
Fundamentals of System Modeling

Version 0.3
January 3, 2009

Pathfinder Solutions
www.PathfinderSystemsModeling.com
+1 508-568-0068

Table Of Contents

Abstract	2
Motivation	2
Models in Systems Engineering	2
Why Bother with Modeling?	2
The Systems Modeling Approach	3
SysML Overview.....	4
The System and the World	5
The System Within	8
Boosting Business Results.....	16
Consistency through Canonical Information.....	16
Simplicity through Multi-Aspect Separation	16
Efficiency through Minimalist Modeling	16
Summary.....	17
Figure 1 – Organization of Systems Modeling Information	4
Figure 2 – Sample Diagrams.....	5
Figure 3 – Context Diagram.....	6
Figure 4 – Use Case Diagram.....	7
Figure 5 – Sequence Diagram	8
Figure 6 – SysML Requirement.....	8
Figure 7 – Value Types	9
Figure 8 – Logical Architecture Diagram	10
Figure 9 – Block Definition Diagram Showing Physical Architecture.....	11
Figure 10 – Allocation of Logical Components to Physical Subsystems.....	11
Figure 11 – Block Definition Diagram of Physical Subsystem Details	12
Figure 12 – State Machine Diagram	13
Figure 13 – Activity Diagram	14
Figure 14 – Parametric Diagram	15
Figure 15 – Integrating Legacy Components	15

Abstract

Systems Engineering has always drawn heavily on detailed sets of drawings to convey high level systems concepts, system architecture and organization, and even detailed graphical depictions of key algorithms and other behavior. With the advent of the SysML language supporting a uniform and standards-based set of semantics for systems modeling and appropriate modeling technology to capture, organize, and transform these models, the state of practice is now pivoting from ad-hoc diagramming to a more rigorous and consistent modeling approach.

This paper outlines a solid foundation in core modeling skills for systems engineers building complex, high-performance systems with the SysML language. With a focus on modeling fundamentals, this paper outlines how teams can get started with proven techniques and a simple step-by-step approach, and deliver substantially improved business results like faster time to market, better quality and reduced cost. These proven techniques build a solid start for teams making the shift from ad-hoc diagramming to a more rigorous model-driven approach.

Motivation

The construction of complex systems can be a challenging endeavor, potentially facing a range of technical challenges - the specification of complex information, capturing and evolving intricate ideas, synchronizing multiple implementation disciplines and mastering key technologies. To be valuable the system must deliver key capabilities within required performance parameters. In many contexts there are additional pressures from competitive markets, constrained resources and timeframes. It is not easy to be successful, and many organizations are not.

While success in systems engineering is often considered in terms of technical results like system capabilities and performance, ultimately the *business results* indicate true systems engineering effectiveness:

- System and component quality
- Time to market
- Repeatability and manageability of the engineering process
- Engineering organization productivity

The effective use of modeling in systems engineering can deliver substantial improvements in business results, resulting in systems that are *better/faster/cheaper*. The first step on the path to effective modeling in systems engineering is to master the Fundamentals of Systems Modeling.

Models in Systems Engineering

Why Bother with Modeling?

Building complex, high performance systems is challenging. Not only is the breadth of system features and capabilities and all their detail a large body of information, but the challenges to achieving system

performance like reliability, response time, capacity, environmental resilience can be each genuinely difficult.

Systems Modeling helps in the construction of these types of systems by giving us an appropriate language, organizing the information we capture about our system, structuring the way we build and evolve this information into a model, and interconnecting various aspects this model. Effective Systems Modeling methods and their underlying technologies also help us rapidly and efficiently build the systems engineering work products based on our models, and helps establish and maintain effective collaboration between systems engineering and development engineering.

The Systems Modeling Approach

It is helpful to establish a vocabulary for our discussion. *We may define some terms here in a manner that is focused on our general topic, so these definitions are not intended to counter or supplant more complete or formal definitions from other contexts.*

This discussion is centered on the challenges of building complex systems:

- **System:** A set of components engineered/constructed and/or integrated, and delivered as a whole to deliver a specific set of user benefits.
- **System Component:** A discrete, homogeneous part of a system, typically developed in some degree of isolation from the rest of the system. This includes physical subsystems, logical components, and other separable elements.
- **Systems Engineering:** The engineering discipline for creating complex systems through their description, specification and architecture, and the coordination of the multidisciplinary development activities required for system component development/acquisition, integration and delivery.

The range of concepts from Drawings to Systems Models shows a progression of formality and rigor:

- **Drawing:** A graphical representation of a related set of technical concepts
- **Model:** A related set of abstractions, adhering to specific (predefined) semantics, where diagrams present views of this information.
- **Systems Modeling:** The disciplined construction of a model that serves as a central repository for the majority of systems engineering information for the system. This approach leverages multiple views into the model, aggregating information from a range of forms and sources, and leverages automation and standards to propagate this information as needed.

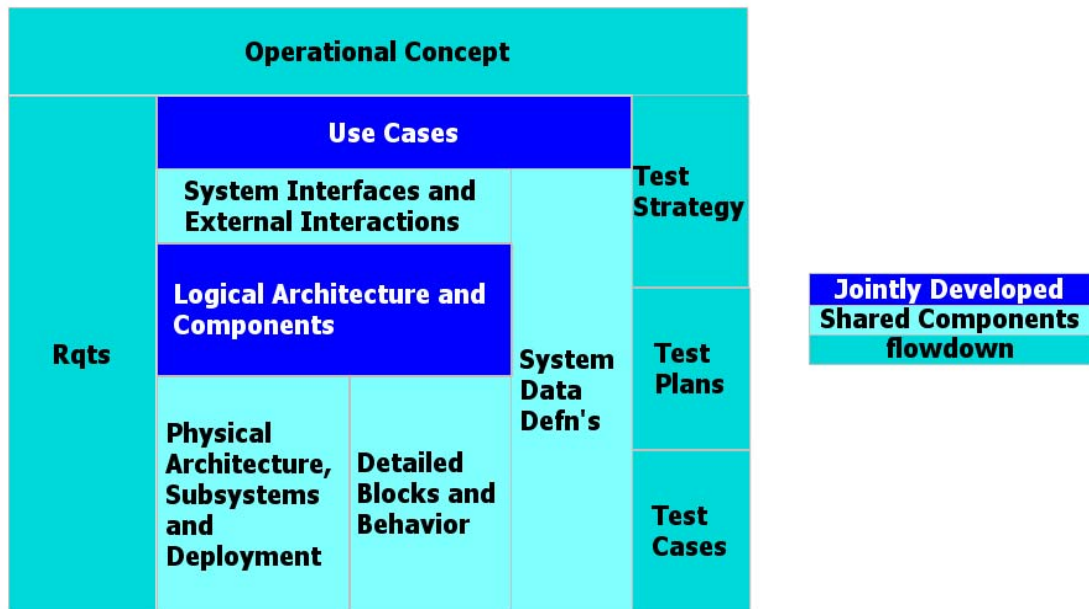


Figure 1 – Organization of Systems Modeling Information

SysML Overview

Many systems engineers use diagrams. Most diagrams are not parts of a model. A model is a coherent body of interrelated information. This coherence is a key element that distinguishes a model from just a set of diagrams. Diagrams of model elements are views the model information, making visible a specific aspect of the model and showing part of the information.

Useful models are expressed using known semantics in a standard language. The Systems Modeling Language (SysML) is a modeling language standard that defines semantic primitives and ways to visualize them at a range of levels of detail and granularity. Managed by the Object Management Group, SysML provides a graphical and semantic foundation for modeling, and is supported by a number of suppliers with drawing tools, training and mentoring in its use.

SysML provides primitives to capture:

- System Requirements
- Behavior
- Structure
- Integration with a broad range of engineering analysis

Developing complex systems generally involves a wide range of engineering and scientific concepts. SysML semantics can express abstractions in the realms of hardware, facilities, software, information, procedures and personnel.

Key SysML primitives include:

- Block
- Flow
- Association
- Requirement
- Parametric Constraint

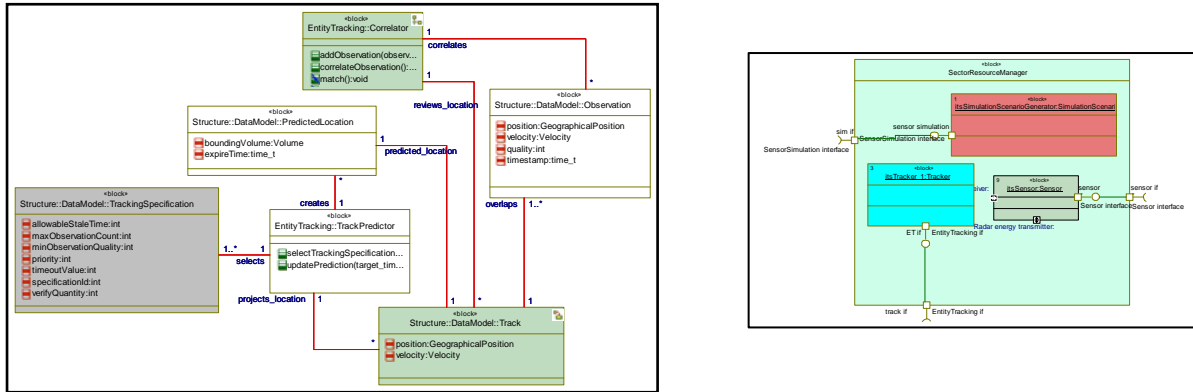


Figure 2 – Sample Diagrams

The System and the World

A system delivers value through its interactions with external elements. The Systems Model captures these interactions in varying levels of detail through the following aspects:

- Operational Concept
- System Context and Systems of Systems
- Use Cases
- System External Interfaces and Interaction Scenarios

The Operational Concept

Often developed well before the start of any systems modeling, the Operational Concept is a high-level summary of the system approach and capabilities, intended as an initial introduction and top-level overview. Often used as the initial introductory material to introduce people to the system, this is generally expressed in prose, pictures and informal diagrams. Depending on the sophistication of the audience for this information and the need to present detail, some popular modeling elements can be used such as use cases and sequence diagrams, but this is uncommon.

System Context Diagram

On the System Context Diagram the system is shown in the context of the things it interacts with, including:

- Other systems
- Users, operators, maintainers

- External components and subsystems

Shown in a Block Definition Diagram, the System Context establishes a boundary around the system.

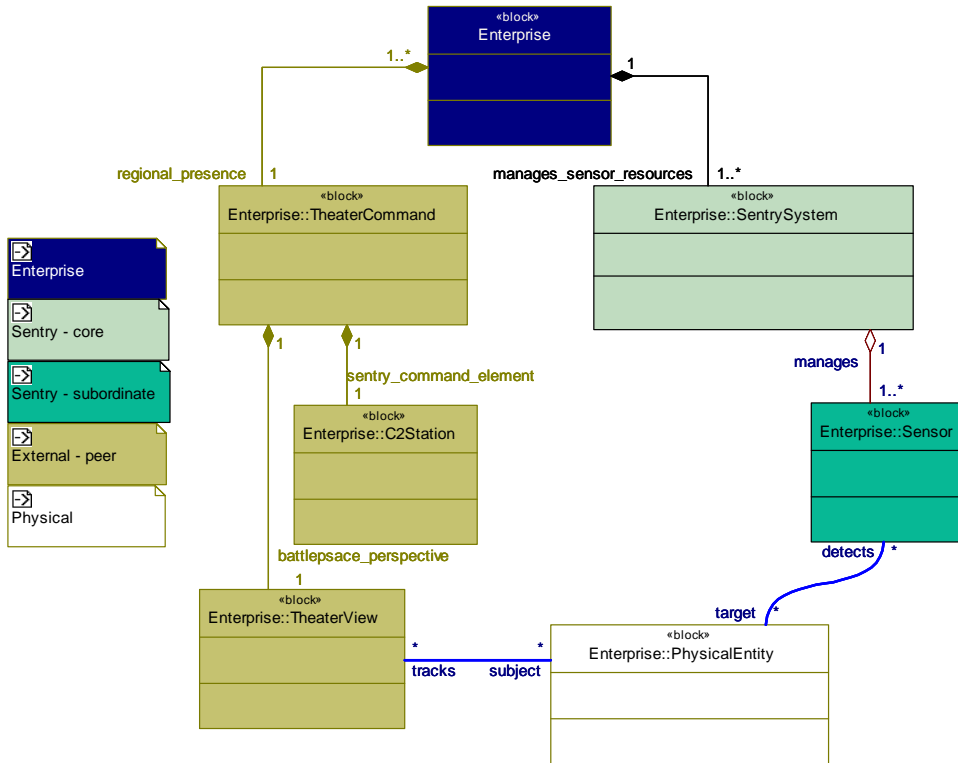


Figure 3 – Context Diagram

In a Systems of Systems context, our system can appear as a subsystem in a larger system. Establishing an opaque boundary around our system manages complexity.

Use Cases

These provide a way to group system capabilities with a focus on system interactions with external entities. They outline key capabilities and identify system boundaries, and also provide a convenient and consistent context for sets of system-level scenarios.

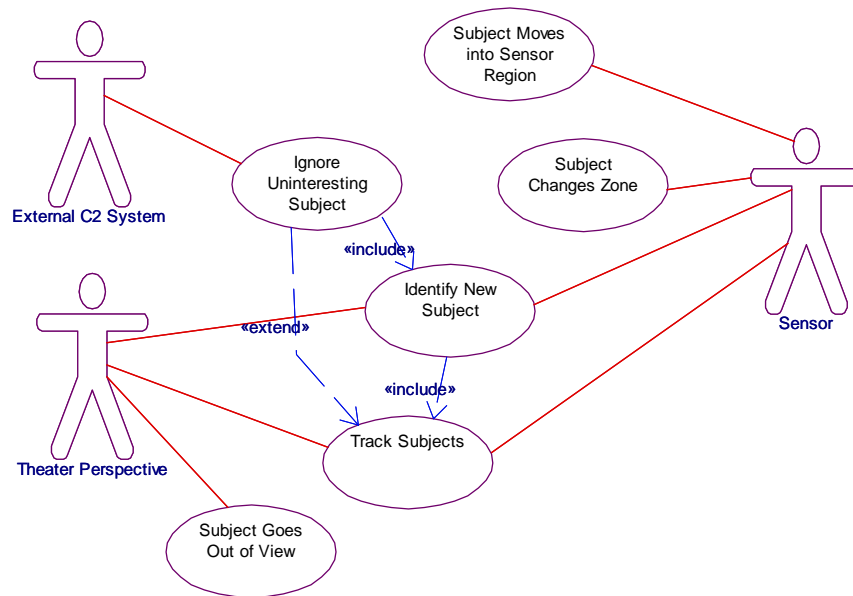


Figure 4 – Use Case Diagram

System External Interfaces and Interaction Scenarios

Illuminating key sets of interactions with external entities, Interaction Scenarios help establish and refine external system interfaces. Captured as a single logical path through a Use Case, these Interactions start to capture significant details and help validate completeness and consistency in this aspect of the system.

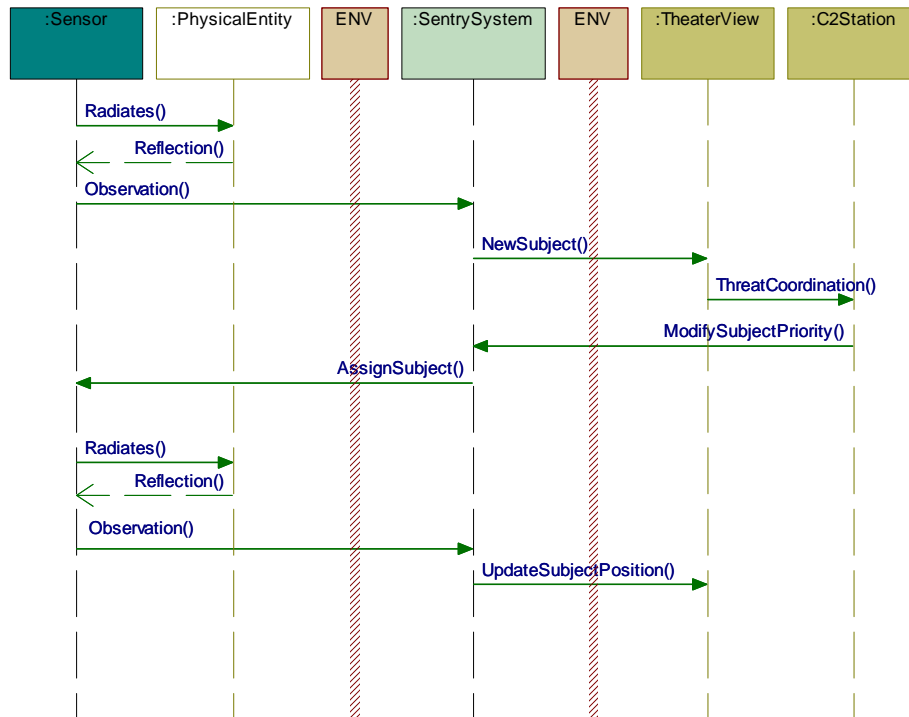


Figure 5 – Sequence Diagram

The System Within

Once the context is established, the focus of modeling turns to the system itself, to specify capabilities required, create system architecture, define major subsystems and components and detail critical behavior. The modeling activity is broken into various specific activities, each with it's own unique aspect or level of detail:

- Requirements Capture
- Value Types
- Logical Architecture and Deployment
- Physical Architecture and Subsystems
- Detailed Behavior
- Parametrics
- Legacy Components

Requirements Capture – Shalls

The specification of the capabilities and other characteristics required of the system is a critical aspect of systems engineering. Although the most efficient and effective capture of textual requirement "shalls" is not graphical, Systems Modeling does provide two key capabilities:

- The specification of requirements through non-textual model elements, using them in a normative manner.
- The inclusion of textual SysML Requirement elements directly, on their own Requirement Diagrams and with other model elements on other diagrams.

The specification of select textual requirements in SysML helps keep them

- Structured, organized
- Broken down to a detailed level, assigned a unique identifier

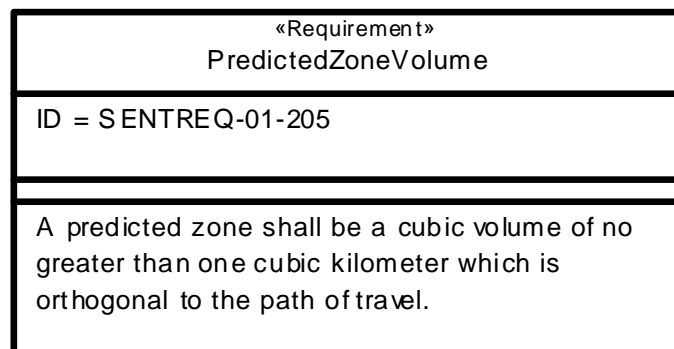


Figure 6 – SysML Requirement

Value Types

Various system elements abstract or carry data including block value properties and parameters. Value Types define these data primitives in a consistent way to ensure consistency and capture essential detail. Common uses of Value Types include:

- Voltage
- Password
- Torque
- Heat Content
- External messages and their fields

«Type»
sys_distance_t

«Type»
sys_linear_velocity_t

The value type provides a single definition for the information description and purpose of key value-based abstraction, including basic meaning, range of values and requirements on format and capacity.

«Type»
sys_time_t

Figure 7 – Value Types

Logical Architecture

Logical components are groupings of capabilities abstracted and organized based on the problem space itself. Logical Architecture organizes logical components based on overall system capability responsibilities and the delegation of lower-level responsibilities to subordinate components. This results in a Logical Architecture Diagram that shows the fundamental relationships between components relative to the mission of the system itself.

These subject-matter based capability groupings are independent of physical architecture of the system or its implementation technologies. Individual logical components, or even subsets of the blocks that constitute them can be allocated to physical subsystems using a range of patterns and automation.

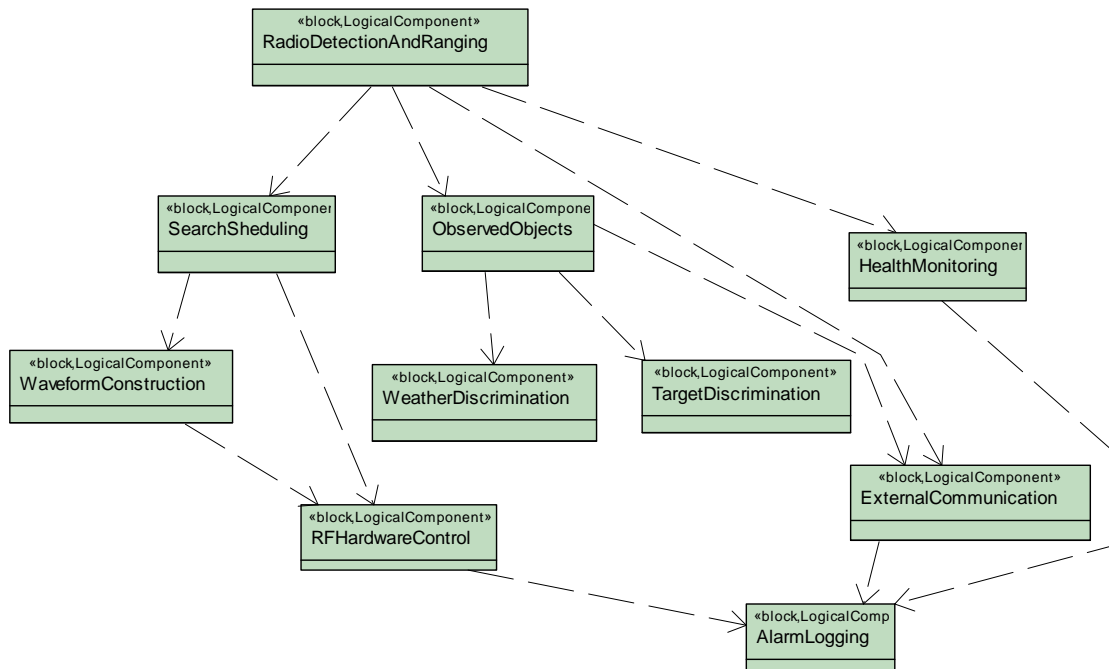


Figure 8 – Logical Architecture Diagram

Logical Architecture is highly durable and flexible because it is independent of physical drivers and other constraints that force Physical Architecture and implementation strategies. This is why it is preferred to create the primary architecture for the system from the Logical perspective (or Aspect), and address the physical subsystems and other aspects of the system in later phases.

Physical Architecture and Subsystems

Physical subsystems are abstracted for the separable physical components or pieces of the system. Although not the first step in determining overall system architecture, Physical Architecture nonetheless is an essential aspect of System Modeling, facing head-on the very challenges that drive the high-performance aspects of the system. In fact the separation of the "feature" capabilities in the logical perspective from the physical and performance capabilities addressed in the physical perspective simplify each, and permit the more direct and effective addressing of each challenge. Maintaining the proper separation of concerns between Logical and Physical Architecture enhances overall system durability and flexibility and reduces end-to-end system development costs.

Physical architecture specifies the system strategy for subsystems in terms of:

- Organization
- Interaction
- Interconnection

Subsystems often have multiple disciplines contributing to their total capability development including Electrical, Mechanical and Software. The physical subsystems show both content and interfaces from these perspectives.

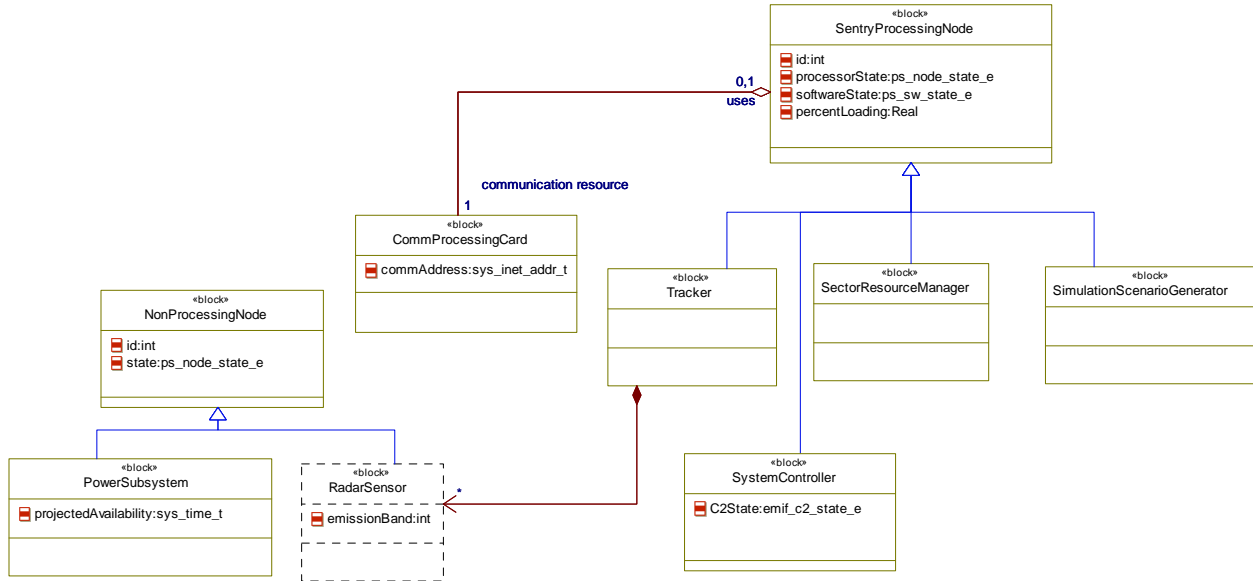


Figure 9 – Block Definition Diagram Showing Physical Architecture

Deployment of Logical Components

Logical components are abstracted separately from the physical realm, and need to be deployed to physical subsystems in order to deliver their capabilities. Physical Subsystems provide an execution/deployment context for Logical Components, and the SysML Allocate relationship capture this strategy.

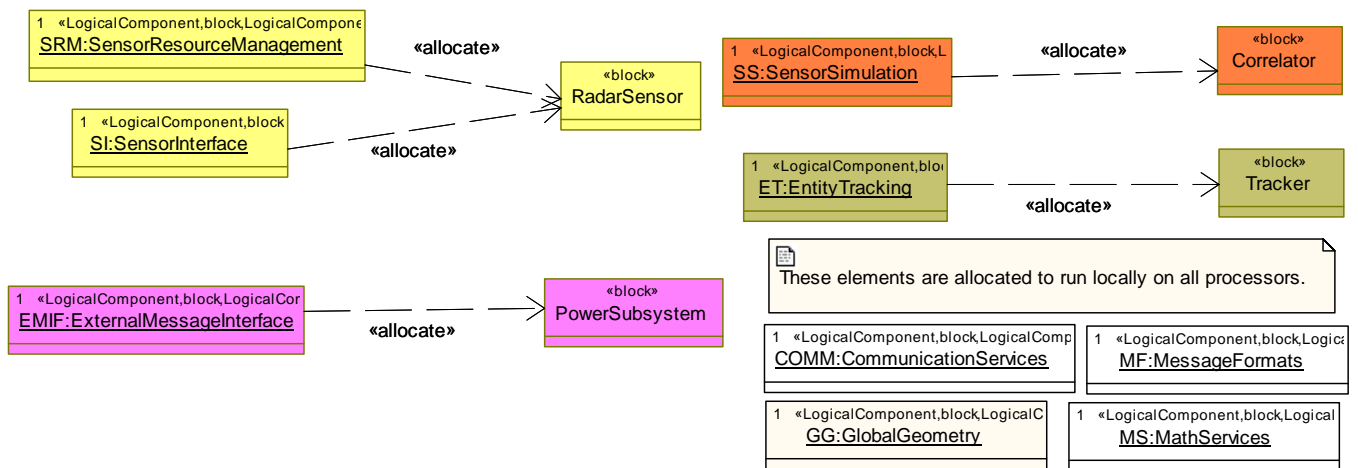


Figure 10 – Allocation of Logical Components to Physical Subsystems

Detailed Blocks

Hierarchical decomposition of blocks and detailed behavioral abstractions:

- Capture component and subsystem structure
- Detail the structure of important data
- Illuminate key concepts and algorithms
- Express requirements-based processes, interactions
- Provide a canonical specification of required behavior too intricate for textual shalls

Detailed blocks show the internal structural details for logical component and physical subsystem structure. Nested decomposition of blocks into further details blocks allows the management of complexity and capture of essential detail.

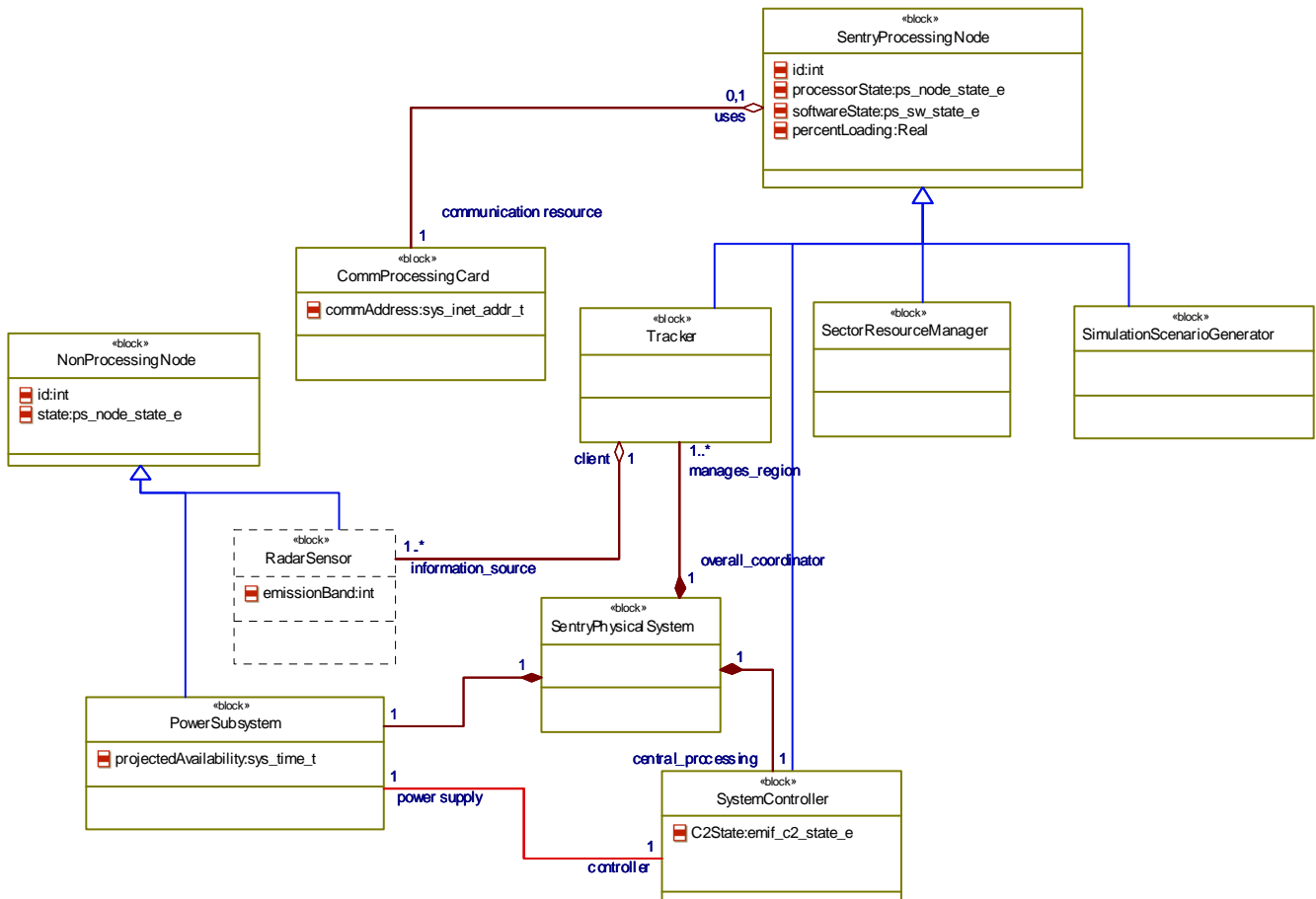


Figure 11 – Block Definition Diagram of Physical Subsystem Details

Detailed Behavior

Textual prose requirements can be flat, bland and expansive. Model-based behavior specification can communicate better using graphical forms and in a standard and language with known semantics. With a higher level of abstraction than software programming languages or other developmental languages, SysML behavioral elements can concisely capture complex details.

State machines are long-standing tools in expressing key concepts about detailed system element behaviors. Abstracting state-dependent behavior where the block instance reacts differently in different states, the State Machine Diagram shows the block's pattern of event → response.

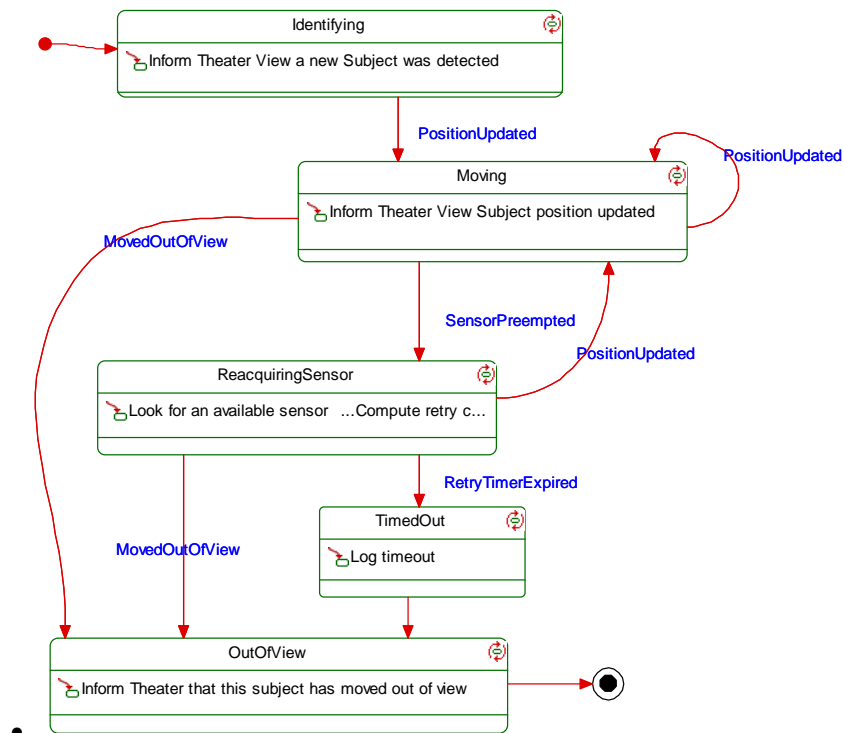


Figure 12 – State Machine Diagram

An Activity Diagram can express the details of an action or process, showing the flow of data and control graphically. It can show allocation of processing to subsystems. Generally an Activity Diagram shows the behavior of a single action such as a Block operation or a State Action, but it can also be applied to show summary-level coordination of processing. For physical-based actions Activity Diagrams can also show Item Flows that can abstract the flow of real-world elements like discrete things or energy or force.

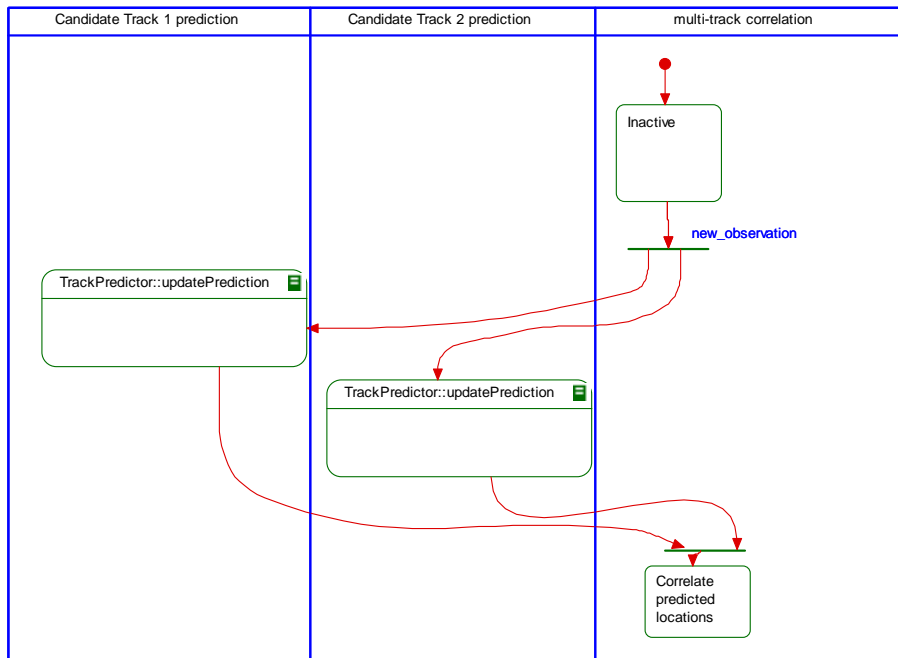


Figure 13 – Activity Diagram

Parametrics

Parametrics express mathematical relationships between system data elements (values) in an equation-based form. This can be used to express values used in requirements, and capture key equations used in detailed behavior. They can also be used to explore specific constraint/capability questions by providing inputs to some elements and resolving the equation system to produce the desired outputs.

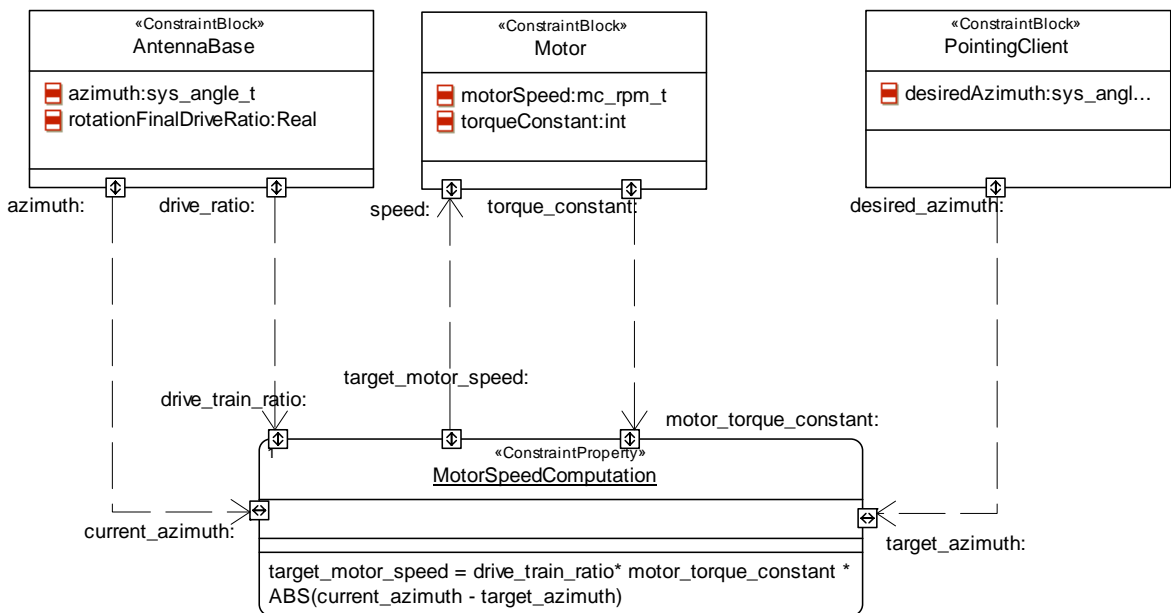


Figure 14 – Parametric Diagram

Legacy Components

In many systems engineering efforts large sets of capability have already been developed and to be used (largely) as-is. These can include physical subsystems or logical components, and it can be a substantial challenge to integrate these existing elements into new system architecture.

The abstraction of logical components and physical subsystems to represent these elements, and the detailing of these component interfaces both for newly developed components and legacy elements is essential to develop and refine the proper connection strategies.

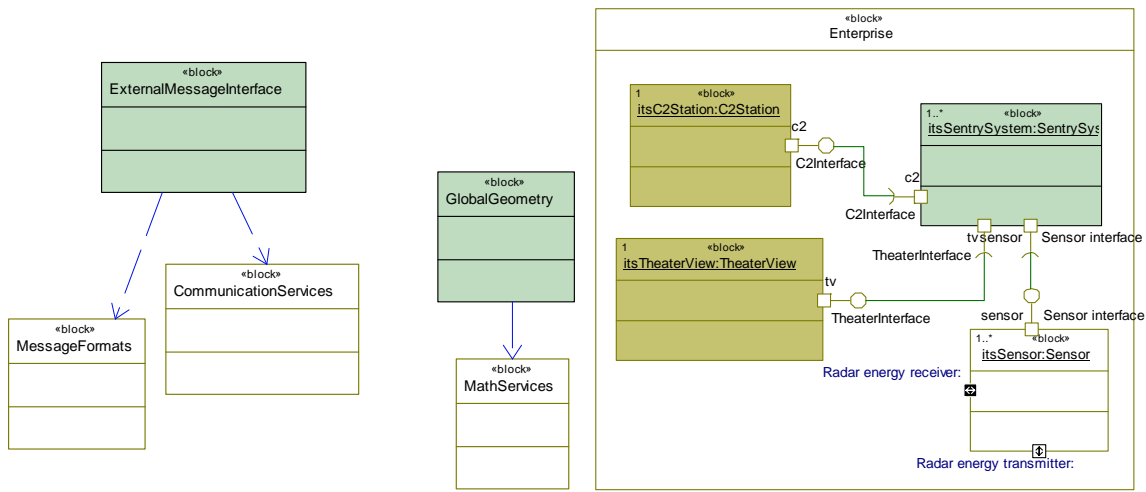


Figure 15 – Integrating Legacy Components

Boosting Business Results

In addition to engineering a functional system, this approach also has specific characteristics that deliver substantial gains in business results, including these critical factors:

- **Schedule Performance** – Reduce time to market
- **Productivity** – Deliver more capability with the same resources
- **Overall Product Quality** – Deliver the product with higher overall quality and concept-to-delivery fidelity

Techniques, guidelines and automation were specifically developed to facilitate the pursuit of these three virtues:

- **Consistency** through Canonical Information
- **Simplicity** through Multi-Aspect Separation
- **Efficiency** through Minimalist Modeling

Consistency through Canonical Information

Also known as "One Fact in One Place", this virtue eschews the cloning or copying of Systems Modeling information and instead promotes the capture of information in its most natural form. Once abstracted in the form that best supports error-free capture and maintenance, appropriate transformation and transport technologies are used to make the information available wherever it is needed and in the form it is needed. While this was always a "good idea" when convenient, modern modeling technologies and frameworks have erased the old practicality barriers and elevated this to a cardinal rule.

Simplicity through Multi-Aspect Separation

All Systems Modeling approaches apply some form of separation and organization of abstractions to address complexity. Most apply a single axis of separation through simple decomposition. While a hierarchy of nested elements is certainly useful, when used as the only aspect of separation the hierarchy can re-introduce complexity and in fact obscure the actual subject-matter of the system itself. By applying multiple aspects of separation, the problem space can be cut in a number of ways, each suited to best expose key aspects of the system. Two common aspects are the Logical Component Architecture with a feature/capability focus, and a separate Physical Subsystem Architecture, each is fundamentally more simple because of the other. Many aspects can be applied to take best advantage of the nature of a specific problem space. Largely a combination of disciplined architectural and perspective techniques, this virtue brings a quantum boost in simplicity delivering the largest business gains.

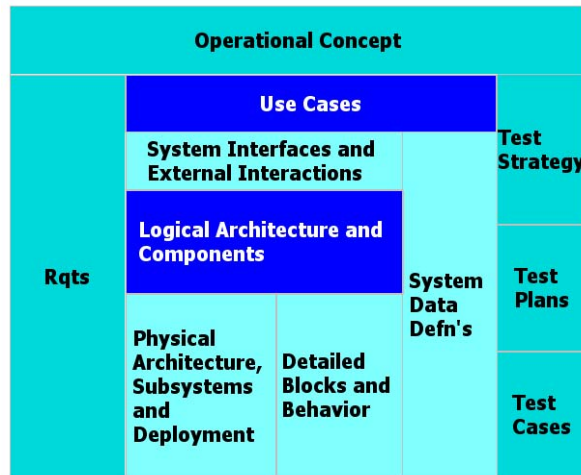
Efficiency through Minimalist Modeling

Embracing the adage that "less is more", this brutally pragmatic discipline builds a straightforward inventory of all Systems Modeling stakeholders and catalogs all work products they consume. From there a modeling plan is built to construct only the model abstractions and elements needed to deliver these work products. Of course some modeling is needed simply to explore concepts and build understanding, but even this "consumer" must catalog their needs in the harsh light of day. Not only does this provide an excellent bill of materials for what the completed systems model needs to contain, but this also brings modeling efforts into sharp focus and avoid unfettered wanderings and stream-of-consciousness modeling that characterize poor schedule performance.

Summary

Engineering complex systems must ensure concept-to-delivery fidelity (build what you say), manage complexity, coordinate multi-disciplinary teams and balance cost, calendar, capability. Systems modeling meets these challenges:

- Build a single, comprehensive body of information
- Apply coherent and integrated set of activities
- Use an industry standard language and automation technologies
- Apply simple techniques proven to meet real-world challenges



Pathfinder Solutions

Our mission is to generate immediate, bottom line, business results in customer organizations through the adoption of model driven approaches and technology for systems engineering, software architecture and development..

- Leading provider of Systems Modeling and MDD/MDA technology, training and expertise:
- Formed in 1995 to bring more complete solutions to customers.
- We provide training, mentoring/consulting, tools, and components.

With deep experience with advanced modeling technologies we are instructors, mentors, and engineers and work directly with your systems engineering teams. As business partners with leading SysML tool vendors, we are also resellers who can build complete solutions for your project.

We can help you bring the concepts in this paper to life in you organization so Systems Modeling can deliver substantial gains in systems quality, engineering effectiveness and overall productivity. Visit us at www.PathfinderSystemsModeling.com, then get in touch - info@PathfinderSystemsModeling.com and +1 508-568-0068.

references

"OMG Systems Modeling Language (OMG SysML™) Specification", OMG document ptc/06-05-04, May 4, 2006

Sanford Friedenthal, Alan Moore, Rick Steiner, "OMG Systems Modeling Language Tutorial", http://www.omgsysml.org/SysML-Tutorial-Baseline-to-INCOSE-060524-low_res.pdf , July 11, 2006

Pathfinder Solutions whitepapers (*various*) , <http://www.pathfindermda.com/resources/whitepapers.php>

"Effective Model Driven Development", Pathfinder Solutions training