math Artif Intell
DOI 10.1007/s10472-011-9267-5

Using formal methods with SysML in aerospace design and engineering

Henson Graves · Yvonne Bijan

© Springer Science+Business Media B.V. 2011

Abstract Maintaining design consistency is a critical issue for macro-level aerospace development. The inability to maintain design consistency is a major contributor to cost and schedule overruns. By embedding The Systems Modeling Language (SysML) within a formal logic, formal methods can be used to maintain consistency as a design evolves. SysML, provided with a formal semantics, enables engineers to employ reasoning in the course of a typical model-based development process. Engineers can make use of formal methods within the context of current engineering practice and tools without needing to have special formal methods training. As component subsystems are introduced to refine a design, their assumptions are checked against current assumptions. If new assumptions do not introduce inconsistency, they are added to the model assumptions. If the assumptions render the design inconsistent, they are detected which minimizes potential rework. SysML has a demonstrated capability for top-to-bottom design refinement for large-scale aerospace systems. SysML does not have a formal logic-based semantics. The logical formalism within which SysML is embedded matches the informal semantic of SysML closely. The approach to integrating formal methods with SysML is illustrated with a typical macro-level aerospace design task. The design process produces a design solution which provably satisfies the top level requirements. The example provides evidence that coupling formal methods with SysML can realistically be applied to solve aerospace development problems. The approach results from a number of detailed design trades employing a model-based system development process which used SysML as the model integration framework.

H. Graves (✉)
Algos Associates, 2829 West Cantey Street, Fort Worth, TX 76109, USA
e-mail: henson.graves@hotmail.com

Y. Bijan
Lockheed Martin Aeronautics Company, Post Office Box 748, Fort Worth, TX 76101, USA
e-mail: yvonne.bijan@lmco.com

Published online: 15 October 2011

Author's personal copy

Keywords SysML · OWL · MBSE · Type theory · Design by refinement · Description logic

Mathematics Subject Classifications (2010) 03B70 · 03C60 · 03G30 · 03C98 · 18B25 · 18C10 · 18C50 · 68Q60 · 68T30

1 Introduction

The technical challenge of managing complex aerospace system development is to keep technical information consistent and to understand the impact of design change. Formal methods have the potential for determining information consistency and change impact. Our goal is integration of formal methods with engineering practice in such a way that all development, analysis, and reasoning are an application of formal methods. Formal methods become the primary tool for management of system development. Product development can be radically improved by integrating formal methods with existing languages and tools. While this goal may seem well beyond what is feasible, there is a direct path. This work takes a step toward integrating formal methods into the way aerospace development is done and with tools currently used.

There is general recognition of an inability to manage the design and analysis of complex aerospace systems as evidenced by cost and schedule overruns. Often, the resulting systems do not meet expectations. Many development problems are traceable to lack of precise verifiable requirements, changing requirements, and to the inability to establish and maintain design consistency as the design evolves. On large scale aerospace programs, the inability to keep design and analysis information consistent is a major cause of development problems. Maintaining consistency requires determining whether the assumptions used by component models are consistent with the overall design context.

Even a good development process does not in itself insulate one from some of the major causes of design error and rework. Many of the really expensive mistakes occur early in the analysis and design process and are the result of not capturing assumptions and checking consistency as the design process develops. A lot of rework and lost time and energy come from not capturing and reconciling the assumptions of component models within a development effort as these models are introduced. These errors of omission are particularly prevalent when different languages and tools are used, even when there are well-defined interfaces between the tools. Relatively simple mistakes that are not caught early result in hundreds of millions of dollars and years of schedule slip. Placing a formal foundation under system development mitigates these very real, costly problems.

Formal Methods [7], the term generally applied to the use of formalized logic-based reasoning, is used in computer processor design, software, control systems, and other disciplines [18]. The goal of enabling formal reasoning in design and analysis for complex physical systems, such as aircraft and automobiles, hardly needs restating [19]. While formal reasoning is used in many areas of system for design refinement and verification [34], formal methods are not common for macro level development of complex systems such as aircraft. Considerable disappointment has been expressed in its lack of use and payoff in Aerospace developments [25]. As we

Author's personal copy

will see, the integration of formal methods in macro level system development can be realistically used and provide payoff.

Many engineering tasks involve reasoning; for example, determining requirements consistency and verification of design implementation properties. These reasoning tasks can potentially be formalized. One of the kinds of reasoning needed consists of representing assumptions that new component models introduce and checking that they are consistent with former assumptions. Consistency maintenance has a very high potential payoff in terms of shorting design cycles and decreasing rework. Automated reasoning can be used to check consistency of design models, in particular to check consistency of component models, requirements and consistency of design refinement. The challenge is to integrate formal methods with the tools and processes in a way that engineers can and will use for large programs.

Integration and retrofit with existing engineering development tools which are robust and have widespread use as an attractive approach if such languages and tools exist. Our experience validates the general premise of the feasibility of Model-Based System Engineering (MBSE) [12] using SysML [30]. The choice of SysML is based on the evidence that engineers can use it, that it scales for aerospace development, and that 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 such as the development and verification of safety critical software. SysML can be made to serve as a unifying framework for system development with detailed specifications exported to other frameworks and the results re-imported. However, SysML does not have a mechanism to incorporate reasoning. Successful integration of reasoning with SysML can be done only when the reasoning is in accord with the SysML informal semantics. As we will see, the informal semantics of SysML accords well with the semantics of an existing formal logic.

By retrofitting SysML with a formal semantics, as opposed to developing new formalisms and tools little additional work is required to integrate formal methods into development beyond what has to be done in any case. Engineers can directly apply formal reasoning to the system models to develop design solutions. There does not appear to be any language and tool candidates other than SysML that have the necessary adoption and record of accomplishment. From our personal experience, we have validated that SysML satisfies the necessary conditions for this approach to work. With the approach of retrofitting SysML with a formal semantics engineers can use formal methods successfully to achieve better designs without special training and with little extra cost, provided the formal methods are correctly integrated with the method and tool use.

The scope of the paper is to introduce a usable logic-based formalism for SysML that engineering practitioners can understand and use within their development processes. The resulting formalism for SysML can be integrated with automated reasoning systems to be effective. The first step is to understand what kind of automated reasoning is needed by working out how the models needed can be represented in SysML and how reasoning tasks can be formulated within this formalism. The paper uses a typical macro-level aerospace development task to show how reasoning is used within an engineering development process. The analysis establishes conditions for reasoning tools to be integrated development tools. One of the essential conditions for integrating a reasoning system based on a different formal system is a semantically well founded mapping between the two formal systems.

Author's personal copy

While determination of the choices of specific reasoning engines is not within the scope of this paper discussion of the kinds of reasoning used is presented. The embedding of SysML within the logical formalism is so direct that it can be viewed as a semantic retrofit for SysML.

This paper demonstrates the practical feasibility of integrating formal reasoning on a broad scale into aerospace system development. The integration by embedding SysML into a logical formalism enables many practical engineering problems to be translated into logical problems. Reasoning can then be integrated with widely used development tools and existing inference engines. In particular, the paper:

- demonstrates how formal methods apply as a general mechanism to keep macro level designs consistent as development proceeds from start to finish
- shows how formal methods may be used in an integrated, top level system development rather than only at the subsystem level
- provides end-to-end integration of models within a unified semantic framework
- enables integration of formal methods with the methodologies and tools that engineers use today
- provides context and methodology for semantic integration of multiple models.

The next step will be to export the SysML models to an inference engine to verify that the manually done proofs can be automated.

1.1 Organization

The approach to integrating formal methods into system engineering is based on (1) the effectiveness of SysML as an integrating language framework, (2) the fact that SysML language constructions and informal semantics correspond closely to the semantics of a well-known logic, and (3) the logic provides a rich language of constructions for representing cyber physical knowledge. The organization of this document reflects this approach.

Section 1 This section introduces the paradigm of encoding an engineering problem as a logical problem by embedding engineering language within logical language. An engineering model translates into an axiom set within a logical language. Issues such as requirements and design consistency translate directly into consistency of the axiom set that encodes the engineering model. An overview is given of the principles used to embed SysML with its graphical syntax within the linear textual syntax of a logical language. The logic used, type theory [8], has been well developed and studied by mathematicians and logicians. Type theory has served as the basis for interactive proof systems used particularly in software development. A SysML model is embedded as an axiom set within a type theory logic. Type theory contains a language fragment that corresponds to Description Logic [3]. Description Logic (DL) is the logical foundation for the semantic web language, OWL [31]. Type theory provides language constructions which can be used to represent embedded operations and part properties which are used to represent composite structures. The constructions needed to formalize composite structures are not present in DL. Simple examples illustrate the potential for automated reasoning is maintaining design consistency. The section makes comparisons of the type theory formalism used for SysML with other logic-based formalisms.

Author's personal copy

Section 2 This section introduces a characteristic aerospace application and develops a series of SysML models. The models successively refine each other to produce a design solution for a requirement. The example is based on a number of engineering applications with which the authors have been involved. The requirements in the original examples motivating this use case were stated in probabilistic terms. This aspect has been simplified to statements of certainty. The macro-level analysis did not require any analysis of time or concurrency beyond satisfying the time constraints needed to perform operations. This aspect has also been simplified to remove statements using concurrency or time. The example illustrates, at least in a simplified form, the complexity of the kind of analysis and reasoning that that occurs at the macro level in aerospace development. In this sense the example provides a realistic use case for the application of formal methods. Using SysML models as the primary design artifact within a Model-Based Engineering development paradigm motivates the necessity to capture system component and environment assumptions as part of the SysML models. The expressivity of the SysML constraint constructions enables capturing many assumptions. We did find cases where SysML language extensions would enable additional assumptions to be captured. The advantage of checking consistency of models as they are developed and integrated motivated the search for a logic-based formalism that can support the kind of reasoning encountered.

Section 3 This section provides a description of the specific form of type theory in which SysML is embedded. This type theory logic has been engineered to have a syntax compatible with SysML where the languages overlap. Conversely, the graphical syntax of SysML can be used with this logic. The SysML language constructions and their informal semantics correlate closely with type theory language constructions and inference rules. Type theory contains constructions not present in SysML. For example, type theory contains individuals and class constructions found in description logic. While these constructions are not present in SysML, they are candidates for inclusion within SysML.

Section 4 This section embeds the SysML application model developed in Section 2 as an axiom set within the logical formalism. A requirement and its design solution are formalized. A sketch of the verification proof is given. The proof is contingent on electro-optical physics laws applied to sensors and on empirically derived laws regarding human performance. With these laws as assumptions, sufficient conditions on humans and sensors are given that imply a composition of a human viewing a sensor and the sensor performance are sufficient to satisfy the requirements. The structure of the proof clarifies what assumptions are made and what physical tests are needed for a contractual verification of the requirement solution. For example, the design solution depends on the validity of electro-optical physics. Physical testing can presume that these laws are well established. The empirical laws regarding human ability to identify an object by viewing a display may be subject to more debate, but having the assumptions explicit enables transparency of assumptions.

Section 5 This section provides an analysis of the effectiveness of the approach, discussion of the choice of type theory for SysML semantics, some lessons learned and recommendations. Working out the ISR example gives strong evidence that the approach to providing a formal semantics for existing engineering languages and tools can become a practical and useful reality. Formal methods can be easily

Author's personal copy

integrated into model-based development processes. Regarding type theory, an observation has potentially important consequences for integrating with automated inference engines. Informally statements of requirements, constraints, and derived conclusions are universally quantified. That is, they must hold for all possible realizations of the SysML model (axiom set in the logic). However, these statements can be stated as subtype relationships within the type theory logic. The assumptions that we have encountered can be represented within type theory either by equational constraints defined for variables representing class attributes or statements using DL class constructions. A modification of the DL reasoning engines should be able to perform the reasoning encountered here. Finally, the use case example suggests useful extensions for SysML and suggests that a formal semantics should be part of the official language specification.

1.2 A useful formal semantics (the approach)

A major issue in system development is keeping design and analysis information consistent as development proceeds. More generally, the inability to analyze the consequences of design decisions early causes the development to proceed down a path which may eventually have to be rescinded. Maintaining design consistency requires determining whether the assumptions entailed by a design change are consistent with the assumptions of the design before the change is made. This problem of consistency maintenance is well suited for employment of formal methods.

A formal system is a formal language with a formal semantics. An axiom set within a formal system is a collection of statements within the language of the system which are assumed to be true. A formal semantics, as used here, consists of a reference semantics and an inference semantics. The reference semantics specifies how an axiom set is to be interpreted within a real or virtual domain. The inference semantics provides rules which can be used to derive conclusions from axioms, i.e., other true statements. The theory generated from an axiom set is collection of the statements which can be derived from the axiom set. Any useful formal semantics for SysML will have to be correct in so far as this is possible, given that SysML does not have a formal semantics. Otherwise, the semantics is potentially dangerous as it may lead to false conclusions. A formal system is said to be sound when the statements derived from an axiom set are true with respect to the reference semantics and is said to be complete when all statements true in the reference semantics can be derived from the axiom set. The informal semantics of SysML as specified by an informal notion of valid interpretation accords well with the formal concept of valid interpretation of type theory within the reference semantics for theory [26] where soundness and completeness results hold.

In engineering, a model is a description of an existing system or a specification for one to be built. In logic, an axiom set describes a class of valid interpretations (models in the sense of logic). The type theory logic provides a precise notion of consequence; the consequences of an axiom set are the statements true in all valid interpretations of the axioms. An axiom set is satisfiable if it has a valid interpretation. An axiom set may have many or no valid interpretations. If an axiom set has no valid interpretations, it is said to be unsatisfiable. When an application is represented by a SysML model the engineering question of consistency of the model translates into the question of whether the axiom set encoding the model is

Author's personal copy

satisfiable, i.e., do the axioms have a valid interpretation (model in the logician's sense). In the logic to be used consistency and satisfiability are equivalent as the logic is complete. Many other engineering problems can be reduced to the question of satisfaction of the axiom set. The reasoning encountered in model management is not complex from the logical viewpoint. The fact that manual consistency checking of large complex models characteristic of aerospace is error prone argues for automated consistency checking.

Integration of reasoning with SysML development enables determination of whether design decisions have led to inconsistencies. The benefits of having a formal semantics go beyond application of inference engines to determine inconsistencies. A modeling language with a formal semantics enables the construction of models whose meaning is precise in terms of what is required to implement or recognize an interpretation of the model (reference semantics). The inference semantics provides the method for determining what can be concluded about its reference semantic from the model. A formal semantics mitigates misunderstanding by human or machine users. A formal semantics will help evolve engineering languages to meet future expressiveness and integration needs. The current work is primarily restricted to the static part of SysML. However, some indications are given as to how type theory constructions can be used for the behavioral part of SysML.

1.2.1 Encoding SysML within a logic

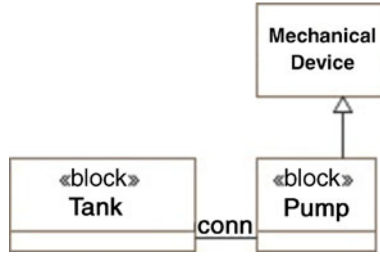
The specific method used to integrate formal methods into macro level system development is to encode the SysML language within the language of formal logic. A SysML model which is the molecular unit of SysML is encoded as an axiom set within the logic. A higher order type theory [8] is used for the SysML encoding. The variant of type theory logic used is called Abstract Block Diagram (ABD) logic. ABD logic was introduced in [14] and has been engineered subsequently. ABD logic has been engineered specifically for application to engineering languages such as SysML. While SysML uses diagrams to represent models, the diagrams can be encoded as formulae within the logic. The logic contains language constructions which are not in SysML but can be added as user defined stereotypes. More detail can be found in Section 3. By using this encoding and making use of the language extensions in the logic, all development, analysis, and reasoning in engineering practice can be potentially an application of formal methods.

SysML uses a graphical syntax with elements for blocks, associations, part properties, and subclass relationships between classes. A block can contain compartments with declarations that include value properties, operators, and constraints. In the encoding of a SysML model as an ABD axiom set a block is encoded as a class and a binary association from one block to another is encoded as a property with domain A and range B. The same symbols will be used in SysML and ABD logic. The ABD axiom set corresponding to Fig. 1 consists of three class symbols, a declaration $p : P(Pump, Tank)$ and a class inclusion $Pump \subseteq MechanicalDevice$.

In ABD logic, one can declare $a : A$. A property p may also have instances. The instances are pairs of class individuals. A pair is written as $< a, b >$. If $Dom(p) = A$ and $Range(p) = B$ and $< a, b >: p$ then $a : A$ and $b : B$. If $a : Pump$, the interpretation of a is an instance of the interpretation of $Pump$. Similarly, the interpretation tests if a pair of objects satisfies the property instance criteria for a pair of instances. ABD logic has an abstraction constructor which is used to define subtypes of a given

Author's personal copy

Fig. 1 Pump model



type. For example, if we have a Boolean valued operation $f(x : X) : Bool$, then the abstraction type

$$A = \{x : f(x) = true\} \tag{1}$$

is a subtype of X and has the property that

$$\forall a.\ a : A \ implies\ f[a/x] = true \tag{2}$$

where $f[a/x]$ is the result of substituting a for the variable x in f. Conversely, if $p(a) = true$ then $a : \{x : f(x) = true\}$. The abstraction type constructor can be used to define the DL class constructions. A consequence of the abstraction construction is that logical operations such as *and* and *or* as well as universal and existential quantifiers can be defined for Boolean typed operators. As a result, the expression

$$\forall x.\ x : A \ implies\ x : B \tag{3}$$

is a Boolean valued operation and subtype relation $A \subseteq B$ is equivalent to

$$\forall x.\ x : A \ implies\ x : B \tag{4}$$

In ABD logic, as in set theory, a property p with $dom(p) = A$ and $range(p) = B$ defines an operation from A to the Power type of B. A property p with $dom(p) = A$ and $range(p) = B$ is functional if

$$\forall x \forall y \forall z.\ x : A \ and\ y : B \ and\ z : B \ implies\ y = z. \tag{5}$$

In ABD logic, a functional property p with $dom(p) = A$ and $range(p) = B$ defines a function of its first argument, A. The result of applying the function determined by p to a is written in "dot" notation as $a.p$. The term $a.p$ is typed as $a.p : B$.

The ability to represent composite structures is one of the features of SysML which enable the representation of complex systems. For example a fuel system is a composite structure as it has components and interconnections between the components. Figure 2 is a SysML Internal Block Diagram. The diagram shows the internal structure of a fuel system model. The fuel system model has three part properties, $itsTank1$, $itsTank2$, and $itsPump$. In the diagram the rectangles are labeled by the part property and the range type of the property. For example, in $itsTank1 : Tank$ the expression $itsTank1$ is a binary property whose range type is $Tank$. The block $Pump$ has two value properties, *input* and *output*; and an operation, $transfer$ to transfer the input to the output value properties.

ABD represents composite structure with declarations which use type constructors. The linear syntax for the SysML model which for which diagram in Fig. 2 is a partial display uses three type constructors: *parts*, *values*, and *operations*. Each of

Author's personal copy

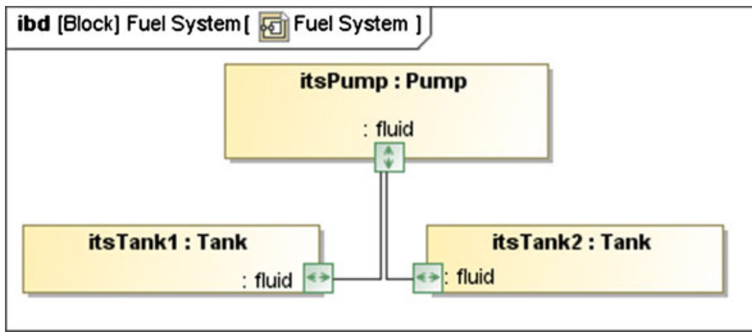


Fig. 2 Fuel system model

these constructions has two type arguments and produces a type as value. These type constructors are definable within ABD logic. For example, the ABD linear syntax for the fuel system parts declarations is:

$$\begin{aligned} itsTank1 &: parts(FuelSystem, Tank)[1] \\ itsTank2 &: parts(FuelSystem, Tank)[1] \\ itsPump &: parts(FuelSystem, Pump)[1] \end{aligned} \tag{6}$$

In the declaration $itsTank1 : Parts(FuelSystem, Tank)$ $itsTank1$ is a named binary property which is an instance of the type, $Parts(FuelSystem, Tank)$. In the model, *Pump* contains declarations corresponding to the *operations* and *values* compartments in *Pump*. They are:

$$\begin{aligned} input &: values(Pump, var\,Fluid) \\ output &: values(Pump, var\,Fluid) \\ transfer &: operations(Pump, (x : Fluid) : Fluid) \end{aligned} \tag{7}$$

In general, the property instances of the parts constructor are not functional. The suffix [1] declares the part property $itsTank1$ to be functional. Both *values* and *operations* are type constructors with two arguments. The type constructors are functional, i.e., there is a unique second argument for any first argument. The first argument of *values* is *Pump* and the second argument is a variable of type *Fluid*. ABD uses the term "attribute" where SysML uses "values". In ABD, the value of an attribute is a variable in that it can be substituted for and its value set. This is consistent with the informal semantics of SysML. SysML value properties can be set and substituted for which are the characteristics of variables. The first argument of the type constructor *operations* is *Pump* and the second argument is the type of an operation with one argument. Since the relational is functional it is represented as a (higher order) functional of the first argument which returns an operation as value. For an individual $p : Pump$, a "dot" notation is used to write the application of $transfer$ to the instance p.

$$p.transfer(x : Fluid) : Fluid \tag{8}$$

Author's personal copy

The compositions $itsTank1.port$ and $itsTank2.port$ are distinct and are typed as operator valued variables. Within the ABD logic, the variables can be bound to the input and output attributes of the pump within *FuelSystem*. For example, in

$$FuelSystem[[itsPump.transfer[itsPump.input/x]itsTank1.port] \tag{9}$$

the argument of the transfer operation the argument *x* is bound to the input attribute of pump and the value of transfer is bound to the output attribute. As we will see later, SysML has a constraint construction that can be used to model such relationships between the input and output attributes of the pump and contents of tanks that are connected to the pump.

1.2.2 Translating engineering problems into logic

A SysML model can introduce named properties, operations and types; it can express type relationships, term equality, and subtype relations. A SysML model is encoded as an axiom set for an ABD theory by directly using the model declarations and typing relations as axioms. The named operations and types occurring in the model are the signature of the ABD theory. The theory of an ABD axiom set (SysML model) is the consequences of the axioms derivable by the inference rules [14] (covered in Section 3). ABD type constructions extend SysML type constructions with product, sum, and operation types, as well as providing description logic (DL) [3] class constructions. ABD has classes including a universal class, *Thing*, an empty class *NoThing*, and has class constructions for universal and existential class constructions following DL. In ABD theory classes are types, but not all types are classes. By converting a SysML model to an ABD theory, model refinement becomes theory refinement. A SysML model may use SysML constraints and subtype relations to express assumptions about an application description. Engineering questions about a model translate into questions about the axiom set that encodes the model. This encoding enables interfacing SysML with an inference engine which take an axiom set as input. As we will see by using class constructions not available in SysML, additional kinds of assumptions can be expressed within the model as subclass relationships. An inference engine can be used to answer questions about the axiom sets which translate back to be engineering answers. The answers consist of blocks in the extended language, added block inclusions and block equivalence with the “null” block.

1.2.3 Integration with reasoning

For reasoning using the encoding of a SysML model as an axiom set to be useful, efficient scalable reasoning algorithms must be available. Our analysis of use cases indicates that many engineering problems can be reduced to consistency problems. The axioms encoding the models are often within a decidable fragment of ADB theory. The semantic analysis suggests model construction principles which ensure that the axioms belong to the description logic fragment of ABD. For the DL axiom sets, efficient deterministic algorithms are available for deciding the logical consistency of the model. In addition the examples often make use of term simplification for which there are terminating algorithms to produce irreducible forms for specific kinds of language terms.

Author's personal copy

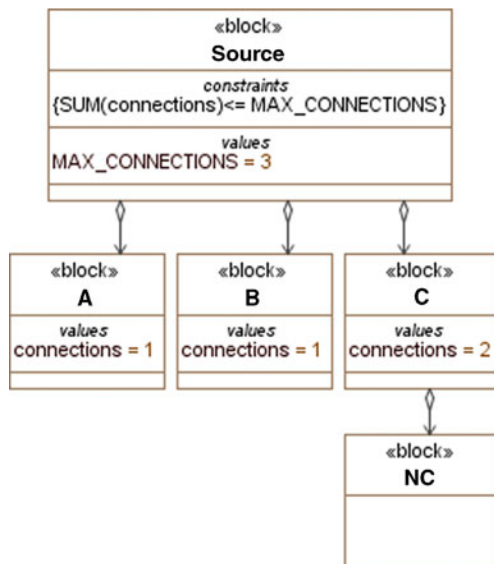
Three simple examples of managing model development illustrate how models that represent application descriptions including assumptions can be encoded in ABD logic and consistency checking used to identify potential design issues. The examples are simplified so that they can be represented with DL class constructions and the reasoning can make use of DL reasoners. The ability to define description logic classes enables a wider range of constraints to be represented by subclass relationships than is the case in SysML.

Example 1 The way in which the models are typically developed is by adding components and making connections between components. When adding a component to a system and connecting it to, for example, a hydraulic system, the connection may violate constraints regarding admissible connections to the hydraulic system. The violation of a constraint may not be apparent from simply making the connection to a component of the hydraulic system. Using DL constructions constraints may be expressed within the model and the consistency of the connection constraints checked when changes are made. Figure 3 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 that specifies the sum of all connections must be less than the maximum of 3. The connections of *Source* are not just the direct connections from *Source* to A, B, and C, but any path connections formed by composition of connection properties. A constraint that the model may have at most 3 path connections which originate from source and terminate at some other block can be expressed with

$$Source \subseteq \leq 3p.Thing \tag{10}$$

where the property p is the union of all properties with Source as domain which are constructed from the connection properties in the signature of the model. This

Fig. 3 Max connections



Author's personal copy

property can be defined within the logic. With the new connection from *NC* in the diagram, one has:

$$Source \subseteq= 4p.Thing \tag{11}$$

However, since $\leq 3p.Thing \bigcap = 4p.Thing = Nothing$, *Source* would be equal to *NoThing* which means the design is not satisfiable. To determine that a new connection makes the model unsatisfiable requires that an algorithm be used to compute the number of connections. This part of the inference is not done by the DL reasoner, but would likely be computed externally. This example gives a hint of how reasoning may require additional computation to proceed with the reasoning.

Example 2 Issues often occur when integrating a component model into a composite 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. The pump model in Fig. 4 contains the subclass relation $Pump \subseteq \forall conn.B$ where *conn* is a property that represent pump connections. This subclass relationship represents the assumption that the pump can only be connected to components of type B. However, if we attempt to connect *Pump* to a component A where A and B are disjoint, the connection violates the original model. To use the pump model, the assumption must be modified as this assumption is incompatible with

$$Pump \subseteq \forall conn.A \tag{12}$$

as A and B are disjoint.

Example 3 Assumptions and requirements are often given in the form of pre-conditions imply post-conditions. For example, if a sensor satisfies stability of motion conditions respect to its environment then the sensor's pointing operator can achieve a specified accuracy. Such a statement may occur as either an assumption or a conclusion to be verified. As an assumption, it could be part of a specification for an existing sensor which is assumed to have been verified for sensors of the appropriate type. The statement about the sensor refers to its operating context. The operating

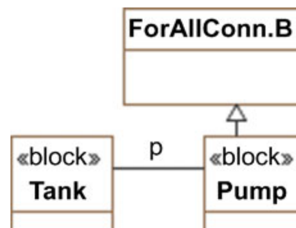


Fig. 4 Pump model

Author's personal copy

context will have its own assumptions. The stability of the sensor in its operating context can be represented as a type

$$SensorStability = \{< s, e >: s : Sensor,\ \ e : Environment, and\ stability(s, e) = true\}. \tag{13}$$

stability is an operator with type (Sensor,Environment):Bool. The pointing accuracy can be represented as a type

$$PointingAccuracy = \{< s, e >: s : Sensor,\ \ e : Environment,\ \ and\ accuracy(s.pointingOp(e) = true\}. \tag{14}$$

In Fig. 5, Sensor contains a component operation declaration

$$pointingOp : Operations(Sensor, (X) : Y). \tag{15}$$

For an individual s of type *Sensor*, the result of applying *pointingOp* to s is written as $s.pointingOp$. The statement can be expressed as:

$$SensorStability \subseteq PointingAccuracy \tag{16}$$

This subclass inclusion gives conditions for the operation to achieve a specific pointing accuracy. For example, one would have to check whether the stability conditions are satisfied when the sensor is mounted on a moving air vehicle under the constraints which have been determined about its motion.

When designing an implementation for an operation, a common technique is to consider known implementable designs and solve for preconditions which will imply that the operation will satisfy required post conditions. The validity of such a solution will depend on whether the preconditions are consistent with the existing system assumptions. If the composite design assumptions are satisfiable, then in the logical encoding the question is whether the original assumptions and precondition are still satisfiable. The additional assumptions are added to the aggregate of assumption about the system and the operating context. If the conjunction of the assumptions cannot be satisfied then the proposed design change will not be valid.

Fig. 5 Sensor block

Sensor
values accuracy stability fieldOfView
operations pointingOp(x:X):Y
pre_post_conditions Pre≤ ∃ f.Post

Author's personal copy

The kinds of inference that occur in applications as illustrated by the examples are not difficult, simply the size and complexity of the theories argue for using reasoning engines as the need for an inference may be missed on manual inspection.

1.3 Comparison with related work

While a logical formalism may have some of the necessary expressiveness, most are deficient in the full spectrum or do not have the breath to serve for macro level system development. Most applications of formal methods within aerospace focus on specific issues such as concurrency analysis [20], design refinement to produce correct executable code [35] or analysis of system reliability (AltaRica) [33]. There appear to be few attempts to integrate the expressiveness of SysML into a logical formalism [32]. Many automated inference procedures are potentially useful for engineering provided they are incorporated into a logical formalism. An inference procedure in itself is not a logical formalism. For example, many inference procedures are special cases of constraint solving inference procedures [21]. Constraint solving can be used in some cases to verify the correctness properties of an operation specified using pre-post conditions, or show that preconditions for the operation cannot be satisfied. However, constraint solving is an inference procedure not a logical formalism.

1.3.1 Theory based formalisms

The formalisms most directly comparable to ABD logic are ones in which specifications as well as descriptions (models in the SysML sense) are represented as an axiom set or the theory generated by an axiom set within a logical formalism. Theory refinement while developed for software is a general paradigm and works equally well for system development. The only difference is that not only the system but its operating context get refined as development progresses. Specification formalisms that are theory based can be compared by kind of signature used, by restrictions made on axioms, and by the kinds of inference is used. For example, SpecWare, as does ABD, uses signatures that include product and function types [2]. In that sense it compares directly with the ABD type theory approach. It is unclear what type constructions are used and what axioms and inference rules are used for the type constructions. For example, it is not clear whether SpecWare has the class constructions for Description Logic and other constructions that are widely used in system modeling. While any specific constructions could be added if they are not already there, providing semantics for constructions is on the order of complexity of developing axioms for set theory. With the expressiveness of SysML extended by ABD logic, functional requirements can be captured as axioms so there is no need to introduce any extra formalism for requirements beyond the constructions discussed here; the gap between requirements and design specifications does not exist [22].

Blocks correspond to classes in many system and software engineering languages such as UML [29], as well as other knowledge representation and conceptual modeling languages such as OWL [31]. Binary associations correspond to properties in these languages. The diagram in Fig. 1 with three blocks, *Pump*, *Tank*, and *MechanicalDevice*, the association *conn* connecting *Pump* to *Tank* and the subclass relation between *Pump* and *MechanicalDevice* expresses that

$$Pump \subseteq MechanicalDevice, \tag{17}$$

Author's personal copy

i.e., any pump is a mechanical device. A common interpretation of the arrow labeled conn is that any pump which is connected to something is connected to a tank and conversely any tank connected to something is connected to a pump. In a conceptual modeling or knowledge representation language such as OWL, this diagram is encoded as an axiom set with essentially the same axioms. The reference semantics of a conceptual modeling language is generally specified in terms of valid interpretation of the axiom set within set theory. The classes in the pump model correspond to sets and the subclass relationship corresponds to set inclusion. The inference semantic of the language is given within first order logic where a block corresponds to a unary predicate and an association is encoded as a binary predicate. In the encoding a subclass relationship

$$A \subseteq C \tag{18}$$

corresponds to

$$\forall x. A^\wedge(x) \ implies \ B^\wedge(x) \tag{19}$$

where $A^\wedge$ and $B^\wedge$ are the unary predicates corresponding to the classes A and B.

Description Logic (DL) [3] provides a logic based formalism using constructions for individuals, classes (types), and properties. In ABD logic, the type constructions have axioms derived from the elementary axiomatization for a topos [24]. The DL class constructions are definable in ABD logic. For example, both representations of the subclass relation above are representable within ABD logic. The importance of DL for ABD logic is that the DL class constructions enable a wide class of assumptions to be expressed as subclass axioms. There are efficient decision algorithms which may be used for the DL fragment of ABD logic. The semantics of the constructions in ABD logic are given by ABD inference rules rather than through model theory. UML class diagrams have been encoded as axiom sets within the Description Logic (DL), ALCQI a sublogic of SHOIQ. The DL encoding for UML and its results carry over to SysML. In SysML, blocks are a stereotype of class and SysML uses associations, as does UML. In this encoding, UML classes are encoded as concepts and UML associations are encoded as roles; to encode the additional information contained in class diagrams, other assertions are needed. The result is that an encoding of a UML class diagram is as an axiom set. The encoding provides a formal semantics for class diagrams which conforms to their informal semantics; the encoding is further validated by comparison of first order logic (FOL) axiomatizations of the UML constructions with the FOL representation of description logic. A consequence of the encoding for integration with reasoning is that a class diagram (class model) corresponds to a knowledge base within a DL [7]. The results of DL consistency checking and derived classes and class inclusions can be reinterpreted within SysML. The DL encoding of SysML is equivalent to the ABD encoding. However, ABD encodes SysML model constructions such as blocks with compartments which are not covered by the DL the class diagram encoding.

Another point of comparison is state machines. A state machine is an abstraction of a computation machine which uses data stores and rules or operations to transform the date stores. A machine is initialized with data in the stores and executes by updating the stores. A snapshot of the machine performing a computation is referred to as a state of the computation. Formalizations of a state machine generally use variables for data stores, and state transformation rules or operations which operate on the variables [28]. Semantics is generally defined in terms of valuations which

Author's personal copy

map variables to their values or rules for transforming the data stores. Formalizations differ in terms of the constructions for concurrent operations and in semantic rules for the operations. ABD as a type theory logic can represent embedded variables, operations, and behavior in terms of state charts. Type theory has the advantage that variables have types and are first class terms, with assignment (substitution) as a first class operation on variables. Thus a formalism that contains variables and transformation rules is grounded soundly in logic. ABD defines a valid interpretation for axioms in terms of valuations on the (state) variables to functions defined on a time domain. In this approach, the state of a machine is an evaluation of the state variables for a time instance which eliminates the need to define an abstract state as a theory or interpretation between theories [2]. In cases where a language is used to describe a state machine, the formalism for the state machines is sometimes referred to as the semantic of the machine [33]. However, it seems more appropriate to describe the semantics of a state machine in terms of a logical language in which it is expressed. From this perspective the language is simply a notation for the state machine and the kind of machine can be compared to the machines that can be described in ABD logic. Concurrency has yet to be addressed in ABD logic.

1.3.2 Translations from SysML to other formalisms

System description and engineering languages that use classes and properties such as SysML are often mapped into logic based language where classes correspond to unary predicates [4]. Such translations can be viewed as making steps toward establishing a formal semantics for SysML. However, the informal semantics of SysML is often not completely captured or preserved when encoded in logic-based languages. Examples include the generation of a B-Specification from a UML class [22] and the generation of a Z-Specification from a UML class [11]. The encoding of the structural part of SysML into DL is described in [14].

A common motivation for encoding SysML/UML into a target formalism is to take advantage of verification procedures for the target formalism. However, when reasoning is integrated through such translations, there need to be arguments for the semantic justification of the translation. Without providing SysML with an accepted semantic foundation, there is no real basis for justifying the correctness of a translation. Many translations simply provide a syntactic translation of languages, often based on translations between meta-models. One risks the possibility of unsound reasoning. However, any translation of SysML into another logical formalism provides some basis for comparison between the ABD type theoretic formalism and the representation of SysML/UML in other formalisms.

One example of using translation for inference is the translation of UML augmented with OCL annotations to Constraint Programming [9]. The resulting specification is translated into a constraint satisfaction problem. The language combination provides considerably expressiveness and integrates with automated inference systems. However, from analysis of examples it is unclear that this language combination provides any more expressiveness than the SysML/ABD approach. Neither SysML nor OCL have a formal semantics and so the translation into a constraint solving paradigm only has an informal justification. Unlike SysML, OCL was not designed for human legibility and there is insufficient evidence that this combination in its present form can be adopted as a general approach to aerospace development.

Author's personal copy

2 ISR use case

This section introduces an Intelligence, Surveillance and Reconnaissance (ISR) design trade study application, illustrates how the application can be represented within SysML as a composite model, i.e., a model that incorporates submodels. The models contain representations of their assumptions. The explicit capture of assumptions enables the possibility of formal design verification. In this sense, the application provides a use case for formalizing SysML. This example wil be formalized in Section 4.

The engineering task is to determine the feasibility of providing an ISR capability to an existing air vehicle platform by adding a sensor system. From an initial domain model of the system under design and its operating context, capability analysis is used to determine the constraints on an air vehicle operating context, including the target and the physical environment, under which the ISR capability is to be exercised. Both the operating context and the air system are further decomposed into components. The result of the capability analysis is a requirements specification for a top level air system operation for target identification assuming prior detection. The air vehicle motion constraints, detection constraints, and target assumptions form the premises for the target identification operation. Development refines the domain model with its constraints to establish the feasibility of meeting the identification requirements while maintaining theory consistency. Feasibility consists of assessing whether the target identification requirements have potential design solutions, i.e., an implementable operation can be defined in terms of operations of components of the air system. Much of the work to produce a design specification is to provide sufficient detail that an instance can actually be physically realized. The end result is an air system design specification that consists of a description of components with their specifications which are used to define the identification operation.

The air system is required to identify vehicles and humans as the air vehicle loiters over an area of interest. In the concept of operations, the air vehicle uses radar to detect and approximately geolocate a target and an Electro-Optical (EO)/Infrared (IR) Sensors to create an image of the target. Actual target identification is performed by an operator crew on the air vehicle as they view a sensor display. The primary engineering task is to determine the allocated EO/IR sensor requirements and to determine if available sensor solutions can be found to meet the requirements. The air vehicle is assumed to be an existing air vehicle accompanied with detailed specification information and validated test results. The identification starts after the initial approximate location for the target has been determined by the air vehicle's radar system. A more detailed presentation of this example would allow for the derivation that the radar can provide geolocation with sufficient accuracy for the EO/IR system to use as an input for identification. Figure 6 illustrates the operation of the air system in its mission environment. The air vehicle flies in a racetrack pattern over the area of interest. The image on the diagram is an operator display of EO/IR image generated by a sensor on an air vehicle flying over the terrain at the distances prescribed in the top level specification.

A top level statement for the capability is given as input to the design process. The capability statement is that the air vehicle system shall have the capability to identify targets, as defined within a threat database, with a cross section of at least 1 m, moving less than 5 mph, from a slant range of at least 5 mi, in clear weather conditions

Author's personal copy

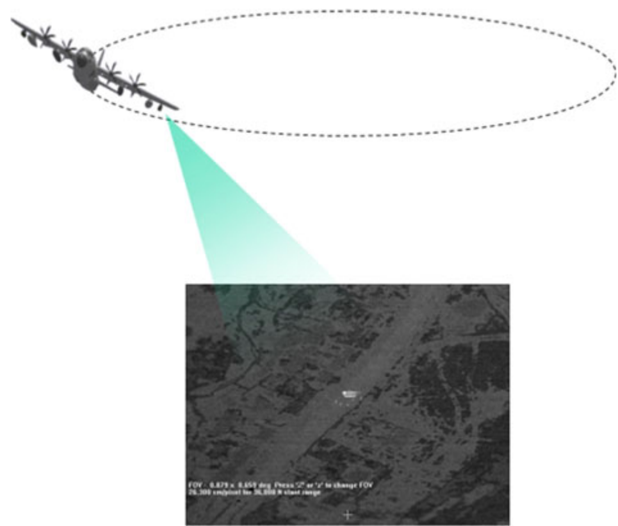


Fig. 6 Concept of operations for the ISR capability

as defined by a weather reference handbook, within ten minutes of initial detection, in an area of interest (AOI) no larger than 100 m by 50 m, with the air vehicle flying at less than 250 knots, with at least 95% probability. The requirement which drives the design trade study is a functional requirement for an air system to have an operation for target identification which provides positive identification when the air system is operating under the conditions to be specified in the Mission Model. A major part of the ISR application task is devoted to making the target identification requirement sufficiently precise so that one can proceed to the design decomposition task. In the end, the statement of the requirement is sufficiently precise that a design solution can be tested with an actual air system in an actual operating context.

A simplification made is to eliminate requirements stated in probabilistic terms, e.g., that correct target identification is only made 95% of the time. We will use point estimates in the detection conditions rather than probabilistic ones. Probabilistic conditions can be treated formally exactly the same as point estimates. What changes is how test in a concrete domain is performed. Trials and probabilistic methods will be required in testing. Also, the dynamic aspects of the problem are replaced with static conditions whose values do not change over time.

The SysML diagrams that represent the ISR application are part of a single composite SysML. Typically, no single diagram contains the entire model. The resulting model was constructed by a design refinement process. During this process multiple refinements were explored to reach the final model. Figure 7 is a SysML Block Definition Diagram (BDD) which is part of a simplified SysML model of an air system and its operating context. This model is typically developed in the requirements analysis phase of a program. It is a formalization of the information indicated by picture in Fig. 6. The Block Definition Diagram shows the structure of the mission domain. The domain includes the air system, target, and physical environment. The boxes are blocks and the closed diamond arrows represent directed compositions, which are referred to as part properties. The whole cannot exist without the parts. Thus, the air vehicle is not a complete system without the sensor

Author's personal copy

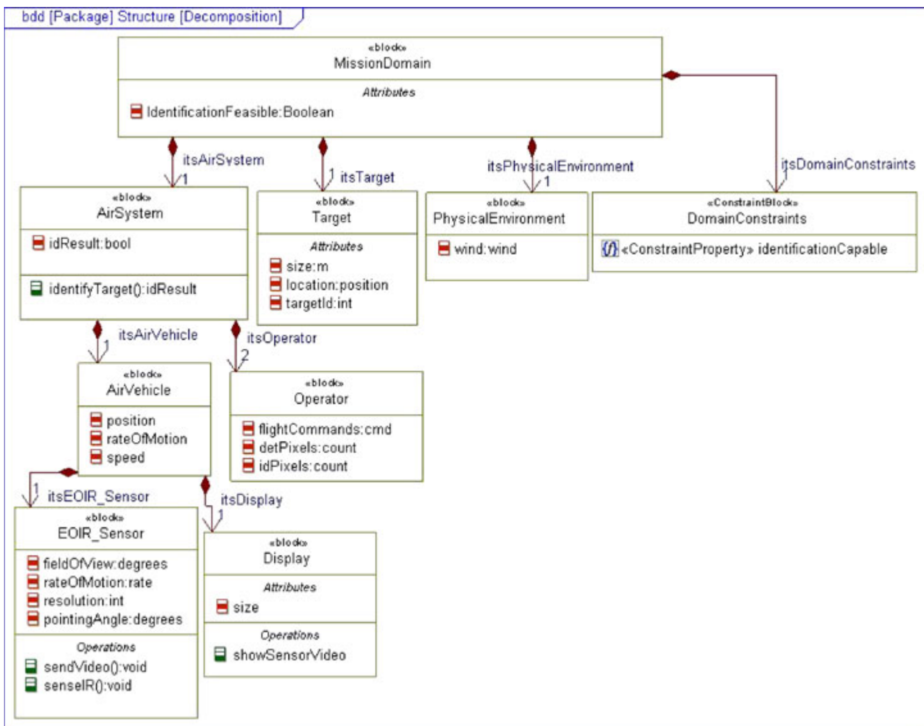


Fig. 7 Mission Model

and the display. The domain is incomplete without the air system, target, and physical environment. *DomainConstraints* is a constraint block that contains the constraints on the mission domain including one for determining if the system is capable of successfully identifying targets. It will be used to check the feasibility of a new sensor to function on the air vehicle and provide the necessary capability given the target characteristics. At the beginning of development the block is a place holder for the constraint to be developed from the domain analysis.

The ISR example design task is to determine the feasibility of providing an ISR capability by adding a sensor system to an existing air vehicle platform. The section will formalize the conditions for target identification requirement and sketch the proof that an operation can be constructed which satisfies the requirement. The primary explicit assumptions for the target identification are aircraft motion and position, and target conditions. For this discussion, we assume that the target has been detected. The three air system components which play a major role in satisfying the requirement are the EO/IR sensor, the display, and the sensor system operator. The component operations which provide the implementation have their own operating constraints. The model which represents the requirement and the design solution are represented within SysML. Additional assumptions will be represented in text and incorporated into the logical formalization.

The design task proceeds by elaborating the Mission Model. This model (see Fig. 7) is used to analyze the desired capability. The domain models used here are

Author's personal copy

the result of performing several pilots which allowed us to understand how to make the models generic. The models also reflect our evolving understanding of how to use SysML and its tools. The diagrams used to illustrate stages of development are actually views of the final resulting model where information is hidden. The elaboration and refinement is based on analysis and execution of simulations of the models. From the logical formalism view point, a simulation is a virtual realization of the Mission Model. Model refinement is continuous throughout the development process. Backtracking occurs when the developers realize that they have modeled the domain incorrectly. Backtracking also occurs when analysis of design decisions indicates that the decisions were not good ones. Domain modeling is highly reusable; physics does not change and if the models are built generically, then they can be specialized for many design tasks.

The first task is to analyze the target identification capability. The result of this analysis contains the functional specification for the target identification function in the form of a SysML parametric diagram. The specification characterizes the conditions under which identification takes place and characterizes what identification means in this context. The functional specification for the air system identification function leads to specifications for the subsystems and allocation of requirements to subsystem operations. The final refinement contains the air vehicle which represents the air vehicle design.

2.1 Setup

The air system is required to identify vehicles and humans as the air vehicle loiters over an area of interest. In the conceptual design, an EO/IR sensor is used to capture an image of the target. Actual target identification is performed by an operator crew on the air vehicle. The air system provides images on an operator display to allow the operator to identify targets. The primary engineering task is to determine the allocated EO/IR sensor requirements and to determine if available sensor solutions can be found. The air vehicle is assumed to be an existing air vehicle accompanied with detailed specification information and validated test results. The identification result required is stated as the probability of successful identification for targets which meet the criteria must be greater than 95%.

Table 1 Air vehicle motion and position characteristics

Name	Condition	Type	Description
Position	$4{,}572 \leq \text{altitude} \leq 10{,}668$ m	position	Air vehicle location includes longitude in degrees, altitude in meters, and latitude in degrees.
Speed	≤ 128.6	m/s	Air vehicle speed.
Direction	0	degrees	Direction air vehicle is traveling in 0 degrees is north.
pitchRate	≤ 5	degPerS	Rate for the air vehicle motion in the pitch direction.
yawRate	≤ 5	degPerS	Rate for the air vehicle motion in the yaw direction.
rollRate	≤ 5	degPerS	Rate for the air vehicle motion in the roll direction.

Author's personal copy

Table 2 Target characteristics

Name	Condition	Type	Description
Size	≥ 1	meters	Target cross section in meters.
Location	θ, ρ, α	position	Target location includes longitude in degrees, altitude in degrees, and latitude in degrees.
Speed	0	m/s	Speed of target.
Direction	0	degrees	Direction target is moving in. 0 degrees is north.

2.1.1 Interaction constraints

The air system identification capability is to be exercised under constraints on an air vehicle operating in its operating context which includes the target and the physical environment. The initial assumptions about these constraints are given in Tables 1, 2, 3 and 4. The conditions in the table are representative for this type of problem. The tables describe air vehicle position and motion, target characteristics, derived distance from the target, and the operating environment characteristics.

2.1.2 Air vehicle motion

The dynamic constraints on the ability of the air vehicle to detect and identify a target depends on the distance from the air vehicle to the target and the on air vehicle motion. We assume, based on empirical evidence, that the range of motion of the air vehicle is less than β degrees per second when it is flying in moderately turbulent air conditions. In Fig. 8, the values for flight commands and wind represent external influences on the motion of the air vehicle. The $rateOfMotion$ is the result of the interaction between the external environment and the air vehicle. Pilot commands and wind turbulence cause the air vehicle to pitch, roll and yaw. The values of the system and operating context (in red boxes) are connected to constraint parameters (squares on the border of the constraint property) in the constraint property (box with rounded corners) via binding connectors (blue lines). Figure 8 represents how motion of the air vehicle depends on both the pilot and on the weather conditions. The rate of motion of the air vehicle depends on pilot commands to fly the air vehicle and on the force of wind blowing against the air vehicle. Flight commands and wind are bound to the constraint property, $avMotion$, as inputs. The $rateOfMotion$ is bound as an output. Typical air vehicle motion was determined empirically from data collected on the air vehicle being modified. We fitted a function of sine waves to approximate the pitch, roll and yaw of the air vehicle under various conditions.

Table 3 Physical environment

Name	Condition	Type	Description
Wind	$\leq \gamma, 0$	m/s and degrees	Speed and direction of the wind blowing against the air vehicle. North is 0 degrees.
Terrain	0	bool	Terrain can affect what the sensor can view. Hills can get in the way of the field of view and needs to be accounted for. 0 means terrain is not an impediment to viewing the target, 1 means it is.

Author's personal copy

Table 4 Distance constraints

Name	Condition	Type	Description
slantRangeEq	≥ 8047	meters	slantRange = f(av.Position, tar.Location)

2.1.3 Target identification conditions

The air vehicle motion constraints, detection constraints, and target assumptions form the premises for the target identification operation to be used for target identification. The diagrams use a probability of success greater than 95%. For simplicity in the formalization discussion we eliminate the probability statement and rephrase the requirement in terms of a Boolean identification. The *TargetIdentificationCondition* contains the assumptions made about identification and the criteria the system must meet in order to perform identification successfully while operating within the conditions of the assumptions. The target detection and identification is to take place when the target has specific characteristics of size (cross section) and the slant range from the air vehicle is within a specified distance. For this example, we assume that the target is not moving. Figure 9 shows the assumptions and constraints about the air vehicle, the target being detected, and interactions for target detection which is a prerequisite of identification. The diagram describes the detection results required as a function of air vehicle attributes (values) and target attributes. It does not represent the constraints under which the air vehicle operates. The diagram shows the distance between the target and air vehicle, *slantRange*, is a function of the location of the air vehicle and target. The probability of detecting the target is a function of the *slantRange*, size of the target, resolution of the sensor and the number of pixels an operator needs to make the determination.

The identification constraints define a region as the set of k-tuples which satisfy the constraints. Informally, a realization of the air system target identification operation is a domain which includes the variables which occur in the mission

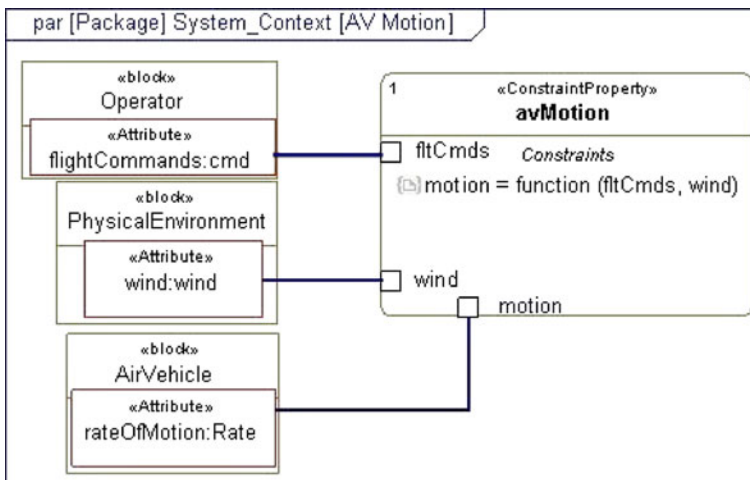


Fig. 8 Air vehicle motion

Author's personal copy

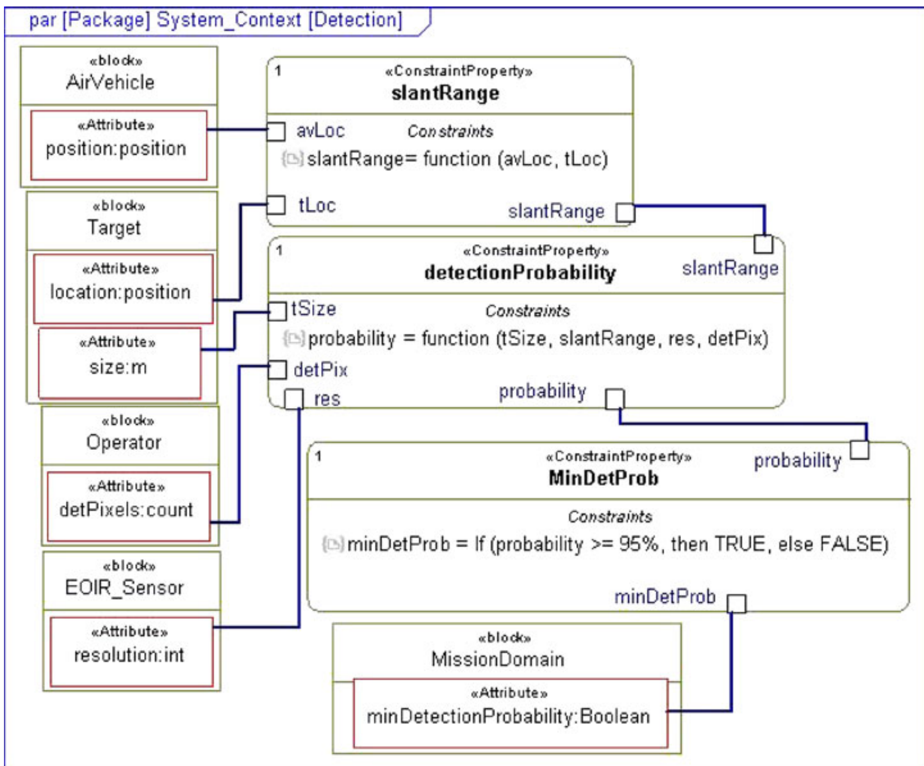


Fig. 9 Target detection conditions

model and which satisfy the motion, detection constraints and target constraints. The identification constraints define a region in k-dimensional number space. The region is the set of k-tuples which satisfy the constraints.

2.2 Design analysis

The mission model is used to establish the feasibility of meeting the identification requirement. Feasibility consists of assessing whether the target identification specification of the air system has potential design solutions, i.e., the design solution space is non-empty. Many conditions beyond the requirement constraints determine the feasibility of the air system's target identification capability. For feasibility analysis, we ascertain the impact of air vehicle motion on the stability of the IR sensor mounted on the air vehicle. This requires verifying that antecedent requirements for sensor stability are satisfied on the air system under operating conditions. If these requirements are satisfied by a sensor with a given specification, then the sensor image can be stabilized. Additional variables and constraints are introduced for the effect that air vehicle motion has on the sensor stability. Other conditions, beyond the requirements constraints, determine the outcome of the air system identification function. These include the mission scenario, i.e., behavior of air vehicle, other entities in the environment, and assumptions about the air vehicle platform. The

Author's personal copy

mission model is used to analyze operations of the air vehicle flying over the terrain in order determine potential solutions and the assumptions for which the solutions are valid, given assumptions about the terrain, constraints on the vehicle, and sensor characteristics. Identification of a target begins once an operator has successfully detected a target. It is difficult for operators to make a definitive identification of targets if the image of the target is bouncing erratically on the display. Therefore, it is necessary to analyze the stability of the sensor mounted on a moving air vehicle. Sensor characteristics such as the field of view, focal length, sensing array, and pixels size are related. It is possible to trade off some of these characteristics against the others while still meeting the overall requirements for identification. An analysis of these characteristics and their relationships is needed in order to determine the design space of feasible sets of values.

2.2.1 Sensor analysis

The target identification operation requires an EO/IR sensor whose rate of motion is capable of keeping images stable enough for identification while the air vehicle is subjected to environmental operating conditions. The sensor stability of motion is a function of the stability of air vehicle motion, as well as target distance. This analysis is used in the following way. The stability of the air vehicle motion is assumed from known characteristics of this particular kind of air vehicle. Recall that we are investigating adding a new sensor to an existing kind of air vehicle. We also assume a specific maximum instability, MAX, is needed for an operator to identify a target within a field of view. The functional relationship allows us to compute the sensor stability of motion which can be used to check against specifications of known sensors. Figure 10 shows parameters (value properties) for the air vehicle and sensor, as well as the assumptions about the physical environment and target that one must consider in order to determine if the sensor is capable of keeping images stable enough for detection and identification of targets. The position of the air vehicle with respect to the target and motion of aircraft in turbulence and rate of sensor motion play a role in the stability of the sensor. Sensor stability is impacted by the air vehicle motion.

We use a constraint property *stability* to represent the sensor stability conditions and a constraint property *avMotion* for the motion of the air vehicle. Pilot commands and wind pushing against the air vehicle are input parameters to *avMotion*. The outputs are the rate of motion for the pitch, roll and yaw of the air vehicle. The sensor must keep up with the motion of the air vehicle in order to keep the target in the field of view and stable enough for the operator to detect or identify the target. The EO/IR senor value property, *rateOfMotion*, represents the rate of motion for the sensor and acceleration. The *rateOfMotion* must be sufficiently high to deal with the impacts of motion of the air vehicle on the sensor. The output of the stability constraint property is a boolean that represents whether or not the sensor can provide a stable enough image. The constraint property, *inFoV*, determines whether or not the target is in the field of view of the sensor. The equation for *inFOV* was exported to a simulation tool and values for the air vehicle position, target location and target size are assumed. The value for *inFOV* was forced to be true in the simulation. The pointing angle of the sensor in the azimuth and elevation directions was allowed to change in the simulation to values that were required for the *inFOV* to be true. The motion of the air vehicle was approximated by an equation made of a combination of

Author's personal copy

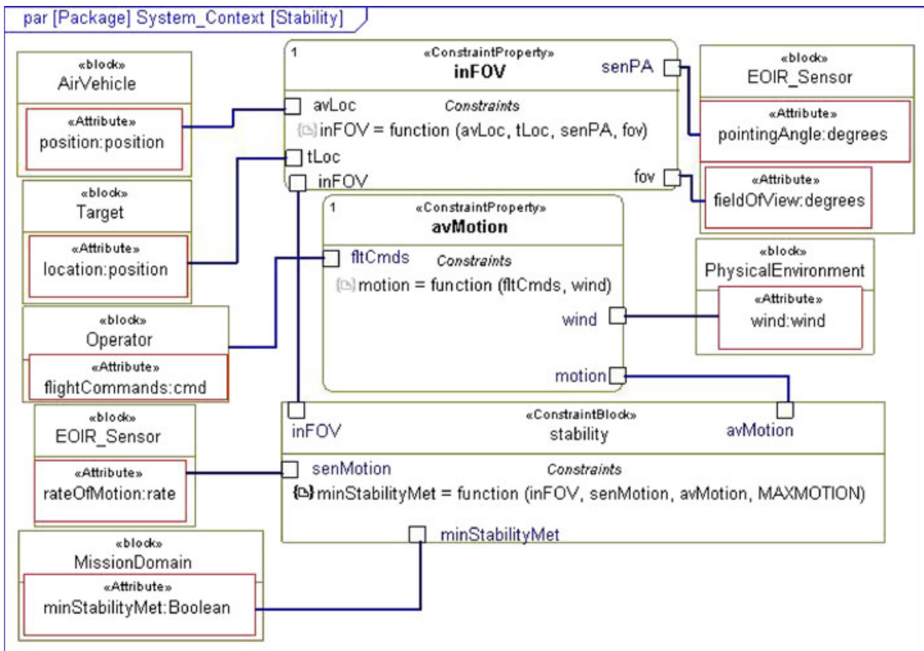


Fig. 10 Sensor stability

sine waves that reasonably matched the collected data for the air vehicle. The speed of the air vehicle was set in the simulation tool to a nominal operating value. The target speed was set to 0 for this example. A constraint for stability was set to be less than $MAXMOTION$ which was determined by experimentation with operators to determine how much motion on the display could be tolerated by the operator trying to identify a target. The simulation was then programmed to keep the sensor pointed at the target at all times. The pointing angle of the sensor was programmed to change in response to the motion of the aircraft. The simulation tool recorded the rate of motion for the sensor slew and acceleration required to keep the sensor pointed at the target while the air vehicle moved. Any sensor that has the required rate of motion and acceleration can be used on the air vehicle being modified.

2.2.2 Operator analysis

Assumptions about operator performance introduce new assumptions which are incorporated into the mission model. These assumptions are about the ability of operators on the air vehicle to be able to identify the target by looking at the sensor display screen.

The minimum number of pixels the operator needs for successful identification of a target is determined statistically from running experiments, with displays that had an assumed size and resolution. Analysis is performed to determine the minimum number of pixels that the target needs to cover on a display in order for the operator to have a 95% probability of successfully identifying the target. The results of the analysis are inputs for the required $idPixel$ count. The equations were exported

Author's personal copy

manually into an optimization tool. Assumptions about the environment, such as target size for targets the customer wants to identify, target location and air vehicle location were set inside the optimization tool and goals such as the 95% probability were set. The optimization tool can then solve for pixels needed. The number of pixels can then be used later when determining the sensor characteristics required to provide the number of pixels.

During the course of design, the identification function, f, is decomposed into sub functions implemented by subsystems. In general, one would not like to have to introduce any new assumptions at the top level which are needed for the component property verification. In the air system, the requirements for sensor stability did not pose any additional top level constraints as the sensor constraints were satisfied by the top level constraints and the sensor specification. On the other hand, if new assumptions imposed by the potential design solutions are necessary, then they need to be added to the top level assumptions together with any evidence that they can be satisfied in the operational context. This situation was the case for the assumptions regarding an operator's ability to identify a target on a display with a specific resolution. Figure 11 displays the Identification constraints. The slant range equation and probability equation were exported into an optimization tool manually. Assumed values of the location and speed of the air vehicle, the location and size of the target were set as inputs. The assumed values were based on customer need. The minimum number of pixels on the target for a 95% probability of success was set as a constraint. The values of focal length and other sensor characteristics can be

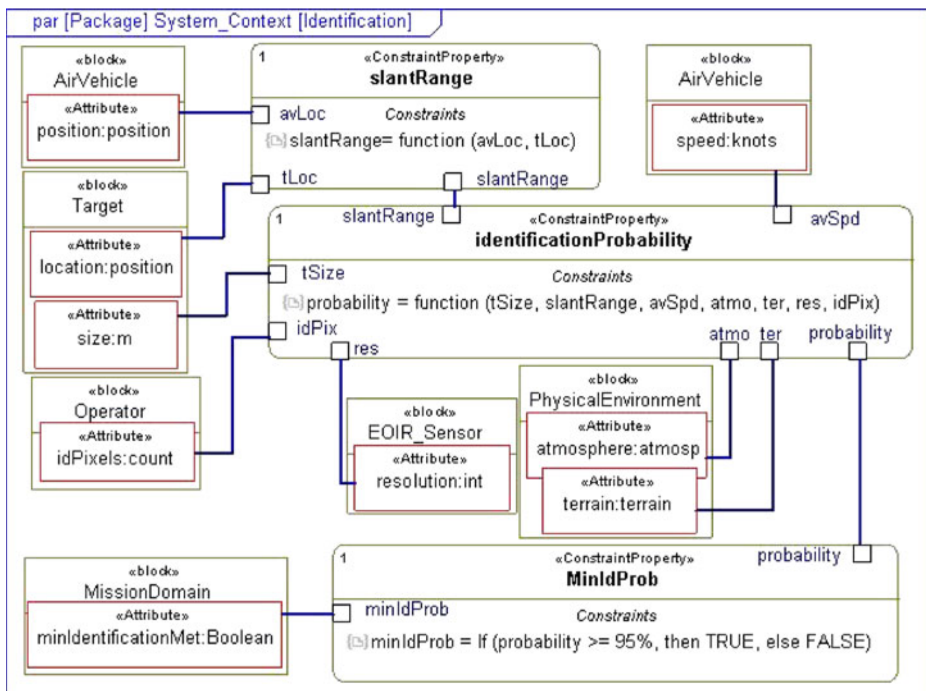


Fig. 11 Identification conditions parametric diagram

Author's personal copy

evaluated to determine if the sensor resolution can meet all off the constraints and assumptions.

2.2.3 Derived specification for display

The target identification requirement is for the air system. However, as we have noted, the delivered system is an air vehicle without the crew. Therefore, our design task is to derive a requirement for the air vehicle with given assumptions regarding operator identification performance. This can be used to prove the air system target identification requirement. The derived air vehicle requirement will be on an operation which displays the sensor video.

2.3 Design construction

Any air vehicle that can meet the constraints can successfully meet the formal specification for target identification for any target that meets the specified assumptions. The design task is to construct an operation for the air vehicle, definable

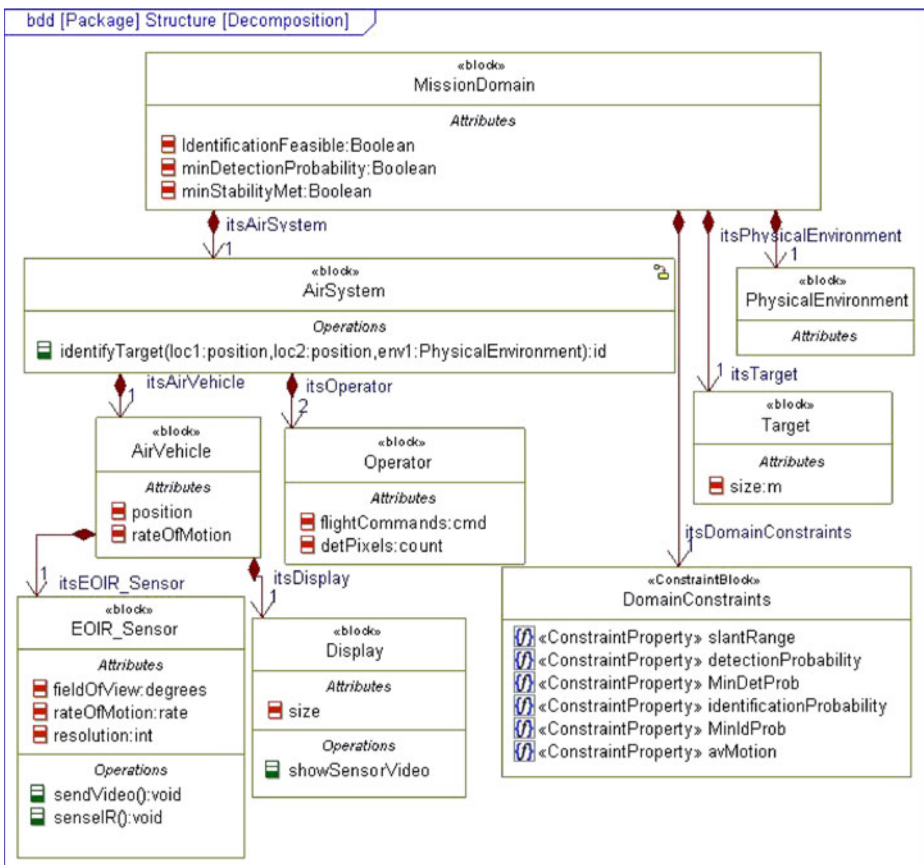


Fig. 12 The final decomposition diagram

Author's personal copy

in terms of component operations. The burden is to show that such a function which satisfies the specification exists. A composition of operations which meet the requirements is defined and a sketch of its verification is given in Section 4. Much of the work to produce a design specification is to provide sufficient detail that an instance can actually be physically realized.

Figure 12 shows the decomposition of the system and the constraints on the system for the identification function. Each of the components has attributes that represent the characteristics of the subsystems determined in the previous models. Operations have been allocated to the component that will be responsible for performing the necessary functions. The model has been used to establish that EO/IR sensor mounted on the air vehicle has the characteristics needed for target identification. The operator has an attribute for the minimum number of pixels needed for target identification; the sensor has attributes for the field of view and resolution. The target has attributes for its size and location. The constraint block contains the relationships that must be considered in determining the value of the attributes. An air system design, as the term is use here, is a decomposition (parts) tree for an air vehicle block, with connections defined between components, and where each component has value properties (variables) and operations. The air vehicle specifies that an air vehicle implementation has two components, the display and EOIR sensor. The display has a specified size.

We have performed sufficient analysis of the air system operation in its mission context that we have a precise formalizable statement of the requirements. We have also performed sufficient design analysis that we can formalize a design solution in terms of an operator viewing a display of the sensor results. These statements will be formalized in Section 4 after we have given more detail on the ABD logic.

3 Formalizing SysML

This section presents more detail on ABD Logic and on how SysML is embedded in the logic. Abstract Block Diagram (ABD) logic [17] is a variant of type theory. ABD logic is more expressive than SysML but in the overlap they correspond sufficiently closely that ABD logic can be viewed as a formal semantics retrofit for SysML. ABD extends SysML, as we have noted by having description logic class constructions and by having individuals. ABD logic also has a concept of operator evaluation which is not explicitly present in SysML. Operator evaluation is used in the proofs of correctness of the target identification operation defined within SysML. ABD has a linear syntax and a graphical syntax modeled on SysML. Not all of the linear syntax has a corresponding graphical syntax. While the graphic syntax of SysML carries over to ABD Logic the presentation will use a linear textual syntax. ABD logic has operation and type constructions for sum, product, function, and power type. While ABD logic is a type theory, it can to all intents and purposes be viewed as a set theory in that familiar set constructions are available as type constructions. Familiar function constructions are available as operator constructions. In ABD logic classes and properties are types. ABD logic has a primitive binary predicate for typing terms, logical equality, as well as a substitution predicate. ABD logic has type class and property equality. Type inequality is definable within the logic. The typing relation is written in infix notation, e.g., $a : A$. The symbol "=" is used for equality between

Author's personal copy

terms and types. Inference rules are given for the term constructions (see Table 5). ABD logic is equational in that all inferences reduce to proving a term equality, i.e., $t1 = t2$ provided that the terms have been verified to have the same type. The inference rules provide the semantics for the term constructions. General type theory results [23] imply that ABD theories are sound and complete.

3.1 An ABD axiom set

An ABD axiom is a binary relation between terms constructed from the primitive predicates, $typeOf$ and $equality$. Several defined predicates are used, such as subtype. The terms are classified into operations and types. A declaration is an axiom that uses the binary $typeOf$ relation to introduce a symbol and types the symbol. The second argument provides the type of the symbol. The type argument of a declaration consists of a type construction defined in Table 5. For example, the two declarations $C : Type$ and $a : C$ declare C to be a type and a is declared to be an instance of C. A collection of declarations specify an ABD language. The atomic operators and types that occur in the declarations are the signature of a language. The type symbols of the signature are called the sorts. The type and operator terms of the language are constructed from the signature using the term constructions with their typing rules in Table 5. A term construction t is well formed only it follows from the typing rules that $t : X$ for some type X. It is well-known that there is no algorithm to decide if an arbitrary program terminates. Of course one can prove that individual programs terminate. For an operator defined without constructions, such as recursion, typing is decidable. For the version of ABD logic used here, operator term typing is decidable and can be used as part of syntax checking. An axiom set is recursively defined by the declarations which introduce symbols and by equalities or defined relations which may only use terms which are in the language of the declarations. The ABD theory of an axiom set are the relationships derived from the axioms using the inference rules in Table 5 and derived inference rules. The relationships may only use terms definable in the language of the axiom set and for which a typing relation has been established.

Table 5 gives syntax and axioms for basic operator and type term constructions. The table is incomplete as we only include the term constructions that are used in the ISR example. The table does not include sum types or the full typed lambda calculus constructions and rules. In applications one often extends an ABD theory with external functions which may or may not be computable. Some of the term constructions and corresponding rules are definable in terms of other ones. DL class constructions are definable; some of these definitions are given below.

3.1.1 Inference rules

The semantics of an ABD axiom set is given by inference rules for the term constructions, the relations and by derivation rules. The derivation rules are used to construct new inference rules from previously constructed rules. Each rule is depicted as a fraction; the inputs to the rule are listed in the numerator, and the output in the denominator. The denominator is a relation and the numerator is a sequence of relations separated by commas. The relations in the numerator are called

Author's personal copy

Table 5 Syntax

Name	Syntax	Typing rules	Equality rules
Type of	t:X	t is an operator term and X is a type term	
Equality	t1 = t2	Equality is only permissible when t1 and t2 have the same type.	The term equality rules are used to simplify and rewrite terms. Term equivalence is an equivalence relation. A term may be substituted for an equal term.
Product type	(X1, ..., Xn), One	Rounded brackets are used for the Cartesian product construction.	One = ()
Variable	x: var:X		
Substitution	[t/x]	t:X, x:Var X ———————- [t/x]:X	x[t/x] = t = t[x/x]
Tuple	$< t1, \ldots, tn >$	$< t1, \ldots, tn >$:(X1,...,Xn)	t:(X1,...,Xn), x1:var X1,...,xn:var Xn —————————— t = [t/x1,...,t/xn] x :var (X1,...,Xn), x1:var X1,...,xn:var Xn —————————— $x = < x1 \ldots xn >$
Function type	(X):Y	f:Y —————— f(X):Y	x :var X ———————— f = f[x/x]
Number	N	A generic number type that includes the natural numbers and is equipped with arithmetic operations	
Operation declaration	f(x1:X1,..., xn:Xn)	f(x1:X1,...,xn:Xn) ———————- f(X1,...,Xn): Y	f() = f f(x1:X1,...,xn:Xn) = f(y1:X1,...,xy:Xn)
Composition	f(g)	g(X):Y, f(Y):Z ———————————— f(g)(X):Z f(x1:X1,...,xn:Xn):Y, t1:X1, ...,tn:Xn —————————- f(t1,...,tn)(X1, ...,Xn):Y	f(x1:X1,...,xn:Xn):Y, t1:X1, ...,tn:Xn ———————————— f(t1,...,tn) = f[t1/xn,...,tn/xn] f:(X1,...,Xn) ———————————— f = $< f/x1, \ldots, f/xn >$
Identity operator	id(X)		id(x1:X1,...,xn:Xn) = $< x1, \ldots, xn >$ var:X = (x:X):X id(X) = $< x1, \ldots, xn >$
Truth type	Bool = P(), true	true:Bool	
Abstraction type	X{p} incl{p}(X{p}):X	p(X):Bool —————————- X{p}, Monic{f}	p(incl{p}) = true

Author's personal copy

Table 5 (continued)

Name	Syntax	Typing rules	Equality rules
Characteristic operation	Char{t}	Monic(f), f(X):Y ———— char{f}(Y):Bool	f = incl{char{f}} char{incl{p}} = p
Membership			a:X{p(x)} ———— p(a) = true a:X, p(a) = true ———— a:X{p}
Power type	Pow(X) ε p*	ε(Pow(X),X):Bool p(X, Y):Bool ———— p*(X):Pow(Y)	p = ε(id,p*)
Relation conversion	r^	r(X):Pow(Y) ———— r^(X,Y):Bool	p*^ = p r^* = r
Equality operation	eq	eq(X,Y):Bool	eq(id,id) = true eq(t1,t2) = true ———— t1 = t2
Logical and	∧	p(X):Bool, q(X):Bool ———— (p ∧ q)(X):Bool	(p ∧ q) = char{X{eq(p,q)}}
Implication definition	p implies q	p(X):Bool, q(X):Bool ———— (p implies q)(X):Bool	(p implies q) = char{X{eq(p,p ∧ q)}}
Universally quantified formula definition	∀x.p	p(X,Y):Bool ———— ∀x.p(X):Bool	∀ x.p = char{{p*}}
Intersection definition	X ⋂ Y	p(X):Bool, q(X):Bool ———— X{p} ⋂ Xq : Type	X{p} ⋂ X{q} = X{ p ∧ q}
Subtype relation definition	X ⊆ Y	p(X):Bool, q(X):Bool ———— X{p} ⊆ X{q}	X{p} ⋂ X{q} = X{p}
Enumeration type	{a1,…,an}	ai: X ———— {a,…,an}⊆ X	b: {a} ———— b = a
Class	Class(X)	p:Pow(Thing) ———— Class(Thing{p})	Class(Thing)

the premises and the denominator is called the conclusion. Substitution of terms by equal terms within a relation is used by the inference rules in term construction. The term equality rules are used to simplify and rewrite terms. Substitution of values for variables and term simplification provides the foundation for the concept of "evaluating" a term. The theory of an axiom set is the collection relationships derivable from the axioms using the inference rules.

The derivation rules are written in a linear form

$$P1, \ldots, Pn \rightarrow Q \qquad (20)$$

Author's personal copy

The Pi and Q are formula. In general, the Pi can be rules. An axiom Q can be written as

$$\rightarrow Q \tag{21}$$

A derivation is constructed using one of the rules:

$$\begin{array}{r} P \rightarrow Q, Q \rightarrow R \\ ------\,-- \\ P \rightarrow R \end{array} \tag{22}$$

and

$$\begin{array}{r} \rightarrow P, P \rightarrow Q \\ ----\,-- \\ \rightarrow Q \end{array} \tag{23}$$

The last derivation rule is used to eliminate premises.

3.1.2 Operations

ABD operations are comparable to SysML operations. In SysML, an operation such as the pointing operation on a sensor is defined as belonging to the sensor. However, operations may be declared globally. The mechanism for representing ownership of an operation is discussed in the paragraph on composite structure. The semantics is given by equational rules for variable substitution and functional application. Functional application, also called composition, is defined in terms of term substitution and satisfies the expected properties. The rules for operations are standard from computer science and logic. A declaration such as

$$f(x : X) : Y \tag{24}$$

introduces f and a variable x. The type X is called the argument type and Y the value type. An operator with no arguments $f() : Y$ is an individual and is equated with $f : Y$. For the declaration $f(x : X) : Y$, the type of f is, $f(X) : Y$. A variable can be declared with $x : var\ X$ where $Var\ X$ is equated with the function type $(X) : X$. An operation can be defined in terms of previously introduced operations. For example, using the number type and arithmetic operations, an operation f can be defined as:

$$f = x + 1 \tag{25}$$

The constructions include n-ary operations and n-ary product types. If $ai : A1, \ldots, an : An$, the notation $< a1, \ldots, an >$ is used for an n-ary tuple. The type is given as $< a1, \ldots, an >: (A1, \ldots An)$ where $(A1, \ldots An)$ is the n-ary product.

One consequence of the rules for product types and tuples that is used frequently is that variables are projection operators. For example, if

$$x = id(X1, X2) \tag{26}$$

then

$$x =< x1, x2 > \tag{27}$$

Author's personal copy

and

$$x1, x2 : var(X1, X2). \tag{28}$$

3.1.3 Power type and formulae

The expressiveness of type theory depends on having a power type [26] construction. The power type construction enables the definition of subtypes and provides a correspondence between a subtype and its "characteristic" Boolean typed operator. For a sequence of types $X_1, \ldots, X_n$ the constructor $Pow(X_1, \ldots, X_n)$ is a type called the power type. The truth value type, *Bool* is identified with $Pow()$. The type Bool has two truth value individuals *true* and *false*. An operation term p with $p(x1 : X1, \ldots, xn : Xn) : Bool$ is called a formula. An instance t of an n-ary power type defines an n-ary formula. For

$$t : Pow(x_1 : X_1, \ldots, x_n : X_n) \tag{29}$$

The term $char\{t\}$ has type

$$char\{t\}(x_1 : X_1, \ldots, x_n : X_n) : Bool \tag{30}$$

with

$$char\{t\}(a_1 : X_1, \ldots, a_n : X_n) = true\ iff < a_1 : X_1, \ldots, a_n : X_n >: t. \tag{31}$$

Note that an instance t of a power type $Pow(X)$ may be the type of an instance of X. A formula $p(x_1 : X_1, \ldots, x_n : X_n) : Bool$ can be used to define a type

$$\{< x1, \ldots, xn >: p(x1, \ldots, xn) = true\} \tag{32}$$

called an abstraction type. The abstraction type characterizes the operation terms that satisfy the formula. That is:

$$\begin{aligned} < a1, \ldots, an >: \{< x1, \ldots, xn >: p(x1, \ldots, xn) = true\} \\ iff\ p([a1/x1, \ldots, an/xn]) = true. \end{aligned} \tag{33}$$

and

$$\{< x1, \ldots, xn >: char\{t\}(x1, \ldots, xn) = true\} = t \tag{34}$$

A subtype relation may be defined between abstraction types using the inclusion relation. Equivalently, the relation may be defined for the instances of a power type, Pow(X) for a type X. For $t1, t2 : Pow(X)$, the definition

$$t1 \leq t2\ ;\ iff\ ;\ char\{t1\}(incl(t2)) = true \tag{35}$$

satisfies the properties expected for a subtype relation.

ABD logic has a number type, N. With the type N we can define types such as $\{x : x : N\ and\ x > 1\}$ as well as types with cardinality restrictions on subtypes of a power type. One has $a : \{x :\ x : N\ and\ x > 1\}$ iff $a > 1$. Also, one can define $\{x : |p(x)| = k\}$ where $p(x) : Pow(Y)$ is a subset of a power type and $|p(x)|$ is the cardinality of the type.

Author's personal copy

3.1.4 Classes and properties

A declaration of the form $A : Class$ declares A to be a class. ABD has a top class $Thing$, and a bottom class $NoThing$. $NoThing$ is a subclass of any class and any class is a subclass of Thing. In ABD logic, there is a natural correspondence between classes and unary Boolean operations, and properties with Boolean operations with two class arguments. In particular, if $p : P(A, B)$, then pair $< a, b >$ may be declared to be an instance of p with $< a, b >: p$. If $p : Pow(A, B)$ and $< a, b >: p$, then the inference rules imply $a : A$ and $b : B$. For $C : Class$ the subtype construction provides an operator $c\hat{} : (Thing) : Bool$ with

$$c\hat{}(a) = true\ iff\ a : C = true \tag{36}$$

This correspondence enables the identification of a class with a unary Boolean typed operator. A binary property in ABD logic has a domain and a range type. A property with domain A and range B is a subtype of the product type of two types. A declaration $p : Pow(A, B)$ declares p to be a property with a domain and a range types A and B. A property $p : Pow(A, B)$ determines an operation $p(A) : Pow(B)$ whose range type is a power type. This construction determines a correspondence between binary properties and binary Boolean typed operators. We write $a.p$ for the relational composition. For any $a : A, a.p = y : p\hat{}(a, y) = true$.

DL class constructions can be defined within ABD logic; their semantics is defined by derived inference rules. For example, ABD has the class constructions $\forall p.C$ and $\exists p.C$ where $p : Pow(A, B)$ and C is a class. For $p : Pow(A, B)$ the DL class construction

$$Max3p = \{x : |\{y : p\hat{}(x, y)\ = true\}| < 4\} \tag{37}$$

has the property that any individual x with $x : Max3p$ has the property that the cardinality of $\{y : p\hat{}(x, y)\}$ is less than 4, i,e., x can participate in at most 3 instances of the property p. More generally we have

$$\begin{aligned} \exists p.C = \{x : \left|\{y : p^\wedge(x, y) = true\ and\ C^\wedge(y)\}\right|\} \\ \forall p.B = \{x : p^\wedge(x, y)\ imply\ B^\wedge(y)\} \end{aligned} \tag{38}$$

A consequence of the power and abstraction type constructions is that in ABD class and property axioms are equivalent to axioms stating that their characteristic formulas (Boolean operations) which they correspond to are equal to *true*.

3.1.5 Composite structure

A block may have an internal structure which can be shown within compartments of a block in a diagram. Corresponding to a SysML block which has compartments, ABD has a form of declaration which uses type constructors including *Parts*, *Attributes*, *Operations*, and *StateCharts*. These constructors have two type arguments and a type value. Examples are illustrated with Fig. 7, the top level SysML Block Definition Diagram for the ISR model. A composite structure declaration has the form

$$name : CompositeConstruction(X, Y) \tag{39}$$

where X and Y are types. The constructed type is a subtype of the power type, $Pow(X, Y)$. Recall that $Pow(X, Y)$ is the type of properties with domain X and

Author's personal copy

range Y. A named instance of one of the composite types is a property. The composite structure instances are all functional properties with the exception of the parts constructor which is not in general functional. For the parts constructor, the suffix [1] declares that a part property instances are functional. For a functional property, there is a unique second argument for any first argument. The composition of two functional properties is functional. For example if:

$$\begin{aligned} p1 &: Parts(A, B)[1] \\ p2 &: Parts(B, C)[1] \end{aligned} \tag{40}$$

then the composition $p1.p2$ written in a left-to-right order is a functional property with

$$p1.p2 \ : Pow(A, C). \tag{41}$$

Different composite constructions place constraints on the form of the second argument type. For example, for an operations declaration such as

$$pointingOp : Operations(Sensor, (X) : Y) \tag{42}$$

The second argument of the operations construction is a function type. The operation *pointingOp* is a functional relation. Again, the dot notation *s.pointingOp* is used to denote the application of *pointingOp* to s for $s : Sensor$. In both SysML and ABD, the value of an attribute (SysML value property) is a variable which can be substituted for and its value set.

3.1.6 Constraints

A SysML constraint declaration is another composite construction. A constraint construction enables defining equations in a finite number of variables and binding the variables to value properties (also variables) in the context of the constraint definition. The top level ISR mission domain model, Fig. 7 shows the constraints blocks of *MissionDomain*. The constraint blocks have constraint properties which are the equations. The SysML parametric diagram is used to bind variables in a constraint property to value properties of blocks. Figure 8 shows a parametric diagram with a constraint property, *avMotion* that represents how motion of the air vehicle depends on both the pilot and on the weather conditions. The rate of motion depends on pilot commands to fly the air vehicle and on the force of wind blowing against the air vehicle. The *avMotion* equation has as variables: *fltCmds*, *wind*, and *motion*. The small boxes displayed on the edges of the constraint property are called constraint parameters. Lines connect the constraint parameters to the value properties of the blocks. The value properties and the constraint parameters are variables and the lines connecting them are binding connectors.

The corresponding constraint can be declared in ABD logic with:

$$\begin{aligned} avMotion &: Constraints(MissionDomain, \\ Equation(&itsOperator1.flightCommands/fltCmds, \\ &itsPhysicalEnvironment.wind/wind, \\ &itsAirVehicle.rateofMotion/motion) = \\ &[function(fltCmds, wind)/motion] \end{aligned} \tag{43}$$

Author's personal copy

In this declaration, value properties of $itsOperator1$ and the physical environment are bound to variables in the equations. The binding of the SysML parameters in the constraint is represented by substitution of value properties for argument variables of the equation. This equation is functional the first three arguments.

The constraint construction provides a mechanism for importing reusable assumptions such as physics models and can be used to enforce relationships between subsystem value properties and value properties within a subsystem. Figure 11 shows a parametric diagram that contains three constraint properties. The three constraints are functional; the diagram represents a composition $minidProb$ with $identificationProbability$ with $slantRange$. The variables in $slantRange$ are bound to $itsAirVehicle.position$ and $itsTarget.location$ which are value properties. For the mission domain we have:

$$\begin{aligned} slantRange : Constraints(MissionDomain, \\ Equation([itsAirVehicle.position/avLoc, itsTarget.location/tLoc]) = \\ [fn1(avLoc, tLoc)/slantRange] \end{aligned} \tag{44}$$

The variables of the parametric diagram (parameters) are bound to the value properties of part properties of $MissionDomain$. With the substitutions into $identificationProbability$ we have:

$$\begin{aligned} identificationProbability : Constraints(MissionDomain, \\ Equation([\\ itsSlantRange.slantRange/slantRange, \\ itsAirVehicle.speed/avSpd, \\ itsTarget.size/tSize, \\ itsOperatoridPixels/idPix, \\ itsEOIRSensor.resolution/res, \\ itsPhysicalEnvironment.atmosp/atmo, \\ itsPhysicalEnvironment.terrain/ter, \\ probability, \\ ]) = \\ function(tSize, slantrange, avSpd, atmo, ter, res, idPix)/probability \end{aligned} \tag{45}$$

Finally, $MinidProb$ checks if the probability result meets the minimum probability requirements and sets the result to $itsMissionDomain.minidentificationMet$ value property.

3.1.7 States

While reasoning about dynamic behavior is beyond the scope of this paper, we indicate how behavior can be represented within ABD and its impact on the interpretations of a theory that contains dynamic behavior. A state machine with behavior may be axiomatized by adding a type Time to an ABD theory. For discrete

Author's personal copy

sequential time, Time has a successor operation written as "+1". The values of a state machine attribute change as a function of time while their structure remains constant. To represent such state machines, their embedded variables are declared with a time argument. For example, a location attribute of an air vehicle changes value over time as it flies. We make extensive use of simulations of state machines in an operational context to determine how the systems respond. The simulations are designed to be valid interpretations of theories extended with time.

For example, a class, *Switch*, can be defined as a sequential state machine with the declarations:

$$display : Attributes(Switch, (Time)Var(X)) \tag{46}$$

The values of the time indexed variable display varies with respect to time, i.e., for $a : Switch$ the embedded variable

$$a.x(t) : var(X) \tag{47}$$

is a function of time whose values are variables of type X.

$$\begin{aligned} mystate &: hasStateChart : (Switch, var(State)) \\ State &= (Time)\{off, on\} \\ mytransistion &: hasOp(State, ((display : X, s : State))(display : X, s : State)) \\ mytransition &= \\ &(if display(t) = off \ then \ display(t+1) = on, \\ &else \ display(t+1) = off)(s(t+1) := off) \end{aligned} \tag{48}$$

The state of $a : Sensor$ at time t, is the value of the embedded variables $a. < s, x > (t)$. The states satisfies the relations imposed by the guards and the state changes corresponding to the actions. The states of $a. < s, x > (t+1)$, for i greater equal zero are a sequence starting with an initial state $< s, x > (t_0)$. A valid interpretation of the theory that contains mySwitch maps the variables to functions from the interpretation of time to the interpretation (Time)X.

3.2 Semantic correlation of ABD logic with SysML

SysML has a rich, if incomplete type system. SysML has blocks, properties, and operations. Blocks may have compartments with components such as parts, value properties, operations, constraints, and state charts. Some of the SysML language constructions are representable in description logic or more generally first order logic. However, SysML operations and composite structures are not representable within first order logic and require higher order operations, such as ABD logic provides. The ABD representation of a SysML model as an axiom set corresponds well with informal notions of interpretations of models. Informally, a realization of a SysML model such as the ISR model is a correspondence of the signature of the model to subtypes of a domain which satisfies the equations and subtype relations of the model. The signature of a model includes a collection of atomic types, properties, and operations and the composite constructions. By power and abstraction type constructions and the internal formula language (Boolean typed operations), the

Author's personal copy

inference semantics of the constructions used in embedding of SysML into ABD logic is defined in terms of inference rules in Table 5 and the derived inference rules. The reference semantics is the model theory of ABD logic. As ABD logic is sound and complete, either the reference or the the inference semantics can be used for correlation with informal SysML semantics. This greatly simplifies the discussion. where the SysML semantics have been represented axiomatically in first order logic or represented with respect to reference semantics common the semantics are equivalent. Some specific notes follow.

In SysML, operators may be declared globally or in a composite construction. An operator may have multiple arguments. A declaration for an operation may give a definition in terms of composition and other function operations in the form of an equality of the name in the declaration with a body. Operators may have variables and operations of substitution are permitted, at least in parametric diagrams. Parametric diagrams allow the substitution of constraint property equation variables to value properties of part properties. No explicit semantics is given for operations. Implicitly the semantics is taken to be that of function calculi found in software languages and in logical formalizations of function calculus.

The semantics of SysML blocks and properties was discussed in Section 1.3. Recall that ABD classes are subtypes of the type *Thing*. The ABD semantics for blocks, properties, and subtype relation are equivalent to the direct first order logic representation which has been suggested as the semantics for SysML. The ABD logic semantics is also equivalent to the description logic semantics.

Composite structure declarations were discussed in the current Section as were constraints. The informal semantics of a SysML model with composite structure can be illustrated with an example. The declarations

$$\begin{aligned} p1 &: Part(A, B)[1] \\ p2 &: Part(A, C)[1] \\ loc &: Values(B, X) \\ loc &: Values(C, X) \end{aligned} \tag{49}$$

define a composite model. This model can be realized as a subtype of the product type

$$M = \{a : a : A \text{ and } a.p1.loc : X \text{ and } a.p2 : C\} \tag{50}$$

where a, $a.p1.loc$, and $a.p1.loc$ are variables. An interpretation of the axiom set is a tuple in M which satisfies any constraint equations that may be given in these variables. This model theoretic semantics coincides with the informal notion of a realization of the SysML model.

4 Formalization of ISR

This section describes the embedding of the SysML ISR model as an axiom set in ABD logic. ISR model components are expressed in the linear ABD syntax. The air system target identification requirement is expressed as a formula within ABD logic and a proof is sketched that a target identification operation defined within the model satisfies the requirements formula. The operation is defined as a composition

Author's personal copy

of a human operator on the aircraft making an identification decision from the sensor display. The proof depends on additional assumptions regarding the ability of an operator to identify a target from a display with a specific resolution. The sensor is assumed to provide an image with a specified resolution at a given distance from the target. Assumptions are also made regarding the motion of the aircraft. The final axiom set contains these additional assumptions. A few changes to the model have been made in the ABD encoding based on the analysis of requirements formalization and design verification. These changes could be reincorporated into the SysML model.

Informally, the statement that the air system target identification operation is satisfied must be true in any instance of the mission domain. Similarly the statement the air system operation $itsAirSystem.identifyTarget$ satisfies its requirements is to be true in any instance of the mission domain. This means that the statements are quantified over all instances of a mission domain. As we will see by examining the declarations of the ISR model and using the ABD logic constructions we can represent $MissionDomain$ as an abstraction subtype of a type X. $MissionDomain$ has the form

$$\{m : m = < m1, \ldots mn > \ \ and\ p(m) = true\} \tag{51}$$

where X is a product type and p is a formula (Boolean valued operation). This eliminates the need for explicit quantification and simplifies proofs.

4.1 Mission domain axioms

SysML uses a Block Definition Diagram (BDD) to describe the top level ISR model. The ABD axiom set encoding the ISR application consists of declarations corresponding to the graphical syntax corresponding to Fig. 7 and other diagrams which are referenced from this diagram. The structure of $MissionDomain$ is defined by the declarations in the ISR model. The part declarations form of a tree of declarations which include

$$\begin{aligned}
itsAirSystem &: Parts(MissionDomain, AirSystem)[1];\\
itsAirVehicle &: Parts(AirSystem, AirVehicle)[1]\\
itstarget &: Parts(MissionDomain, Target)[1];\\
itsEnvironment &: Parts(MissionDomain, Environment)[1]\\
itsOperator1 &: Parts(AirSystem, Operator)[1]\\
itsOperator2 &: Parts(AirSystem, Operator)[1]\\
itsSensor &: Parts(AirSystem, EOIRSensor)[1]\\
itsDisplay &: Parts(AirVehicle, Display)[1]\\
itsEOIRSensor &: Parts(AirVehicle, EOIRSensor)[1]\\
view &: Operations(Operator, ID)
\end{aligned} \tag{52}$$

Author's personal copy

The value property declarations define operations whose range types are variables. Informally, these variables are the state variables of the model. For example,

$$\begin{aligned} identificationResult &: Attributes(AirSystem, var(ID)) \\ location &: Values(AirVehicle, var(Loc)); \\ location &: Values(Target, var(Loc)) \end{aligned} \quad (53)$$

are operations. The operation is typed

$$itsAirSystem.itsAirVehicle.location(MissionDomain) : Loc \quad (54)$$

To define the ABD type that represents *MissionDomain* note that each of these operations is unique. For all of the variable value operations occurring in the ISR declarations, let $xi : Xi$ be a variable with its corresponding type and let

$$Domain = (x1 : X1, \ldots, xn : Xn) \quad (55)$$

be the product type $(X1, \ldots, Xn)$ Let

$$allConstraints(x1 : X1, \ldots, xn : Xn) : Bool \quad (56)$$

be the conjunction of all constraints with the substitutions made. Then *MissionDomain* is the subtype of *Domain* defined as:

$$MissionDomain = \{< x1, \ldots, xn >: allConstraints(x1, \ldots, xn) = true\}. \quad (57)$$

The axioms and statement to be proved will be represented as subtype relations defined in terms of *MissionDomain*.

4.2 Air system requirements

The air system target identification requirement states that the air system can correctly identify a target provided conditions regarding the distance to the target and air vehicle motion are met. The requirement does not mention any operation to be used for identification. The target has a location attribute and the air system has as attribute used to store the value of an identification operation when one is defined, and a constraint to check that suitable conditions are met; its arguments are distances, locations, speeds, etc. The mission domain has a Boolean attribute for the result of the conditional test that the air system attribute has the correct value when the suitable conditions are met. Given the representation of *MissionDomain* as

$$\{< x1, \ldots, xn >: allConstraints(x1, \ldots, xn) = true\} \quad (58)$$

The formalized requirements will be the subtype of *MissionDomain* consisting of those tuples which satisfy the further constraint *identificationCapable*. We introduce the name *MissionDomain* {*identificationCapable*} for this type.

Author's personal copy

Figure 11 shows a parametric diagram that represents the requirement for the air vehicle to be able to identify a target when constraints are met. The constraint can be declared in ABD logic with:

$$\begin{aligned} identificationCapable &: Constraints(MissionDomain,\\ &Equation(identificationFeasible,\\ &[itsAirVehicle.position/avLocation],\\ &[itsAirVehicle.itsSensor.resolution/res],\\ &[itsTarget.size/tSize],\\ &[itsTarget.location.tLocation)]) =\\ &if(function(avLocation, res, tSize, tLocation)) < k\\ &then identificationFeasible = True\\ &else identificationFeasible = False) \end{aligned} \tag{59}$$

The parametric diagram can be modified slightly to serve as a formal requirement for target identification. The constraint property tests whether $IdentificationFeasible$ is true and compares the results of the Air System $idResult$ attribute with the target identification attribute. The modified version is:

$$\begin{aligned} &identificationCapable : Constraints(MissionDomain,\\ &Equation(idFeasible, avLocation, res, tSize, tLocation) =\\ &if\ IdentificationFeasible(avLocation, tLocation, tSize, res)\\ &eq\ true\ then\\ &[identificationResult/idr] == [itsAirsystem.sensedid\ eq\ true \end{aligned} \tag{60}$$

The requirements subtype has the form

$$\begin{aligned} MissionDomain\{identificationCapable\} &=\\ &\{< x1, \ldots,\\ &itsAirvehicle.location,\\ &itsTarget.size,\\ &itsTarget.location,\\ &itsSensor.resolution,\\ &\ldots, xn >:\\ &(function(itsAirvehicle.location,\\ &itsTarget.size,\\ &itsTarget.location,\\ &itsSensor.resolution,\\ &) < k = true\} \end{aligned} \tag{61}$$

Author's personal copy

Note that *function* only uses some of the components of $< x1, \ldots, xn >$. Further note that we are assuming the requirements are consistent, but have not proved it. For the requirements to be consistent the type *MissionDomain* {*identificationCapable*} must not equal *NoThing*. However, on real programs requirements are often inconsistent.

4.3 The air system targeting operation

This subsection defines the target identification operation and the type of those tuples where applying the operation gives the correct value. The arguments available for the air system target identification operation are the environment attributes such as *Environment.wind* and attributes which represent atmospheric conditions. 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. The operator views the sensor display to make the actual identification. Assumptions about operator the ability of the operator to identify a target provided the sensor resolution is sufficient are part of the model.

The air system target identification operation is declared with:

$$identifyTarget : Operations(AirSystem, (en : Env) : ID \tag{62}$$

The operation is defined as a result of the design analysis. The operation is a composition of first using the sensor to produce a video image which is displayed and viewed by the air system operators which make the identification from the image. The only input that the sensor system has is the environment. The declarations are:

$$\begin{aligned} showSensorVideo : Operations(Display, (env : Environment) : VideoStream) \\ view : Operations : (Operator, (Display) : ID) \end{aligned} \tag{63}$$

Finally, the identification operation of the air system is defined as

$$\begin{aligned} identifyTarget(AirSystem) = \\ itsOperator.view(itsDisplay.showSensorVideo(itsSensor([AirSystem.env/e]))). \end{aligned} \tag{64}$$

To verify that *itsAirSystem.identifyTarget* satisfies the target identification condition means that we have to show that whenever the constraints in *identificationCapable* are satisfied then

$$\begin{aligned} itsAirSystem.itsOperator.view(itsDisplay.showSensorVideo \\ (itsSensor(itsEnvironment))) = itsTarget.id \end{aligned} \tag{65}$$

Author's personal copy

The states in *MissionDomain* for which the targeting operation is correct can be written as:

$$MissionDomain\{eq(itsAirSystem.itsOperator.view(itsDisplay.showSensorVideo \\ (itsSensor(itsEnvironment))), itsTarget.id) = true\}. \quad (66)$$

We abbreviate this to $MissionDomain\{IdentificationCorrect\}$. The type $MissionDomain\{identificationCorrect\}$ is defined using the result of evaluating the target identification operation.

4.4 Verifying the air system targeting operation

The verification will have the form $MissionDomain\{identificationCapable\} \subseteq MissionDomain\{identificationCorrect\}$ which means that the correct identification results whenever the preconditions are met. The objective now is to verify the correctness of the result that the target identification operation applied to arguments meet the identification preconditions. The air system's target identification operation is the composition of the operator on the air vehicle viewing the sensor display which displays the result of the sensor's target identification operation. The conclusion depends on a number of assumptions which are expressed in the model but whose justification is outside the mission model.

The target identification operation is composed of an operator on the air vehicle viewing the sensor display. The image display that the operator uses to identify the target depends on the sensor image and on degradation of video feed, Screen resolution, refresh rate, and Screen size. 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. This assumption can be represented as:

$$MissionDomain\{stability\&distanceCondition\} \subseteq \\ MissionDomain\{identificationCorrect\} \quad (67)$$

With this assumption what has to be shown is that the sensor satisfies the stability and distance conditions when the *identificationCapable* conditions are satisfied.

4.4.1 Operator identification assumptions

The operator assumptions give conditions on a sensor display image for which an operator can make a correct identification. The assumptions are derived from empirical studies of how operators perform. 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 less than MAXMOTION.

Author's personal copy

MAXMOTION was determined by experimentation with operators. The functions which give sufficient conditions are:

$$\begin{aligned} itsOperator &: Parts(OperatorDomain, Operator)[1]; \\ itsDisplay &: Parts(OperatorDomain, Display)[1] \\ itsSpec &: constraint(OperatorDomain, Equation(\\ & imageStability(display) < MAXMOTION, \\ & imageResolution(display) < MINRESOLUTION \\ & display.pixels > MINPIXELS) \end{aligned} \tag{68}$$

These declarations define a subtype of MissionDomain

$$\begin{aligned} ImageGood = \{m : mvar : MissionDomain, \\ itsSpec(m) = true\} \end{aligned} \tag{69}$$

The assumption that we make about operators can be expressed as:

$$ImageGood \subseteq MissionDomain\{IdentificationCorrect\} \tag{70}$$

If we can show that

$$MissionDomain\{identificationCapable\} \subseteq MissionDomain\{IdentificationCorrect\} \tag{71}$$

Then the target identification condition has been verified.

4.5 Sensor assumptions

The design solution for the air system targeting operation is contingent on finding a sensor which under the mission domain assumptions has the property that its display produces video images from which an operator can correctly identify the target as indicated above. The specification for the sensor is based on electromagnetic laws of physics applied in their engineering form. For the sensor they are expressed as a constraint. In the form applied to air systems they are represented by Fig. 10. *imageStability&resolution* is given as 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.

$$\begin{aligned} imageStability\&resolution(inFOV(avloc, tloc, sensor.pA, sensor.fov), \\ avMotion(fltcmd, wind) < k \end{aligned} \tag{72}$$

We assume that $avMotion(fltcmd, wind)$ is less than some constant for which the value of the *imageStability&resolution* function has an acceptable value.

Author's personal copy

The composite declarations for $EOIRSensor$ include:

$$\begin{aligned} resolution &: Values\{Sensor, varN\} \\ rateOfmotion &: Values\{Sensor, varrate\} \\ fieldOfview &: Values\{Sensor, vardegrees\} \\ pointingAngle &: Values\{Sensor, vardegree\} \\ displaySenseIR &: Operations\{Sensor, e : Environment) : Video\} \end{aligned} \tag{73}$$

The sensor specification can be written in the form

$$\begin{aligned} &sensorspec : Constraints(MissionDomain, \\ &Equation(imageStability\&resolution(inFOV(avloc, tloc, sensor.pA, sensor.fov), \\ &avMotion(fltcmd, wind)), MAXMOTION) \end{aligned} \tag{74}$$

If we write the sensor specification as a subtype of $MissionDomain$ we have:

$$SensorSpec = \{m : m : MissionDomainandsensorspec(m) = true\} \tag{75}$$

Then the assumption regarding $SensorSpec$ is that:

$$SensorSpec \subseteq ImageGood. \tag{76}$$

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

$$\begin{aligned} MissionDomain\{IdentificationFeasible\} \cap SensorSpec \cap \\ ImageGood \subseteq MissionDomain\{IdentificationCorrect\} \end{aligned} \tag{77}$$

Thus we have shown that *MissionDomain{identificationCapable}* $\subseteq$ *MissionDomain{identificationCorrect}* by showing that all of the antecedent conditions for identification are a subtype of the requirement type.

What needs to be done more carefully is to sort out the physics laws and formulate them as reusable models which can be imported into any application. In this case we used the empirically derived law for operators and the application of electromagnetic laws to sensors. Note that a generic model for the operator will contain a

Author's personal copy

representation of the operator context. The variables in the model which represent the general context will have to be matched (unification) with the variables of the application model; in this case the air system operating in its context.

5 Conclusion

This paper demonstrates the validity of the claim that formal methods can be used in macro level system engineering to solve everyday tasks. By embedding a SysML model within an ABD axiom set, the informal SysML semantics is in accord with the formal ABD semantics. Common engineering questions regarding a model are equivalent to questions regarding an axiom set. Maintaining model consistency during development is one of the many uses axiom set consistency checking. The ABD logic is amenable to automated inference to check problems equivalent to axiom set consistency. The SysML to ABD embedding has a very high potential payoff in terms of shorting design cycles and decreasing rework.

5.1 Lessons learned

The first thing learned by working out examples such as the ISR application is that SysML has the expressiveness to represent aerospace applications. It was initially surprising that SysML constraint constructions could be used to express domain constraints and functional requirements. Once one understood the SysML language constructions it was clear that they have direct logical representation. The process of refining capability analysis into requirements and refining requirements into designs does not present any insurmountable technical objectives, provided the domain model is refined in parallel to the system under design. Execution of concurrent behavior via simulation plays a major role in understanding possible system interaction with its operating environment in the upper levels of system analysis and design. In our experience with the ISR application, concurrency was not the most pressing issue in the first several refinement iterations. From early capability analysis through preliminary design there are time and accuracy budgets. However, the analysis and reasoning can be represented with static assertions. For more detailed analysis and design one of course needs concurrency analysis. While the reasoning for the initial analysis and design does not involve a lot of concurrency, almost all of the component models in our examples have executable state charts. They play a major role in constructing simulations, which are the primary avenue to understand what a capability means and the emergent behavior as the system interacts with its physical environment. The results of the assumptions are translated back into static conditions.

A SysML model translates so naturally into an axiom set in a type theory logic that one can simply view a model as an axiom set. Viewing a SysML model as an axiom set enables using meta-logical methods to analyze models. For example, to determine if a model has the correct level of detail for an application translates into the question of whether the intended interpretations correspond to the logical valid interpretations. A model is sufficiently detailed when its interpretations are exactly what is intended. One may want a very general model with many valid

Author's personal copy

interpretations exactly so that it can be used in multiple contexts. There are many meta-matematical techniques which might assist in methodology for model development.

Representing assumptions as part of a model enables many engineering problems to be translated into logical problems of axiom set consistency. Axiom consistency lends itself to automated inference. The ISR example model assumptions were primarily of two kinds; SysML constraints which are equations between value properties and part property and interconnection assumptions can be expressed using extended DL class constructions of ABD logic. Both these kinds of assumptions lend themselves to automated inference for consistency checking. Many of the problems can be represented in decidable fragments of ABD logic.

5.2 Integration into engineering practice

As SysML is retrofitted with a formal semantics, the results of inference can be presented back to the users in the context of their development tools, typically as the statement that some class is void. The integration of inference engines directly with SysML means that users are not required to learn and use some separate logical language in order to employ reasoning tools. For example, users are not required to translate a SysML model into a B-Spec or an OWL model, perform inference in an external environment, and translate the result into the SysML context. The usefulness of the SysML semantic retrofit is hardly limited when the inference engines operate directly on the SysML models and results can be directly presented to users in their tool environment. After adoption of an MBSE model refinement paradigm no additional paradigm change is needed to adopt a formal theory refinement approach to system development. In our experience, an MBSE design refinement approach does not take more time to set up than more traditional produce development approaches, provided one uses good modeling principles such as refining both the system and its operating environment. We have executed applications in parallel with their execution performed using more traditional non-model based approaches. The MBSE approach wins. The work done in the MBSE approach will have to be done sooner or later; the later the work is done the more costly it is and more rework is required. Even if only a single project is done the cost is less. When multiple projects are done, much of the modeling work can be reused, if it is done correctly. As SysML becomes formalized, no additional work will be required for deployment for model refinement to become theory refinement. The SysML formalization retrofit does not add appreciable development overhead as there is no need to learn a new language or set up separate systems and translations between them.

5.3 Type theory as formalism choice

Type theory, also sometimes called higher order intuitionist theory, is any of several formal logical systems [5, 23, 27]. Type theory is used as a basis for many programming languages and theorem proving systems [10] and has been used as a foundation formalism alternative to set theory for mathematics and computer science [27]. Type theories underlie many interactive theorems proving and proof construction

Author's personal copy

systems [18]. Type theory contains type constructions for product, sum, function (operation) types, and abstraction types, and even an internal logic of formulas as the theories have a truth value type. Type theory has higher order term constructions associations of parts, variables, operator. Type theory has the potential to accommodate dynamics in a simple way with the inclusion of a time type as a subtype of One, the terminal type. Type theories have an inference semantics given by inference rules and a reference semantics given by a notion of valid interpretation (model in the sense of logic). The approach of using type theory has been outlined is a series of papers by the authors [13–15]. SysML and ABD theory corresponds so closely that ABDs can be viewed as a formal semantics retrofit for SysML. While the SysML type system is not fully closed under all of the standard type theory constructions, ABD logic is closed under the type constructions present in type theory [6]. The type theory constructions and their axioms are closely modeled on topos theory and have been implemented as an interactive theorem proving system [16] called Algos. The theory generated from an axiom set with the inference rules is sound and complete with respect to models defined by a topos [6]. A topos looks for modeling intents and purposes like set theory and so makes a good venue to talk about interpretations and model theory. Lawvere [24] proposed using topos theory as an alternative to set theory as a foundation for mathematics. Each topos is associated with a type theory and each type theory is defines a topos. A topos can be constructed from a type theory by identifying terms which are provably equal.

Section 3 demonstrated that SysML language constructions and their informal semantics match type theory language constructions and inference rules closely. Many of formalisms encountered for aerospace systems have a general similarity in that they typically use concepts such as individuals, properties, and operations; they have some form of type system, whether it is called types, classes, or sets. Generally variables and assignment statements are used. The choices for formalizing SysML are to translate SysML models into one or more logical formalisms, or to provide SysML with a direct formal semantics either in the form of inference rules or in the form of a model-theoretic semantics. Both the direct and the translation formalization approaches have the issue whether the formal semantics accords with the informal semantics of the modeling language. Of course the issue is not always clear cut. The informal semantics may be incomplete, inconsistent, or simply not accord with what is found in formal systems. All of these formalization approaches can be used to critique the informal semantics of SysML and suggest improvements.

The full expressiveness of type theory provides the additional advantage that it has an internal logic which consists of the Boolean valued operations. These operator terms represent a full first order logic with quantification. For example, a subtype axiom may be expressed as

$$\forall x.x : B \; implies \; x : A \tag{78}$$

an inference rule (where A and B are variables in the metalogic). Further, this rule is provably equivalent to

$$A \subseteq B. \tag{79}$$

The latter form is much more amenable for theorem proving.

Author's personal copy

5.4 SysML limitations

Our approach builds on the demonstrated effectiveness of SysML to represent large scale aerospace systems and model their operational contexts. Our experience leads us to believe that the effectiveness of SysML in product development depends on SysML having language constructions that are well suited for modeling aerospace systems. Engineers can and do use SysML to develop complex aerospace systems. There is good tool support. We illustrated how a typical macro-level aerospace design problem can be represented as a composite SysML model. SysML, with its constraint language features provides expressiveness to represent assumptions in the form of equations. For engineers to make use of formal methods, we have chosen to retrofit SysML with a type theoretic formal foundation. The alternative is to develop or adopt a logical formalism and find or develop tools and get them used on a broad scale. While one could consider integrating multiple formalisms without a unifying language and bridging with transformations between formalisms, this requires a lot of work and provides a lot of opportunity for errors. Retrofitting SysML is more likely to be more successful than choosing a formal logical system lacking wide spread robust tool support. SysML and type theory (abstract) syntax and semantics are sufficiently close that they can be integrated and the result has a very high potential payoff in terms of shorting design cycles and decreasing rework.

SysML language is known to fall short in representing continuous time models (e.g. dynamics) but also when it comes to concurrency in general. As noted, aerospace systems are usually hybrid reactive systems. Behavioral constructions can be added to ABD theory as additional axioms. The use of topos theory reference semantics provides a sound foundation for modeling dynamic systems. Many function spaces defined on some base space such as Euclidian space-time are toposes and some can be axiomatized by elementary (first order) axioms. Their distinguishing feature is the Boolean type is no longer a two element set but a more general kind of algebra. At present, the authors do not yet have enough experience with verification of dynamic properties in the SysML context and so we are not yet able to make a good choice in this regard.

A formal semantics along the lines of ABD logic should be part of the formal specification of SysML. Specific extensions to SysML that would be practically useful based on the ISR use case are: to (1) include a complete set of DL class constructions, add individuals, and (2) provide a mechanism to bind operation arguments to attribute values and bind the value of an operation to an attribute. SysML could easily adopt the Description Logic class constructions. A simplification, or expansion, of the parametric binding capability could make it easier to express pre and post conditions on operations. More generally it would make sense to add the full collection of ABD language constructions with their inference rule semantics to the formal specification of SysML. One would of course have to add time, concurrency, and a more explicit concept of execution.

References

1. Abrial, J.R.: The B-Book. Cambridge University Press, Cambridge (1996)
2. Anlauff, M., Pavlovic, D., Smith, D.: Composition and refinement of evolving specifications. In: Proceedings of Workshop (2002)

Author's personal copy

3. Baader, F., Calvanese, D., McGuinness, D.L., Nardi, D.: The Description Logic Handbook. Cambridge University Press, Cambridge (2010)
4. Barendregt, H.: Handbook of Logic in Computer Science, vol. 2. Oxford University Press, Oxford (1992)
5. Bell, J.: From absolute to local mathematics. In: Synthese. Springer, New York (1986)
6. Bell, J.: The development of categorical logic. In: Handbook of Philosophical Logic, vol. 12. Springer, New York (2005)
7. Berardi, D., Calvanese, D., De Giacomoa, G.: Reasoning on UML class diagrams. Artif. Int. **168**(1–2), 70–118 (2005)
8. Boileau, A., Joyal, A.: La logique des topos. J. Symb. Log. **46**, 6–16 (1981)
9. Cabot, J., Clariso, R., Riera, D.: Verification of UML/OCL class diagrams using constraint programming. In: IEEE International Conference on Software Testing Verification and Validation Workshop (2008)
10. Coquand, T., Huet, G.: The calculus of constructions. Inf. Comput. **76**(2/3), 95–120 (1988)
11. Dupuy, S., Ledru, Y., Chabre-Peccoud, M.: An Overview of Roz: a tool for integrating UML and Z specifications. In: 12th International Conference CAISE'00, Stockhom, Sweden (2000)
12. Estefan, J.A.: Survey of Model-based Systems Engineering (MBSE) Methodologies. Rev. B, INCOSE Technical Publication, International Council on Systems Engineering (2008)
13. Graves, H.: Constructions for modeling product structure. In: OWL Experiences and Directions October Workshop (2010)
14. Graves, H.: Logic for modeling product structure. In: Proceedings of 23rd International Workshop on Description Logics (2010)
15. Graves, H.: Ontological foundations for SysML. In: Proceedings of 3rd International Conference on Model-Based Systems Engineering (2010)
16. Graves, H., Blaine, L.: Algorithm transformation and verification in algos. In: Third International Workshop on Software Specification and Design. IEEE Computer Society Press, Silver Spring (1985)
17. Graves, H., Guest, S., Vermette, J., Bijan, Y.: Air vehicle model-based design and simulation pilot. In: Simulation Interoperability Workshop (SIW) (2009)
18. Harel, D., Pnueli, A.: On the Development of Reactive Systems. Springer, New York (1989)
19. Hoare, C.: An axiomatic basis for computer programming. Commun. ACM **12**(12), 576–583 (1969)
20. Hoare, C.: Communicating sequential processes. Commun. ACM **21**(8), 666–676 (1978)
21. Jaffar, J., Michael Maher, J.: Constraint logic programming: a survey. J. Log. Program. **19/20**, 503–581 (1994)
22. Laleau, R., Semmak, F., Matoussi, A., Petit, D., Hammad, A., Tatibouet, B.: A first attempt to combine SysML requirements diagrams and B. Innovations in Systems and Software Engineering **6**(1–2), 47–54 (2009)
23. Lambek, J., Scott, P.J.: Introduction to Higher-Order Categorical Logic. Cambridge University Press, Cambridge (1986)
24. Lawvere, F.W.: An elementary theory of the category of sets. Proc. Natl. Acad. Sci. **11**, 1–35 (1964)
25. MacKenzie, D.: Mechanizing Proof. MIT Press, Cambridge (2001)
26. Marquis, J.-P., Gonzalo, E., Reyes, G.: The history of categorical logic. In: Kanamori, A. (ed.) The Handbook of the History of Logic vol. 6. 1963–1977. (to appear) webdepot.umontreal.ca
27. Martin-Lof, P.: Constructive mathematics and computer programming. In: Logic, Methodology and Philosophy of Science (1982)
28. Michel, D., Gervais, F., Valarcher, P.: B-ASM: Specification of ASM a la B. In: Abstract State Machines, Alloy, B and Z: Second International Conference, ABZ 2010, Orford, QC, Canada, February 22–25 (2010)
29. OMG Formal Ontology Definition Metamodel. http://doc.omg.org/formal/09-05-01
30. OMG Systems Modeling Language (OMG SysMLŮ), V1.1 (2008)
31. OWL 2 Web Ontology Language, W3C Working Draft 11 June 2009
32. Padawitz, P.: Swinging UML. Lect. Notes Comput. Sci. **1939**, 162–177 (2000)
33. Point, G., Rauzy, A.: AltaRica: constraint automata as a description language. Eur. J. Autom. (Hermes) **33**(8–9), 1033–1052 (1999)
34. Rushby, J.: Formal methods and the certification of critical systems, SRI-TR CSL-93-7 (1993)
35. Srinivas, Y., Jullig, R.: Specware: formal support for composing software. Lect. Notes Comput. Sci. **947/1995** (1995)