Model-Based Systems Engineering, Real-Time Operations, and Autonomy

Fernando Figueroa,[1] Lauren Underwood,[2] Duane Armstrong[3]
NASA Stennis Space Center, Stennis Space Center, MS 39529, USA

Model-Based Systems Engineering has been enabled by the development of the SysML language and software tools to create systems models. Systems models described in SysML incorporate frames (Diagrams) that represent behaviors (activities, sequences, state machines, use cases), requirements, and structure (definitions, internal structure, parametric formulation, and packaging). The SysML models are, in turn, used by applications to do analysis and studies of the designs and operational capabilities. These uses of the model are based on simulations, and do not include hardware. This paper presents a software environment and processes that enables more comprehensive systems models for MBSE, and use of these rich models for real-time operations. The paper describes a software platform that enables creation of comprehensive models, beyond what is now possible with SysML and related software tools, called the NASA Platform for Autonomous Systems (NPAS). The platform encapsulates a paradigm and infrastructure for creating systems models with complexity levels comparable to the ones handled by SysML software tools, but with additional fidelity that includes detailed design diagrams encompassing sensors, components, and design topologies. Furthermore, NPAS enables incorporation of data, information, and knowledge (DIaK) to implement autonomy and Integrated System Health Management (ISHM) and the inherent integration of content encompassing SysML structure and behavior diagrams throughout the NPAS modelAnd lastly, the NPAS models are used in real-time operations, taking advantage of the fidelity and complexity encompassed in the models in order to implement "thinking" ISHM and/or autonomous operations. . Incorporation of SysML model content into an NPAS model is briefly discussed.

I. Nomenclature

NASA = National Aeronautics and Space Administration
SSC = Stennis Space Center
NPAS = NASA Platform for Autonomous Systems
ISHM = Integrated System Health Management
DIaK = Data, Information, and Knowledge

II. Introduction

A model typically is the outcome of a design process and encompasses all the information that describes the system (e.g. parts, assembly, operation and maintenance). In this respect or viewed in this way, a model is a representation of a system that will be built. Another interpretation of model refers to descriptions of how the various elements of a system are expected to behave/operate, so a model may be an equation or other transformation artifact that describes the relation between inputs to the system/element and its outputs. The construct of Model-Based Systems Engineering (MBSE) encompasses the process that leads to a final design whereby behavioral models are used to achieve the desired final design of the system. Moreover, the MBSE process implies capture of all the information and its inter-connectedness, along with requirements and concepts of operation that flow down from the stakeholders to the systems engineers.

[1] Lead Autonomous Systems and Operations, Test Technology Branch, AIAA Associate Fellow.
[2] Project Manager, Autonomous Systems Lab, Test Technology Branch.
[3] Chief, Test Technology Branch..

The MBSE paradigm for design has prompted the development of a standard language, SysML [1], and also software platforms that facilitate creation of models where complexity is systematically managed. All this results in models that are sustainable, expandable, and evolvable; where formal methods for validation and verification can be naturally applied.

Models created using SysML can be very comprehensive and can include multiple levels (e.g. component, system, and operational models). "Diagrams" are used to create the models, and "packages" are used to organize the model. Also, "views" are used to collect information from multiple packages for focused access by developers and stakeholders.

The Autonomous Systems Laboratory (ASL) at NASA Stennis Space Center (SSC) started development of what is currently the NASA Platform for Autonomous Systems (NPAS) in 2004 [2, 3]. There are important commonalities and differences between NPAS as an enabler for intelligent applications (e.g. intelligent systems, and particularly intelligent autonomous systems); and SysML and external tools developed to support design of complex systems utilizing SysML language and constructs. This paper discusses these commonalities and differences, as well as platforms that use SysML such as MagicDraw [3] or Rational Rhapsody [4] or various other platforms.

III. NPAS

NPAS is a platform that enables development of "thinking" autonomous systems, as well as deployment and operations. When the preliminary version of NPAS was instantiated, a decision to use G2 [4] was made; G2 was identified as a software environment that embodied key paradigms and capabilities for implementing "intelligent" applications that would run in real-time. Figure 1 shows the software building blocks of NPAS. NPAS was developed and continues to evolve as a G2 application, augmenting G2 with the capabilities required for implementing autonomous systems and operations. G2 was originally developed at MIT in the 1980's, and resulted in an offshoot company called Gensym Corporation. Since then, G2 has been used worldwide in commercial applications. However, NPAS is the first implementation of G2 in the area of autonomy. G2 was acquired by Ignite Technologies [2], which currently owns and supports the product today.

The G2 engine provides a full-scale object-oriented programming environment encompassing a combination of graphical and procedural code in the G2 Language (using structured natural language expressions). In addition, G2 can include layered products, also developed in G2, that provide important capabilities. The layered products used by NPAS include communications bridges and Symcure a G2 product which enables reasoning using cause-effect diagrams), and is a powerful capability to implement Failure Modes and Effects Analysis (FMEA) that makes possible diagnostics and prognostics, as well as other reasoning needs such as for planning and task scheduling and execution.

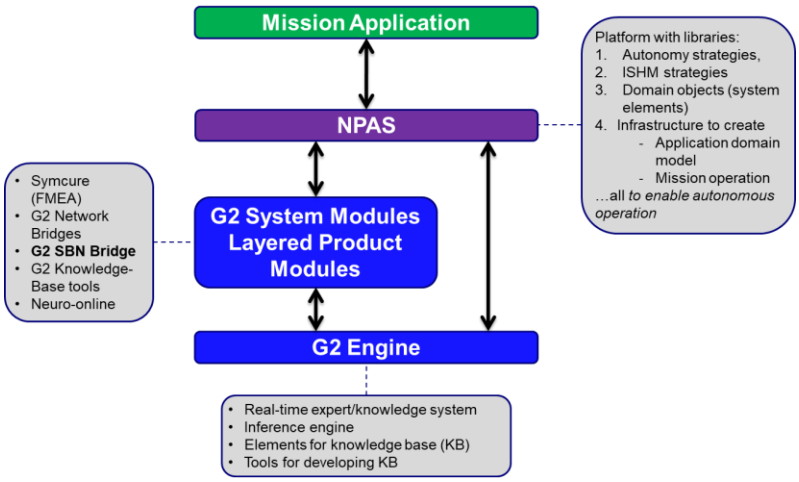

Figure 1. NPAS build

NPAS incorporates infrastructure and core libraries, functional models, analysis, reasoning, decision making, activity execution, network communications, and user interfaces that enable capabilities required for implementation of autonomy. Figure 2 shows the NPAS software architecture. NPAS comprises four primary modules (white blocks-(1) ISHM Strategies, (2) Autonomy Strategies, (3) Tools to create application domain model, and (4) Tools to create mission operations).

Block (3) "Tools to create application knowledge domain model" enables creation of a knowledge model of the system (blue box "NPAS system application knowledge domain model"). This model can include every object (part, component) that encompasses the system. In fact, schematic drawings are replicated as part of the knowledge model, and in doing so, the topology and semantic definitions, such as "connectivity", are automatically established and captured. This knowledge model encompasses the reference representation of the system in the same manner as a SysML model does, but NPAS additionally includes other attributes that will be addressed in detail in sections V and VI.
Block (4) "Tools to create mission operations", enables the creation of tasks, plans, and timelines, as well as the execution of the timelines used to achieve the missions (blue box "autonomous mission plans, schedules, and execution").

Block (1) "ISHM Strategies", encompasses code that implement strategies to carry out ISHM functions including: (1) anomaly detection, (2) diagnostics, (3) consequences of faults, (4) prognostics and (5) awareness about the health state of all objects that encompass the KM of the system. Information relevant to availability or lack of resources is used by Block (2) "Autonomy Strategies" which is represented as another block, and is used in order to determine the best way to manage consequences of anomalies that affect negatively plans and timelines.
NPAS also includes tools to create graphical user interfaces (GUIs), as well as a set of standard communication bridges and tools to implement costume bridges. This enables implementation of distributed applications across networks.

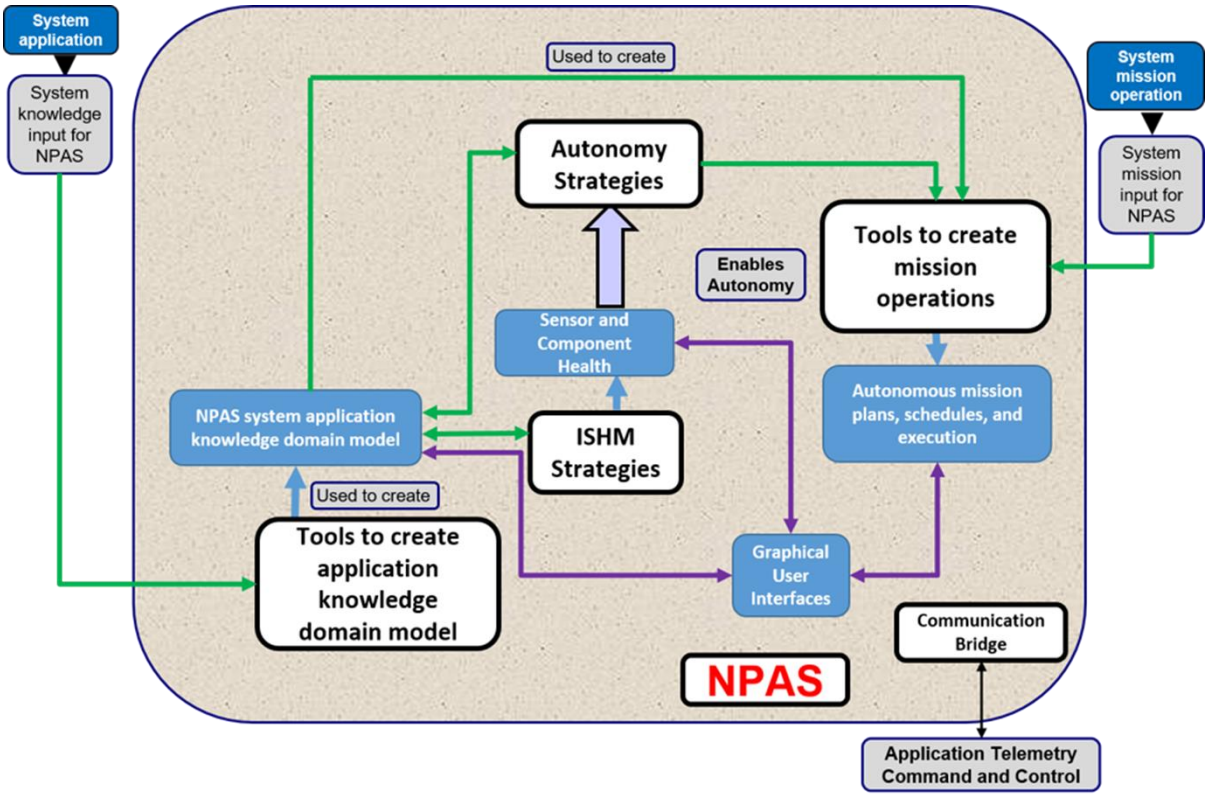

Figure 2. NPAS Software Architecture.

IV. System Model Development using SysML

SysML is a general-purpose graphical modeling language that supports the analysis, specification, design, verification, and validation of complex systems. A SysML model is then established as a definitive reference that simultaneously evolves and is used in all phases of the system design process. The reason for using SysML to design a system is to create a standards-based comprehensive model. Comprehensiveness is captured and visually represented in the diagram shown in Figure 3. this comprehensiveness is a taxonomy/architecture that captures all aspects of a model of a system represented throughout a hierarchy of diagrams. The Structure Diagram encompasses classes of diagrams that enable a very complete description of a system. Block Definition Diagram and Internal Block Diagram are used to capture systems components and topology/assembly. Parametric Block Diagram is used to enable parametric studies needed to meet requirements, and Package Block Diagram is used to organize all diagrams in a way that is conducive to efficiently understanding the system model.

The Behavior Diagram encompasses classes of diagrams that enable a description of what the system does, or how it behaves. Activity Diagram describes behavior as a network of process steps, Sequence Diagram describes behavior as message exchanges between system components, State Machine Diagram describes transitions between states of the system, and Use Case Diagram provides a description of functions, from the user perspective.
Software platforms that support creating SysML models provide graphical user interfaces to represent diagrams (e.g. MagicDraw). Diagram Frame is shown as a square container where other diagrams are placed, and Package diagrams are used to organize diagram frames so that development can be collaboratively done, whereby different individuals work is on different diagrams.

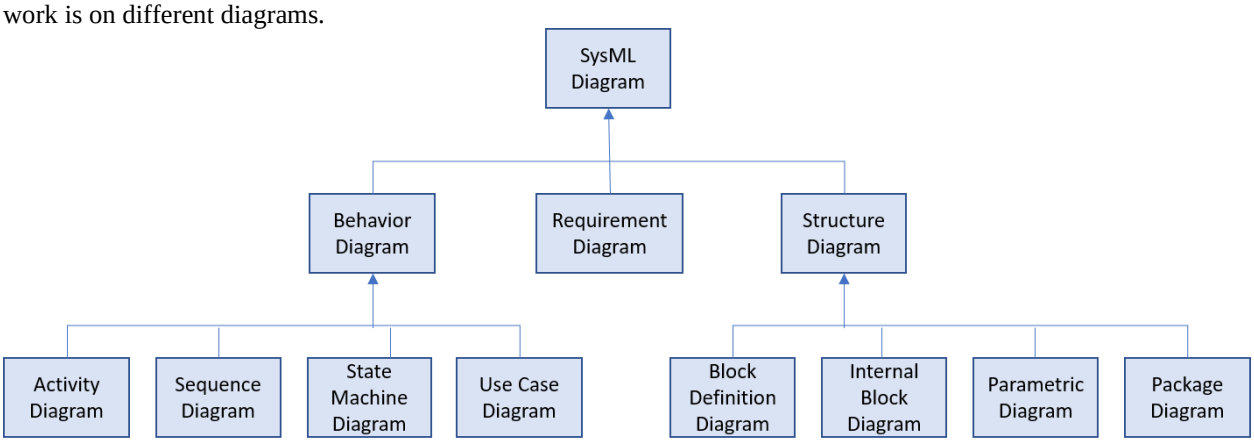

Figure 3. SysML diagrams for model development.

V. G2/NPAS Model Development

G2/NPAS is a platform that uses models in real-time and includes capabilities for system model development as well as system operation . Development includes graphical and procedural coding using the G2 language. The G2 language uses structured natural language expressions, which makes the code intuitive to understand. When creating a model in G2/NPAS, the platform generates multiple knowledge bases (KBs), one for each module that is part of the model. The modules are loaded according to a hierarchy, whereby a model that is at a level of the hierarchy loads all modules that are required by this module and are below in the hierarchy.

G2 is an object-oriented software development environment, with additional capabilities that enable “intelligent” applications and real-time operations.

Just like platforms like MagicDraw, which have tools to create SysML models, G2/NPAS also has tools to create which are referred to as Application Knowledge Domain Models (AKDMs). All content that can be specified in any SysML diagrams can be included in NPAS AKDMs. NPAS models, however, also include additional content that

enable autonomous operations, and the overall model represents a “live” model, since it is used in real-time operation of the system. Note that graphical formats and nomenclature are not the same in NPAS as in SysML. Window containments such as those representing diagrams in SysML, are called workspaces in G2/NPAS. Figures 4 and 5 display a SysML diagram and an NPAS Workspace, respectively. Workspaces have names that can be defined just as header descriptions are in SysML, however, the developer standardizes the definition of header components .
Additional content in NPAS models can be seen as represented as additional frames in SysML, corresponding to “Structure” and “Behavior”, as shown in Figure 6. The Autonomy and ISHM model elements correspond to SysML Behavior diagrams, and Structure Model element correspond to SysML Structure diagrams. Autonomy encompasses mission plan creation, scheduling, execution; as well as strategies to resolve issues that may arise. ISHM encompasses real time anomaly detection, diagnostics, prognosis, and FMEA. Structure Model enables schematic level topology and behavior capture. Autonomy and ISHM capabilities process information, in real-time, using the reference AKDM, and then corresponding update the model with health, availability, relationships, and other attributes that are updated by ISHM and utilized by autonomy as these changes arise.

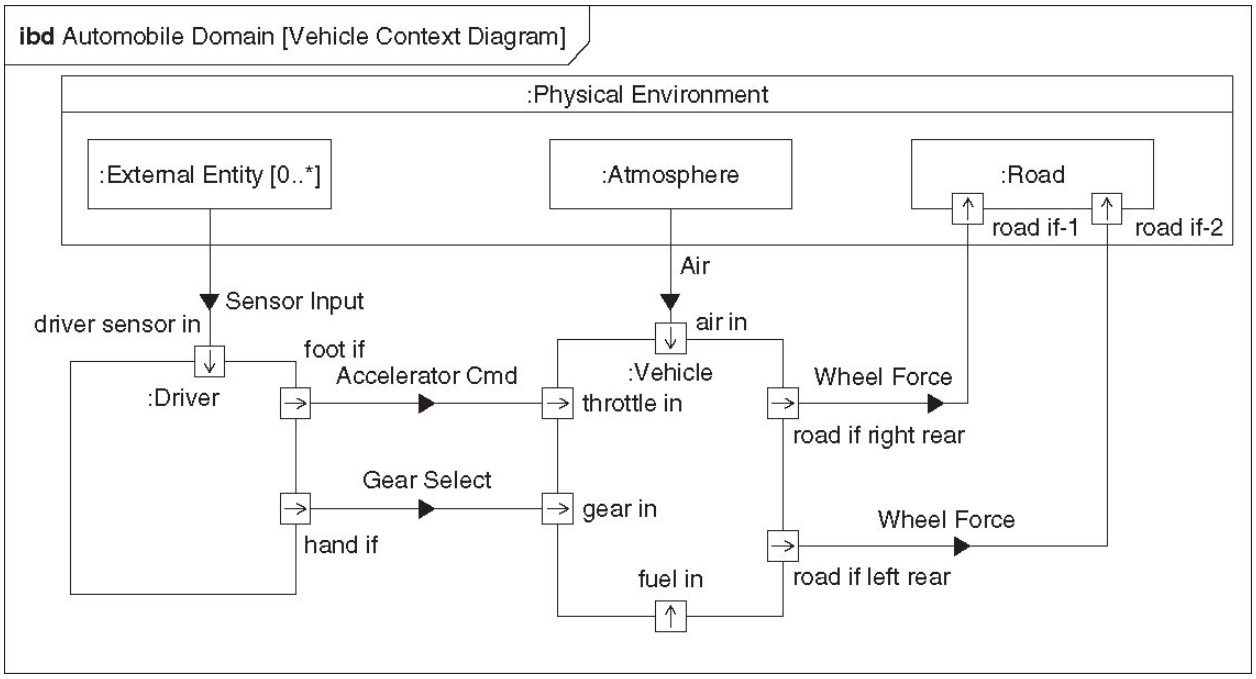

Figure 4. Internal Block Diagram in SysML (Structure)

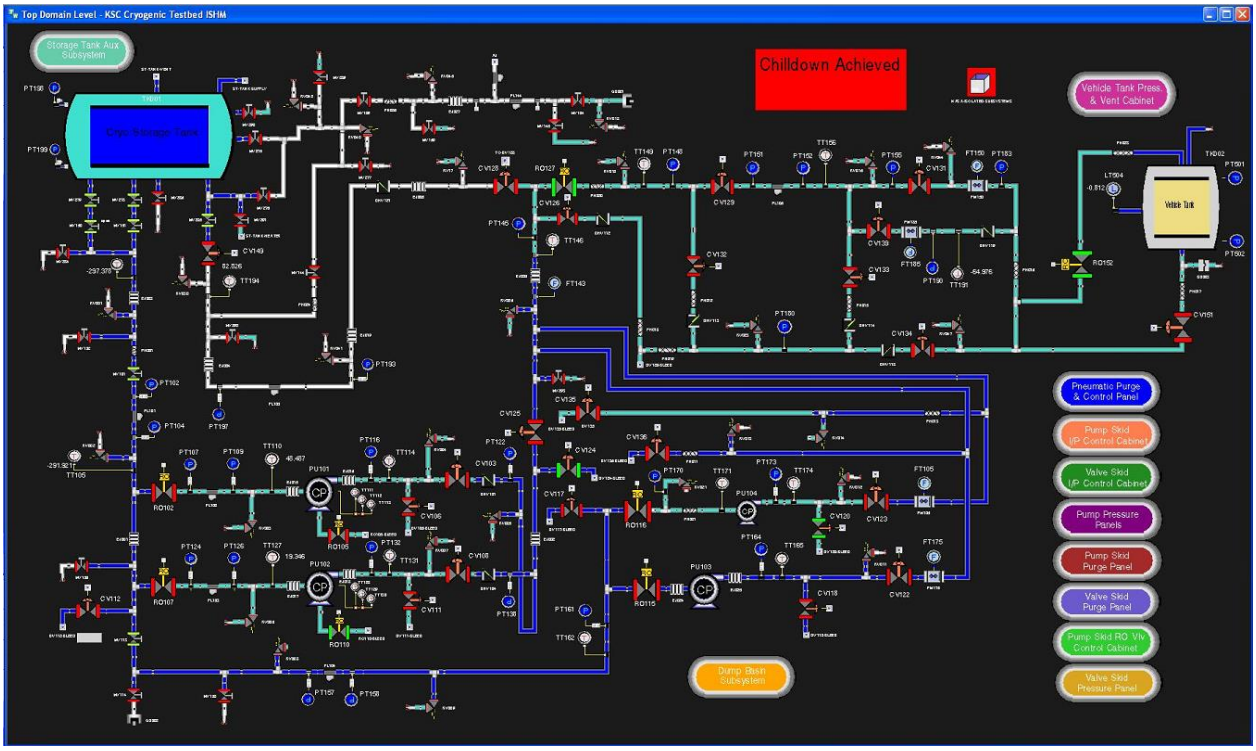

Figure 5. Schematic level diagram in NPAS (Structure).

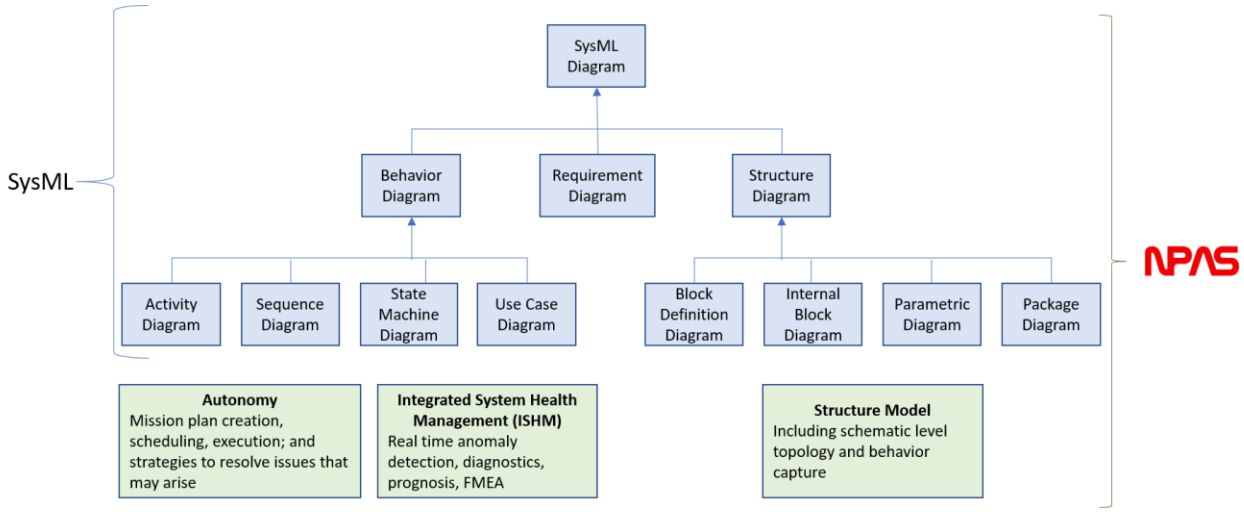

Figure 6. Autonomy and ISHM model elements in NPAS correspond to Behavior Diagram. Schematic level modeling correspond to Structure Diagram

Purposeful decomposition into functional elements (e.g. encapsulated in separate SysML diagrams) is not necessary in NPAS, but it is acknowledged/incorporated automatically as functions and relationships are recognized to be associated with concepts that are then employed in models that enable analysis and decision-making in real-time. All structure diagrams can be innately integrated on the basis of schematic level diagrams. Figure [7] shows a schematic of a battery system that includes sub-diagrams (contained in sub-workspaces) that include details of a battery, down to the cells. What is shown in Figure 7 is graphical code in NPAS. Each sensor or component is an instance of its class, selected from a library of domain objects (e.g. voltage sensor, battery, wire, switch). When the objects are connected the schematic is graphically built, the AKDM is created automatically, and includes connectivity

relationships. Additional G2/NPAS language code has the ability to automatically navigates the schematics to recognize concepts. This is how behavior diagrams content is integrated with the structure diagrams.

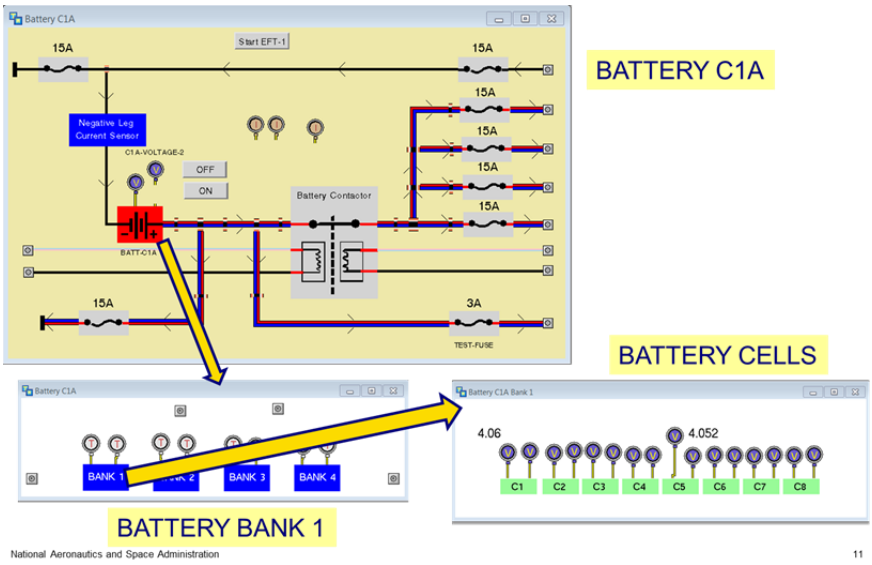

Figure 7. Schematic level system structure showing details that include objects, their connectivity, and the ability to implement reasoning including all objects.

An example of a state analysis that identifies inconsistencies (anomaly events) and logs anomalies is represented in Figure [8]. The graphic is a precise representation of graphical code that is used to create cause-effect diagrams. The nodes represent events related to anomalies in the elements of an instrumented valve that can be linked to inconsistencies when a physics valve model is applied using data from the valve sensors and the command signal. These events are considered to be health states of the classes of objects (e.g. hm-physical-analog-sensor).

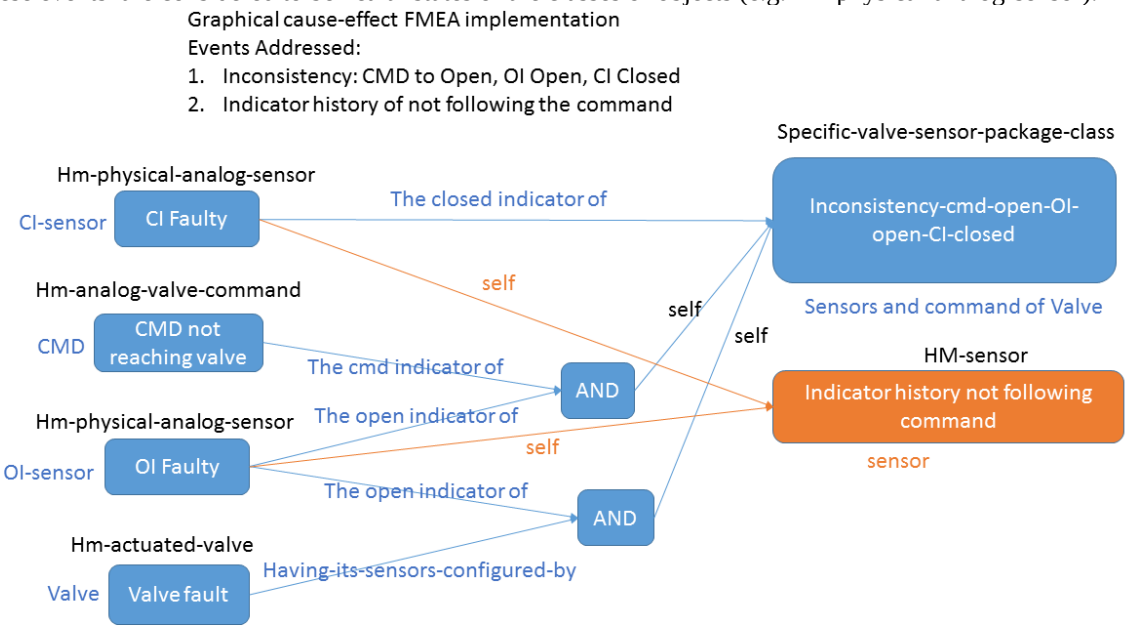

Figure 8. Reasoning with events associated with classes of objects to infer anomaly conditions for an instrumented valve.

A core concept in commodity distribution systems (e.g. power system) is "flow-subsystem" which is defined as an ordered collection of objects that include, at its extremes, a source and a sink for the commodity. For example, in a space power system, a source could be the solar panels, and sinks are systems powered such as the life support system. Flow subsystems change dynamically according to configuration changes (opening/closing of flow-control-objects such as switches), or when anomalies that affect flow control occur, i.e. a bad switch. In this case, , the G2/NPAS code would be running in the background, applying flow models (physics or otherwise) to every flow-subsystem in order to determine anomalies, predict future values, and participate in autonomous operations activities. This is an example of how commodity distribution behaviors are incorporated and integrated into the NPAS systems models.
Another concept example that integrates model-based reasoning in NPAS models, is flow-section (Figure [9]). Flow-section is defined as a collection of objects bounded by stops or flow control objects (e.g. valves) that could be considered/conceived as one piece, and would therefore filled with the same commodity at any given time; and the expectation would be that the properties of the commodity (e.g. temperature of a fluid) would be the same throughout the flow-section.

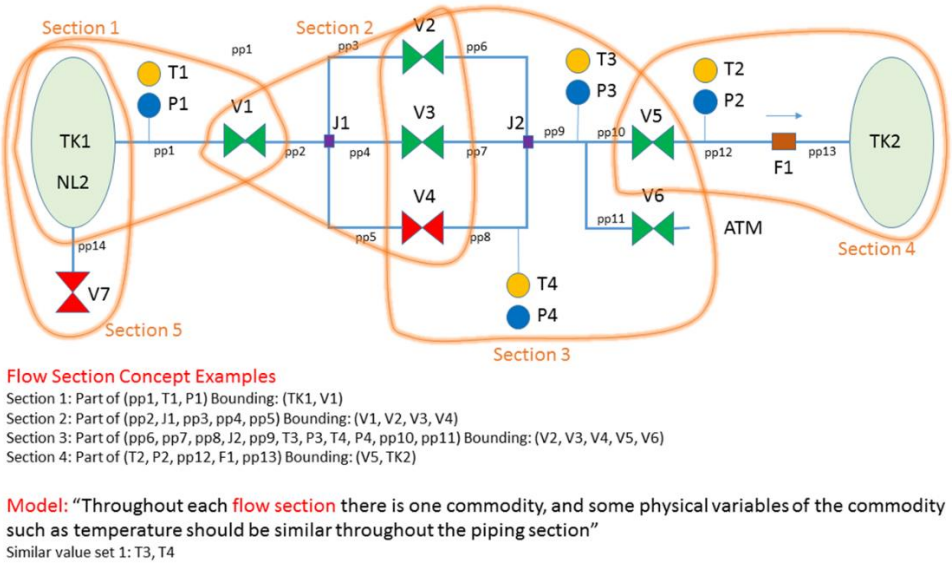

Figure 9. Flow section concept and use of the concept for a behavior model.

Figure [10] represents an example of how object behavior capability is integrated with Structure Diagram type of information. In this case, behavior for the class of rotary pumps is implemented. The Workspaces help organize the modeling. The top-left workspace enables navigation to the workspace containing methods (G2 structured natural language code) to calculate efficiency, and then the workspace containing procedures (G2 structured natural language code) that is sued to apply models that evaluate consistency with physics-based models of nominal behavior as well as off-nominal such as cavitation. When events such as cavitation, off-nominal efficiency, or inconsistency with the nominal model are determined, corresponding events are set to "true" in cause-effect diagrams (FMEA), which triger diagnosis and determination of consequences (shown on the lower right of the figure). The top right show alarms that correspond to anomaly or other events.

VI. G2/NPAS software environment capabilities for integrated system model and real-time operations

G2 is a unique full object-oriented environment that enables knowledge representation using graphical tools and coding using structured natural language expresions. In addition to the model development capabilities described above, NPAS leverages the following specific G2 capabilities for knowledge representation [x].
Temporal knowledge: to represent, for example, behavior over periods of time
"if the rate of change of the pressure over the last 45 seconds …"
"if the valve has moved, then I should see the temperature responding about 10 seconds later."
Dynamic Models: to represent commodity management systems such as in plants.
"d/dt (the concentration of oxygen in the habitat module) = …"

Object Oriented knowledge representation: knowledge expressed in terms of object class… extends to many objects.
"if the pressure of any habitat … and the rate of change of the temperature of the habitat …"

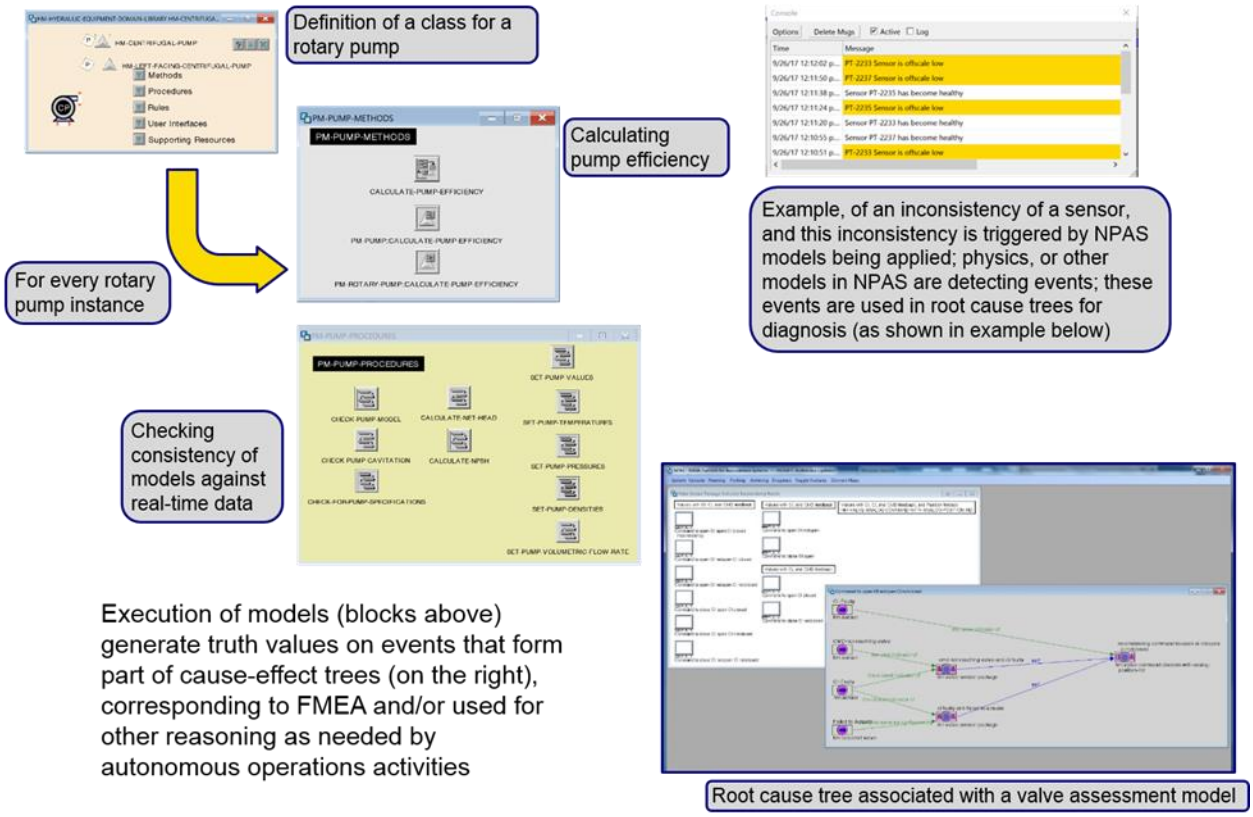

Figure 10. Integration of rotary pump behavior with procedural code (in structured natural language) and structure information.

Connectivity of objects: connectivity interpreted from schematics rather than being coded is a powerful capability. Connectivity is used in analysis, and reasoning about the connections of objects. A powerful use of connectivity, for example, is automatic channelization in systems like a power, fluid, mechanical, and communications.
"if the pressure of any pressurizer PR < 150 and the temperature of PR > 450 then invoke emergency rules for the reactor connected to PR"
Generic knowledge: applied to classes of objects. For example, code and rules about sensor behavior can be applied to hundreds or thousands of objects in a system. The example above applies to any pressurizer in the system.
Procedural knowledge: applies to when the order of the reasoning process is important. The G2 language includes coding of procedural knowledge using natural language expressions (Figure 11 shows an example of procedural language).

"Do in parallel [until one completes]
<statement> [;<statement>] … [;]
And
do
for T = each tank
do
for V = each valve connected to T
… etc."

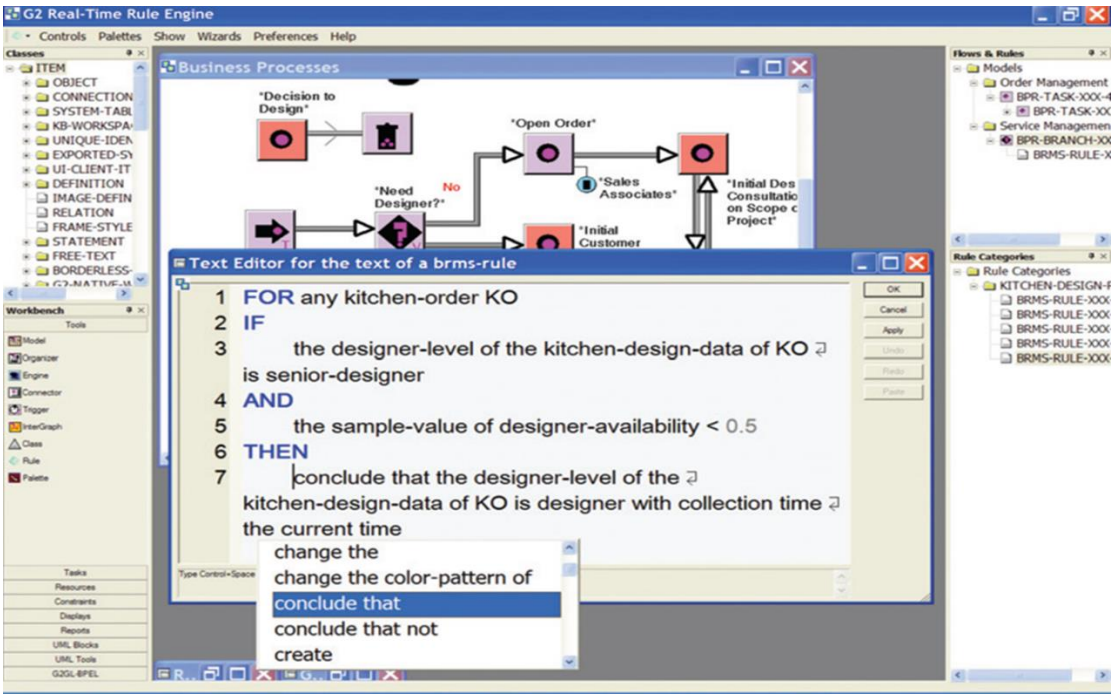

Figure 11. G2 procedural language (structured natural language).

Reasoning using "meta-knowledge": knowledge about knowledge. For example, knowledge about reasoning along flow chanels would be indexed as being appropriate for this reasoning. It enables the ability to focus on employing the knowledge that is appropriate for a task.

Reasoning using deep knowledge: deep knowledge enables early detection of anomalous conditions, when there is still time to mitigate. This is done using a combination of models (physics based knowledge) and rules about expected behaviors. This is a model-referenced methodology that embodies an accrual of intelligence.

Finding the best answer in a fixed time: the following structure addresses this problem.

" if (the first of the following that has a value, ...) ... then ...

The first element in the list is the preferred reasoning, and if the information to evaluate it exists, then an immediate answer exists. If not, evaluations are launched for consecutive elements in the list, and conducted at a set time limit, and then, the best available answer is used.

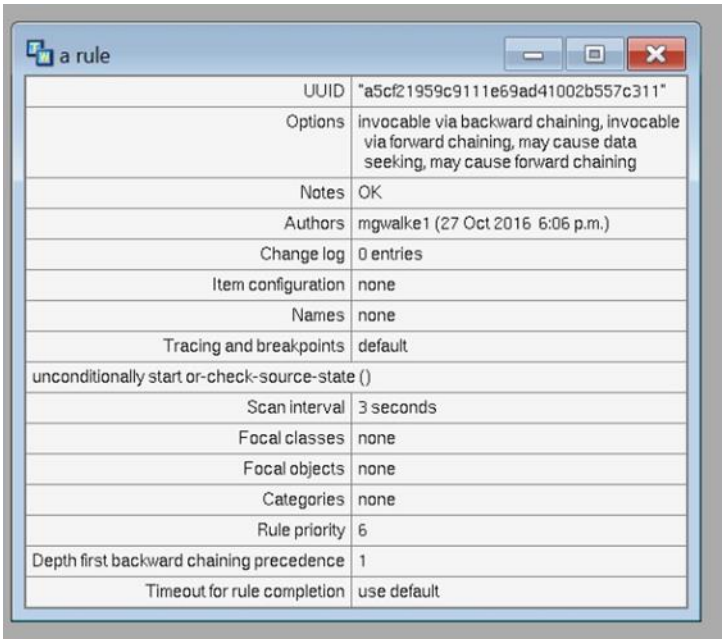

Figure 12. Example of rule definition.

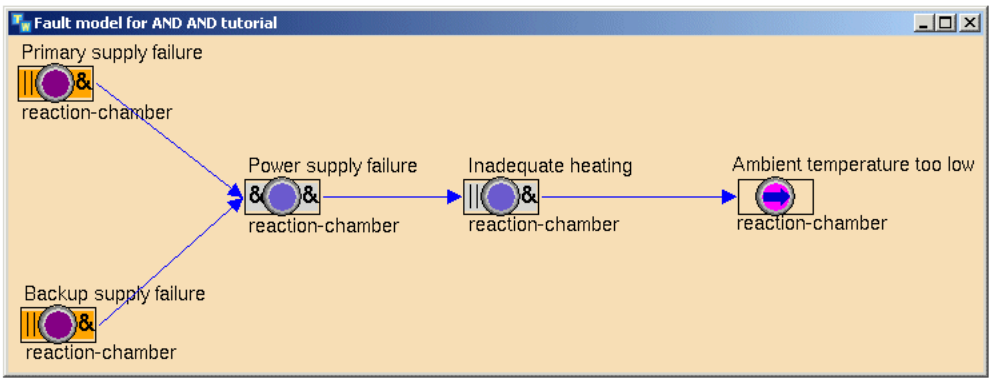

Figure 13. Example of fault model coding (FMEA).

Real-time truth maintenance: as time passes, conclusions need to have a validity time limit, even without new information/knowledge being added. Every measurement and everything reasonded with the measurements have an associated validity interval. Validity can then affect higher level reasoning and conclusions to assign values of true or false or some value in between.
Scheduling G2 code tasks and priorities: tasks (procedures, methods, rules) are scheduled with priorities, allowing concurrent reasoning strings at the same time, with the ability to promptly respond to dynamically changing priorities and attending to high-priority tasks as needed.
Additional capabilities to develop applications in G2 include the following.
Structured natural language (Figure 11): an editor is used for creating and editing objects, definitions of rules, and other forms of knowledge. The structured natural language is means towardsa truly natural language for human interface. The syntax lets the human designer freely express rules, procedures, models and other knowledge. Structured natural lanaguage provides a “look ahead” view of allowed entries.
Defining objects: there is a structured schema to define objects. An object class may be changed (evolved) and existing instances would therefore be automatically changed to conform. There are special object classes to enable data flow from/to communications’ networks.
Expanding the domain: Cloning and connecting enables rapid expansion of the domain models (e.g. schematics).
Knowledge libraries: may include tens of thousands of objects, rules, procedures, models, and other items that are re-usable. The objects are organized in workspaces (e.g. diagrams in SysML).

Inspecting knowledge: knowledge contained in the model can be inspected and displayed in a temporary workspace. This helps understand and manage the knowledge bases.

- *" Show the object hierarchy of pump"*
- *" Show on a workspace every rule"*
- *" Show on a workspace every rule containing the word pump"*

Dynamic simulation testing: simulation capability can be quickly built to test behavior throughout the knowledge system.

Interactive knowledge capture: G2 is an interactive development environment which encompasses the use of graphics and structured natural language, as well as animated graphics; enabling capture of expert knowledge, in an environment where engineers and experts can work together.

VII. Conclusions and Recommendations

SysML modeling platforms enable application software tools for analysis during the project/system development process (for example, tools to capture the total weight and center of gravity of a spacecraft). This is done by interacting with behavior diagrams. The objective is to determine if the model behaves according to requirements and concepts of operations established by the stakeholders. However, using SysML models to embody real time operations is a completely different capability. In reference [8], the authors report about performing a first assessment of the strengths and the weaknesses of SysML as a real-time modeling language. The paper describes creation of a SysML model of the Generalized Crossing Gate Problem. The authors found that the SysML language is not fully satisfactory for modeling precise time requirements.

However, evaluations made in this paper, , revealed that G2/NPAS is not just be able to include elements to describe real-time behavior, but also able to have a model that is used in real-time to operate the system. Corredspondingly,, one way to utilize SysML models would be to develop a translator that would enable SysML model information tobe ported automatically into the G2/NPAS environment. This porting application would enable incorporation of the content from SysML models into the G2/NPAS system application knowledge domain model, and then be used for autonomous operations. One major challenge would be to create standarization of classes of objects, like the ones that now exist in the NPAS libraries of domain objects, and then use them in creating G2/NPAS models. This conceptualization would require an ontology and language that is currently substantially developed in NPAS. Because G2 can load XML code, once ontology and language are defined, NPAS could just read the SysML code to populate NPAS models.

VIII. Acknowledgements

This work was made possible by funds from NASA's Advanced Exploration Systems (AES), Project NPAS.

IX. References

1. http://www.omgsysml.org/
2. https://www.ignitetech.com/gensym/
3. No Magic's MagicDraw https://www.nomagic.com/products/magicdraw
4. IBM Engineering Systems Design Rhapsody.
5. Figueroa F., Underwood, L., Walker, M., and Morris, J., "NASA Platform for Autonomous Systems (NPAS)," AIAA SciTech., 7-10 Jan 2019.
6. Fernando Figueroa, Lauren Underwood, Jonathan Morris, and Ben Hekman, "Hierarchical Distributed Autonomy: Implementation Platform and Processes," IEEE Aerospace Conference, Yellowstone Conference Center, Big Sky, Montana, Mar 7 - Mar 14, 2020.
7. Robert L. Moore, "G2: A Software Platform for Intelligent Process Control," Proceedings of the 1991 IEEE International Symposium on Intelligent Control," 13-15 August 1991, Arlington, VA, USA.
8. Pietro Colombo, Vieri Del Bianco, Luigi Lavazza, and Alberto Coen-Porisini, "An experience in modeling real-time systems with SysML," International Workshop on Modeling and Analysis of Real-Time and Embedded Systems, Genova, October 2006.