A META-MODEL FOR CONSISTENT & AUTOMATIC SIMULATION MODEL SELECTION

Sangeeth saagar Ponnusamy
Airbus Operations SAS & CNRS,LAAS
316, Route de Bayonne
Toulouse-31060
sangeeth-saagar.ponnusamy@airbus.com

Vincent Albert
CNRS, LAAS
7 Avenue de Colonel Roche
Toulouse-31400
valbert@laas.fr

Patrice Thebault
Airbus Operations SAS
316, Route de Bayonne
Toulouse-31060
patrice.thebault@airbus.com

KEYWORDS
Modeling, Simulation, Ontology, SysML, Reasoning.

ABSTRACT

A meta-model of modeling abstractions is presented and discussed in the context of using simulation as a means for system verification and validation activities. Extending the classical results of presenting modeling as a relevance reasoning problem, abstractions are classified to build an ontology based on the concepts of teleological modeling and relevance reasoning. The domain model is built in the standard ontology tool of Protégé to exploit the reasoning and inference capabilities to build a model abstraction library. Lattice structures of model instances are formed and a recursive algorithm is implemented as an activity diagram in SysML for automated and consistent model selection. The approach is presented with a battery model example from the literature. Challenges and future work in implementation of such a semi-formal approach in model selection is briefly presented in the context of improving fidelity of simulation in the industry.

1. INTRODUCTION

In using Modeling and Simulation (M&S) as a means for the system Verification & Validation (V&V), often the difficulty is finding and implementing abstractions of the system being simulated with respect to the simulation requirements. Such difficulties gives raise to the problem of simulation fidelity i.e. the effectiveness of simulation in reproducing the reality for the System Under Test (SUT) validation. In the field of artificial intelligence, this model construction problem is posed as a reasoning problem (Iwasaki and Levy 1994) i.e. inclusion of relevant information about the system being modeled and reasoning about them. However, identification, organization and further classification of these information i.e. abstractions for models simulating a phenomenon are often a tedious task. This is even more so true in modeling complex systems and it becomes imperative that this identification and organization of domain knowledge about the model be (semi)automated in a Model Based System Engineering (MBSE) context.

Ontology, which is a formal representation of a set of concepts within a domain and the relationships between those concepts, could serve as an answer to this problem of building a domain model due to its inherent reasoning and knowledge exploitation capabilities. This domain model, also called meta-model, serves as a common framework for better understanding and making domain assumptions explicit. Since a model can be interpreted as a set of concepts with some relationship between them, ontologies could be used in their representation, organization and exploitation by the practicing engineers with ease in this MBSE framework.

In this paper, a domain model of modeling abstraction is proposed and implemented in Protégé tool, (Protégé). This domain model serves as a standard template for instantiation by system designers and simulation users. The reasoning and querying abilities are used by the model developer over these instances to identify and classify abstraction to build a model abstraction library. The recursive algorithm (Lickly 1992), implemented in a SysML activity diagram is then used to identify a consistent yet simple modeling abstraction to be implemented by the model developers. A simple example of a battery model from the literature is taken as a case study.

2. ONTOLOGIES & REASONING FOR M&S

Reasoning in general is the process of deriving facts that are not explicitly stated. In (Iwasaki and Levy 1994) modeling abstractions were discussed as inclusion of relevant information about the system being modeled and reasoning about them. Based on this relevance reasoning, Levy et al (Levy et al 1997) proposed a recursive procedure to find the consistent yet simplest model from an existing model library. This library is assumed to be well formed i.e. modeling assumptions behind each model is clearly identified and classified. Unfortunately this is not often the case especially when system development and model development are done by different entities who do not necessarily share the same domain knowledge and its associated vocabulary. This problem is further compounded in system V&V activities by simulation when the models of system need to be developed according to some user defined scenarios. Thus the problem of fidelity too can be posed as a reasoning problem i.e. inclusion of the relevant information according to a given scenario and reasoning about them.

In order to ensure this view point consistency, standardization of knowledge exchange and its exploitation, ontologies are proposed in this paper for developing models with required fidelity. Originally intended for semantic web, ontology practices has been increasingly used to improve semantic interoperability and consistent modeling in a MBSE framework (Jenkins 2012) (Man et al 2009). Greves et al (Greves 2009) discussed a reasoning aided MBSE approach by integrating ontology and SysML, a general purpose system engineering language. Ontology, as a means of incremental knowledge addition will help in formalization of these inclusion relations by domain experts over time with their validation experiences. The availability of the standard Web Ontology Language called OWL (OWL) and tools such as Protégé with its query and reasoning capabilities makes it

an attractive option to perform these activities. In Protégé complex concepts can be incrementally built up from simpler concepts using rich set of operators, after which plug-in reasoners such as Fact++, Hermit (Hermit) are used to draw inferences and check consistency. Reasoners infer this relationship by reification, a concept in logic where an instance of a relation is made the subject of another relation. The inferred ontology can be queried for specific needs with SPARQL, a query language which is used to retrieve and manipulate data stored as Resource Description Framework (RDF), a standard for the semantic web. Queries are constructed in triple pattern of subject, predicate and object with conjunctions, disjunctions and optional patterns such as to filter, sort etc.

Broadly the contribution of this paper is twofold, building a model abstraction library based on ontology reasoning and exploitation based on SysML implementation of model selection and automated assembly based on ontology queries. The paper is structured as follows, the classification of abstractions are presented

3. A META-MODEL OF MODEL ABSTRACTIONS

A meta-model for M&S must include different viewpoints of the system being modeled in a teleological perspective. Since a model is an abstraction i.e. simplification of a system and intends to represent a system phenomenon at some operating condition through some quantities and relationship between them, the meta-model need to have different classes of abstraction to represent these aspects. Thus the objective of such meta-model is to translate system knowledge usually expressed in natural language to explicit and model based form for standardized exchange between the stakeholders.

3.1 SBFIO Ontology

In understanding and design of complex systems, teleological modeling in the form of Structure, Behavior and Function (SBF) framework is important. The SBF ontology was proposed as a set of distinct activities in design science as a basis for modeling (Garo 2001). This can be extended with notation of interface (I) and Operating mode (O) to describe interconnected system with different modes of operation. The SBFIO ontology is briefly explained below

System is composed of a Structure (S) i.e. architecture in the form of system-subsystem-equipment-component hierarchy i.e. a System is composed of subsystem which in turn composed of equipment and equipment is composed of component, e.g.: battery is composed of terminals, resistor, capacitor etc.

System exhibits a Behavior (B), e.g.: battery exhibits voltage discharge as function of time.

System performs a Function (F), e.g.: battery provides power.

System communicates via an Interface (I) which are ports of exchange, energy and data, between physical and cyber systems respectively. e.g.: battery connected to electrical circuit via terminal interface.

System operates in an Operating Mode (O), Mode is a partition of state space of system () and for interconnected system, Operating Mode refers to causal dependency between modes of interconnected systems or components. e.g.: when battery is in mode ‘charging’ the electrical circuit is in mode ‘off’, when the battery is in mode ‘discharging’ the electrical circuit is in mode ‘on’.

In general, a model is essentially a representation of any or all of these system perspectives. In addition, behavior or consequence is essentially an outcome of relationship between model quantities which in turn characterizes a function. Their relationship is briefly illustrated in figure 1 with other relationships and concepts of the domain model being hidden for the sake of brevity

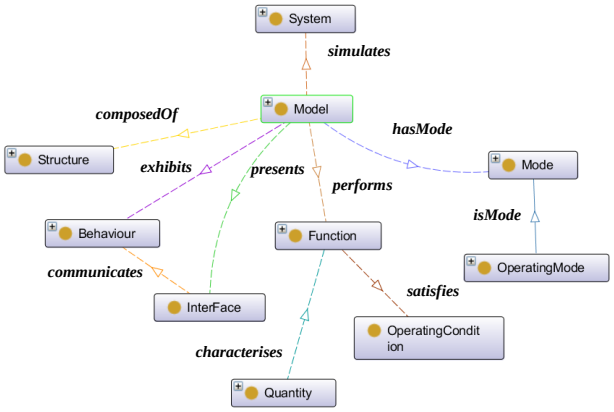

Figure 1: SBFIO Framework

For example, consider *calculate altitude* function performed by the aircraft navigation system at cruise which is characterized by the quantity, altitude (ft) but from two different sources namely, radio altitude and GPS height data for redundancy reasons. This domain model will help in maintaining design consistency such as: all altitude quantities has same syntactics e.g.: unit as ft and semantics e.g.: absolute height not relative height. These ontological relationships help in top down traceability from high level functions to low level behavior with corresponding architectural granularity. These concepts and relationships are being implemented in the context of a system V&V by simulation ontology where the entire processes, from fidelity requirements capture to its implementation vis à vis consistent modeling abstractions are covered.

3.2 Classification of Abstractions

A model is built to represent one or more system viewpoint described in section 3.1 via abstractions. There exist different taxonomies for abstractions employed in M&S by (Frantz 1994). In our approach, modeling abstractions are broadly classified into four classes namely, *architecture*, *data*, *computation* and *time* (Albert 2009). For the sake of brevity, the class definitions are not discussed in detail since the focus is, for a given a class description, how to reason and select corresponding modeling abstractions. A part of this class definition is implemented in Protégé and intuitively one can see that architectural class corresponds to structural viewpoint whereas computation class corresponds to behavioural viewpoint. Only computation, architecture and data dimension will be discussed henceforth.

In general, the abstraction classes are identified $c \in \mathfrak{M}$, where $\mathfrak{M}$ is the domain model shown in figure 1. Consider a model M_i defined by an abstraction operation α_i, where α_i^c is a member of the abstraction class α^c set as described above. This model definition is valid for a certain condition called Operating Condition (OC). For example, an aerodynamic

model with abstraction of only laminar flow is valid for a range of Reynolds number, $Re<Re_{limit}$.
The hierarchy of abstractions is related by binary relation forming a partial order ($\preccurlyeq$) as follows.

$$M_0 \xrightarrow{\alpha_1^1} M_1 \xrightarrow{\alpha_2^1} \dots M_N \quad (1)$$

where M_0 refers to concrete model and *n*=1..N are possible abstractions.

The model abstraction library lists the models and their corresponding abstraction and operating conditions as described in the Table 1 below. The abstractions defined manually by the developer or user are indicated by the '*' sign and those which are inferred then by reasoning capabilities of the ontology to complete this table to the extent possible, are denoted by '+' sign.

Table 1. Model Abstraction Library

Model	Abstraction				Operating Condition
	α_i^c	α_{i+1}^c	α_i^{c+1}	α_{i+1}^{c+1}	
M_0	*			*	OC_1
M_1			+	*	OC_1
….					OC_1
M_N	+	*	*		OC_1

* defined, + inferred

The models described by such a partial relation forms a lattice. Lattice or Hasse diagram is a mathematical diagram of this partial order relation. Such models described over lattices are grouped based on the abstractions. Since a valid abstraction is an operation from a concrete model to an abstract model, where, whatever is true about the concrete model is true in the abstract model but the reverse is not necessarily true, the properties can be inferred from such inheritance relations. From Eq. (1) for models M_n and M_{n+1} and their requirements φ defined over some temporal logic such as Linear Temporal Logic (LTL) or Signal temporal Logic (STL), if

$$M_{n+1} \vDash \{\varphi_{p=1..P}\} \Rightarrow M_n \vDash \{\varphi_{p=1..P}\} \quad (2)$$

Thus for an abstraction belonging to the same class $\alpha_{i=1..n}^c$ arranged over the lattice, implementation of an abstraction α_{i+1}^c also mean the implementation of abstraction α_i^c due to partial order relation $\alpha_i^c \preccurlyeq \alpha_{i+1}^c$. The model abstraction library is thus filled based on these inheritances and dependencies identified by reasoning over the partial order relations. These inclusion relations are exploited to fill the modeling abstraction library and this approach is illustrated with a battery example in next section.

4. APPLICATION CASE

The application case is a battery system similar to the one described in (Levy et al 1997). The battery is connected to a solar panel of a satellite and the function of the battery is to provide power to the panels when the satellite is at the far end of earth without the sunlight.

It is known that a phenomena exhibited by the system can be modeled in different ways. Thus the battery can be modeled in different perspectives (e.g.: model voltage phenomena, charge level or a combination of both) and for each perspective it can be modeled in varying granularity of details (e.g.: voltage is independent or dependent of charge level). Every such model may correspond to different operating condition and the challenge is to find an abstraction consistent with the required operating conditions and phenomena.

The model abstraction library based on table 1 for this application case has models with Voltage (V) as output with different abstractions on ChargeLevel (CL), time (t), Temperature (T). The model ids are given by the following set, $M_{i=1..6}$ = {*Constant Voltage, Binary Voltage, Normal Degrading-1, Normal Degrading-2, Charge Sensitive, Temperature Sensitive*}. The Operating Condition (OC) corresponds to state of *damage* and *rechargeable* conditions. For this case, there are only two conditions namely {*not damaged*} and {*not damaged, rechargeable*} denoted by OC_1 and OC_2 respectively.
Let us denote a class and its instance by a notation *Class: Instance*. Consider a sample model M_5, *Model:ChargeLevel* which describes the evolution of voltage as function of charge level and time under a condition *not damaged*. The *Quantity: ChargeLevel* and *Quantity:Time* is defined to *characterize* the battery function, *Function:Recharge.* An instance could be defined or inferred and the objective is to minimally define these instances and infer the rest. For example, a model with *OperatingCondition: rechargeable* upon inference becomes *not damaged* too. This rule is encoded in ontology through a *subsumes* relation such that if OC_i *subsumes* OC_j then $OC_1 = OC_1 \cup OC_2$. Similarly other domain specific rules could be implemented by domain experts and such template will be useful for other stakeholders to find the dependencies through inference.
In addition, queries can be made on the instances to extract required data or match related data. For example, models could be grouped under an assumption classes based on the output quantity, Voltage (V) in this case. Then, using SPARQL queries, all models having same outputs can be extracted and grouped. Similarly, instances of a class *ParameterDependancy* defining the quantities characterizing the function under an operating condition can be queried to answer teleological questions such as listing functions which depends on same parameters etc.

In the following section, only a few abstractions for each class are explained and this method can be extended for others too, provided a hierarchy can be built with binary relationship between them as described by Eq.1.

4.1 Architectural Abstractions

Architecture relations such as system-subsystem-equipment-component are expressed through *Structure_Composed_of* relationship. For example the battery system is composed of component such as terminals, switches etc. An instance *Model:Binary_voltage_Model* with the relation *Structure_Composed_of* to another instance *Structure_part: Binary_voltage_Model_Terminal* which in turn related to other instance such as port etc. Intuitively, a simulation user requirement of simulating a battery port implies simulation of its parent system.

4.2 Data Abstractions

Similar to previous example, a hierarchy of data types could be created using *data_part* property. A simulation model data type abstraction is deemed valid if the data type is at least less abstract than required by the user. For example, describe data types (DT) as Float $\preccurlyeq$ Int $\preccurlyeq$Boolean, and the simulation user required data type DT_{user} as Int and

that of model developer, DT_{dev} as float. It is inferred that Int is also a float and hence the data type abstraction is deemed valid. These lattice declarations could be extended to other concepts in the context of static model analysis for mitigating model composition errors (Lickly et al 2011).

4.3 Computation Abstractions

Consider a type of computation abstraction such as accuracy which is the difference between exact solution and approximate solution due to modeling abstractions of the behavior. One such abstraction is the Model order which refers to the degree of freedom or in other words the ability of model to capture the rate of change of the dynamics. The model dynamics defined with same input quantities could be related with *Model_Order_part* relationship with the dimension of its space i.e. the complexity of the model. Let the order be defined as, $\mathbb{O}$:M→$\mathcal{N}$, where $\mathcal{N}$ is set of natural numbers. If $(M_2) \preccurlyeq (M_1)$, M_2 is more capable than M_1 and it intuitively implies the former model captures the dynamics of the later as well. Hence the model abstraction at higher order infers the model simulates lower order dynamics too.

Consider, battery example which models the output voltage as function of different parameters based on their Input-Output (IO) relations. The abstraction hierarchy α_i^{IO} corresponds to the number of inputs for the function, f_m where m ∈ $\mathcal{N}$ is the order of function. As described in section 3.2.1, for the models of same IO inputs, the hierarchy can be further decomposed on the model order. In the *Normal Degrading-1 & 2* case, the second order model, M_4 also simulates first order behavior given by the model M_3.

Similar such reifications i.e. information enrichment can be done for members of other classes such as architecture etc. as described in sections 3.2. Upon completion of the model abstraction library, the next task is to select the model consistent with requirements which necessitates the construction of the lattice which will be explored by the recursive algorithm. The lattice structure can be generated by a lattice plug-in or Formal Concept Analysis (FCA) tools such as Lattice Miner where the abstraction library is given as input in the form of objects and attributes. Similar to the lattice described in (Levy et al 1997), the generated lattice for models with Voltage as output is shown in the figure 2. The objects i.e. models are noted in red and attributes are noted in blue and the inclusion hierarchy can be seen. For example, the *Model:Temperature Sensitive* is modeled by temperature, CL and time whereas the *Model: ChargeSensitive* does not model temperature effect. In other words the latter model is an abstraction of the former or lower the lattice element higher is the complexity.

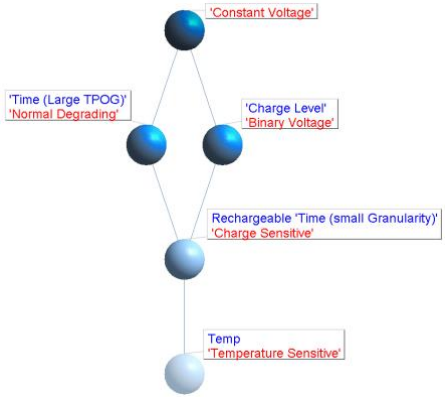

Figure 2: Lattice for Voltage Assumption Class

Similar lattice can be generated for other consequence quantities or any other assumption classes.

5. AUTOMATED MODEL SELECTION

In this section, a SysML implementation of the recursive algorithm to identify a necessary and sufficient simulation model is presented. This implementation consists of block diagrams to define the domain model and activity diagrams for the description of the algorithm. The algorithm is executed over the instantiated domain model i.e. model abstraction library previously constructed in section 3.3. The resulting output is a selection of consistent model with necessary and sufficient abstraction which is built in the form of parametric diagram to be directly simulated. The modeling tool used is MagicDraw SysML (NoMagic) with its Cameo Simulation Toolkit plugin for the execution of built models, in our case execution of activity diagram over the instantiated domain model. Also it could be possible to transform ontology models to SysML using OWL2UML plugin in Protégé 4.1 and then initiating the algorithm. It may be noted that the domain model built is same as described in section 3.2, except for the SBF class of functions and the abstraction taxonomy.

The model selection problem is to find a necessary and sufficient consistent model called *scenario model* i.e. model attribute of *ModelSelection* class, from the given input of *domain theory*, i.e. a set of model abstraction from the library, called *assumption classes*, and a *query*. A query is characterized as follows

- a list of quantities, *quantity* whose value to be predicted by simulating the system,
- a list of exogenous quantities, E_{input} whose elements are assumed to be given and to be outside the scope of the simulation for which scenario model is constructed.

A domain theory is characterized by a set of assumption classes. An assumption class is a set of models which describe the same phenomenon, i.e. having the same *output quantity* in their consequence based on different and often contradictory modeling conditions. *Quantity*, as described in section 3.1, is an atomic expression denoting time dependent attributes associated with the participants in a model instance. On the other hand, *Consequences* are statements that are true whenever the phenomenon represented by the models takes place. Consequences can also be any other logical assertions that are true in a state in which an instance of the model exists.

Activation conditions are statements that indicate when the phenomenon represented by the model takes place by specifying constraints on the participants of the model and on its quantities. The conditions include both structural constraints on the participants as well as constraints on the ranges of quantity values.

Models are related to each other by a refinement/generalization relationship *Rel*. A model can be related to zero or many other models which are simpler i.e. more abstract or complicated i.e. less abstract. It is assumed that every assumption class has a single most complicated model and a single simplest model. In other words the lattice is finite with a minimum and maximum.

These concepts were implemented in SysML and lattice structures are instantiated according to this SysML implementation. The selection algorithm implemented as activity diagram, which is not shown here due to lack of

space, is then used to recursively find the consistent yet simplest model. It may be noted that an activity diagram specifies input to output transformation through controlled sequence of actions and the model selection algorithm is formalized in it and executed. We have used iterative expansion region for list iterations,"readStructuralFeature" and "addStructuralFeature Value" actions for attributes' getters and setters of classes, call behavior actions for modularity and reusability of functions, merge and decision nodes for choices and conditions. The results for the models which correspond to the query Voltage for conditions *not damaged* is given below in the figure 3. Informally, the algorithm starts with simplest model and progressively adds the assumption according to the requirement until all the neceassry assumption classes are added out of which a simplest model is chosen. In this case, the final scenario model is {*battery-damaged, charge-sensitive, accumulation-with-ageing*} and each selection is highlighted in grey at the end of each iteration in the figure 3.

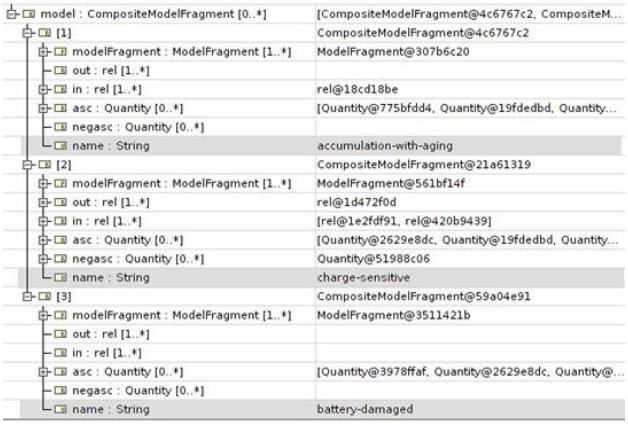

Figure 3 : Model Selection Results

Though the results differ in CL assumption class at the third iteration (Fig 11, Levy et al), the objective is not the algorithm implemented as an activity diagram and its results per se but a description of model library and further model selection in graphical system engineering notion such as SysML for better standardization and understanding of the underlying semantics of the process coupled with ontologies.

6. FUTURE WORK & CONCLUSION

A possible drawback of our approach is the implementation of class definition in Protégé and model selection in SysML. This is done in order to leverage the flexibility, scalability, query and reasoning powers of ontology with the control flow execution, graphical interface capabilities of SysML. However this approach has limitations in terms of effort and at times redundant. This necessitates an integration of SysML and OWL as remarked by (Greves 2009) and (Wagner et al 2012). Such a mutual transformation between SysML and ontology will help practising engineers to capitalise on their graphical syntax and reasoning capabilities respectively and thereby ensuring seamless design and product V&V activities. In addition, studies are being carried out (Ponnusamy et al 2014, 2015) to extend this apparoach in the established M&S framework of Experimental Frames proposed by Zeigler (Zeigler 2000).

This domain model approach based on ontologies & SysML could be integrated in the standard M&S process (Thebault et al 2015). Such ontology aided simulation design process will enable different stakeholders in simulation to define, solicit and manage knowledge usable for M&S in a consistent way. Realization of such an objective will help improve the level of confidence in simulation results for the system V&V and help better utilization of simulation resources by selecting the best available resource according to the test objectives.

REFERENCES

Albert, V. 2009. *Simulation validity assessment in the context of embedded system design*. PhD Thesis. University of Toulouse, CNRS, LAAS, Toulouse, Unpublished.

Frantz, F K. 1994. "A taxonomy of model abstraction techniques", *Proceedings of the 27th conference on winter simulation*, Arlington, Virginia, United States, 1413-1420.

Gero, J.S., Kannengiesser, U. 2004. "The situated function-behaviour-structure framework", *Design Studies*, 25(4), 373-91.

Greves, H. 2009. "Integrating SysML & OWL", *Proceedings of OWL:Experiences and Directions*, 2009.

Iwasaki, Y., Levy, A. 1994. "Automated Model Selection for Simulation". *Proceedings of the Twelfth National Conference on Artificial Intelligence*, 108-116.

Jenkins, S., Rouqette, N. 2012. "Semantically-rigorous systems engineering using SysML and OWL", *International Workshop on System & Concurrent Engineering for Space Applications*, (Lisbon, Portugal, October 17-19, 2012).

Levy, A., Iwasaki, Y., Fikes, R. 1997. "Automated model selection for simulation based on relevance reasoning", *Artificial Intelligence*, (Nov 1997), Vol 96, Issue 2, 351–394.

Lickly, B., Shelton, C., Latronico, E., Lee, E. 2011. "A Practical Ontology Framework for Static Model Analysis", *Proceedings of the Ninth ACM international conference on Embedded software*, NY, USA, 23-32.

Man-Kit-Leung J., Mandl T., Lee E., Latronico E., Shelton C., Tripakis S., Lickly B., 2009. "Scalable semantic annotation using lattice based ontologies". *Lecture Notes in Computer Science*, Vol 5795, pages 393-407.

Ponnusamy, S. S., Albert, V., Thebault, P. 2014. "A simulation fidelity assessment framework", *International Conference on Simulation and Modeling Methodologies, Technologies and Applications 2014*, (Aug 2014, Vienne, Austria), 463-471

Ponnusamy, S. S., Albert, V., Thebault, P. 2015. "Consistent behavioral abstractions of experimental frame", *AIAA Modeling & Simulation Technologies Conference*, Accepted.

Thebault, P., Ponnusamy, S. S., Albert, V. 2015. "A Multimodal Approach to Simulaton Fidelity", *The 12th International Multidisciplinary Modeling & Simulation Multiconference*, Accepted.

Wagner, D.A., Bennett, M.B., Karban, R., Rouquette, N., Jenkins, S., Ingham, M. 2012. "An ontology for State Analysis: Formalizing the mapping to SysML", *IEEE Aerospace Conference*, (Mar 3-10, 2012, Montana, USA), 1-16.

Zeigler B.P., Praehofer H., Tag G.K., 2000. *Theory of modeling and simulation*, San Diego, California, USA: Academic Press.

WEB REFERENCES

http://hermit-reasoner.com/
http://www.nomagic.com/
http://owl.man.ac.uk/factplusplus/
http://protege.stanford.edu/
www.w3.org/TR/rdf-sparql-query/