Proc. 23rd Int. Workshop on Description Logics (DL2010), CEUR-WS 573, Waterloo, Canada, 2010.

Logic for Modeling Product Structure

Henson Graves

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

Abstract. *A fragment of type theory with OWL class constructions for types and binary properties is used to formalize SysML Structural Block Diagram models. A structural SysML block diagram model is a model that does not have behavior in the sense that values of properties do not change. A structural model may include properties, variables and operations. Individuals are a special case of operators with no arguments. Type theory is chosen as the target semantic formalism as SysML constructions correspond closely to type theory term constructions. The type theoretic semantics defined in terms of introduction and elimination fits well with the informal SysML semantics. An abstract version of a structural model, called an Abstract Block Diagram (ABD), is introduced. An ABD is a theory closed under specific type, property, and operator constructions with additional axioms. The ABD corresponding to the SysML model contains axioms in the form of equations for types, and properties, and operators. This formalism captures the syntactic constructons of the SysML models and the type theoretic semantics appears to be in accord with the informal semantics, as documented in the OMG specification. ABD theory gives an explicit mechanism for introducing instances for types defined by property restrictions. This construction is useful for parts decompositions. ABD theory constructions have a limited kind of property union used to construct parts decompositions. An ABD determines a Description Logic (DL) closed under union, intersection, and existential type constructions and property constructions restricted by typing relations. The ABD constructions are useful in identifying potential extensions for SysML and may be useful, as well, for adding operator terms to Description Logic.*

Keywords: Description Logic, Ontology, OWL, Product Model, SysML, Type Theory, UML.

1 Introduction

The systems engineering language SysML [3] is a natural starting point for developing a formal logic for product modeling. SysML is sufficiently expressive to represent complex product structure such as occurs in aircraft and automobiles [4], has a graphical syntax, engineers can use it, and it has good commercial tool support. There is no other language in this category. The SysML graphical syntax uses several kinds of diagrams which are all views of a single SysML model. Providing SysML

with a formal semantics allows engineers to work with the tools they use today and apply formal reasoning to the results.

A fragment of type theory with OWL class constructions for types and binary properties is used to formalize SysML Structural Block Diagram models. A structural SysML block diagram model is a model that does not have behavior in the sense that values of variables do not change. A structural model may include variables and operations. Type theory is chosen as the target semantic formalism as SysML constructions correspond closely to type theory term constructions. The semantics of a type theory, presented in terms of introduction and elimination rules, is close to the informal semantics of SysML. An abstract version of a structural model, called an Abstract Block Diagram (ABD), is introduced. An ABD is a theory closed under specific type, property, and operator constructions with additional axioms. The ABD corresponding to the SysML model contains axioms defined in terms of equations for type, and properties, and operators.

The constructions in an ABD are needed for modeling product structural properties such as parts decompositions and (horizontal) relationships between product components. In an ABD each property P is typed with a domain and range type. ABD properties are closed under composition, inverse, restriction, and union, provided the typing conditions are met. The use of typed properties enables an exposition of typed “parts” properties [19,20,21]. The type theory semantics provides an explication and semantics for the "dot" notation to provide a fully qualified name for a part whose existence is guaranteed by an existential restriction. For example for a typed property P(A,B) and an individual a:A, the ABD contains a term a.P which has type B. Operators while not fully exploited in static models are never the less useful in that they allow value properties to be defined on a type algebraically. For example, the weight can be define as the sum as the weight of the product’s components.

OWL2 [16] has been used as a formalism for capturing the structural part of a block diagram model [8,9,10]. The correspondence between SysML and OWL constructions is well known [16]. While SysML does not have all of the type constructions found in OWL2, these constructions are needed to capture the semantics of a SysML model. SysML properties are typed with a domain and range types; they can be represented as OWL2 properties with axioms which express that the domain and range properties. Information regarding subtype and equality relations between types implicit in SysML models is translated into OWL2 axioms [10]. A structural SysML model without operators and variables can be translated into an OWL KB in a semantics preserving way. However, operators are not included in OWL2. The translation into OWL KB requires explicit axioms for the domain and range classes for each property in the SysML model.

However, a type theory with type constructions closely modeled on OWL2 class constructions and typed property constructions provides SysML with semantics which is not currently supplied by OWL2. The ABD formalism simplifies the characterization of those SysML models which represent structural SysML models. ABD constructions include typed operators and variables as they are used in SysML. The ABD formalism suggests possible extensions for SysML block diagram models and a method for adding operators with variables to Description Logics.

2 Abstract Block Diagrams

An Abstract Block Diagram (ABD) consists of a finite set of basic type, property, and operator symbols with constructions for each kind of term. Property and operator terms symbols have type signatures. The syntax of an ABD theory is characterized by a recursive definition, in which the constructors that can be used to form terms are stated. The ABD types are closed under intersection, union, finite enumeration types, existential restriction, and have type constants for top and bottom types. A property symbol P has a type signature P(A, B) where A and B are types. The ABD properties are closed under property composition, inverse, restriction, and union, provided the typing conditions are met. An operator symbol has a signature f(A1,…,An):B where A1,…,An, B are types; operators may have variables are closed under composition, and variable substitution. Types are closed under products. An operator f() with no arguments is identified with an individual. Constructors for forming products, tuples and case properties are used.

Type, property, and operator terms have rules for determining which equations are valid. The operator and property terms are recursively constructed the signature using term constructors introduced below. Formulas within an ABD theory are constructed from equation, subtype, subproperty, and operator type relations. An ABD may have as axioms equations and typing assertions.

Each inference rule is depicted as a fraction; the inputs to the rule are listed in the numerator, and the output in the denominator. The inputs to the rules may be terms or other theorems. Inference rules state that equality between terms and types is reflexive and transitive. Rules of inference allow one to substitute new terms for the free variables in a theorem and allow one to substitute new types for the type variables in a theorem. The inference rules provide mechanisms for defining new constants and new types. The usual presentation of introduction rules within logic uses a line to separate an antecedent condition above a line and the term introduced below the line. In the following A, B, C, and D are types and P, Q, and R are properties, and lower case letters are operators.

2.1 Type constructions

ABD types and individuals have a tuple construction. For any types A1,…,An and any individuals a1,…,an with ai: Ai, the expression (A1,…,An) is a type and (a1,…an) is an individual with (a1,…an):(A1,…An). We write proji for the projection constructors defined for products which project an individual t with t:(A1,…An) onto the ith coordinate. ABD types have a finite union construction with case constructors. ABD types are closed under finite enumeration types which will be written as {a1,…, an} including integers. For a finite sequence of types A1,…An, the expression Union(Ai,0,..n) is a type. We write case for the properties which have typing case(Union(Ai,0,..n), {0,…,n}). For a property P with P(A,B) and C subtype B, then (P some C) is a type with (P some C)(A,C). The rules for existential types and numeric restriction types are accompanied by an individual introduction rule which uses a “dot” notation. The expression a.P is an individual with a.P:C. ABD provides a “dot” notation and semantics for ``fully qualified names”. The “dot” constructor is

used to introduce terms dependent on an individual. When P(A,B) the abbreviation (P exactly 1) is used for (P exactly 1 A). The inference rule for (P exactly 1) introduces for an individual a with a:A an individual a.P1. Similarly the notation a.Pk is used for (P exactly k).

Inference Rules		
Thing	a: A --- a : Thing	
NoThing	a: NoThing --- a: A	
Enumeration	**a1:A, …,an:A** --- **{a1,…an} Subtype A,ai:**	**b:{a1,…an}** --- **b = ai:A, for some i**
A and B	**a:A, a:B** --- **a: (A and B)**	**a: (A and B)** --- **a:A, a:B**
A Subtype B is defined as	**(A and B) = A**	
Union(Ai, 0,…n)	**a: A** --- **a: Union(Ai, 1,…n)**	**a: Union(Ai, 1,…n),** **<a,i>: case, for some i** --- **a: Ai**
case	--- **case:(Union(Ai,1,…n), {0,…,n})**	
tuple	**t:(A1,…,An)** --- **Proji(t):Ai, t = (proj1(t),…,projn(t)**	**a1 :A1,...,an:An** --- **(a1,…,an):(A1,…,An)**
P some C	**P(A,B) C subtype B,** **a:A, c:C, (a,b):P** --- **a:(P some C)**	**P(A,B) C subtype B,** **a:(P some C)** --- **a.P1:B, (a, a.P1):P**
P exactly k C	**P(A,B) C subtype B,** **a:A, c:C, (a, c):P** --- **a:(P exactly k C)**	**P(A,B) C subtype B,** **a:(P exactly k)** --- **a.Pk:B, (a, a.P):P**
A disjoint B is defined as	**A intersection B = NoThing**	

2.2 Property constructions

ABD properties are closed under composition, inverse, restriction, and union, provided the typing conditions are met. Properties are used to represent parts properties as well as properties such as the property that an engine in a vehicle drives the front wheels of the vehicle that the engine is part of.

Property Instances	**t:P(A,B)** **----------** **first(t):A and second(t):B,** **t = (first(t),second(t)):P.**	
Composition	**P(A,B), Q(B,C)** **----------** **QoP(A,C)**	**(a,b):P, (b,c):Q** **----------** **(a,c): QoP**
Inverse	**t:P(A,B)** **----------** **(second(t),first(t)):P***	
Intersection	**P(A,B), Q(C,D)** **----------** **(P and Q)(A and C, B and D)**	**(a,b):P and (a,b):Q** **----------** **(a,b):(P and Q)**
subproperty is defined as	**(P and Q) = P**	
Property Restriction	**P(A,B), A1 sub A, B1 sub B** **----------** **P|A1,B1(A1,B1)**	**t:P, first(t):Ai, second(t):Bi** **----------** **t :P|A1,B1 and conversely**
Property Union	**Pi(A,Bi), i:{0,…n}** **----------** **Union(Pi)(A,Union(Bi), i:{0,…n})**	**t:Pi** **----------** **t: Union(Pi, i:{0,…n})**

Note that any ABD type defined by restriction properties which are unions or restriction properties does not introduce any new types. For example, consider a restriction class (P some C) where P = Union(P1,P2) is a union property. Then

a:Union(P1,P2) some C

and

a: (P1 some C) or a:(P2 some C)

so

a: (P1 some C) union (P2 some C).

2.3 Parts decomposition structure

The informal concept of a parts decomposition structure is made precise using a collection of typed properties called a decomposition structure. In the informal concept a decomposition structure of a product is specified by a product design. The design specifies a root class in a parts decomposition and what parts are necessary to have a product. In this concept specific part instances may be replaced by other instances of the required type. An individual instance of the root type has a parts decomposition which determines the type of specific parts and may determine the number of parts provided the parts properties specify exact cardinality. The existence of an instance of the root depends on the existence of the parts in its decomposition. However, individual parts may be replaced, and so extensionality does not in general hold for a parts decomposition. The individuals in a parts decomposition are irrefliexive and antisymmetric. Product identity is generally defined in terms of a

unique identification number. The concept of a detailed design is that any two individuals of root type, i.e., a1:Root, a2:Root have the same parts decomposition.

A parts decomposition structure for an ABD is defined as a family of (typed) properties $\mathscr{P}$ for which the signature $\mathscr{S}$ of the ABD and the family $\mathscr{P}$ is an irreflexive, antisymmetric, acyclic, connected graph. The types that occur in the typing of the P in $\mathscr{P}$ are assumed to be disjoint. If P(A,B) is in $\mathscr{P}$ there is no P1 in $\mathscr{P}$ with P1(A,A) and no P1(B,A). Since there is exactly one P(A,B) for each pair of types A and B in the signature of the ABD, we can, by abuse of notation, use the same symbol for each P in the family $\mathscr{P}$. If the S and P form a tree with root V, then the ABD is called a design. The concept of parts decomposition structures can be used to characterize the ABD theories which describe designs in the sense that they have a well defined parts decomposition. An ABD may have multiple parts decomposition structures. We use a decimal index notation P(i…j) for these compositions obtained by starting at the root. For a design ABD with a parts decomposition $\mathscr{S}$ and $\mathscr{P}$ we add the axioms that

Root Subtype (P(1) some B) and (P(1.1) some C1) and ….

A number of questions about parts decompositions while not expressible within an ABD theory can be answered regarding a parts decomposition structure. For a design ABD with root V the parts for a design instance v of V are represented as an enumeration class

{v, v.P(1),…,v.P(i….j),…}

where the P(i….j) are the compositions of are properties in the parts decomposition structure with V as the root. The parts decomposition is a tree, the cardinality of the enumeration set is the number of parts in an implementation. Since each part has a typing, the number of distinct types used by the decomposition can be determined as can the number of occurrences of a given type. For an arbitrary property the collection of individuals reachable from a given individual can be determined. A detailed design is a design ABD which does not use any subclass axioms between the basic types and all of the parts properties have an exact numeric restriction. All of the parts decompositions for a detailed design have the same graph structure.

The union construction can be used to define the property which is the union of composition of parts properties within a parts decomposition whose starting point is the root. Two parts decompositions of a root can be made disjoint by adding an axiom that insures that if an individual is in two decompositions then the instances of the root are equal. The restriction construction can be used to start with a property such as Drives within a vehicle ABD and define restrictions such as Drives|Vehicle,Engine. For example, a vehicle ABD may want to represent a drive property for an engine that represents both driving wheels and driving a generator. Both drivesFrontWheel and hasFrontWheel are subproperties of Drive.

2.4 Variables and Operators

Having operators and variables is useful even in the absence of an ABD having behavior is useful. It enables a type to describe that its instances have properties such as weight without having to bind the variables. The model theoretic semantics of an operator term with variables is defined is defined as a function defined on product domains. An operator symbol f has a type signature f(A1,…,An):B where A1,…,An, B are types. An operator symbol a():B is written a:B. Operator terms are constructed using the constructions in the table below.

Operator Expression Constructions	Syntax	
hasOperator	hasOperator(A,B)	Parts property structure used to introduce operators and associate them with an individual
hasVariable	hasVariable(A,B)	Parts property structure used to introduce variables and associate them with an individual
Variable declaration	x1: C1,…xn : Cn	The symbols x1,…xn are variables and xi is said to have type Ci
Operator Declaration	f(x1: C1,…xn : Cn):B	The symbols x1,…xn are variables and xi is said to have type Ci
Composition	f(g1,…,gn)	Where f has arity n
Substitution	t[t1/x1,…,tn/xn]	Replaces distinct variables with operator expressions
Tuple	(t1,…,tn)	Where t1,…tn are operator expressions

In an ABD theory variables (SysML value properties) and operators are all introduced using the same dot construction as is used for parts properties. An operator is always declared as belonging to an individual a. The association of variables to an individual provides a state description for the individual. For example, the state of a vehicle, *v1*, is a list of attributes (variables) of *v1* and of its components. The value of the vehicle state is a substitution or binding of values to variables. Substitution of values for variables provides the foundation for the concept of ``evaluating" the state of an individual. Usual rules for substitution of variables by terms, equality, and typing statement hold. We will write *a.x* for the term a.hasVariable1. For two instances a1 and a2 of A, note that a.x and a2.x are distinct variables. The sequence (x1, x2, f1,f2) of parts of an instance a of A is represented by

(a.x, a.y, a.f)

which is equivalent to

a.(x:X, y:Y, f(x:X):Y)

using the rules for tuples.

4 The correspondence between an ABD and a SysML Block Diagram

The graphical syntax of a SysML block diagram model identifies a collection of basic symbols sorted into types, properties, and operations, called the signature of the block diagram. The graphical representation of a SysML block diagram model uses rectangles for blocks (types). A directed line between P between two blocks, A and B is a property with domain A and range B. SysML uses properties to represent connections between blocks. A SysML structural model can be used to construct an ABD. The ABD starts from the signature of types, properties, and operators in the structural model. SysML employs several "parts" properties that satisfy the properties of a parts structure as defined above.

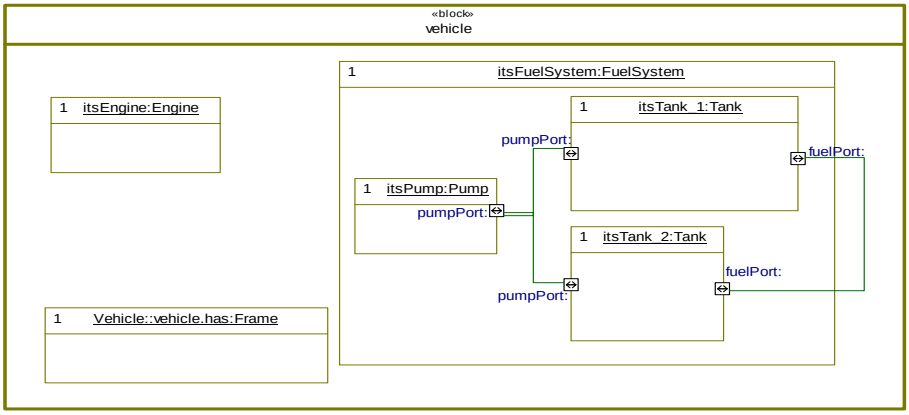

Fig. 1 The Internal Block Diagram illustrates a simplified specification for a Vehicle. The diagram uses blocks: Vehicle, Engine, Frame, FuelSystem, Pump, Tank, and properties itsEngine, itsFrame, itsFuelSystem, itsPump, itsTank1, and itsTank2. This diagram also has flowports with associations between them. The ports are distinct parts of the blocks that they are attached to. The diagram specifies that a vehicle has exactly 1 FuelSystem and a FuelFsystem has exactly two tanks. The tanks are connected through ports with a specific connection. The itsTank1 and itsTank2 are properties that specify that the fuel system has two tanks.

In an internal block diagram, such as Figure 1, the number and kind of parts of a block are described by the parts decomposition structure. itsFuelSystem is one of the part properties in this structure. The range type of itsFuelsystem is the type Fuelsystem, and the number of instances that satisfy itsFuelsystem for a vehicle instance is 1. Multiple occurrences of a rectangle with a given block are used to specify multiple occurrences for a part of an individual instance of the enclosing block. In Figure 1 the block containment relationships are defined using part properties. In the header of a block on the diagram such as itsEngine:Engine, itsEngine is a property with domain Vehicle and range Engine.

The semantics of the vehicle system block diagram model is defined in terms of its parts and their connections. The informal notion of a vehicle implementation is a parts decomposition and a description of how the parts fit together. In Figure 1 a vehicle

instance v has a parts decomposition consisting of an engine, a frame, a fuel system, where the fuel system has three parts.

If a SysML structural model has an arrow P connecting A to B then the ABD has a property P(A,B) with a numeric restriction k. While SysML does not have a restriction type construction the ABD restriction constructions is used to capture the meaning of the property within the context of a SysML model. The meaning of a block diagram arrow P drawn from A to B with a restriction of 1 on B is that any instance of A has only one P connection to B. this can be represented using the axiom

A SubType (P exactly 1)

In this approach block diagrams translate directly into subtype relations. The axiom that A is a subtype of the restriction (P exactly k) captures the meaning that for any a: A then there are k instances of B implied by the property with (a,bi):P, for i:{1,..,k}. The restriction construction can be used to represent user defined relations. The explicit interpretation of ABD type and properties captures the informal, semantics of a SysML structural block diagram model.

5 The Correspondence between an ABD, Description Logics, and axiomatizations of higher order logic in FOL

ABD type constructions correspond directly to DL constructions while not all DL class constructions are used. The property constructions contain standard DL property constructions which are restricted by property typing rules. However, the use of ABD property construction in defining types does not add any new types to the DL. As with Description Logics in general not all of the constructions are logical constructions in FOL. Some constructors are related to logical constructors in first-order logic (FOL) such as intersection or conjunction of concepts, union or disjunction of concepts, negation or complement of concepts, universal restriction and existential restriction. Other constructors have no corresponding construction in FOL including restrictions on roles for example, inverse, transitivity and functionality. However, the full set of constructions can be axiomatized within a multi-sorted FOL where the sorts are type of the ABD theory. The FOL axiomatization of ABD is similar to a FOL axiomatization of Cartesian closed category.

For each type A and each Property P of the ABD the FOL generated by the ABD contains a unary predicate and a binary predicate. By abuse of notation we use the same symbol for both the type and predicate. The context of use will make the usage clear.

A	For any a. A(a)
P	For any a,b. P(a,b)
a:A	There exists x. A(x)
a:(A and B)	(A(a) and B(a))
A = B	For any a. A(a) implies B(a) and for any a. B(a) implies A(a)
a:(P exactly 1)	For any a. there exists a unique b with P(a.b) and B(b)
t:P(A,B)	P(first(t),second(t)) and A(first(t)) and B(second(t))

In the correspondence all judgments derivable from the ABD axioms are provable in the generated FOL theory. Conversely, any theorem provable in the FOL using the logical axioms is a derivable judgment in the ABD. However, the ABD theory provides an explicit term constructions for restriction types rather than just an existence statements. A model for an ABD is defined in the same way as a model for the FOL theory using the axioms for the term constructions. Any non-empty model of a design ABD will contain an implementation of the root in the sense that any model will contain a parts decomposition of the root.

6 Conclusions

The ABD term constructions represent constructions needed for structural modeling of products. These constructions are used within SysML block diagrams. An ABD is an abstraction of SysML structural block diagrams. An ABD can be generated from the signature of the SysML model. ABD has both a type theoretic semantics defined in terms of the inference rules that appear to be in accord with the informal SysML semantics. A model theoretic semantics can be defined as well. The importance of establishing a logical formalism for SysML is that the reasoning required in engineering tasks for design and analysis can be formalized. Automated reasoning techniques can be employed or at least arguments can be automatically checked. The translation of a restricted SysML Block Diagram model into an ABD which preserves the intended semantics is a first step toward integrating product development with formal reasoning. The result provides a formal semantics for a fragment of SysML. Conversely, SysML graphical syntax can be used to develop ABDs.

ABDs can be used to represent parts and connectivity structure. An ABD with a root class under a part decomposition structure has a parts decomposition graph. The parts decomposition structure does not depend on restricting the models as is done in the Description Graph extension to OWL2. An ABD can be used to answer questions such as what parts two designs have in common, and what kinds of transformations and parts replacements to an implementation produce a valid implementation. For product development one would like to characterize ABDs (SysML models) all of whose models have some predetermined similarities. The realization of these goals requires a much richer logical system than has been presented here. However, in order to assess the impact of changing a part in a design on the properties of the design, one needs to be able to define value properties using variables which are recomputed as the value of the variables change. For example, one wants to define the total weight of a product as the sum of the weights of its components and have the total weight change as the parts are changed. An extended version of the type theory presented is a candidate formalism for SysML. Further, an ABD can be viewed as a DL which is sufficient for representing product structure. The next step in the development of the ABD formalism is to add behavior in the form of state charts.

References

1. Bock, C., *UML Composition Model*, Journal of Object Technology, vol. 3, no. 10, November-December 2004, pp 47-73.
2. Barendregt, H., *Handbook of Logic in Computer Science*, volume 2, Oxford University Press, 1992.
3. Friedenthal, S., Moore, A., and Steiner. F., OMG Systems Modeling Language (OMG SysML™) Tutorial, INCOSE Intl. Symp, 2006.
4. Graves, H., Guest, S., Vermette, J., and Bijan, Y., *Air Vehicle Model-Based Design and Simulation Pilot*, Spring 2009 Simulation Interoperability Workshop (SIW)
5. Graves, H. *Design refinement and requirements satisfaction*, Proceedings of 9th NASA-ESA Workshop on Product Data Exchange, 2007.
6. Graves, H., *Ontology engineering for product development*, Proceedings of the Third OWL Experiences and Directions Workshop, 2007. Available at www.webont.org/owled/2007/PapersPDF/submission_3.pdf
7. Graves, H., Horrocks, I., *Application of OWL 1.1 to Systems Engineering*, OWL Experiences and Directions April Workshop, 2008.
8. Graves, H., *Representing Product Designs Using a Description Graph Extension to OWL 2*. OWL Experiences and Directions October Workshop, 2008.
9. Graves, H., Leal, D., *Using OWL 2 For Product Modeling*, Proceedings of 11th NASA-ESA Workshop on Product Data Exchange, 2009.
10. Graves, H. *Integrating SysML and OWL*. OWL Experiences and Directions October Workshop, 2009.
11. Haley, T., Friedenthal, S., *Assessing the application of SysML to systems of systems simulations*, Proceedings of the Spring Simulation Interoperability Workshop, September, 2008.
12. Harrison, J. Formal Verification at **Intel**, presentation June 2002.
13. Howard, W., To H.B. Curry: *Essays on Combinatory Logic, Lambda Calculus and Formalism*. Academic Press, 1980.
14. Martin-Lof, P., constructive mathematics and computer programming. *Logic, Methodology and Philosophy of Science*, 1982.
15. OMG Systems Modeling Language (OMG SysML™), V1.1, November 2008.
16. OMG Formal Ontology Definition Metamodel
17. OWL 2 Web Ontology Language, W3C Working Draft 11 June 2009.
18. Sattler, U., A concept language for an engineering application with part-whole relations. Proceedings of the International Workshop on Description Logics, 1995.
19. Artale, A., Franconi, E., Guarino, N., Pazzi, L., Part-Whole Relations in Object-Centered Systems: An Overview, Data and Knowledge Engineering 20, North-Holland, Elsevier, 1996.
20. Barbier, F., Henderson-Sellers, B., Parc-Lacayrelle, A., Bruel, J., Formalization of the Whole-Part Relationship in the Unified Modeling Language, IEEE Transactions on Software Engineering, Vol. 29. No. 5, May 2003.