Towards an Ontology-Driven Framework for Simulation Model Development

Sangeeth Saagar Ponnusamy, Patrice Thebault, Vincent Albert

Towards an Ontology-Driven Framework for Simulation Model Development

Sangeeth saagar Ponnusamy[(a)], Patrice Thebault[(b)], Vincent Albert[(c)]

[(a),(b)]Airbus Operations SAS, 316 Route de Bayonne, Toulouse-31060, France
[(a),(c)]CNRS, LAAS, 7 Avenue Colonel de Roche, Toulouse-31400, France
[(a)]sangeeth-saagar.ponnusamy@airbus.com, [(b)]patrice.thebault@airbus.com, [(c)]valbert@laas.fr

An ontology driven domain model approach for improving the fidelity of the simulation by developing models through the inclusion of simulation objectives for the system Verification & Validation activities (V&V) is presented. The system V&V by simulation ontology used to build this domain model is briefly outlined in the system teleological framework of Structure, Behavior, Function, Interface and Operation. The concept of operating mode is proposed and discussed with an example. The ontology approach is demonstrated with an aircraft nacelle anti-ice system presented in the experimental frame formalism. An example of using the inference and query capabilities of the domain model approach to identify and justify abstractions consistent with the test scenarios is illustrated with a failure mode case study for this application case. The relationship with formal behavioral approach through operating modes is briefly discussed at the end where some theoretical results on the behavioral compatibility of the experimental frame components with interface simulation distances are briefly presented. The paper concludes with a discussion on the benefits of this approach from an industrial perspective along with an overview of the challenges ahead and the future work.

Keywords: *Modeling & Simulation, Ontology, Domain Model, Fidelity, MBSE, Operating Mode*

1. INTRODUCTION

In the development of complex engineering systems, Modeling and Simulation (M&S) is becoming a key capability to perform design and validation studies. However, in developing models to represent the system, often the difficulty is finding and implementing abstractions of the system being simulated, particularly with respect to the context under which it will be used. This not only leads to model validity problems identifiable only at the simulation runtime, but also results in over or under specification and sub optimal development of systems. These challenges in simulation model development necessitate a Model Based Systems Engineering (MBSE) approach which enables a common understanding by making domain assumptions explicit and separate domain knowledge from the operational knowledge. In addition, since modeling can be interpreted as a 'reasoning' problem i.e. inclusion of relevant information about the system being modeled, it is important to identify, relate and organize this information. However, this is often a tedious task which necessitates a domain model approach with reasoning and knowledge exploitation capabilities. Ontologies serve as a good candidate for building such a domain model approach due to their standardization in terms of OWL[1] language, scalability, and availability of tools such as Protégé[2] with SPARQL[3] query capabilities. The flexibility of ontology in expressing different domain knowledge in a succinct and standard form could significantly improve the modeling activities by explicitly incorporating the model context of usage and thereby ensuring better simulation fidelity.

1.1 STATE OF THE ART

The interest of ontologies in the M&S domain has been discussed in [Fishwick,2004] and an ontology based dictionary of generic M&S terms has been given in [Oren,2011]. Similarly, an ontology for system V&V has been proposed in [Kezadri,2010] with various formalisms and techniques for the purpose of knowledge sharing between stakeholders. However, a holistic application of ontology in simulation model development for system validation has not been explored adequately to the best of our knowledge. This study envisages such an integrated approach which consolidates knowledge capture via domain model and exploitation techniques to build a modeling abstraction library and automated assembly of model for near seamless deployment. In addition, as remarked in [Wagner,2012], [Jenkins,2012], ontologies could be used in conjunction with industry standard SysML based MBSE and this will help engineers to capitalize on the graphical syntax of SysML and reasoning capabilities of ontology.

[1] http://owl.man.ac.uk/factplusplus/
[2] http://protege.stanford.edu/
[3] www.w3.org/TR/rdf-sparql-query/

The overall ontology based approach to simulation model development is discussed in section 2 and the system V&V by simulation ontology concepts are elaborated in section 3. The classical system teleological notions of Structure, Behavior, Function, Interface (SBFI) are given in [Goel,2009] [Garo,2004]. However, these notions could be restrictive in expressing the test scenarios in the V&V context and are extended with the concept of Operation into SBFIO ontology and implemented in the Protégé tool. The Operating Modes formalism proposed in this approach is similar to mode automata [Maraninchi,1998] but is more flexible and amenable to ascribe functional or system behaviour at higher levels of abstraction. In section 3.2, extending the concept of abstraction hierarchy defined over lattice in formal verification [Cousot,1992] and semantic annotation [Lickly,2011] to V&V domains, a distance notion is ascribed to the elements of lattice since an absolute lattice inclusion relation could be too restrictive. This relative distance approach improves the application of SPARQL query capabilities of the ontology approach to the simulation model assembly [Novk,2011]. This domain model approach is briefly discussed in a process oriented perspective in section 4. An example of using the inference and query capabilities of the domain model approach to identify and justify abstractions consistent with the test scenarios is illustrated with a failure mode case study in section 5.

In addition, the concept of operating modes could serve to bridge the existing gap between the rigorous behavioral abstraction frameworks such as bisimulation [Girard,2005] and less formal system engineering approaches [Retho,2013]. In this context, a brief discussion on using ontology-aided quantitative behavioral interface refinement [Ĉerny,2010] is given in section 5 followed by a brief description of the future work and conclusion.

2. DESIGNED FIDELITY APPROACH

In the classical simulation model development process, models are usually developed independently of their context of usage and this bottom up approach often results in over or under detailed models which are inadequate to perform V&V activities. Instead of this 'measured fidelity' approach, a 'designed fidelity' approach is proposed where fidelity needs are incorporated apriori in the model development process. This necessitates collection of knowledge about the system to be modeled and scenarios under which it will be operated called System Description (SD) knowledge and Test Description (TD) knowledge respectively. In other words, SD defines the system capabilities whereas TD defines the context of usage and the expected outcomes. The system validation process normally involves interaction between system designers responsible for SD, testers i.e. simulation user responsible for TD and model developers. It is imperative for the model developer to understand and incorporate only the essential elements needed for the test and usually this set of model requirements MR is given by the model specialist. However, owing to the complexity of different domains of knowledge involved which are often implicit and incomplete, it is a tedious task to define this MR manually.

The limitations of the manual approach in its inability to handle the complexity, error prone nature, lack of archival and reuse capabilities necessitates a domain model i.e. a predefined 'template'. Such a template needs to cover the different perspectives of the knowledge description usually expressed in informal natural languages. Since a model can be interpreted as a set of concepts with some relationships between them, ontologies could be used to build such a template i.e. a domain model. This single domain model is instantiated by different actors such as the simulation user and system developer and this ontology serves as a basis for translating text based TD and SD into a standardized model to write MR. An additional advantage of ontologies is its reasoning i.e. ability to infer knowledge which is otherwise hidden or scattered. The existence of plug-in reasoners with Protégé tool such as Fact++, Hermit[4] helps 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 a 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. In the next section this ontology is presented followed by some applications of using of queries over them.

3. SYSTEM V&V BY SIMULATION ONTOLOGY:

In developing a domain model it is important to incorporate different viewpoints in the system teleological perspective such as SBF ontology [Garo,2004]. 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 presented in this paper has different such generic and domain specific concepts with a modest size of about 900 triples and a part of this ontology is illustrated below in figure 1.

[4] http://hermit-reasoner.com/

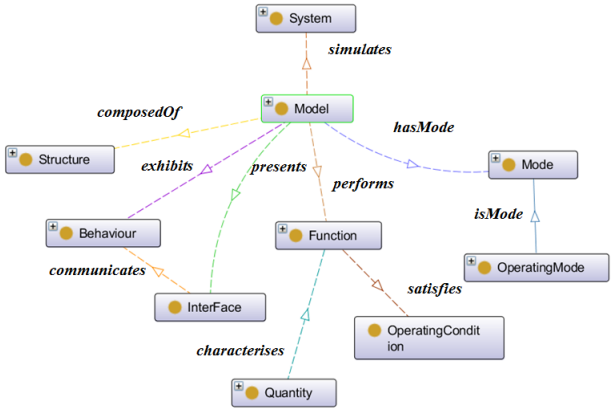


Figure 1: SBFIO Domain Model

The key concepts of the SBFIO ontology covering such a perspective are briefly given as follows:

Structure: In addition to classical architectural descriptions of how the system is built (eg: composition) [Ponnusamy,2015], spatial information is included in our domain model. Besides ensuring geometric consistency aspects, the spatial information could be related to the corresponding physical phenomenon and the interaction between the systems.

Behavior: A system behavior is the temporal evolution of the system when subjected to some scenario and behavioral abstraction will be briefly discussed in section 5.

Function: Function describes the system objectives and how they are achieved. A system's function is essentially an energy flow manipulation and ascribing domain specific laws to such flow type the phenomenon can be modeled. For example: an aircraft actuator's function is to move the control surface according to pilot's command which involves electrical to mechanical energy conversion. Based on such abstract information the associated laws can be inferred from the library developed by the domain experts.

Interface: Interface refers to how the systems interact among themselves (eg: I/O ports) or with the external user (eg:push button). Interface defines the system boundary and can have different attributes such as range, precision etc. It may also be seen as a manifestation of the observable behavior and is essential in ensuring the consistency at interconnection and composition.

Operation: Operation generally refers to the concepts of operating modes and operating condition of the EF.

Operating condition implies the conditions of environment of the SUT and is used to ascribe assumptions behind models especially at higher abstraction level. In other words it refers to the assumptions of the EF components and is used in succinctly expressing and identifying operational domains and dependencies. For example, an operating condition of a flight control system at '*takeoff*' phase implies associated assumptions for the engine performance model at this phase. In the next section, one particular concept of the domain model, namely, operational modes are explained. In [Ponnusamy,2015] a brief description of other concepts in this ontology in the context of building a model abstraction library and automating extraction of relevant abstraction has been discussed.

3.1 Operating Modes

The *Operating Modes (OM)* proposed in this paper extends the classical notion of mode-charts [Jahanian,1994], and is akin to automata. Modes are essentially partition of a system's state space and a system can have different possible modes (eg: Switch-On/off, Engine-Start/Stop). Then the OM definition is based on a simple causality relation for interconnected systems with interdependent modes (eg: Switch-On THEN Engine-Start). This definition is amenable to ascribe functional behaviour or a semantic behaviour vis à vis the system description. In contrary to the rigorous but less flexible formalisms such as mode automata [Maraninchi,1998], our definition refers to the operational manifestation of a model under a given scenario and eases the TD and SD at different levels of abstraction in a static perspective. In other words, the effects of a component's mode on other components can be observed statically and this helps in better understanding the necessary elements to be modelled whose real dynamic behaviour will be analysed later using established formalisms such as mode automata.

Let us denote a system component by C^i having modes $M_j^i \in M^i$ where i and j refers to the component id and the corresponding mode respectively. The dependency between modes are given by mode inter-connection $I_i^k : M_j^i \rightarrow \bigcup_j M_j^k$ i.e. a mode of a component, C^i may affect one or more modes of other component, C^k such that $I_i^k \subseteq \{M^i \cup$

$M^k\}$. The OM then becomes a tuple, $OM^{ik} =< C^i, I_i^k, C^k >$ and the connected modes of C^i are called guards i.e. causative and that of C^k are called states i.e. resultant. Transitions between modes occur whenever the guard mode changes. For example, consider a system with four components, $C^{i=1..4}$ each having different modes. The dependencies in between them are shown as dotted lines below in figure 2, for example, the mode M_1^1 affects M_1^2 which in turn affects M_1^4 i.e. components C^4 is dependent on C^1.

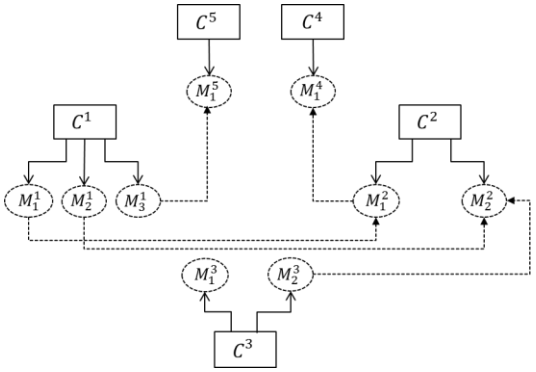


Figure 2: Mode Dependency Example

Such dependencies could be illustrated using OM in the following figure 3, which could be then reasoned and queried to find implicit information such as modes (un)affected by a particular mode or its attributes (e.g: type of system, associated designer etc). In practice the system designer need only gives the component and its dependent modes and the link between different such pairs are extracted automatically. This is useful since the designer usually knows the causality relation only few components upstream and downstream and it is thus important to relate between all such information to have holistic view before modelling the system. In other words, this helps in capturing each component's operational environment assumptions in terms of modes. In the figure below, the causality relation in mode is denoted by solid arrow line and the transition between modes by dotted arrow lines. In addition, transition can be constrained, for example, once mode M_3^1 is activated it cannot be switched to other modes of the component and hence the end state will always be M_1^5. Thus the transitions can be primary i.e. affects other OM or secondary i.e. does not affect other OM e.g.: OM_5.

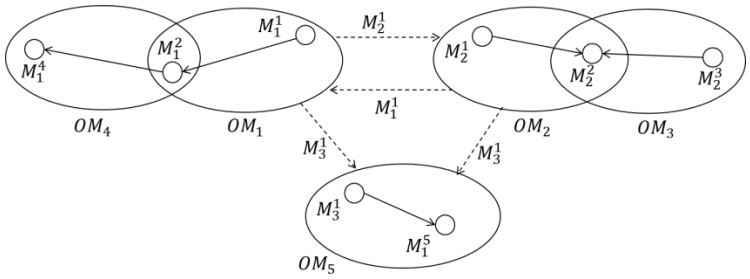


Figure 3: Operational Modes

From such illustration queries can be made on the instantiated domain model for applications such as identification of the transitions between modes and the necessary dependencies to be modelled. For example, reachability notions such as the mode M_1^4 can be reached from M_2^1 by changing the mode to M_1^1 can be queried. Similarly there are two ways of reaching M_2^2 and associated (or the shortest) path can be queried.

This description will also be useful in high level functional failure mode and effect analysis. A failure could be interpreted as the inability of the system mode to transit in response to its associated causality conditional i.e. guards change. Consider a TD stating simulate C^2 failure and this requirement necessitates inclusion of components associated to C^2 such that any mode change in upstream component i.e. guards does not have effect on C^2 since it is already failed and effect of downstream components i.e. states with respect to it. From SDD it is known that C^2 can fail at M_1^2 or M_2^2 and in case of failure at M_1^2 it can be easily infered that M_2^1 will not have any effect and it must be included to check the effect. In addition, OM_4 can be abstracted for simulation of OM_5 since it does not have any transition associated. Similarly recovery procedures such as in case of M_1^2 failure to respond to transition M_2^1, M_2^2 can

be reached through OM_5 if there exists a transition i.e. guard change M_3^1 to M_2^1 can be seen. It may be reminded that all such inferences are static i.e. from instantiated domain model through queries and this helps in inclusion of necessary abstractions to be implemented for a given test requirement before dynamically simulating.

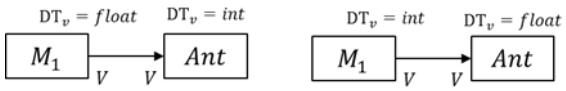
In the next section, notions of hierarchy between concepts of the domain model to build inheritance relations are discussed. These inheritance relations are then exploited to identify necessary abstractions.

3.2 ABSTRACTION HIERARCHY

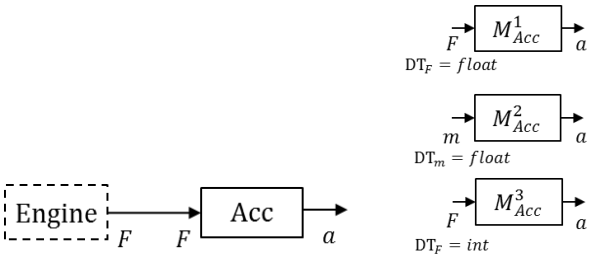
In general, a system can be modeled at different levels of abstraction which could be related to each other by a binary relation ($\preccurlyeq$). This hierarchical notion of simulation preorder represented as a lattice has been widely studied in the field of formal verification [Cousot,1992]. An application of such approach to consistency checking of semantic annotation of models has been explored in [Lickly,2011]. Our study extends such property annotation to V&V domains and ascribes a distance notion to the elements of lattice. In other words, elements of lattice which are closer to the desired element than the others have relatively higher fidelity. For example, the lattice for the variable data type concept with elements *boolean*, *int* and *float* are ordered as *boolean* $\preccurlyeq$ *int* $\preccurlyeq$ *float* where $\preccurlyeq$ refers to 'is also' relation i.e. *int* is also a *boolean* but the reverse is not true. Assuming the TD demands a variable type *float* whereas SD offers only *boolean* or *int*, intuitively it can be seen that the abstract data type *int* is closer to concrete data type *float* than the other abstract data type *boolean*. Similarly inclusion relations could well be extended to other relevant annotations such as in domain specific laws (e.g. a geopotential model with spherical harmonics is also a flat earth model), model versions etc. These inclusion relations are useful to engineers who do not necessarily share the same domain expertise. Such annotations and inclusion relations are useful in mitigating redundant modeling effort, especially in modeling system derivatives where a combination of legacy and new models will be used in tandem for the V&V activities. A similar approach could be used to document assumptions behind models in a hierarchical manner which in turn could be exploited to find the simplest consistent model meeting the simulation requirements [Ponnusamy, 2015].

3.2.1 Automated Model Assembly

In addition to standardization of knowledge and exchange, query capabilities are exploited to select the model with consistent interfaces based on parameter matching using this distance notion. In addition, it is possible to assign weights to each attribute and the model whose interface having the closest consistency is chosen. In a component based design framework, the assembly of components is an important but often ignored aspect and many integration problems arise due to interface compatibility. In assembling i.e. connecting two models, compatible models are selected from a library of models by matching their input and output parameters of their interfaces. In [Novàk, 2011] this task is discussed via queries of ontology but the matching is exact i.e. two models are compatible only if the output of first model is same as the input of second model. This could be true for matching parameters, units etc. but for conditions such as matching data types etc. it could be stringent. Consider an example where a battery model (M_1) modeling voltage is connected via an electrical circuit to an antenna model (Ant). The battery output datatype could be '*int'* whereas the antenna model input datatype is '*float'*. A boolean type checking gives an error despite a *float* is also an *int* datatype. In our ontology, when such an instance occurs, the connection is deemed compatible as shown in figure 4(b), since in the datatype lattice described in section 4.2, *Float* $\preccurlyeq$ *Int*. This is evaluated by simply measuring the length of its relative position in the lattice chain (e.g.: *int* is located lower than *double* hence it has higher length and only elements with lower length are chosen for input type compatibility). Let us consider an example where the engine model is connected to the accelerometer (*Acc*) model to measure the acceleration, a, induced by the thrust, F. The acceleration can be calculated either as function of force or mass or both and from the set of candidate models shown in Fig.4(d) it is evident that second model cannot be used here. From the two available models the first one is chosen for its higher precision if the output datatype is the same (or better). The associated pseudo-queries for this example are given in the appendix. Similar queries can be written to match or extract other system attributes.



(a) Incompatible Assembly (b) Compatible Assembly



(c) System Assembly (d) Model Candidates

Figure 4: Model Assembly

4. PROCESS OVERVIEW

In [Ponnusamy,2015], the utilisation of such a domain model to build a model abstraction library, and automate the model selection from the library using an algorithm in SysML activity diagram is presented. The overall process of the domain model development, building and exploitation of the model abstraction library is briefly discussed in this section with an illustration of the process [Thebault,2015]. In developing a domain model, an important aspect is to describe its integration or improvement of the existing M&S process in an end user operational context. It can be seen from figure 5 that the proposed approach replaces text with domain model concepts, reasoning over implicit information to make them explicit and evaluate their consistency with respect to each other. This is followed by model selection process and the selected model is instantiated in a classical simulation tool such as Modelica etc. The phases of modeling and simulation along with the respective stakeholders can be seen from the figure below. The meta-model and selection algorithm [Ponnusamy,2015] are denoted in dotted ellipse.

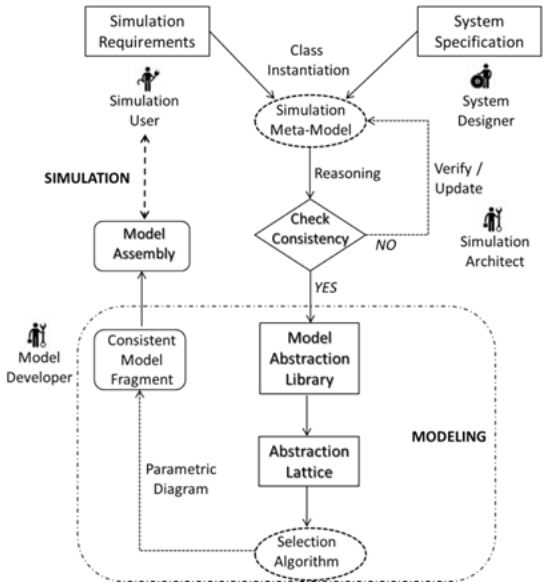


Figure 5: Operational View of M&S domain model

The meta-model instantiation by simulation users and system designers is reasoned by the simulation architect to find implicit data and evaluate its consistency to write the model requirements. In addition, the model developer documents his existing models in a library using the same meta-model, and based on the requirement from architect, a consistent abstraction is selected from this hierarchy of abstractions using the SysML algorithm and then composed with the other models. This assembled model is then deployed on a simulation platform and executed by the simulation user according to the defined V&V plan. An industrial perspective of such approach in the context of simulation fidelity is presented in [Thebault,2015]. This process can be integrated easily in the standard M&S process in industry and it can be seen that this is a non-intrusive method for the engineers since building and exploiting abstraction library is intended to be automated with minimal effort. However, as with any domain model approach in industry, initial effort will be high for tool development, workforce training, process management and

deployment. But as several studies demonstrate MBSE is an important enabler in system development especially due to rapid and complex evolution of corpus of engineering knowledge in an organization and the need to capture systematically this engineering knowledge for standardization and exploitation.

5. APPLICATION CASE: AIRCRAFT NACELLE ANTI-ICE SYSTEM:

A generic description of the aircraft Nacelle Anti-Ice System (NAIS) is presented, followed by instantiation of the domain model built from the ontology defined in section 3. The NAIS is used to prevent ice accretion at the engine nacelle inlet by using hot gases from the engine exhaust. The system is comprised of controllers, valves, solenoids, ducts etc. and is connected to other aircraft systems. In order to perform tests on a component(s) (eg: controller of NAIS), the problem of selecting elements of NAIS and the associated systems (eg: Flight Management System), environment (eg: engine) with respect to this component(s) and the test scenario is outlined in the EF formalism. The following figure illustrates the environment representing the context under which the controller will be tested in the EF formalism. The general system interaction is shown by solid lines and the scenario specific observability of phenomenon (eg: pressure data from sensor) is denoted in dotted lines. Thus the EF helps in a lucid visualization of what is being tested and what is needed for the test in addition to how it is tested (controllability) and what is expected of the test (observability).

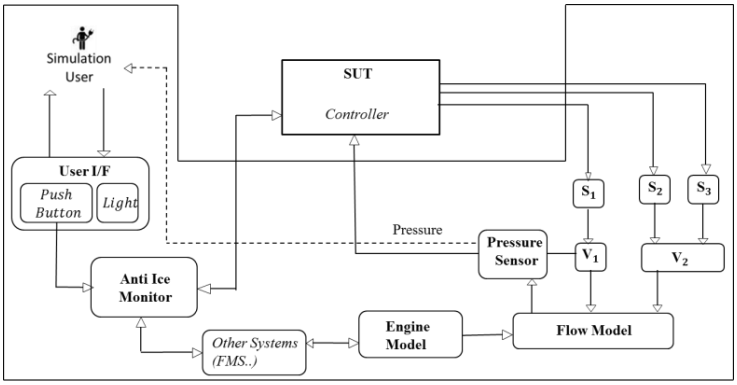


Figure 6: Experimental Frame of NAIS Controller

An application of the OM concept to the failure mode simulation of NAIS valve is illustrated in the following section similar to the example in section 3.1.

5.1 NAIS Failure Simulation

Let us consider a test scenario where TD requires the simulation of valve V_2 failure at closed mode. The test request typically says at which conditions the failure is triggered, where and what are the expected outcomes. On the other hand, SD of NAIS describes all the possible behavior of system, in this case, dependency of valve V_2 modes with the solenoid $S_{2,3}$ modes (e.g. : valve is open when solenoid is energized & closed when solenoid de-energized). It then becomes imperative to identify the components and its associated modes causally affected by this failure condition. Inferring the instantiated OM concepts and querying over this knowledge, desired information such as dependent component or the components that can be abstracted can be obtained with ease. It alleviates the burden of the tedious and often error prone task of keeping track of disparately located but hidden information which is related to each other. Following the notation given in section 3, the SD then becomes

$C^1 = \{V_2\}$ M_1^1 = open M_2^1 = close M_3^1 = regulating
$C^2 = \{S_2\}$ M_1^2 = de-energised, M_2^2 = energised
$C^3 = \{S_3\}$ M_1^3 = de-energised, M_2^3 = energised

The OM is built from the mode data and is illustrated below, for the sake of clarity each OM is shown separately.

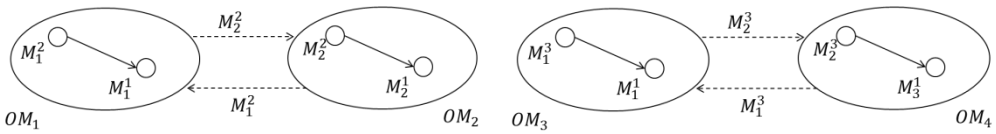


Figure 7: Operating Modes of Valve and Solenoid

Consider a test on the controller to validate its failure monitoring and reconfiguration of valves. It can be seen that, in order to simulate the valve failure when closed, it is imperative to simulate the solenoid S_3 in de-energized mode to see it does not have any effect. However this information is not explicitly given in TD as it describes expectations on the system at higher levels of abstraction whereas SD describes all possible behaviors of the system. Thus it becomes important to identify only the necessary functions and associated systems to be modeled to avoid over or under detailing of models.

In addition such an approach will help visualize and identify possible emergent behavior which may not have been modeled otherwise. For example, from the valve which is failed at the closed position, the regulating mode can be reached in two steps by having S_3 de-energised and S_2 energised. Similar extensions are possible and such information is usually not given explicitly either in SD or in TD, and this formalism helps the model specialist in writing a MR with autonomy. This particular example, though done manually, is found to increase the efficiency during test since provisions for failure triggering are explicitly identified and provided along with necessary functionalities to model the failure propagation.

6. MAPPING TO BEHAVIOURAL FRAMEWORK

The problem of quantitative transition systems and their abstraction has been widely studied by [Alfaro,2004], [Thrane,2009]. However, there exists a gap between the rigorous behavioral abstraction frameworks such as (bi)simulation relations and less formal system engineering approaches [Retho,2013]. The complexity of current engineering systems requires integration of different layers of abstraction and consistency between them. The concept of operating modes could serve as a connection between high level functional description through the domain model approach and low level behavioral description through quantitative transition system. Since OM are high level behavioral descriptions, this would lead to better identification and modeling of transitions to capture the low level behavior, especially during incremental model synthesis. Similarly, the notion of lattice distance leads naturally to behavioral distance quantification based on approximate bisimulation [Girard,2005] and simulation distances [Ĉerny,2010]. In addition, in the context of assume-guarantee contract based design [Benviste A,2012], behavioral refinement of models by extending the concept of interface with interface simulation distances [Ĉerny,2012] is also are being studied. However these studies are still in their infancy and future work includes developing the theory to bridge this behavioral approach with the semi-formal domain model approach to build a unified framework addressing the simulation needs capture from high level to low level model behavioral requirements definition.

7. CONCLUSION & OUTLOOK

This paper gives a preliminary version of domain model and the SBFIO ontology explained in this paper is being improved with other domain specific concepts in the industry and validated with stakeholders before its integration in the engineering process. The study is currently at the identification and experimentation of solutions phase and preliminary indications are encouraging. This study is also being used to assess and provide feedback to ongoing feasibility studies on using the Cappella tool based on the Arcadia framework [ARCADIA] for aircraft system architecture definition and simulation. Future work includes development of a user-friendly graphical interface for domain model instantiation, queries, and formalization of a centralized ontology management process which are all imperative for the utilization across the enterprise.

The ontology driven domain model approach helps to ensure traceability between different abstraction layers and ensures viewpoint consistency and thus enables seamless integration of models and deployment. It helps the test team to optimize the test scenario through inclusion principles and the modularization of ontologies helps in test independence to reduce redundant test combinations. It alleviates the general difficulty of the lack of synchronization and standardization between system development and testing by incrementally and iteratively improving the systems design and testing knowledge along the program schedule. This not only helps in modeling knowledge archiving and reuse for streamlined development of system variants but also for better coordination and decision making in program development.

However, it is often the case that not all abstractions are or can be documented through such a domain model as it is a time consuming and arduous task especially when multiple stakeholders are involved. This is also compounded by the fact that parsing of documents written in natural language into the domain model concepts described in section 3.2 is a complicated task in itself. Though there are some initial studies such as [Ileiva, 2005], this problem needs to be studied with cognitive techniques such as data analytics and deep mining based on iterative learning techniques for better usage of this domain model.

8. REFERENCES

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

ARCAIA, Systems Modeling with the ARCADIA method and the Capella tool, Workshop, EclipseCon 2015, Toulouse, 2015.

Benveniste A et al, "Contracts for Systems Design", Research report No:8147, INRIA, France, 2012.

Cousot, P. 1992. "Abstract Interpretation Frameworks". *Journal of Logic and Computation*, Volume 2, 511-547.

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.

Girard A., Pappas G.J., 2007. "Approximation Metrics for Discrete and Continuous Systems". *IEEE Transactions on Automatic Control*, Volume 52, Issue 5, 782-798.

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

Ilieva, M.G, Ormandjieva, O., 2005. "Automatic Transition of Natural Language Software Requirements Specification into Formal Presentation", *Natural Language Processing and Information Systems*, Lecture Notes in Computer Science Volume 3513, 392-397.

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.

Natalya F. Noy., Deborah L. McGuinness. 2001. "Ontology Development 101: A Guide to Creating Your First Ontology". Stanford Knowledge Systems Laboratory Technical Report KSL-01-05 and Stanford Medical Informatics Technical Report SMI-2001-0880, (Mar 2001).

Novàk, P., Šindelář, R. 2011. "Applications of ontologies for assembling simulation models of industrial systems", *Proceedings of the 2011th Confederated international conference on the move to meaningful internet systems*, Vol 7046, 148-157.

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

Ponnusamy , S. S., Albert, V., Thebault, P. 2015. "A Meta-Model for Consistent & Automatic Model Selection", *30th European Simulation and Modelling Conference*, Oct, 2015, UK, Accepted.

Thebault, P., Ponnusamy, S. S., Albert, V. 2015. "A Multimodal Approach to Simulaton Fidelity", *The 12th International Multidisciplinary Modeling & Simulation Multiconference*, Sep, 2015, Italy, 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*, Montana, USA, 1-16.

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

APPENDIX

```
PREFIX mm:<http://instantiated model name .owl#>
PREFIX nn:<http:// instantiated model name _infered.owl#>

#sample code to compare three simulation models input interface with system model input interface. Two of the models have
#same parameter (e.g.Force, F) but different datatypes (e.g: double, int) – first match the models having same parameters then list
#lattice length
#the query needs to be customized to suit the respective class, object and data properties respectively

SELECT  DISTINCT ?iclist ?source ?dest ?system_var_out_name ?system_var_in_name ?sim_var_in_name ?sim_block
(COUNT(DISTINCT ?sim_var_class) AS ?sim_var_class_no)
WHERE
{

#List all system Interconnection
 ?iclist rdf:type mm:Block_InterConnection;
       mm:connectsFrom ?source;
       mm:connectsTo ?dest.

#check Source Port and Destination Port have same variable names eg:F for force
?source_port a mm:SourcePort;                    owl:sameAs ?system_port_out.?system_port_out mm:isAssociatedTo
?system_param_out. ?system_param_out mm:representedBy ?system_var_out. ?system_var_out mm:hasVariableName
?system_var_name. ?system_var_name mm:hasVariableNameString ?system_var_out_name.
?dest_port a mm:DestinationPort;                    owl:sameAs ?system_port_in.
?system_port_in mm:isAssociatedTo ?system_param_in.?system_param_in mm:representedBy ?system_var_in. ?system_var_in
mm:hasVariableName ?system_var_name1. ?system_var_name1 mm:hasVariableNameString ?system_var_in_name.

#check variable datatypes

FILTER(CONTAINS(?system_var_out_name, ?system_var_in_name))
?sim_block mm:Simulates ?source;           mm:hasBlockParam ?q.
 ?q a mm:InputParameter. ?q mm:representedBy ?b. ?b mm:hasVariableName ?d.  ?b mm:hasVariableDataType ?jj. ?ii
rdfs:subClassOf* mm:VariableDataType. ?jj a ?sim_var_class. ?d mm:hasVariableNameString ?sim_var_in_name.

FILTER(CONTAINS(?sim_var_in_name, ?system_var_in_name) )
?ii rdfs:subClassOf* mm:VariableDataType.
?jj a ?sim_var_class.
}

GROUP BY ?sim_block ?iclist ?source ?dest ?system_var_out_name ?system_var_in_name ?sim_var_in_name
```