INCOSE SG SYSTEMS ENGINEERING DAY | SEPTEMBER 27, 2024

MBSE – PERSPECTIVES ON THE PAST, THE PRESENT AND THE FUTURE

Presented by Robert J. Halligan FIE Aust CPEng IntPE(Aus)

DASSAULT SYSTEMES IBM Capella VIETSOL BLUEKEI SOLUTIONS INCOSE SINGAPORE

rhalligan@ppi-int.com

ROBERT J. HALLIGAN

FIE Aust CPEng IntPE(Aus)

CAREER HIGHLIGHTS

- **Managing Director** | Project Performance International
- **Content Contributor** | EIA/IS-632, EIA 632, IEEE 1220, ISO/IEC 15288 SE standards
- **Past INCOSE Head of Delegation** | ISO/IEC SC7 on Software and Systems Engineering
- **Past Member** | INCOSE Board of Directors
- **Past President** | Systems Engineering Society of Australia
- **Consultant/Training** | BAE Systems, Mitsubishi, Airbus, Thales, Raytheon, General Electric, Boeing, Lockheed, General Dynamics, OHB, Nokia, AREVA, BHP Billiton, Rio Tinto, Embraer, Halliburton and many other leading enterprises on six continents

Model:

"*A simplified representation of some aspect of the real world*"

Source: EIA/IS-731.1 Document2 - Systems Engineering Capability Model

MBSE:

"*Model-based systems engineering (MBSE) is the formalized application of modeling to support system requirements, design, analysis, verification and validation activities beginning in the conceptual design phase and continuing throughout development and later life cycle phases*"

Source: INCOSE SE Vision 2020

3000 BC to date: mathematics

1954: MBSE described in a Rand Corporation report

1977: development by TRW of SREM, initially for software engineering, applied to systems engineering

1991: commercial release of RDD language derived from SREM, and associated RDD software tool

1991: Object Process Methodology (OPM) first conceived by Dov Dori

1993: commercial release of CORE tool and associated language, also derived from SREM

1996: release of OMG UML

2001: commencement of INCOSE/OMG SysML project

2006: release of OMG Business Process Modelling Notation (BPMN)

2007: release of OMG SysML 1.0

2015: publication of ISO/PAS 19450:2015 OPM standard

2025: anticipated release of OMG SysML 2.0

"Systems Engineering"

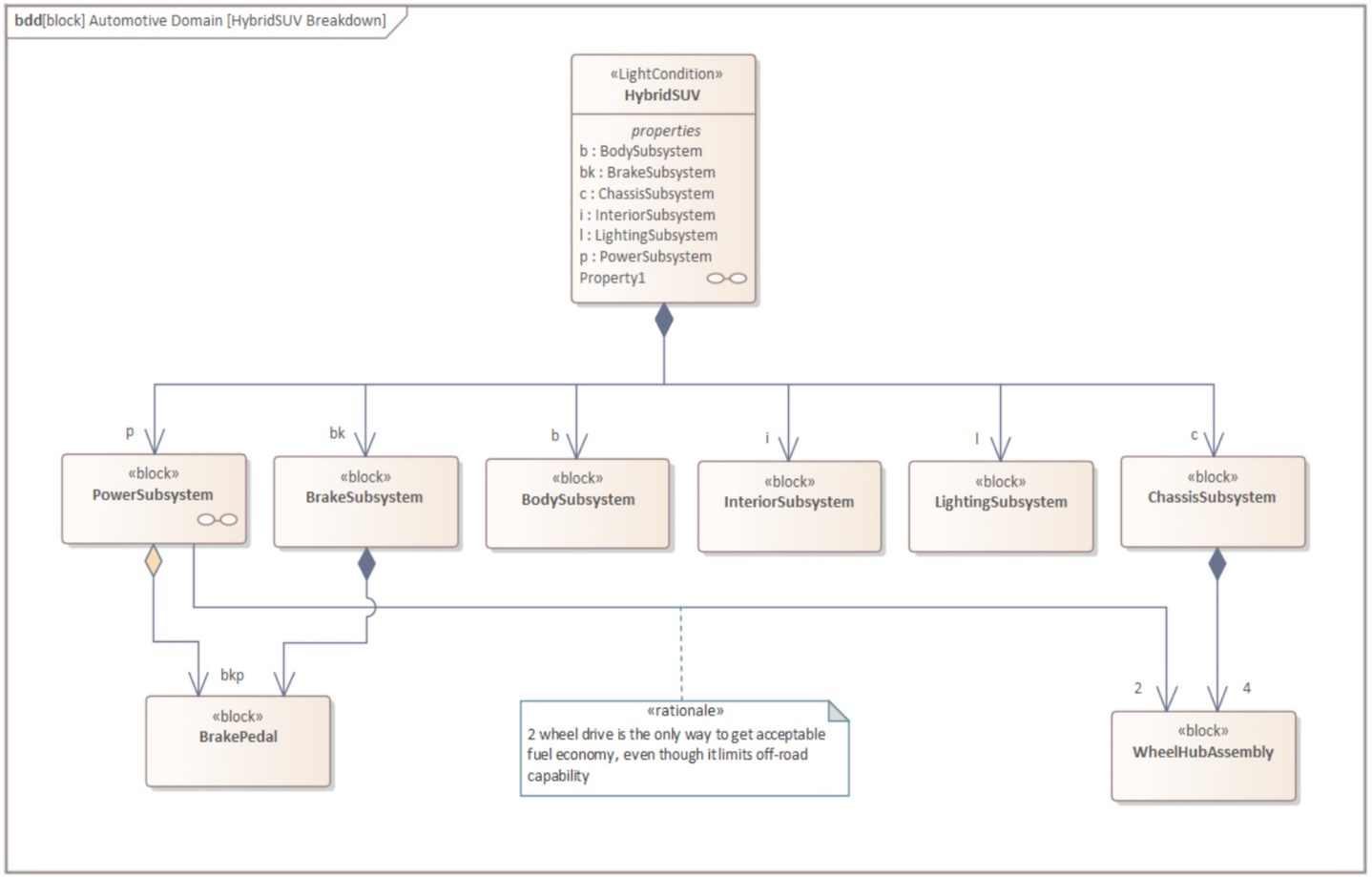
Alexander W. Boldyreff

1954 Rand Corporation paper available in hard copy.

- Currently the most widely used MBSE language, but with strong proprietary competitors
- Was supposed to be developed against RFP requirements (see the PPI Systems Engineering Goldmine)
- Open source for distribution and use
- Conceived by INCOSE in 2001
- A joint development by OMG and INCOSE
- Released as OMG v1.0 in 2007
- Currently at OMG v1.7 released in 2022

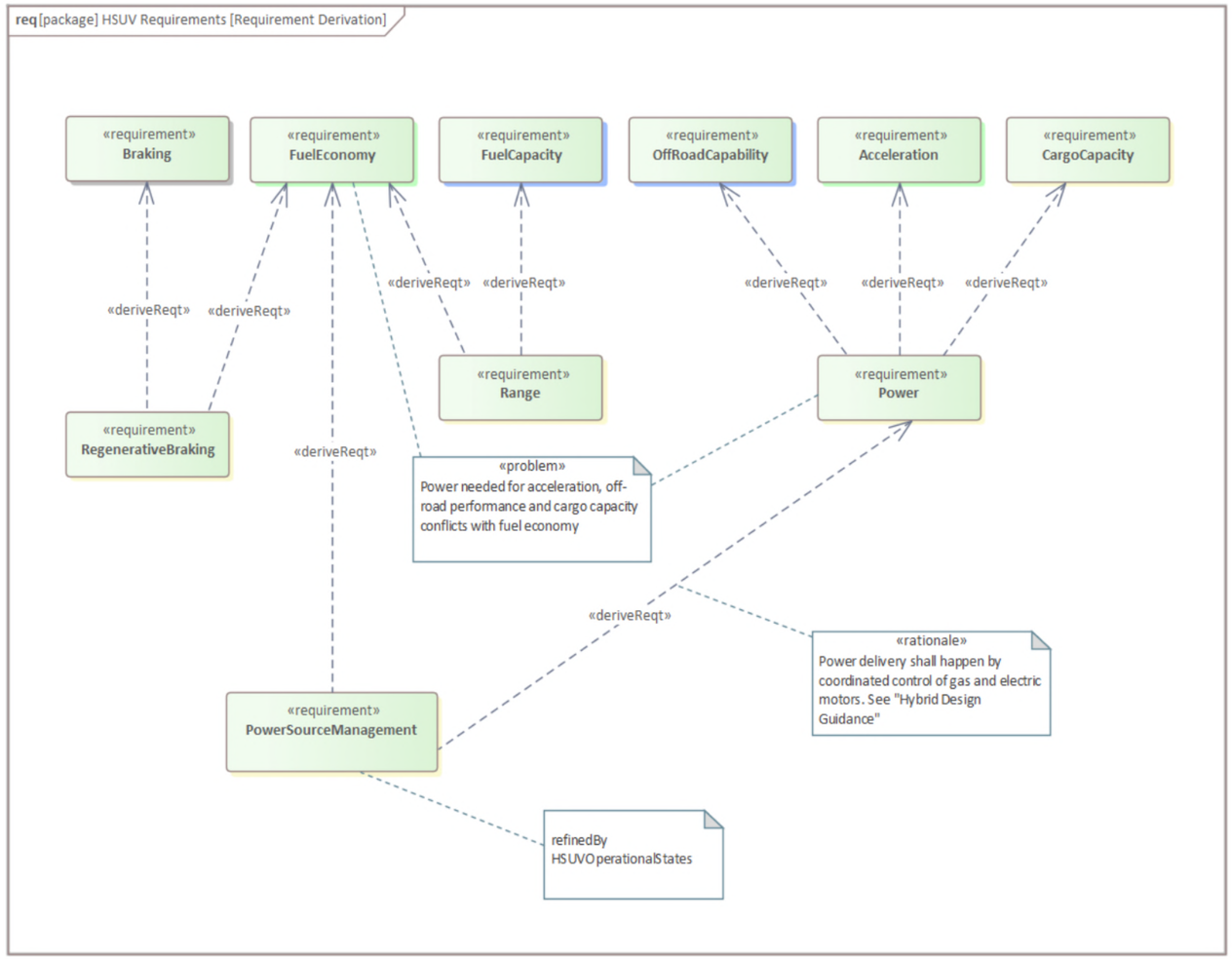
- Block definition diagram
- Internal block diagram
- Package diagram
- Use case diagram
- Requirement Diagram
- Activity diagram
- Sequence diagram
- State machine diagram
- Parametric diagram

The Block Definition diagram is the most widely-used of the SysML diagrams; it is used to model Blocks, their relationships to other elements (including other Blocks) and their features in the form of Properties, Operations and Receptions. Blocks are modular units of system description and provide a way of modeling systems as a graph or tree of modular units. Other elements, such as ConstraintBlocks and Properties, can also appear on the diagram and help describe the system being modeled.



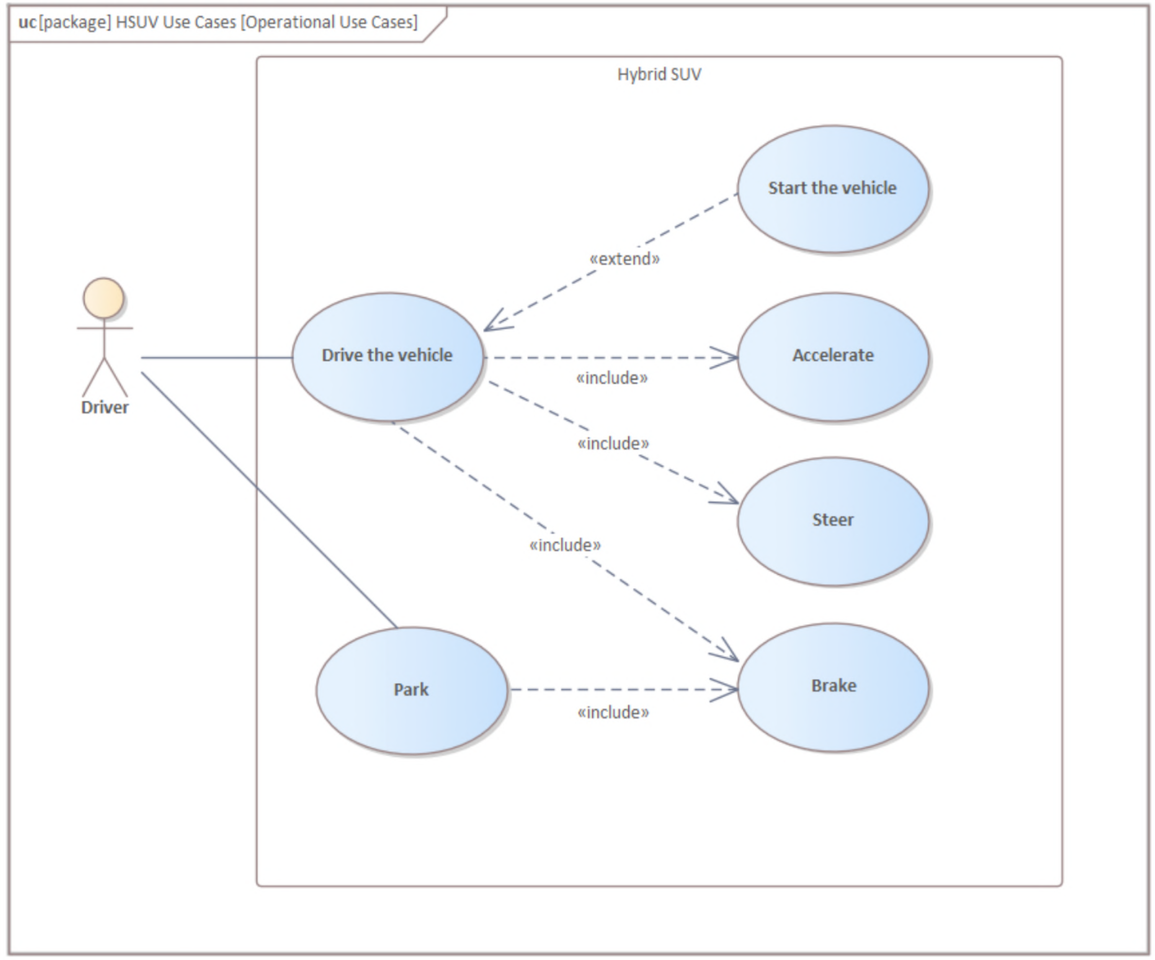
Source: Sparx Systems

The Requirements diagram is used to create and view Requirements and their relationships to other elements, including other Requirements. Requirements can be specified at any level, from strategic enterprise or business requirements through stakeholder requirements down to low-level engineering and even software and transition requirements.



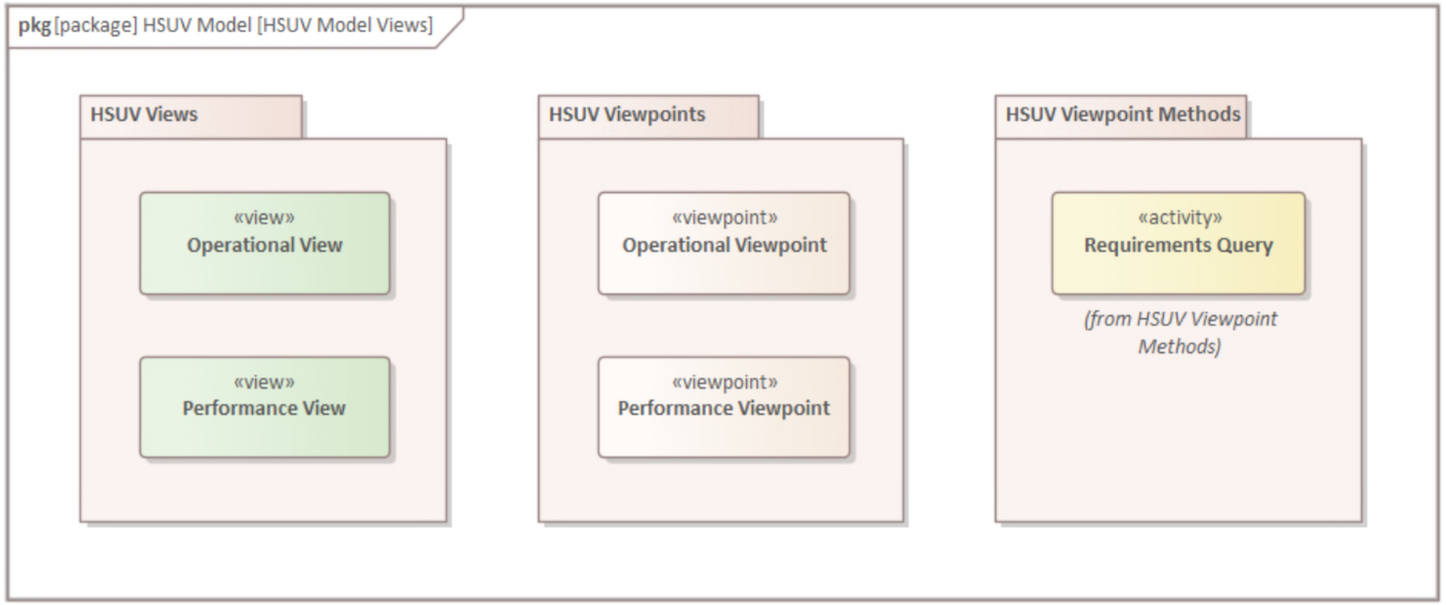
Source: Sparx Systems

The Use Case diagram is used to define and view Use Cases and the Actors that derive value from the system. The Use Case diagram describes the relationship between the Actors and the Use Cases, enclosing the Use Case within a Boundary that defines the border of the system; the Actors, by definition, lie outside the Boundary. While the Use Case diagram can appear simplistic, it is a powerful communication device that describes the value or goals that external roles obtain from interacting with the system. Each Use Case can be detailed with descriptions, constraints and any number of scenarios that contain sets of steps performed alternately by Actor and system to achieve the desired goal.



Source: Sparx Systems

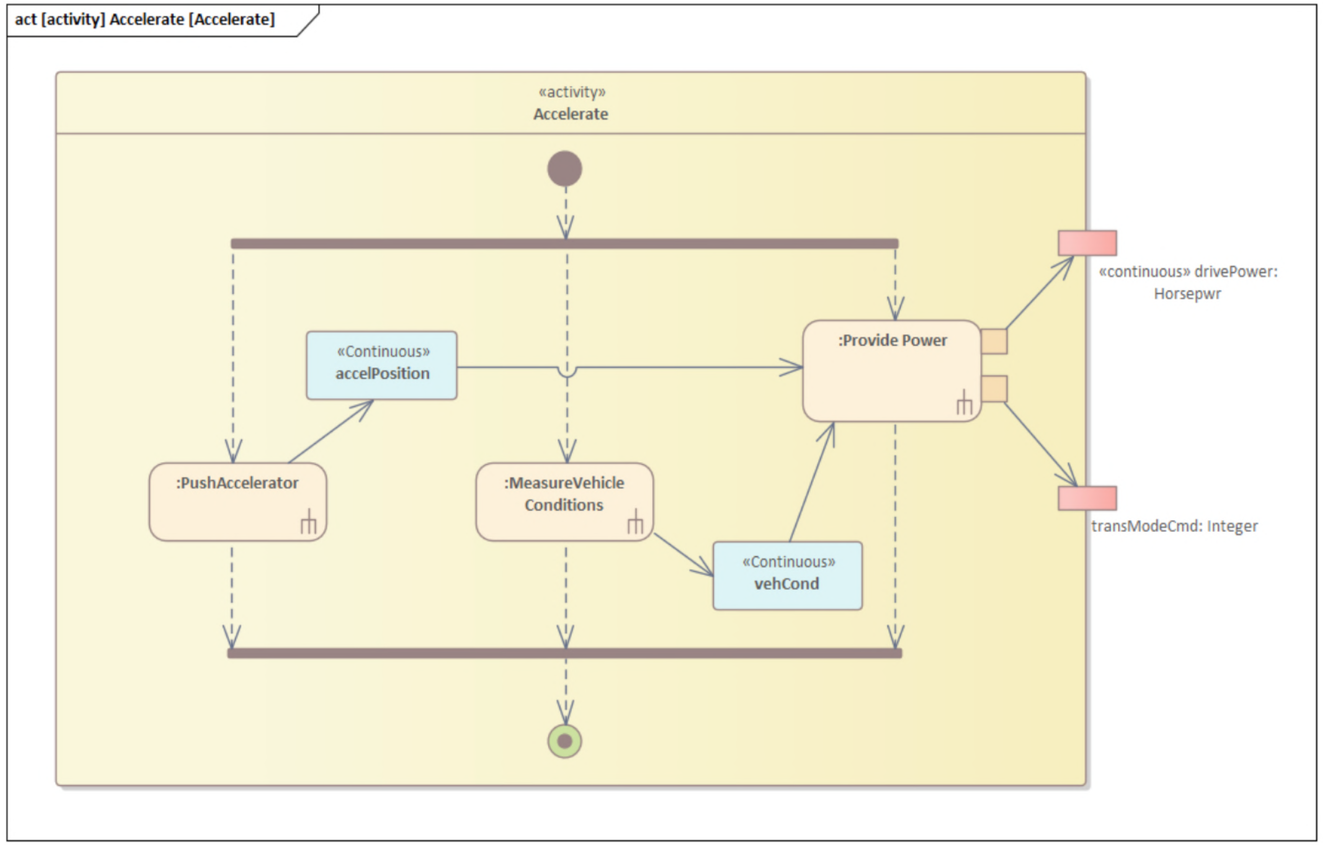
The SysML Package diagram is used to define or view the Packages that provide the fundamental organization of the repository. These can include name-spaces and their sub-Packages and other less formally defined groups of elements. The Packages that appear in diagrams can also be viewed in the **Browser window** and their hierarchy can be navigated by expanding and collapsing the tree.



Source: Sparx Systems

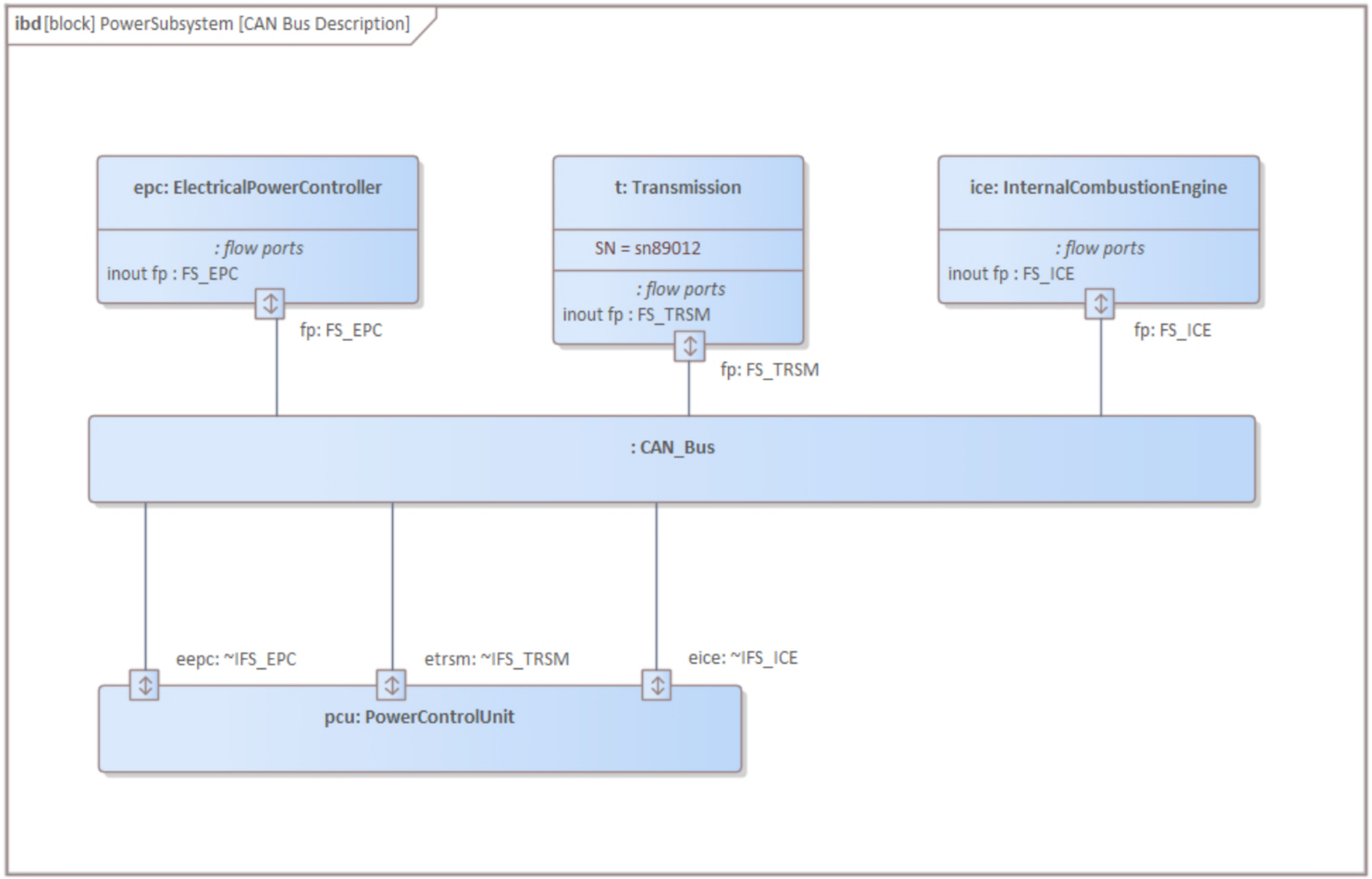
The Activity diagram is the most important behavior diagram and can be used to model flow (discrete or continuous) based behavior where inputs are converted to outputs by traversing a sequence of actions that perform work on the items. They are analogous to the common flow chart diagram but have more sophisticated semantics and also allow Activities and Actions to be related to elements such as Blocks, Requirements and Use Cases.

The Actions that appear on the Activity diagrams can contain input or output pins that represent the interaction points where inputs are fed into an action and outputs are emitted.



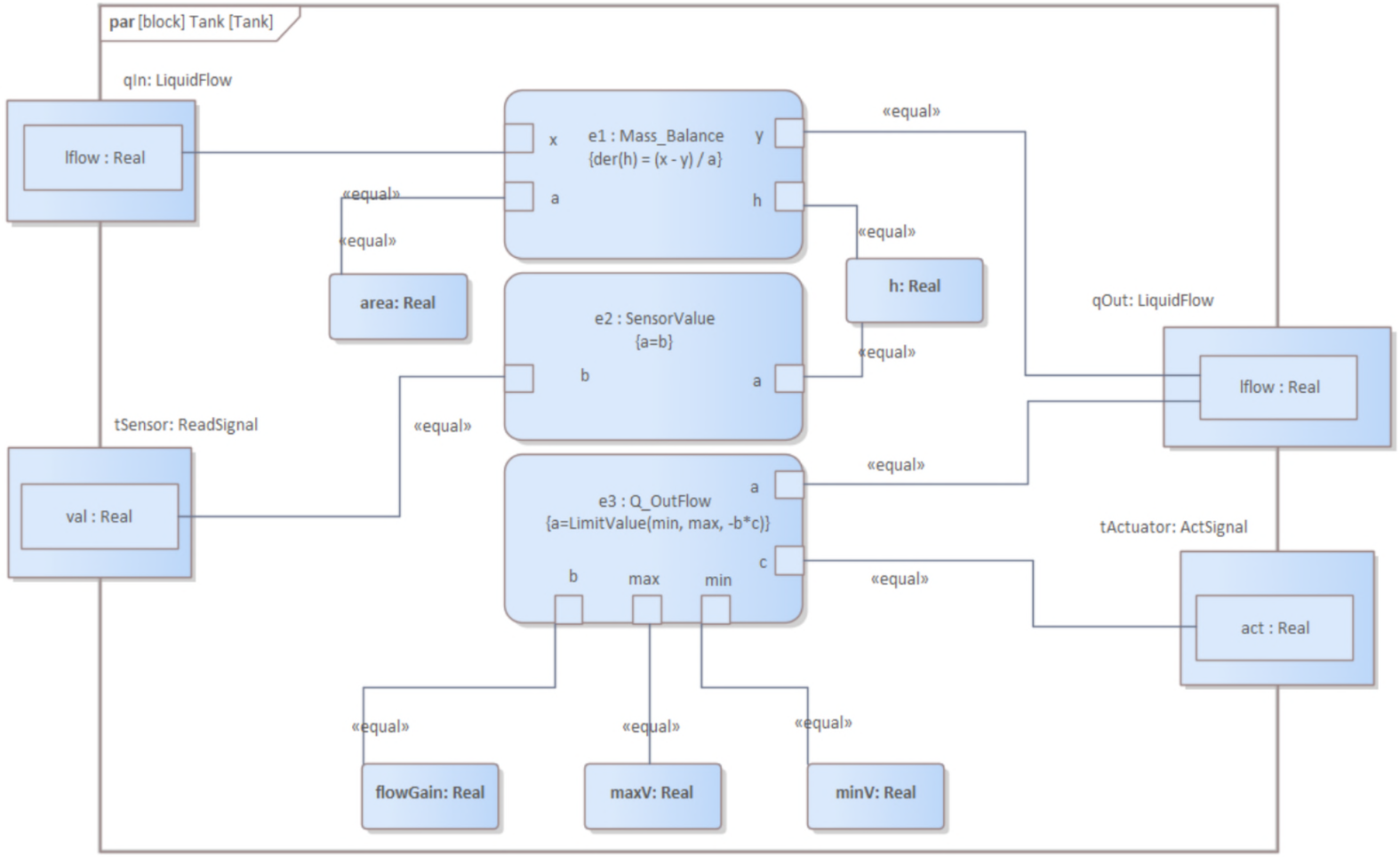
Source: Sparx Systems

The Internal Block diagram provides a way of viewing the composition of a block using part properties connected together using ports and connectors. The diagram is useful for showing a Block's (represented by the diagram frame) composition and the flow of inputs and outputs between the various parts that make up the block, when required the direction of flow can be indicated on the connectors.



Source: Sparx Systems

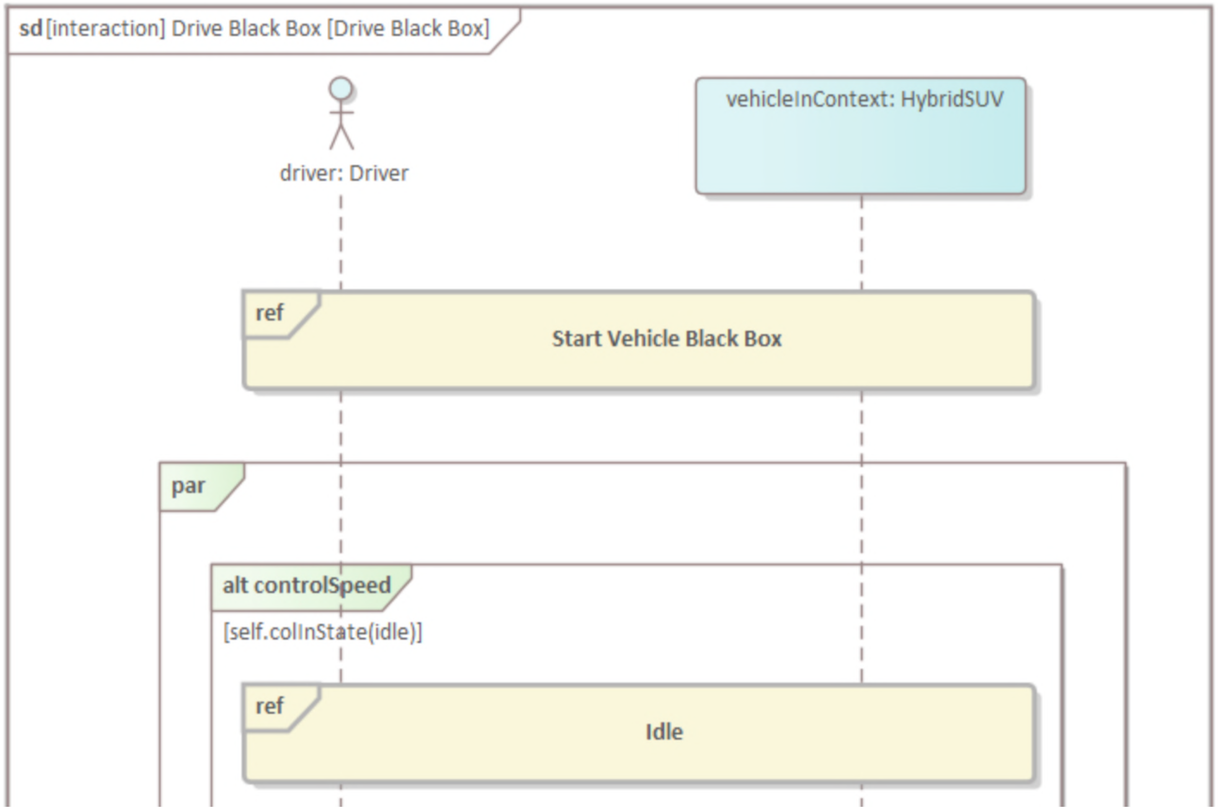
The SysML Parametric Diagram is a type of **Internal Block Diagram** (with some restrictions) that is used to model equation with parameters. They are an important tool that can be used to describe equations and their parameters and are important when performing trade off analysis and assessing design alternatives as they can be combined into systems of equations and related to Measures of Effectiveness MOEs.



Source: Sparx Systems

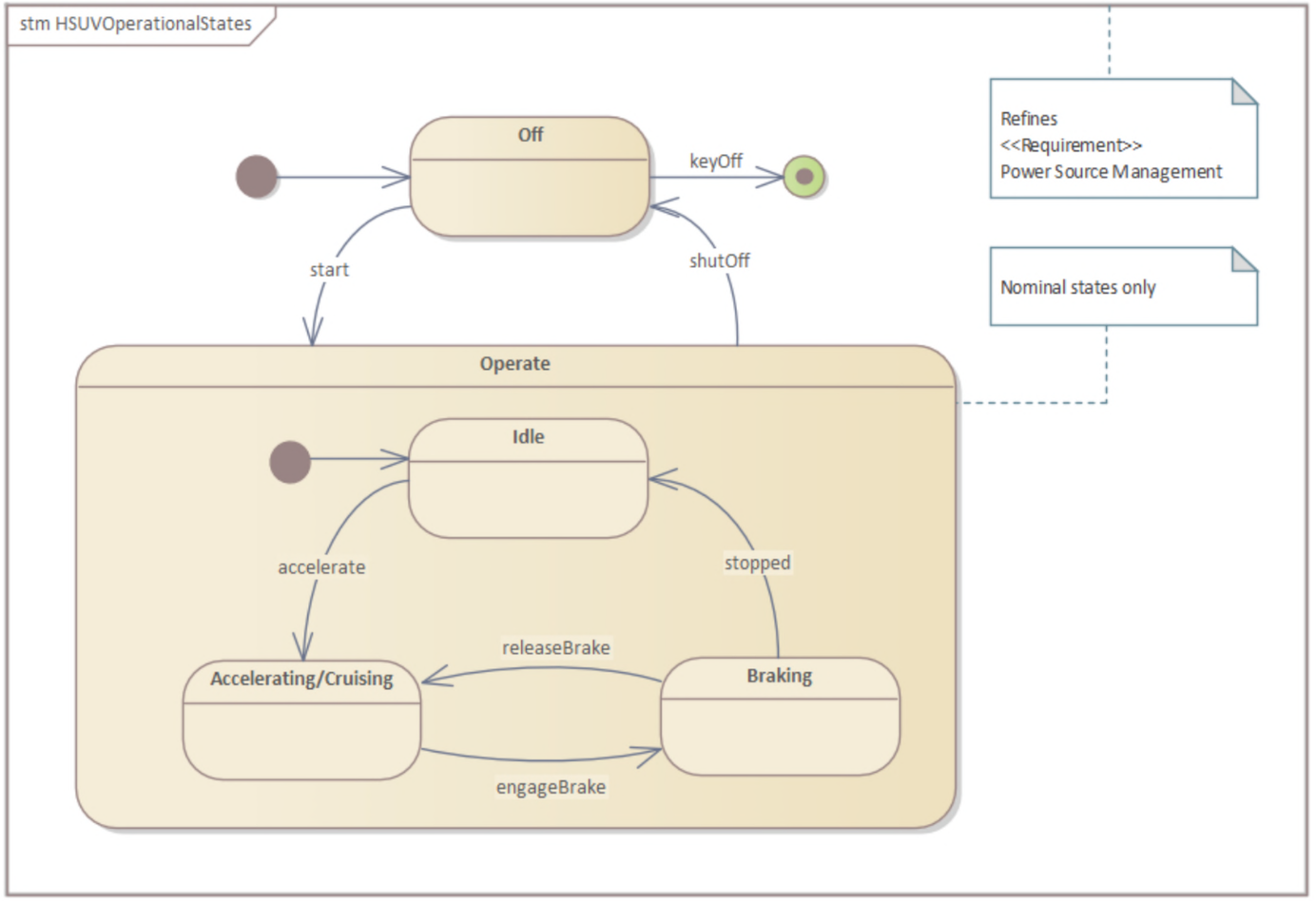
A Sequence diagram is a type of Interaction diagram that shows the time ordered interaction between objects. The diagram has two axes; the vertical axis represents time and the horizontal axis represents the objects that take part in the interaction, typically ordered in a way that best illuminates the interaction. These diagrams have their origin in the modeling of software interactions, but they can be used with systems engineering to be prescriptive of how elements (such as Blocks) should interact, or descriptive in showing how they do interact, in practice.

This Sequence diagram shows the interactions and sequence of message flows between a driver and a vehicle. The diagram expresses the necessary interactions for the 'Drive the Vehicle' Use Case. The interaction is owned by the 'AutomotiveDomain' Block.



Source: Sparx Systems

A **StateMachine diagram** is a powerful way of presenting information about the lifetime of a system element such as a Block. It can be used to describe the important conditions (States) that occur in an entity's lifetime or cycles. Typically only entities that have important stages in their lifetime are modeled with StateMachine diagrams. The entity is said to transition from one State to another as specified by the StateMachine. Triggers and Events can be described that allow the state transition to occur and Guards can be defined that restrict the change of state. Each State can define the behaviors that occur on entry, during and exit from the State.



Source: Sparx Systems

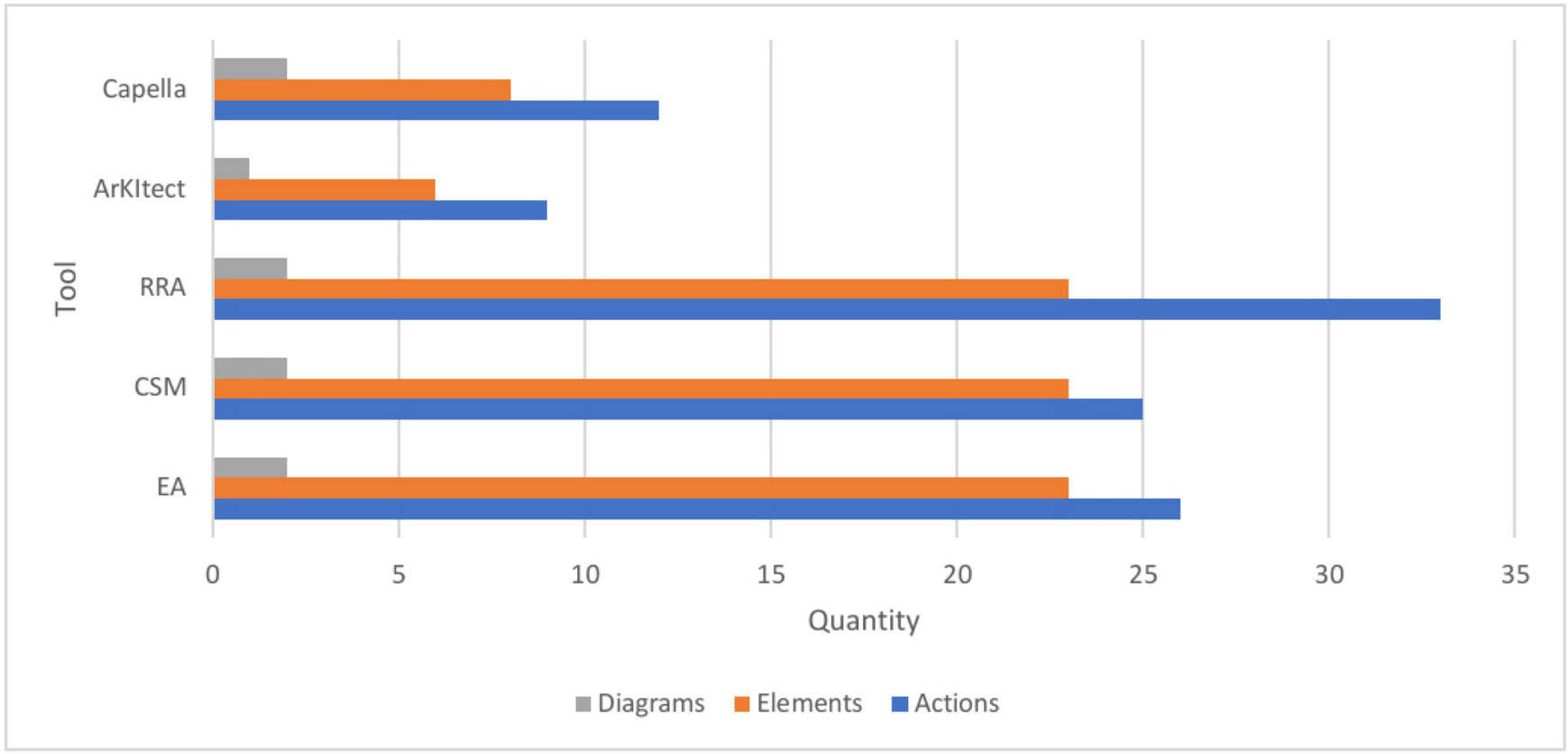
- Standardized and open source
- Based on UML, therefore assisting software engineers who are familiar with UML
- Many proprietary and some open source tools available
- Powerful backing of the Object Management Group (OMG)

SysML 1.x DISADVANTAGES

- Departs greatly from the (sound) requirements, reflecting need, against which SysML was supposed to be developed.
- In practice, no model interchange between SysML tools, a key need.
- Many technical issues that prevent efficient use.
- Requires very large effort to learn, especially without prior UML experience – in practice limits routine use to a few experts.
- Inefficient in use – many steps to create and show simple relationships.

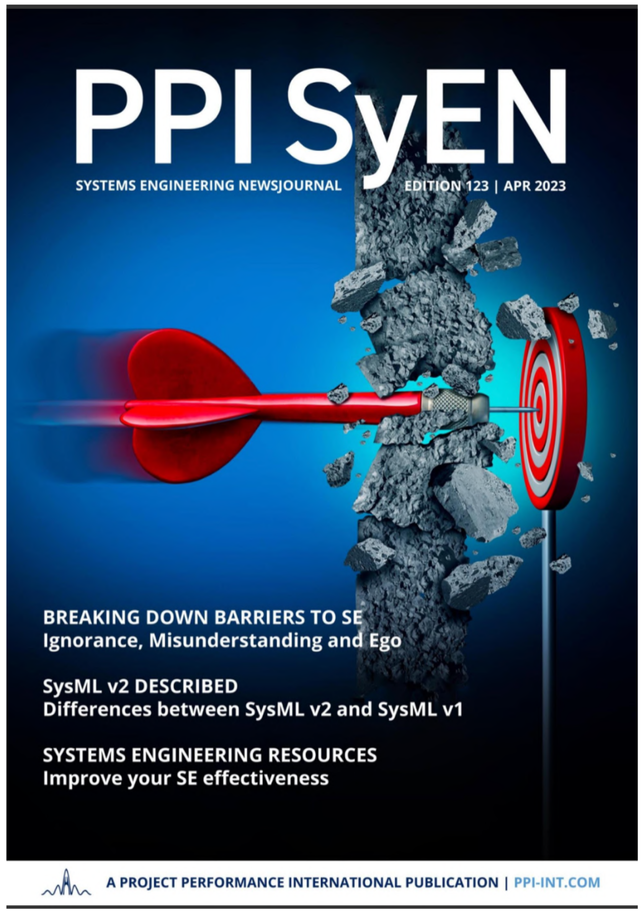
Friedenthal, S., Seidewitz, E., SysML v2: *Highlighting the Differences with SysML v1*, Project Performance international (PPI) Systems Engineering Newsjournal, PPI SyEN 123 – April 2023

- **Action**: (not necessarily atomic, i.e., could involve multiple user steps) a user action resulting in the creation, modification or suppression of an element or a diagram.
- **Element**: (model) element (including link) that is created, modified or deleted by a user action in a diagram.
- **Diagram**



Source: [Regis Casteran,"Functions in systems model", Sep. 9, 2019]

- Close to what SysML 1.0 was supposed to be, and *much* more
- Model interchangeability between tools via a clever API strategy
- Executable
- MUCH easier to learn
- MUCH easier to use
- Free of software bias – no longer a profile under UML
- Bimodal – graphical and textual representations (which has pros and cons)



References: PPI SyEN, Edition 123, April 2023 and Friedenthal, S., Seidewitz, E., A Preview of the Next Generation System Modeling Language (SysML v2), Project Performance international Systems Engineering Newsjournal, PPI SyEN 95 – 27 November 2020.

EXAMPLE OF SysML v2.0 TEXTUAL REPRESENTATION

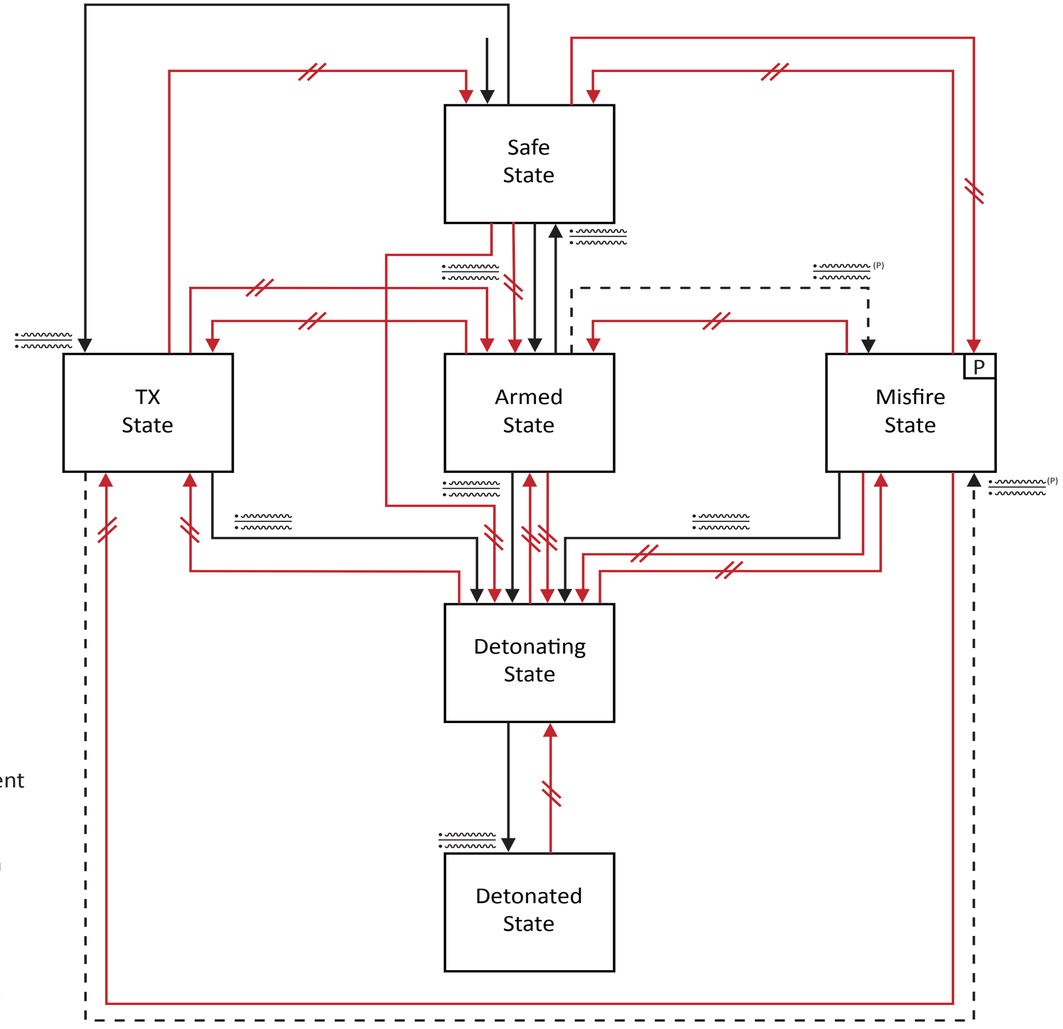
```
action def MowLawn {
        first start;
        then startMower;
        action startMower:StartMower;
        then loop{
                first monitorBattery;
                action monitorBattery : MonitorBattery {
                        out batteryLevel;
                }
                then decide checkBattery;
                if monitorBattery.batteryLevel <= 10.0 then done;
                if monitorBattery.batteryLevel > 10.0 then cutGrass;

                action cutGrass : CutGrass {
                        in cuttingHeight;
                        in lawnArea;
                        in lawnHeight;
                        out grassCut;
                }
        then done;
        } until body.monitorBattery.batteryLevel <= 10.0;

        then returnToBase;
        action returnToBase : ReturnToBase;
        then done;
}

part def Battery;
part def Cutter;
part def BaseStation;

part def Lawnmower {
        part battery: Battery{perform mowLawn.body.monitorBattery;}
        part cutter: Cutter{
                perform mowLawn.body.cutGrass;
        }
        part baseStation: BaseStation{perform mowLawn.returnToBase;}
```

PPI, a member of the OMG, has proposed three future extensions to SysML v2:

- Improved state-based modeling for requirements capture and validation
- Requirement pattern (for requirements analysis, manual authoring and potentially automated authoring of requirements – partially or fully)
- Decision patterns and decision traceability

Example 1 - State Transition Diagrams

STATES DIAGRAM FOR A BOMB



Legend:

P	Permissive guidance (”may”) statement
TX	Time Expired
→	Required transition
- - - →	Permitted but not required transition
=// =>	Prohibited transition

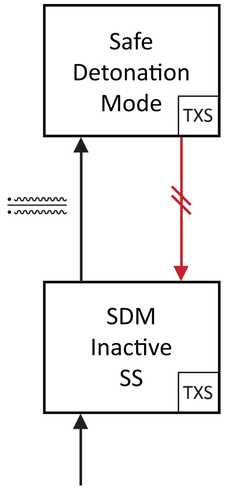
Unconnected arrow : Default state or mode

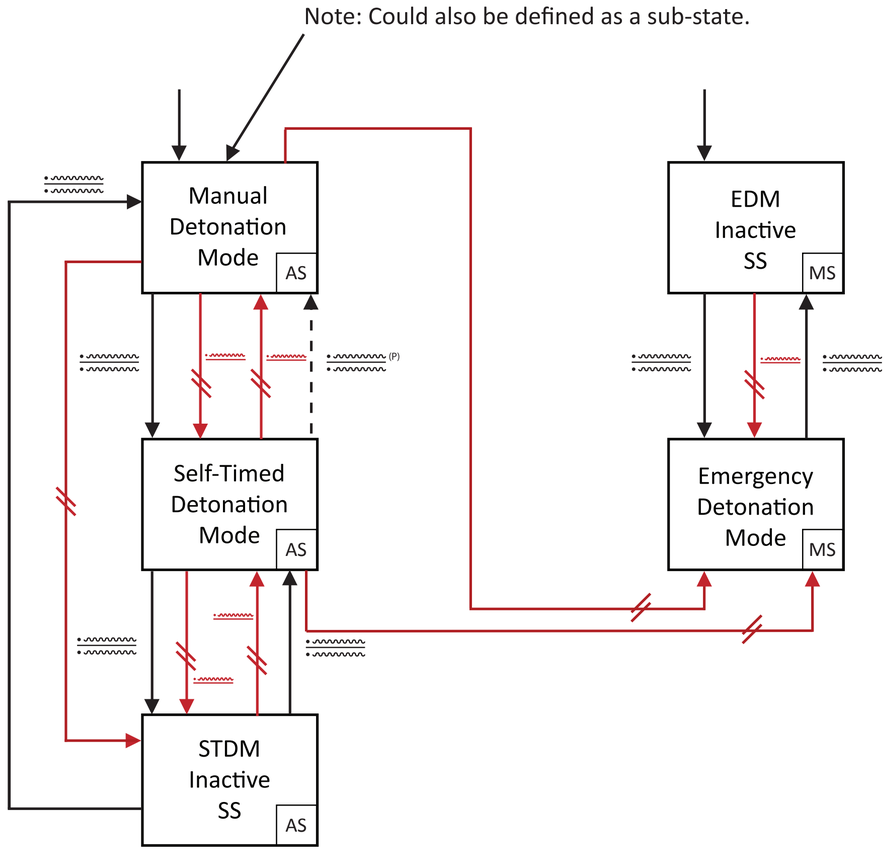
Event(s) that are to cause transition
External response (if any) as a direct consequence of the transition having occurred

PPI-008223-1

Example 2

MODES-IN-STATES DIAGRAM FOR A BOMB





Legend:

AS	Armed State
EDM	Emergency Detonation Mode
MS	Misfire State
SDM	Safe Detonation Mode
SS	Sub-State
STDM	Self-Timed Detonation Mode
TXS	Time Expired State
——→	Required transition
- - - →	Permitted but not required transition
⫽→	Prohibited transition

Unconnected arrow : Default state or mode

Event(s) that are to cause transition
———————————
External response (if any) as a direct consequence of the transition having occurred

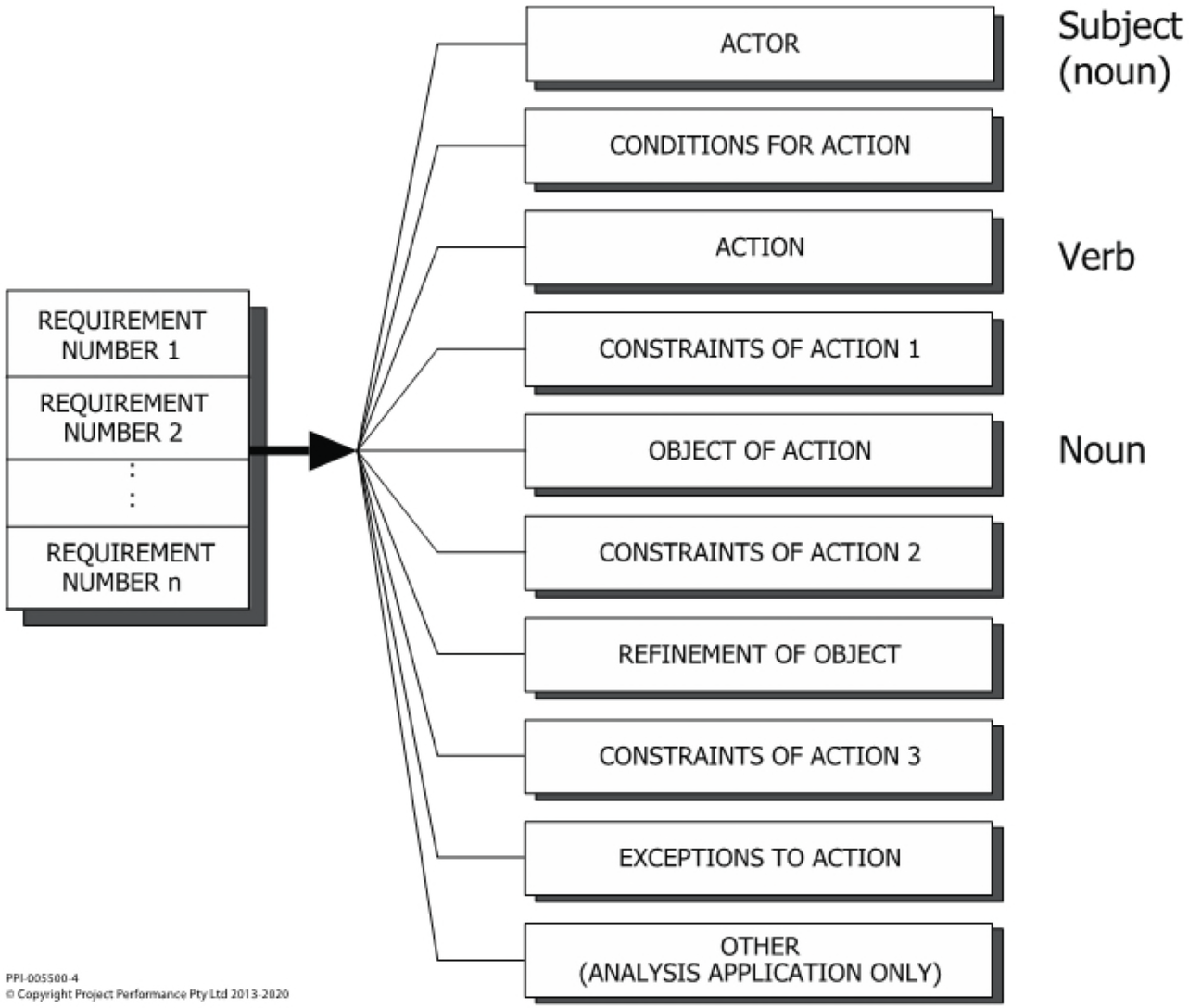
STATE-BASED MODELLING NOTATION EXPRESSIVENESS COMPARISON (WIP)

PPI State-Based Modeling Expressiveness: used in PPI States & Modes Analysis Capture and Visualize:	Expressiveness: Based on Harel statecharts(?) Capture and Visualize:
System states and their definitions	System states and their definition
System modes and their definitions	
System submodes and their definitions	
Required-to-be-potentially-concurrent states, modes and submodes	Composite states and their substates
Required-to-be-mutually-exclusive states, modes and submodes	
Permitted-but-not-required-to-be-potentially-concurrent states, modes and submodes	
Default system state(s), including conditional defaults	Initial state?
Default system mode(s) upon entry to a state, including conditional defaults	
Default system mode-inactive-substate(s) upon entry to a state, including conditional defaults	
Default system submode(s) upon entry to a mode, including conditional defaults	
Required state-to-state transitions (triggering event(s) + system external response if any)	State-to-state transitions (trigger[guard] / action)
Permitted state-to-state transitions (triggering event(s) + system external response if any)	
Prohibited state-to-state transitions (unconditional and conditional)	
Required mode-in-state to mode-in-state transitions (triggering event/events + system response if any)	
Permitted mode-in-state to mode-in-state transitions (triggering event/events + system response if any)	
Prohibited mode-in-state to mode-in-state transitions (unconditional and conditional)	
Required mode-in-state to/from mode-inactive-substate transitions (triggering event/events + system response if any)	
Permitted mode-in-state to/from mode-inactive-substate transitions (triggering event/events + system response if any)	
Prohibited mode-in-state to/from mode-inactive-substate transitions (unconditional and conditional)	
	Functional requirements in each state/substate

PPI advocates a "one-size-literally-fits-all" requirements structural pattern for:

- Parsing Analysis (performed as a part of requirements analysis)
- Requirements authoring.

This pattern breaks down a requirement into up to 7 constituent elements, two of which are mandatory and one of which is purposefully replicated.



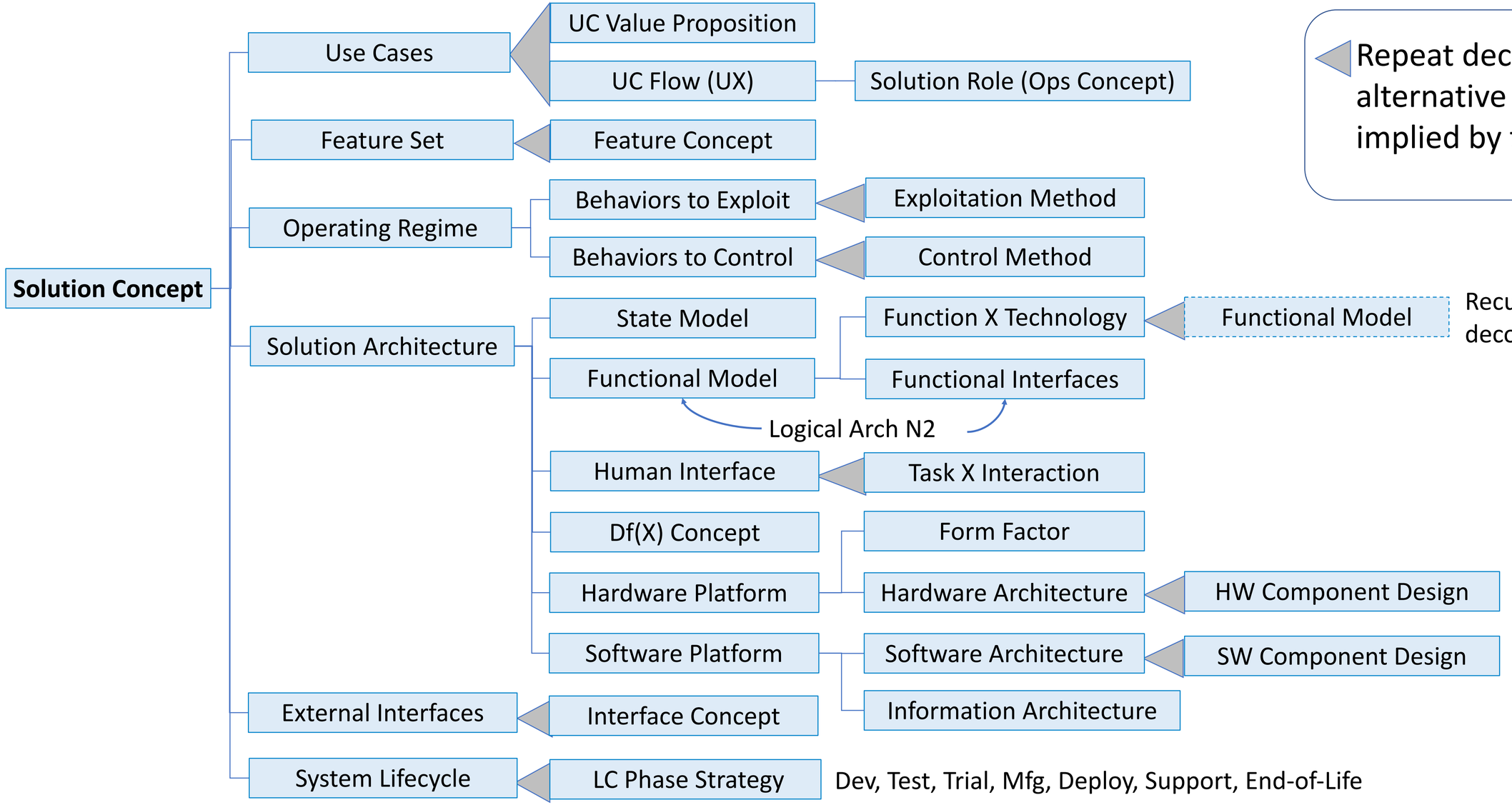
Element	Text
1. Actor	The Actor is grammatically the subject of the sentence, and is therefore a noun or a gerund (noun-verb combination). Examples are: “The system”, “The app.”, “the vehicle”, etc.
2. Conditions for Action (CfA)	Conditions for Action are the conditions during which the characteristic is required to be present, or the triggering and/or initiating/terminating conditions, in any number and type of these conditions. CfAs do not change the Action. Examples are “when in executing state”, “upon receipt of an ON command”, “from receipt of an ON command to receipt of an OFF command”.
3. Action	The Action is the verb, as it relates to the Actor, and includes any auxiliary verb such as “shall” or adverbs such as “not”, Examples are “shall display”, “shall not be displayed”.
4. Constraints of Action 1 (CoA)	Constraints of Action qualifies the action. CoA can be performance, or can take other forms. Examples are: “within 10 ms”, “at a resolution of 720x729 pixels”, “as specified herein”.
5. Object of Action (OoA)	Object of Action, if any, is the thing acted upon, and is usually a noun. OoA can also be a gerund (noun-verb combination), OoA is easily identified by taking the action and turning it around, for example, “what shall be displayed?” – “current time” would be an object of action.
6. Constraints of Action 2 (CoA)	Constraints of Action qualifies the action. CoA can be performance, or can take other forms. Examples are: “within 10 ms”, “at a resolution of 720x729 pixels”, “as specified herein”.
7. Refinement of Object (RoO)	Refinement of Object qualifies the object, for example “present at the user interface”, “for the location”, “compliant with the format standard”.
8. Constraints of Action 3 (CoA)	Constraints of Action qualifies the action. CoA can be performance, or can take other forms. Examples are: “within 10 ms”, “at a resolution of 720x729 pixels”, “as specified herein”.
9. Exceptions to Action (EtA)	Exceptions to Action limit the applicability of the requirement. Examples are “unless already in storage”, “except in Off State”. May include additional CFAs.
10. Other	Other is used only in the Analysis application of the template, and is the metaphorical garbage can. Examples of “Other” content include assertions of fact, and statements of purpose. These have no place in a requirement.

REQUIREMENTS PARSING PATTERN MAPPING TO SysML v2.0 CLASSES (WIP)

PPI Requirements Analysis Parsing Pattern Elements	Corresponding SysML v2.0 Classes:
Actor	
Condition(s) for Action	Constraints?
Action	Action (approximate correspondence)
Constraint(s) of Action 1	Constraints?
Object of Action	
Constraint(s) of Action 2	Constraints?
Refinement(s) of Object	
Constraint(s) of Action 3	Constraints?
Exception(s) to Action	
Other	

- Definition: A decision is a fundamental question or issue that demands an answer or solution - not the alternative chosen
- Design = solution decision making
- A system design is the result of numerous decisions (that must be consistent and correct)
- These decisions follow patterns that can be used to jump-start any project
- An explicit decision model enables proactive, efficient & effective design; ad hoc decision-making is just the opposite
- Decisions create requirements, i.e., all requirements are derived requirements
- Decision traceability demands capture of decision rationale and consequences (a rich data structure)

PRODUCT/SYSTEM DESIGN DECISION PATTERN



REQUIREMENTS ANALYSIS (CAPTURE AND VALIDATION) METHODOLOGY

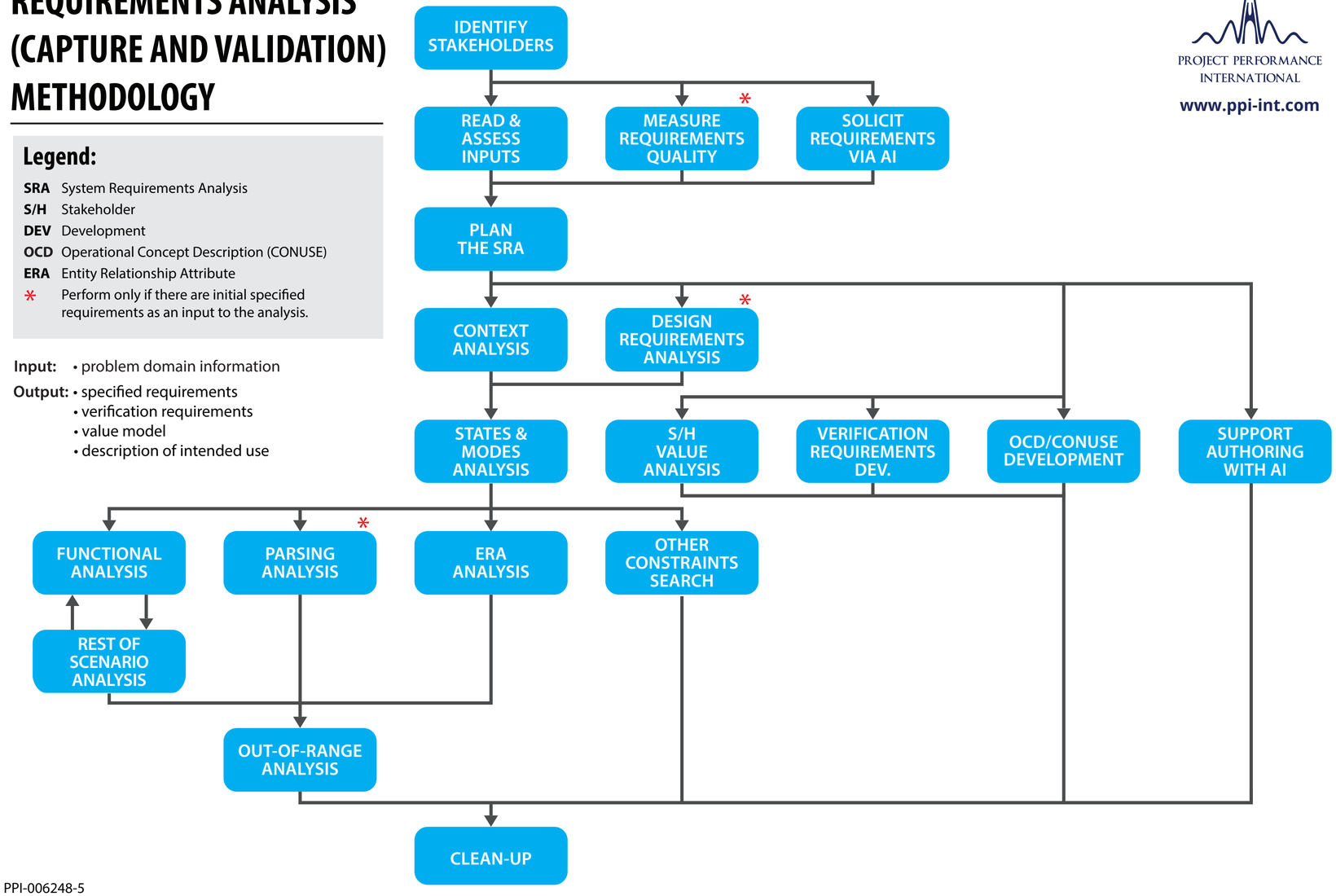
PROJECT PERFORMANCE INTERNATIONAL
www.ppi-int.com

Legend:

SRA System Requirements Analysis
S/H Stakeholder
DEV Development
OCD Operational Concept Description (CONUSE)
ERA Entity Relationship Attribute
* Perform only if there are initial specified requirements as an input to the analysis.

Input: • problem domain information

Output: • specified requirements
• verification requirements
• value model
• description of intended use



PPI-006248-5

Thank you for your interest ☺ Any questions?

Robert J. Halligan

Email: rhalligan@ppi-int.com