An Ontology for State Analysis: Formalizing the Mapping to SysML

David A. Wagner
Jet Propulsion Laboratory
4800 Oak Grove Dr.
Pasadena, CA 91109
818-354-1148
David.A.Wagner@jpl.nasa.gov

Matthew B. Bennett
Jet Propulsion Laboratory
4800 Oak Grove Dr.
Pasadena, CA 91109
818-393-0836
Matthew.B.Bennett@jpl.nasa.gov

Robert Karban
European Southern Observatory
Karl-Schwarzschildstr. 2
85748 Garching, Germany
+498932006542
Robert.Karban@eso.org

Nicolas Rouquette
Jet Propulsion Laboratory
4800 Oak Grove Dr.
Pasadena, CA 91109
818-354-9600
Nicolas.Rouquette@jpl.nasa.gov

Steven Jenkins
Jet Propulsion Laboratory
4800 Oak Grove Dr.
Pasadena, CA 91109
818-354-6055
Steven.Jenkins@jpl.nasa.gov

Michel Ingham
Jet Propulsion Laboratory
4800 Oak Grove Dr.
Pasadena, CA 91109
818-393-6426
Michel.Ingham@jpl.nasa.gov

Abstract— **State Analysis is a methodology developed over the last decade for architecting, designing and documenting complex control systems. Although it was originally conceived for designing robotic spacecraft, recent applications include the design of control systems for large ground-based telescopes. The European Southern Observatory (ESO) began a project to design the European Extremely Large Telescope (E-ELT), which will require coordinated control of over a thousand articulated mirror segments. The designers are using State Analysis as a methodology and the Systems Modeling Language (SysML) as a modeling and documentation language in this task. To effectively apply the State Analysis methodology in this context it became necessary to provide ontological definitions of the concepts and relations in State Analysis and greater flexibility through a mapping of State Analysis into a practical extension of SysML. The ontology provides the formal basis for verifying compliance with State Analysis semantics including architectural constraints. The SysML extension provides the practical basis for applying the State Analysis methodology with SysML tools. This paper will discuss the method used to develop these formalisms (the ontology), the formalisms themselves, the mapping to SysML and approach to using these formalisms to specify a control system and enforce architectural constraints in a SysML model.**

TABLE OF CONTENTS

1. INTRODUCTION 1
2. STATE ANALYSIS 2
3. EARLIER MODELING EFFORTS 3
4. ONTOLOGY DEVELOPMENT 5
5. ONTOLOGY SUMMARY 5
6. MAPPING TO SYSML 8
7. APPLYING THE PROFILE 10
8. CONCLUSIONS AND FUTURE WORK 13
ACKNOWLEDGMENTS 14
REFERENCES 14
BIOGRAPHIES 15

1. INTRODUCTION

State Analysis is a formal methodology that extends basic concepts from control theory and software architecture to aid in the design of complex control applications. Over the last ten years as we have been elaborating and formalizing this methodology we have also been searching for modeling frameworks and tools that might help to transition this from a collection of ad-hoc methods into a more integrated process that more cleanly bridges the gap we see between system analysis and control software specification. The goal of this process is to make it easier for system engineers to precisely express design intent in a tool that actively helps to ensure consistency.

Over this period significant progress has been made in the field of model based system engineering (MBSE), which shares similar goals in a somewhat broader engineering perspective. Much effort has focused on the development and application of the Unified Modeling Language (UML), its derivative the Systems Modeling Language (SysML), and tools that support these languages. While these languages and tools help significantly to formalize the expression, exchange, and graphical representation of system models, they remain ambiguous and in need of extension to capture the specific semantics of a given engineering domain. In the same way that

978-1-4577-0557-1/12/$26.00 ©2012 IEEE

English or another human language requires the creation and definition of new words to enable the precise discussion of new concepts, modeling requires the formal definition of concepts and relations that are unique to a given domain so that models elements can have precise meanings. Then, in order to express these concepts in a formal modeling language like SysML, a mapping is required between the domain concepts and relations and those of the language.

In this paper we will first present a review of the architectural principles and methodology of State Analysis and summarize some earlier efforts in order to set the stage for describing our current work. We will then describe our current ontology, its mapping to SysML, the process used to develop and validate it, and an example system analysis to demonstrate how the concepts are applied to the analysis of a real system. We will conclude by describing some of the limitations discovered during this effort and future work we have identified to resolve them. We begin with a discussion of the relevant architectural principles that guide our modeling approach.

2. State Analysis

Spacecraft design is reaching a threshold of complexity where customary methods of control are no longer affordable or sufficiently reliable. At the heart of this problem are the conventional approaches to systems and software engineering based on subsystem-level functional decomposition, which fail to scale in the tangled web of interactions typically encountered in complex spacecraft designs. A straightforward extrapolation of past methods has neither the conceptual reach nor the analytical depth to address the challenges associated with future space exploration objectives.

Furthermore, there is a fundamental gap between the requirements on software specified by systems engineers and the implementation of these requirements by software engineers. Software engineers must perform the translation of requirements into software code, hoping to capture accurately the systems engineer's understanding of the system behavior, which is not always explicitly specified. This gap opens up the possibility for misinterpretation of the systems engineer's intent, which lead to preventable implementation and operational errors

State Analysis [1] addresses the above challenges by asserting the following basic principles:

- Control subsumes all aspects of system operation. It can be understood and exercised intelligently only through models of the system under control. Therefore, a clear distinction must be made between the control system and the system under control.

- Models of the system under control must be explicitly identified and used in a way that assures consensus among systems engineers.

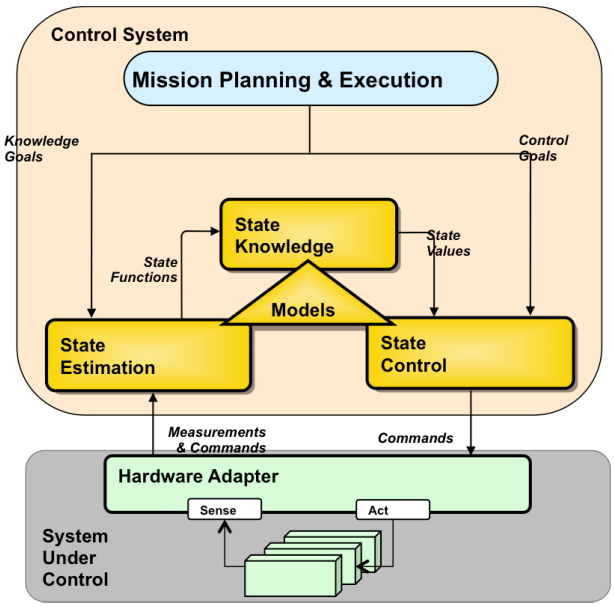

Figure 1 State-Based Control Architecture

- Understanding state is fundamental to successful modeling. Everything we need to know and everything we want to do can be expressed in terms of the states of the system under control because ultimately those are the things we wish to control.

- The manner in which models inform software design and operation should be direct, requiring minimal translation.

State Analysis improves on the current state-of-the-practice by producing requirements on system

and software design in the form of explicit models of system behavior, and by defining a state-based architecture for the control system. It provides a common language for systems and software engineers to communicate, and thus bridges the traditional gap between software requirements and software implementation.

State Analysis provides a uniform, methodical, and rigorous approach for:

- Discovering, characterizing, representing, and documenting the states of a system;

- Modeling the behavior of state variables and relationships among them, including information about hardware interfaces and operation;

- Capturing the mission objectives in detailed scenarios motivated by operator intent;

- Keeping track of system constraints and operating rules; and

- Describing the methods by which objectives will be achieved.

For each of these design aspects, there is a simple but strict structure within which it is defined: the state-based control architecture (also known as the "Control Diamond", see Figure 1).

The architecture has the following key features:

- State is explicit: The full knowledge of the state of the system under control is represented in a collection of state variables.

- State estimation is separate from state control: Estimation and control are coupled only through state variables. Keeping these two functions separate promotes objective assessment of system state, ensures consistent use of state across the system, simplifies the design, promotes modularity, and facilitates implementation in software.

- Hardware adapters provide the sole interface between the hardware in the system under control and the control system: They form the boundary of our state architecture, provide all the measurement and command abstractions used for control and estimation, and are responsible for translating and managing raw hardware input and output.

- Models are ubiquitous throughout the architecture: Models are used for both execution (estimating and controlling state) and higher-level planning (e.g., resource management). State Analysis requires that the models be documented explicitly, in whatever form is most convenient for the given application.

- The architecture emphasizes goal-directed closed-loop operation: Instead of specifying desired behavior in terms of low-level open-loop commands, State Analysis uses goals, which are constraints on state variables over a time interval.

- The architecture provides a straightforward mapping into software: The control diamond elements can be mapped directly into implementation artifacts in a modular software architecture.

In summary, the State Analysis methodology is based on a control architecture that is inherently model-based and has the notion of state at its core. In the following section, we describe our early efforts at capturing the products of State Analysis.

3. Earlier Modeling Efforts

From its earliest stages State Analysis has developed graphical notations to concisely express its concepts and relations. As in many other fields we quickly discovered the limitations of trying to do engineering with simple drawing tools such as PowerPoint and Visio. Although they could create pretty pictures, these tools had no representation of what the boxes and lines meant, and so could not enforce any kind of consistency with the rules of State Analysis or even the implied semantic conventions (e.g., color coding) in an individual diagram. Thus began our search for tools to enable more formal modeling.

The first prototype for a State Analysis database was built in 2003 with a fairly simple entity-relation schema. It supported multi-user web-based sharing through a browser form interface built using a simple web interface over a file-

based data manager. This development was seminal in that it could be used to specify a control system architecture in terms of State Analysis concepts. However, the schema was manually encoded in the source code of the tool, making it difficult to analyze and update.

The second prototype was built using Common Lisp and the LOOPS object oriented database framework. Again, a web-based form user interface was provided to allow users to capture State Analysis artifacts.

A formal description of the ontology was initiated to guide the design of the database in this second prototype. The ontology was described in terms of an object-oriented design, with object types for each of the State Analysis elements, each having properties, and typed links to describe relationships between the State Analysis element 'objects'. This tool performed better than the earlier one, but it still suffered in terms of usability and maintainability because the ontology was manually encoded in the tool implementation.

The last prototype, the State Analysis Database [2], was implemented using a commercial backend relational database that was formally specified using XML and Entity-Relationship diagrams. The structure of the State Analysis Database schema was developed to enable enforcement of the correct relationships between different model elements, between different software specification elements, and between models and the corresponding software specifications. Thus, the schema prevented a class of architectural engineering errors, and provided a guide for doing a complete and consistent engineering analysis. The front end of the database was a standalone web-client application. The client application could create html reports desired by users. For example, a prototype command dictionary report generator was implemented. In addition to the form-based user interface, a graphical interface was developed in the client to draw some of the diagrams specified by State Analysis

The State Analysis Database had a client-server distributed system design. A single central database was the repository for requirements and models. The database was hosted using a commercial database manager. Communication with clients and other tools was through SQL queries and other industry standard mechanisms so that the database could be easily re-hosted on a variety of systems. The database could be accessed through the internet using the HTTP protocol – a database web server serviced HTTP database requests from clients and tools.

We designed the database schema to reflect the formal State Analysis process. Each of the kinds of architectural elements that could be modeled or specified in State Analysis had a corresponding table in the database. Relationships between architectural elements were captured as references to related elements in tables, or through the use of linking tables between tables. The design was guided by the nature of each kind of relationship and the desire for the schema to enforce architectural rules where possible and reasonable, rather than having to rely completely on external consistency checking scripts.

Although we learned much about modeling, and refined our concepts at each step, none of these earlier tools gained much traction with the system engineering community for doing real system analysis because they just weren't very easy to use, and they required significant effort to implement and maintain. The desire for a more user-friendly graphical modeling interface based on customizable tooling led us to the approach described in this paper. The current approach is to leverage existing graphical interfaces provided by commercial system modeling tools, which are founded on the industry-standard modeling languages UML and SysML. SysML is created as a UML profile[1], i.e. it extends the UML meta-model using the stereotype mechanism.

Murray [3] first prototyped a UML customization for State Analysis using the MagicDraw modeling tool. The profile, consisting of UML stereotypes, was manually created in MagicDraw[13],

[1] A *UML Profile* is a collection of definitions for *stereotypes*, *tags* and *constraints* that customize UML for a domain, redefining the semantics of the modeling language by extension.

demonstrating for the first time the viability of this general approach.

Karban [4] is leading a team using State Analysis in the design of the control system for the European Extremely Large Telescope (E-ELT), a highly complex hardware/software system. Because of their need to specify hardware, software, and system-level interactions, allocations, and other non-software properties, SysML is a more appropriate modeling language choice than UML. They have developed a handmade profile for MagicDraw that defines many of the basic concepts and relations from State Analysis.

In both Murray's and Karban's efforts, customized stereotypes were used to apply domain-specific meanings to model entities (Classes or Blocks) in order to help reduce ambiguity. In the current version of UML/SysML, stereotypes are the only way to specify a domain-specific meta-model that can be readily mapped to model entities and their relationships.

Although these approaches have significantly improved the ability of graphical models to express design intent, and have made it much easier to integrate control analysis into system models, they have only limited ability to verify model consistency with the underlying semantic meaning of the applied stereotypes and relations. To achieve that it became clear that we needed to more precisely formalize the concepts of State Analysis, and then map those concepts to SysML.

4. Ontology Development

Our aim in this effort has been to provide a modeling framework for future applications of State Analysis. that enables us to leverage recent advances in graphical modeling tools, while also enabling us to perform formal analyses of consistency and correctness with respect to State Analysis domain relationships and constraints. Ontology is the study or analysis of the fundamental concepts and relationships in a domain. Here, the domain is that of software-intensive control systems. The State Analysis domain is defined as an ontology in OWL2 [6] using the open-source Protégé ontology tool [7]. The mapping of the State Analysis ontology into a profile extension of SysML is defined as a mapping ontology that relates concepts from the domain ontology onto SyML elements that will be used to represent them in SysML models. This integration of OWL2 and SysML facilitates using OWL2 as a language and formalism for representing heterogenous conceptual domains, for reasoning about properties of the domains, and for simplifying the model-to-model transformation workflow required for projecting information from one domain into another [10,11]. Not only does this process allow us to cleanly separate the ontology from the semantics of SysML, it also allows us more flexibility to change the mapping of domain concepts (as stereotypes) into SysML entities or relations without affecting the concepts themselves. This is important as the SysML standard continues to evolve. The mapping flexibility allows us to maximize the usability of a State Analysis extension of SysML given the evolving constraints and limitations of the SysML abstract syntax language and of the SysML concrete syntax diagrams.

The State Analysis ontology described here is constructed on top of a set of related ontologies describing the organizational and engineering context in which State Analysis is performed. Ultimately, the product of State Analysis is a specification for a deliverable control system product. It is beyond the scope of this paper to describe them here, but these base ontologies describe and relate the concepts of *project*, *mission* (that is, the mission the system is designed to perform), and deliverable *components*. Building upon these common base concepts makes it easier in the long run to relate discipline-specific models built using the State Analysis ontology with models from other domains such as software, electrical, or mechanical design that share the base concepts.

5. Ontology Summary

The State Analysis ontology is divided into two layers, one for the fundamental concepts of physical system modeling, and a second layer defining the fundamental concepts of control

system modeling. This layering enables modeling of the physical system (the "plant" in control system terminology, or "system under control" in State Analysis) to be independent of any influences of the control system design, enforcing the methodological emphasis on completing the analysis of the physical system before beginning control system design. As described in the previous section, these mapping to SysML is then defined in another separate layer in order to help decouple the semantics of the State Analysis ontologies from those of SysML, and allow the mappings to evolve with the SysML standard. In this section we will discuss key concepts in both layers. Note that the diagrams presented in this section were generated from the OWL2 model and a documentation generation transformation.

Physical System Modeling Concepts

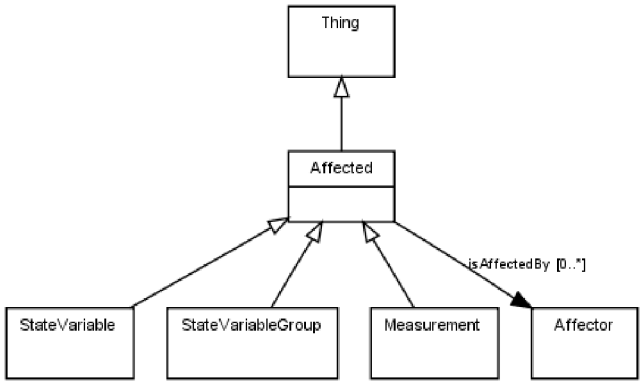

Figure 2 *Affected* Concept

A fundamental concept in State Analysis is the modeling of state effects, or physical (causal) relationships between state variables of the system under control, the commands, and the measurements of the system. To model this relationship we used the abstract entities *Affector* and *Affected*. A *StateVariable* of the system under control can be both the source and target of an effect, and so derives from both. A *Measurement* can only be affected, and a *Command* can only cause an effect as shown in Figure 2 and Figure 3. The base concept *Thing* is simply an anonymous abstract container used as an OWL convention to keep our model distinct from other ontology models in the same system of models. Thus, there is one base *Thing* from which all of the State Analysis concepts derive, for the sole purpose of identifying that they are all part of the same ontology.

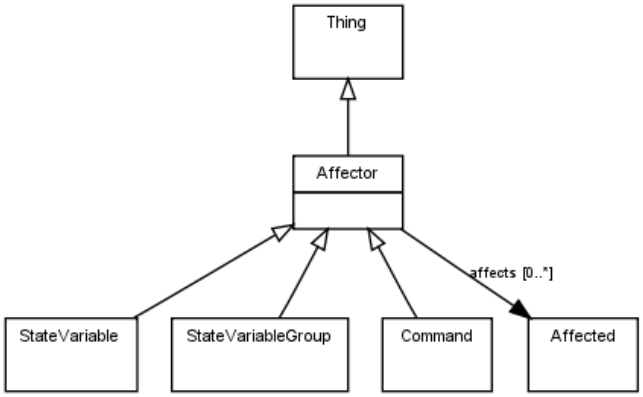

Figure 3 *Affector* Concept

The state effect relation is reflected in the named filled-in arrow (representing an OWL2 property) in Figures 2 and 3. The square brackets indicate the multiplicity of the relationship. In this case it says that an *Affector* can affect zero or more *Affected* elements.

The *StateVariableGroup* entity is a modeling concept intended merely as a container for an arbitrary group of state variables, measurements, or commands in a diagram so that the model need not express every single state effect in a single diagram. This allows diagrams to express effects between groups that can then be detailed in separate diagrams for each group. In large complex systems this enables a more understandable and incremental presentation of the model details.

Control System Modeling Concepts

In State Analysis, the control system is specified in terms of estimators and controllers that are associated with the state variables of the system. *Estimator*s and *Controller*s are the main "active" elements that the ontology represents in the abstract concept *ControlSystemComponent* (see Figure 5). A *ControlSystemComponent* can be hardware or software that actively performs a function (i.e., in software this would be a function that is scheduled to execute). An *Achiever* can actively perform the function of achieving a goal. A *Controller* achieves control goals (goals that constrain state variables of the physical system), whereas an *Estimator* achieves knowledge goals

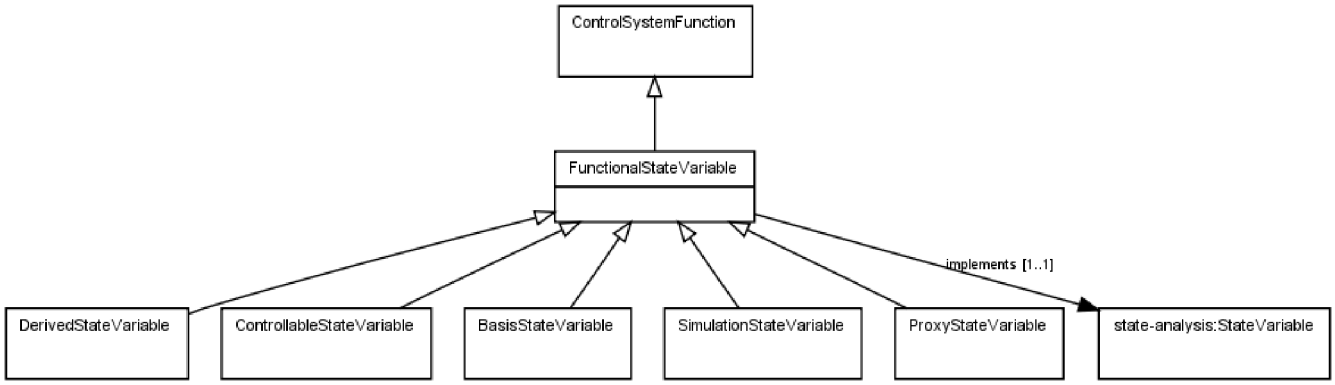

Figure 4 Implementation State Variable Types

that specify constraints on the quality of state knowledge (level of acceptable uncertainty in the estimated values of state variables) needed to achieve given control goals.

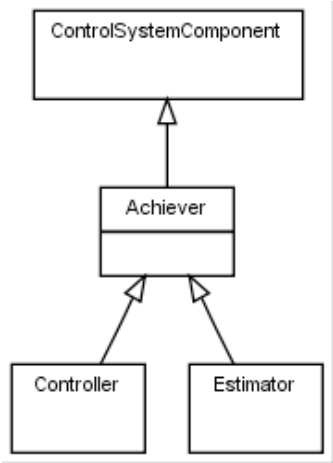

Figure 5 Achiever

A *FunctionalStateVariable* is a control system artifact responsible for representing the value of a state variable from the system under control. As depicted in Figure 4478, there are several kinds reflecting distinct relationships with other elements in the control system[2]. Two we will describe here are *ControllableStateVariable* and *BasisStateVariable*. As laid out in Figure 6, a *BasisStateVariable* has a relation with one *Estimator* (i.e., it provides the knowledge basis for control), and a *ControllableStateVariable* (Figure 7) has a relation with one *Controller.*

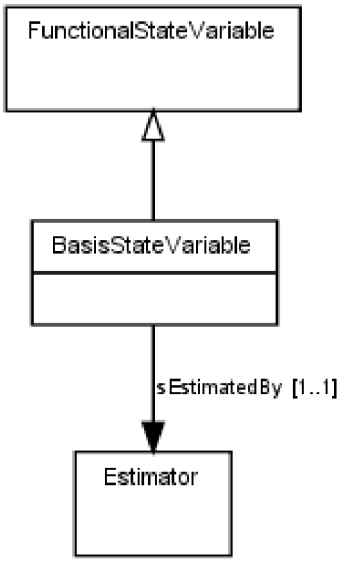

Figure 6 Basis State Variable

Note that a functional state variable can perform both of these roles. That is, a given implementation state variable can be both controllable (representing a controllable state variable of the system under control, and having an associated controller to effect control), and basis (having an associated estimator to provide updates).

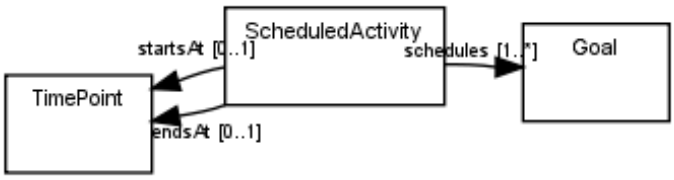

Figure 7 Goal Associations

In the process of developing this ontology and its associated mapping to SysML, a few refinements to basic concepts were added as a result of the

[2] Functional state variables are also described as software state variables in previous work. The new name recognizes that they do not have to be implemented in code; for instance they could alternatively be implemented as switches in hardware or registers on an FPGA board.

need to remove ambiguities or overloaded meanings of certain concepts. One in particular had to do with the concept of a *Goal*. As shown in Figure 4, the concept of a *Goal* was divided into separate concepts for scheduled and unscheduled goals. *ScheduledActivities*[3] are those that have been coordinated on a timeline with all of the other goals and sub-goals needed to accomplish a plan, whereas *Goal*s more abstractly specify intent through elaboration dependencies on other goals [1].

6. MAPPING TO SYSML

In general, our strategy for mapping the ontologies into SysML is to define the ontological concepts and relationships as SysML stereotypes that can be applied to appropriate modeling entities: concepts to blocks, and relationships to semantically compatible SysML relationships. In a SysML tool such as MagicDraw, this can be accomplished by creating a profile module to define the stereotypes and any associated diagram and tooling customizations. In other attempts [3,4] the translation from ontology to profile had to be performed by hand, which is not only tedious, but also prone to error because they were not based on a semantically verifiable ontology.

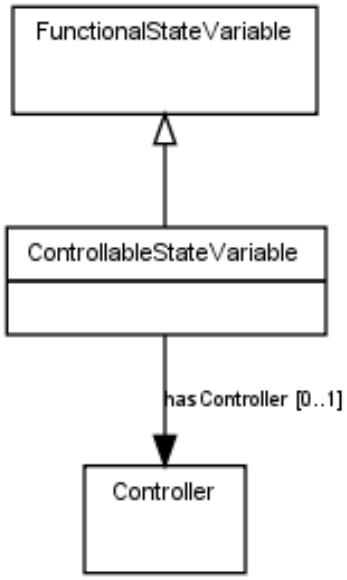

Figure 8 Controllable State Variable

Thanks to some advanced model transformations developed by JPL's Integrated Model-Centric Engineering [10] team, the SysML profiles can now be generated automatically from the OWL2 ontologies, and validated in the process against a set of consistency assertions derived from the semantic relationships defined in the ontologies. Furthermore, similar transformations, in the inverse direction, can then be applied to the SysML models to verify that any models that use these stereotypes conform to the semantics expressed in the ontologies, and thus conform to the rules of the architecture that they express.

One of the key challenges in this entire effort was finding appropriate mappings of the domain concepts defined in our ontology concepts onto SysML modeling entities (Block, Port, etc.). Although the basic entities in SysML are defined with very abstract semantics, so that they can be specialized to model a wide variety of systems (by specializing the SysML stereotypes or specializing UML meta-classes), one must be aware of these meanings and relationships in order to avoid using them with domain semantics that conflict with the SysML abstract semantics. This is not something SysML itself would or could enforce, but is something that could cause problems for semantic reasoners attempting to apply formal methods to verify model correctness. Keeping the mappings to SysML distinct from the domain ontology makes it easier to refine the mappings or to define alternate mappings without affecting the domain semantics.

Context diagram

A concept that came to State Analysis through SysML is that of a modeling context. State Analysis establishes the need to formally distinguish between the control system being designed and the plant, or system under control, that it will interact with. SysML convention further requires that the combination of control system and plant be defined in a containing context in order to establish the extent of what is being modeled using an internal block diagram. We model the context in SysML as a block stereotyped as a modeling context (see Figure 111111), which will then contain the models for control system and system under control. The diagram depicting this relationship is called the

[3] Also called executable goals, or XGoals in earlier papers.

context diagram. The line between control system and plant is depicted in the context diagram through the containment of model elements into either control system or plant parts of the model. The plant model can then be further decomposed, for example into a physical system and an environment, or into multiple elements of a distributed physical system.

State Effects Diagram as ibd

The SA concept of a state effects diagram (SED) is implemented in SysML using an internal block diagram (ibd) associated with the modeling context block (see example in Figure 12). Using an ibd associated with the modeling context allows the model to express relations between specific state variables contained within the context. Different modeling context blocks can be created if it is necessary to consider different effect models for the same physical model (e.g., models of different fidelity might be used at different times in a development process), or different configurations of the system as is likely to be the case during concept studies. A block definition diagram (bdd) would only express relations between state variable *types*, and although one can define very context-specific types, it seems more appropriate to express relations between specific instances defined within a context.

State Effects Models as Parametrics

The use of an ibd to represent state effects is basically just an abstraction of SysML's concept of parametrics. The state effect relation is modeled as a stereotyped dependency arrow that indicates a causal effect. The behaviors behind these relations can then be specified in SysML parametric (par) diagrams (or possibly state charts or activity diagrams, depending on the nature of the behaviors) to document the details of the relations in the form of mathematical formulae or algorithms.

Goal Elaborations

In State Analysis, goals express user intent as explicit constraints on state variables of the system under control over time. Goals are usually defined abstractly so that the time domain, and often the specific constraint, is expressed parametrically as a goal type rather than as a specific goal instance. This concept of a goal type can readily be expressed in SysML as a stereotyped block. We have also tried modeling goal types as use cases, with distinct properties and diagrams. The use case "includes" relation, where one use case depends on another is semantically quite similar to the SA "elaborates" relation (see Figure 9). Thus, applying the SA stereotypes to these SysML elements makes it possible to express certain aspects of goal elaboration directly in use case diagrams (those stereotypes extend the SysML UseCase Meta-Class).

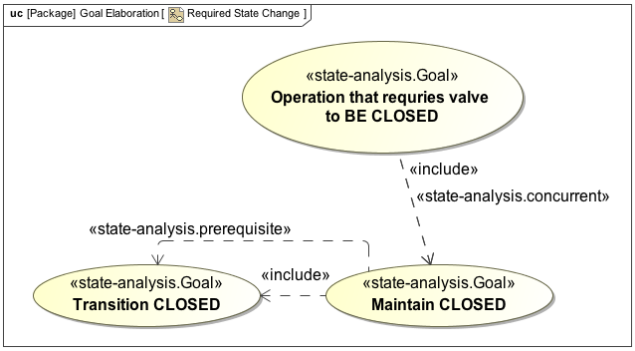

Figure 9 Simple Goal Elaboration

The main drawback of this approach is that use case diagrams have no notion of time. Goal elaboration diagrams must indicate the temporal dependencies between elaborated sub-goals and their parent goals. Our solution to this was to add stereotyped dependency relations between goals to express a few simple temporal relations including "concurrent," and "prerequisite." In the State Analysis theory such relations are expressed in the form of a mini temporal constraint network (TCN). SysML use case diagrams have no such notion of time. This problem could be addressed by using parametric diagrams and SysML Constraint Blocks to express temporal constraints in such networks.

Use case diagrams also permit the use of generalization to model goals that have alternate tactics as shown in Figure 10. This clearly expresses the notion that there are two ways the

parent goal can be achieved, but does not provide a simple way to formally express the selection criteria, or specification of which tactic to choose in a given set of conditions.

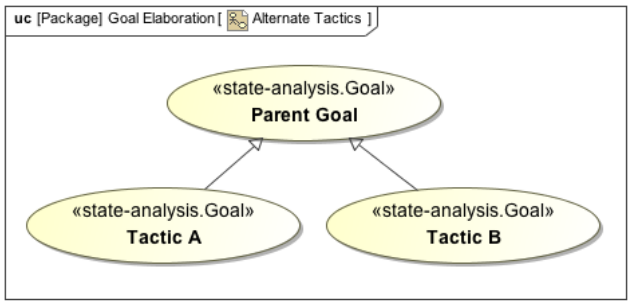

Figure 10 Representing Alternate Tactics

Architectural Diagrams

Goal-oriented control systems are constructed using *Estimator* and *Controller* components that associate with state variables to achieve knowledge and control goals. The functional architecture of the system can be modeled in SysML using block definition diagrams (bdd) to define *Estimator*, *Controller*, and *FunctionalStateVariable* types as parts of the Control System context block, and then using ibds to model the internal connections between them (see example in Figure 16). The stereotypes for these entities defined in our ontology constrain the kinds of information flow relations they can have with each other. For example, an Estimator can have an update relation with a State Variable, but a controller cannot. A Controller can issue commands to a hardware adapter, but an Estimator cannot.

SA defines the concept of Hardware Adapter as an interface between the control system and system under control. In SysML we can model hardware adapters as interface blocks and nested ports. Ports are all that is necessary to document simple interfaces; aggregating several of them into an interface block allows the model to express a little more design intent, or detail about how the system constrains the interfaces.

7. Applying the Profile

For the purpose of exposition we define a simple system we can model using the concepts described in the previous section. Our system consists of a controllable valve in a pipeline. Figure 111111 depicts the context diagram for this system. We have defined a Physical System State Analysis block and a separate Control Software block, both contained in an Analysis Context block (those stereotypes specialize the SysML <<Block>> stereotype).

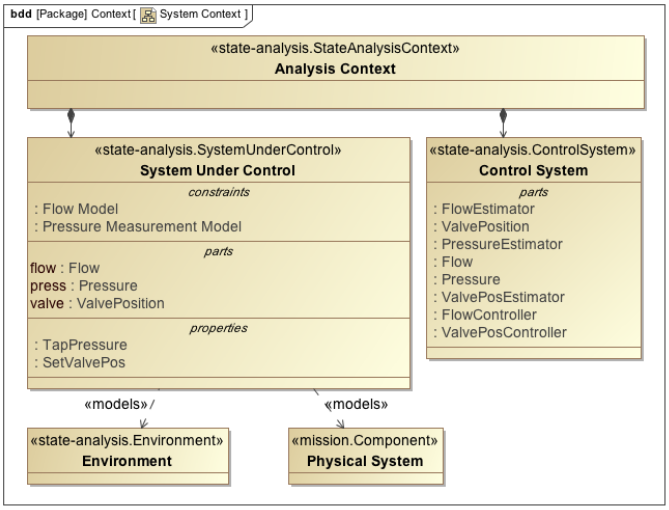

Figure 11 System Context Diagram

The Analysis Context block provides a context in which we can model relations between the control system and the system under control it interacts with. The diagram indicates that the system under control is a characterization of the actual physical system and its environment, using a stereotyped dependency relationship (alternatively this could be expressed with a realization or specialization relationship). Similarly, the control system context characterizes the actual software implementation. This separation between an analysis context and the thing being analyzed can be important in a larger modeling environment in which there may be multiple characterizations of the same physical thing. Note that the order of creation may be different in these two cases. Whereas the physical system may exist prior to its characterization, the software implementation may be created after conceiving the State Analysis functional architecture.

The System Under Control block is shown containing a number of State Variable part properties. The types of these State Variable part

properties are defined in separate diagrams (not shown), and stereotyped as State Variables.

The next step in the State Analysis process is to define abstract state effects. Figure 12 shows how this is done using an ibd. State effects are indicated using dependency arrows stereotyped with the Affects stereotype. Also shown are the *TapPressure* measurement, and *SetValvePos* command that have been defined to enable communication between the physical and control systems. State effect relationships identified in this diagram must subsequently be elaborated into detailed state effect models, or measurement models using parametric diagrams or other behavior representations.

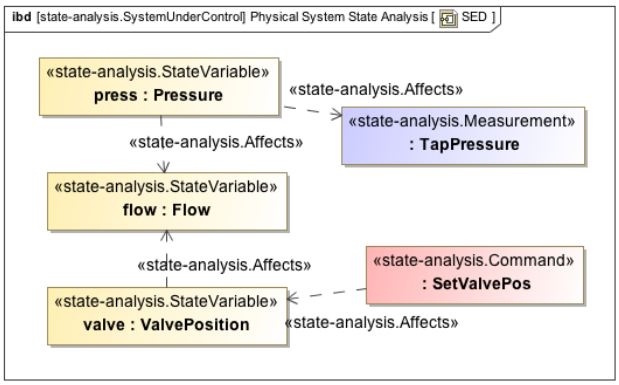

Figure 12 State Effects Diagram

Parametric diagrams provide a good way to document state effect and measurement models as shown in Figure 13 and Figure 14. The constraint function inside of the constraint block can be in the form of a mathematical relationship, or pseudo-code logic (SysML does not currently specify the details of constraints).

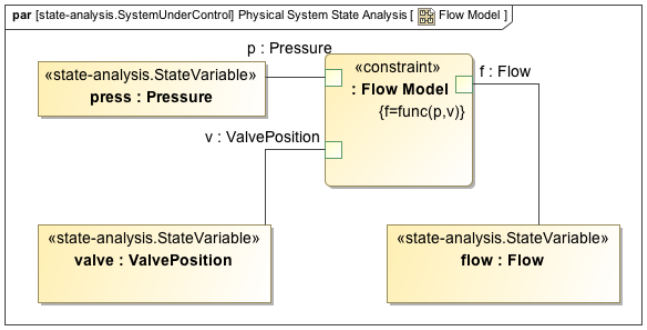

Figure 13 State Effects Model as Parametric

Figure 14 elaborates the measurement model for the TapPressure measurement. This diagram preserves the abstract *Affects* relationship from the State Effects Diagram in order to show how this abstract relation is realized in a parametric model.

The measurement model constraint block details how actual pressure in the pipe, as represented by the *Pressure* state variable, causes a particular measurement value to be produced. In State Analysis, measurement models are important because they document factors such as sensor sensitivity and range, noise, and latency that must be compensated for in the estimation process.

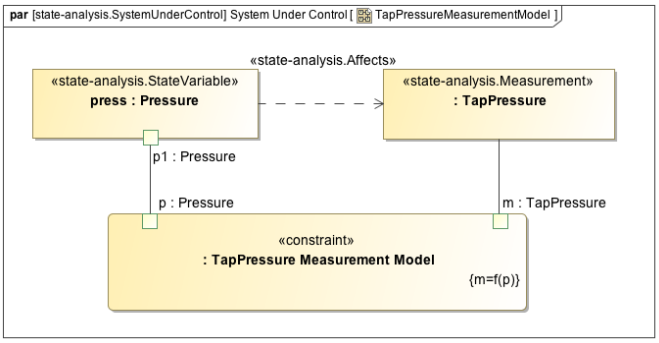

Figure 14 Measurement Model as Parametric

Once the system under control has been sufficiently modeled, the design of the control system can begin. In this system we define functional state variables to represent each of the state variables identified in the state analysis (see Figure 15). Stereotyped dependencies indicate that each functional state variable implements one or more state variables in the system under control (the control system design can opt to aggregate closely-related state variables into a single data structure, particularly if they are all updated at the same time). Estimator components are defined for each state variable, and a controller is defined for the Valve Position (similar relations are defined for the Flow state variable, not shown). All of these elements are defined as part properties in the control software context block described above.

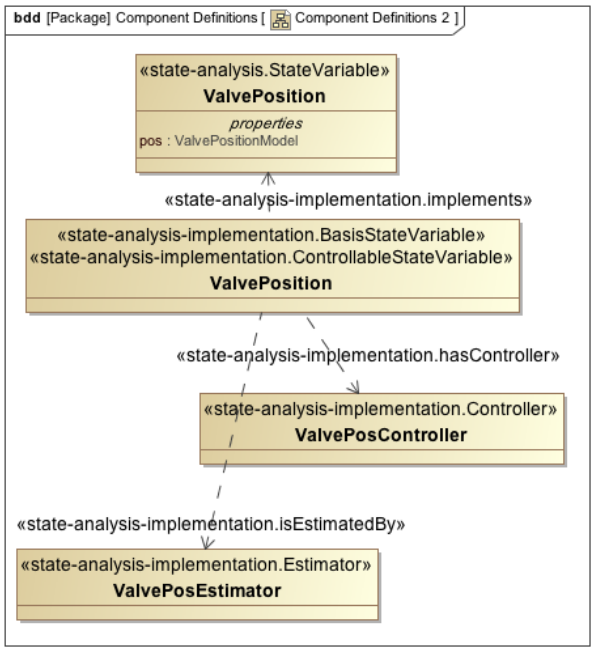

Figure 15 Control Component Definitions

Finally, we can describe the architecture of the control system in terms of its information flow using another ibd (Figure 16). Note that the interface between the elements inside the control software context (the content of the diagram) and the system under control is depicted using an interface block described as a *HardwareAdapter*. This block is defined elsewhere in the model containing typed ports for each of the command and measurement types that the control system will exchange with the system under control. Command and Measurement elements are modeled in SysML as signals that flow through flow ports. The flow ports indicate the direction of the flow (in SysML 1.3 this will be modeled with flow properties). Measurements always flow from the system under control into the control system. Commands normally flow from the control system to the system under control, but we define our hardware adapter so that it can buffer one or more previously sent commands for reference by the control system. In our very simple example, the valve position estimator has no direct sensing of the valve position (see Figure 13), so it uses the last sent command to infer the valve position.

The profiles for MagicDraw are constructed so that the stereotype menus will only offer those stereotypes that apply to the selected SysML entities or relations. For example, to create the state effect relations in Figure 12, the part blocks (whose types are already stereotyped as <<state-analysis.StateVariable>>) are automatically added when the diagram is created. An effect is modeled by drawing a dependency arrow from the affecting state variable block to the affected state variable block, and then applying the <<state-analysis.Affects>> stereotype. If an effect arrow had been drawn from a measurement to a state variable the stereotype would not have been offered as an option because this is an invalid relation according to the ontology. In this way the semantics of the ontology inform the modeling tool to help enforce model consistency with the rules of the domain. Similarly, these underlying constraints are available to MagicDraw's built-in validation tool, or to external tools that can parse the model and assess consistency of relations. This improves significantly the usability for the modeler, and the ability to validate models during modeling.

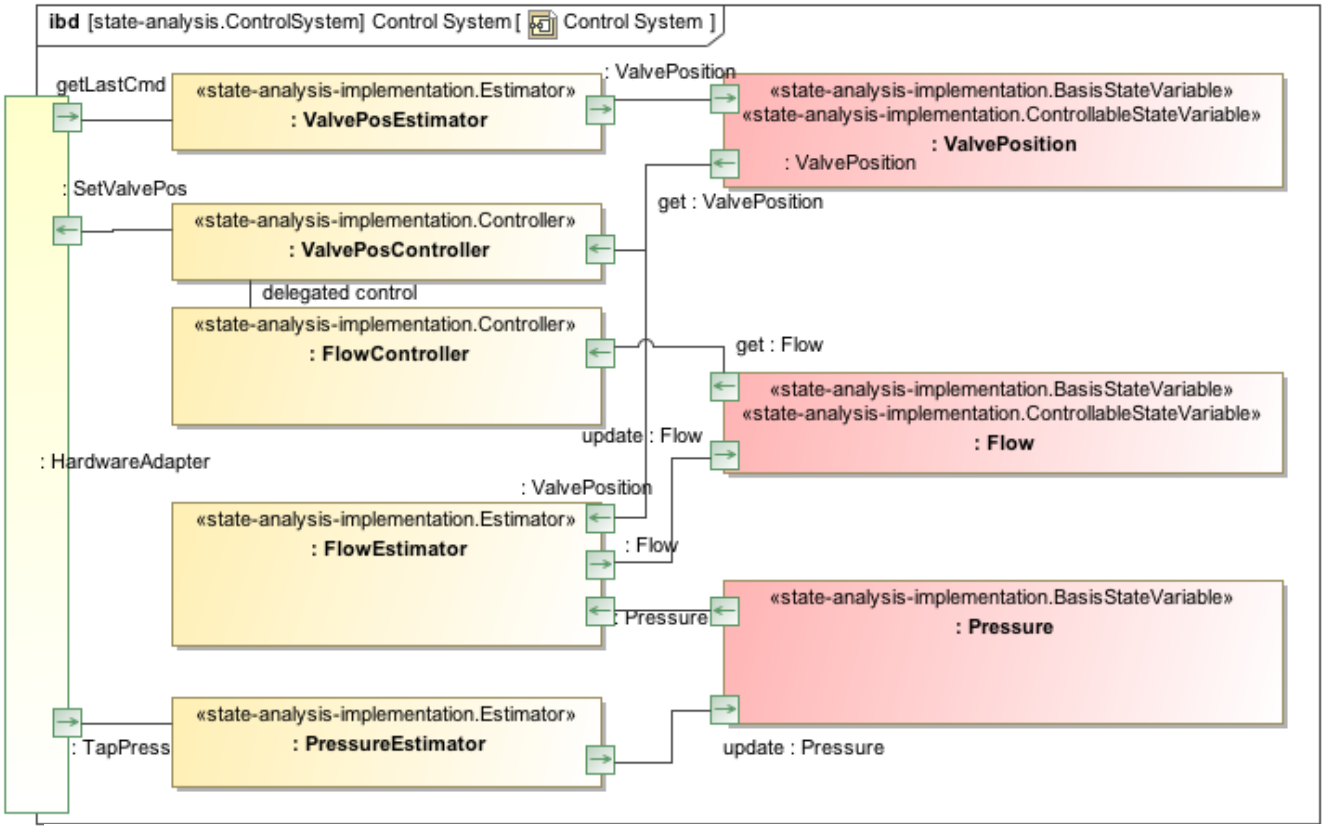

Figure 16 Functional Architecture

8. Conclusions and Future Work

Applying stereotyped relations in models allows the model to be analyzed to compare the semantics and constraints expressed in the stereotype definitions with the details of the model, in order to verify that the model conforms to the semantics of the domain expressed in the ontology. This work has demonstrated that it is possible to define meaningful domain-specific stereotypes using a model transformation from OWL2, apply them in a SysML modeling tool, and then use those stereotypes to verify correctness properties in the model. When complete, the State Analysis ontology and SysML profile should enable control system engineers to model system behaviors, specify control behaviors and intent, and have the model enforce semantic consistency rules established by the principles of State Analysis.

This work improves on previous efforts to enable the use of SysML to perform State Analysis by providing a set of tool-specific customizations and modeling patterns that achieve much of the intent of State Analysis. However, some SA concepts have yet to be modeled and mapped, and we continue to refine others as our technique evolves (several concepts evolved significantly over the course of writing this paper). Because this remains a work in progress we have only described the modeling and mapping of a few key concepts from SA. In particular, we have yet to find entirely satisfactory representations for intent, including goals and goal networks in SysML. Consequently, our current effort is focused on formalizing the modeling concepts related to intent and behavior, including goals, scheduled activities, temporal constraints, and the ways these concepts associate with state variables and the flow of time. The modeling of behaviors is the focus of significant effort in the modeling community and at JPL [5, 12]. While much of the standards-focused effort focuses on descriptive modeling of behaviors, our work also intends to model the relationships between intent (goals) and behavior that explains, through State Analysis, how the behaviors satisfy the specified intent.

We have experimented with using activity diagrams (stereotyping activities as “scheduled

goals"), and using fork-join relations to represent time points. This is problematic in that activity diagrams cannot easily express temporal constraints on the activities, and because the token-passing semantics expressed in the SysML standard are inconsistent with those of goal networks. More recently we have begun developing a separate timeline ontology [5] and applying its concepts as stereotypes to these activity diagrams, to express temporal semantics distinct from those of SysML This is a focus of current work.

Considerable work remains to formalize the graph state variable concept [8] in SA. Graph state variables relate position, orientation, or other relative states within frames of reference. Describing these relations formally requires defining frames of reference, coordinate systems, and the mathematical structures for representing multidimensional quantities. Since those concepts are meaningful outside the domain of SA it seems best to define those in a separate ontology and reference them in the SA ontology. As of this writing that work remains incomplete.

We continue to reconcile and refine our JPL ontology and SysML mappings with ones developed separately by our colleagues at the European Southern Observatory. While the concepts and relations defined in our separate ontologies are mostly consistent, small differences remain to be resolved. Most of the refinements at this point involve deep relations within base ontologies that will help to relate concepts across modeling domains.

ACKNOWLEDGMENTS

Special thanks to Robert Rasmussen, Michel Ingham, Daniel Dvorak, and Seung Chung who also contributed to the ontology development and review.

This research was carried out at the Jet Propulsion Laboratory, California Institute of Technology, under a contract with the National Aeronautics and Space Administration and at the European Southern Observatory.

REFERENCES

[1] Ingham, M., Moncada, A., Bennett, M., Rasmussen, R., Engineering Complex Embedded Systems with State Analysis and the Mission Data System, AIAA Journal of Aerospace Computing, Information, and Communication, Vol. 2, No. 12, December 2005.

[2] Bennett, M., Rasmussen, R., Ingham, M., A Model-Based Requirements Database Tool for Complex Embedded Systems, International Council on Systems Engineering (INCOSE) International Symposium, Washington D.C., 2005.

[3] Murray, A., Rasmussen, R., A UML Profile for State Analysis, IEEE Aerospace Conference, Big Sky, MT., 2011.

[4] Karban, R., Kornweibel, N., Dvorak, D., Ingham, M., Wagner, D., Towards a State Based Control Architecture for Large Telescopes: Laying a Foundation at the VLT, 13th International Conference on Accelerator and Large Experimental Physics Control Systems (ICALEPCS), Grenoble, France, Oct. 2011.

[5] Chung, S., Delp, C., Fosse, E., Sarrel, M., Bindschadler, D., Representing Information using Timelines for System Design to System Operations, IEEE Aerospace Conference, Big Sky, MT., March 2012.

[6] Web Ontology Language (OWL), http://www.w3.org/2004/OWL/

[7] Protégé OWL Ontology Editor, http://protege.stanford.edu/

[8] Bennett, M., R. Rasmussen, Modeling Relationships Using Graph State Variables, IEEE Aerospace Conference, Big Sky, MT., March 2003.

[9] QVT (Query/View/Transformation) - Operational, http://www.omg.org/spec/QVT/, http://www.eclipse.org/m2m/.

[10] Bayer, T.J., Bennett, M., Delp, C.L., Dvorak, D., Jenkins, J.S., Mandutianu, S., Update – Concept of Operations for Integrated Model-Centric Engineering at JPL, IEEE Aerospace Conference, Big Sky, MT., March 2011.

[11] Rouquette, N., Jenkins, S., Transforming OWL2 Ontologies into Profiles Extending SysML,12th NASA-EST Workshop on Product Data Exchange, Oslo, Norway, May 2010.

[12] Conrad Bock, James Odell, "Ontological Behavior Modeling", Journal of Object Technology, Volume 10, (2011), pp. 3:1-36, doi:10.5381/jot.2011.10.1.a3.

[13] MagicDraw modeling tool, https://www.magicdraw.com/

BIOGRAPHIES

David Wagner *is a software system engineer and architect in the System Architectures and Behaviors group at JPL and was a principal developer of the Mission Data System in 2000-2006. Since then he has continued to apply MDS technology and State Analysis in several applications. He is currently a member of the project system engineering team on the Europa Habitability Mission formulation project. He has a BS in Aerospace Engineering from the University of Cincinnati, and MS in Aerospace Engineering from the University of Southern California.*

Robert Karban *is a Software Engineer at the European Southern Observatory (ESO), developing highly distributed real-time control systems for telescopes since 1996. His current main task is the modeling and development of interdisciplinary systems, in particular the control system of the E-ELT, as its system architect. His roles have varied from project manager to developer, including mentoring, and devising development standards. Robert has also been leading INCOSE's Model Based Systems Engineering (MBSE) Challenge team on Telescope Modeling since 2007, providing practical applications of SysML in the astronomy domain. He is a certified OMG Systems Modeling Professional - Advanced. Robert Karban received his M.S. in computer science in 1990 from the Technical University of Vienna, Austria. Afterwards he worked several years in the Medical Engineering industry, and developed in the European Organization for Nuclear Research (CERN) embedded software for accelerator control systems.*

Nicolas Rouquette *is a Principal Engineer in the Systems and Software Division at the Jet Propulsion Laboratory. Nicolas pioneered comprehensive code generation of flight/ground software from high level systems engineering specifications for the Deep Space One mission. As a leading expert in modeling and model transformation, he represents NASA's interests at the Object Management Group (OMG) where he recently chaired the 2.4 revision of the Unified Modeling Language (UML) and produced the last two versions of the Systems Modeling Language (SysML). Dr. Rouquette received his M.S and Ph.D. in Computer Science from USC.*

Steven Jenkins *is a Principal Engineer in the Systems and Software Division at the Jet Propulsion Laboratory, currently supporting JPL's Integrated Model-Centric Engineering Initiative. His interests include application of semantic and modeling technologies to systems engineering. Dr. Jenkins holds a B.S. in Mathematics from Millsaps College, an M.S. In Applied Mathematics from Southern Methodist University, and a Ph.D. In Electrical Engineering (Control Systems) from UCLA.*

Matthew Bennett *is a Senior Software Systems Engineer in the Systems Engineering section at the Jet Propulsion Laboratory. His research interests include model based engineering, software architecture, fault protection, and spacecraft autonomy. He has designed, developed, and delivered architectures, software and technologies for human-robotic interaction, model-based engineering, fault protection, autonomous planning and scheduling, control systems, data visualization, guidance and control, performance analysis, and simulation. He holds an MS from the University of Washington in Computer Science, and a BS from the University of California at San Diego in Computer Engineering.*

Dr. Michel Ingham is the supervisor of the System Architectures and Behaviors group, in the Systems Engineering section at the NASA Jet Propulsion Laboratory. His research interests include model-based methods for systems and software engineering, software architectures, and spacecraft

autonomy. He is a core contributor to JPL's Integrated Model-Centric Engineering initiative, and model-based systems and software engineering efforts across the Laboratory. He has played an important role in the development and formalization of the model-based systems engineering methodology called State Analysis, and its application to the design and implementation of state-based flight and ground software architectures. Dr. Ingham received his Sc.D. and S.M. degrees from MIT's Department of Aeronautics and Astronautics, and his B.Eng. in Honours Mechanical Engineering from McGill University.