UNIVERSITÀ DEGLI STUDI FIRENZE

DINFO
Dipartimento di Ingegneria dell'Informazione

Software Engineering for Embedded Systems

An introduction to the Systems Modeling Language

Laura Carnevali

Software Technologies Lab
Department of Information Engineering
University of Florence
`http://stlab.dinfo.unifi.it/carnevali`
laura.carnevali@unifi.it

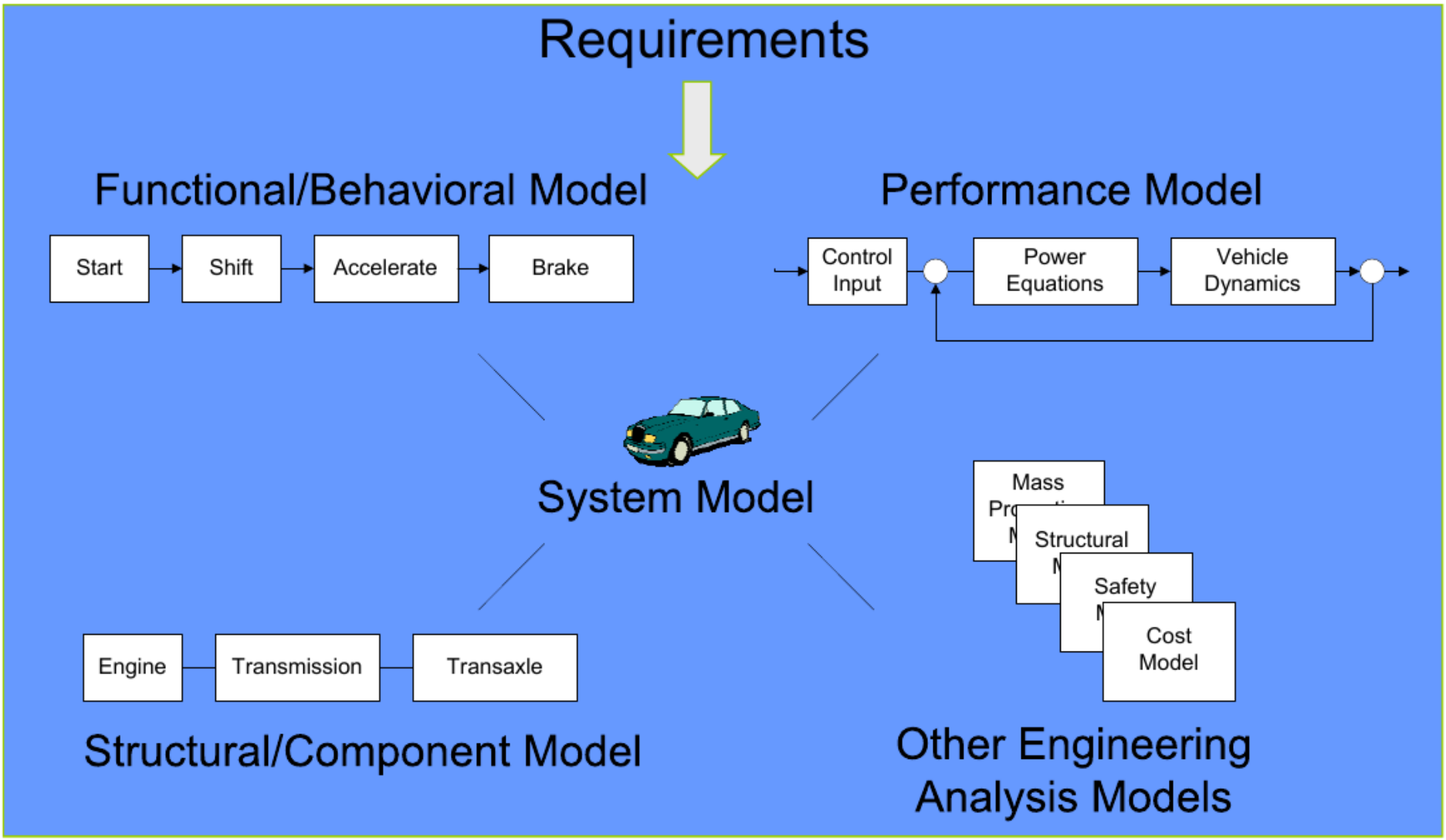
Outline

Systems Modeling Language

What is Systems Engineering?

- Systems Engineering is a discipline that concentrates on the design and application of the **whole (system)** as distinct from the parts. It involves looking at the problem in its entirety, taking into account all the facets and all the variables. (*Federal Aviation Agency FAA-USA, Systems Engineering Manual, Definition by Simon Ramo, 2006*)
- Systems Engineering is an iterative process of top-down synthesis, development and operation of a **real-world system** that satisfies, in a near-optimal manner, the full range of requirements for the system. (*Definition by Howard Eisner, Essentials of Project and Systems Engineering Management, Wiley, 2002*)

System modeling



Benefits of model-based Systems Engineering

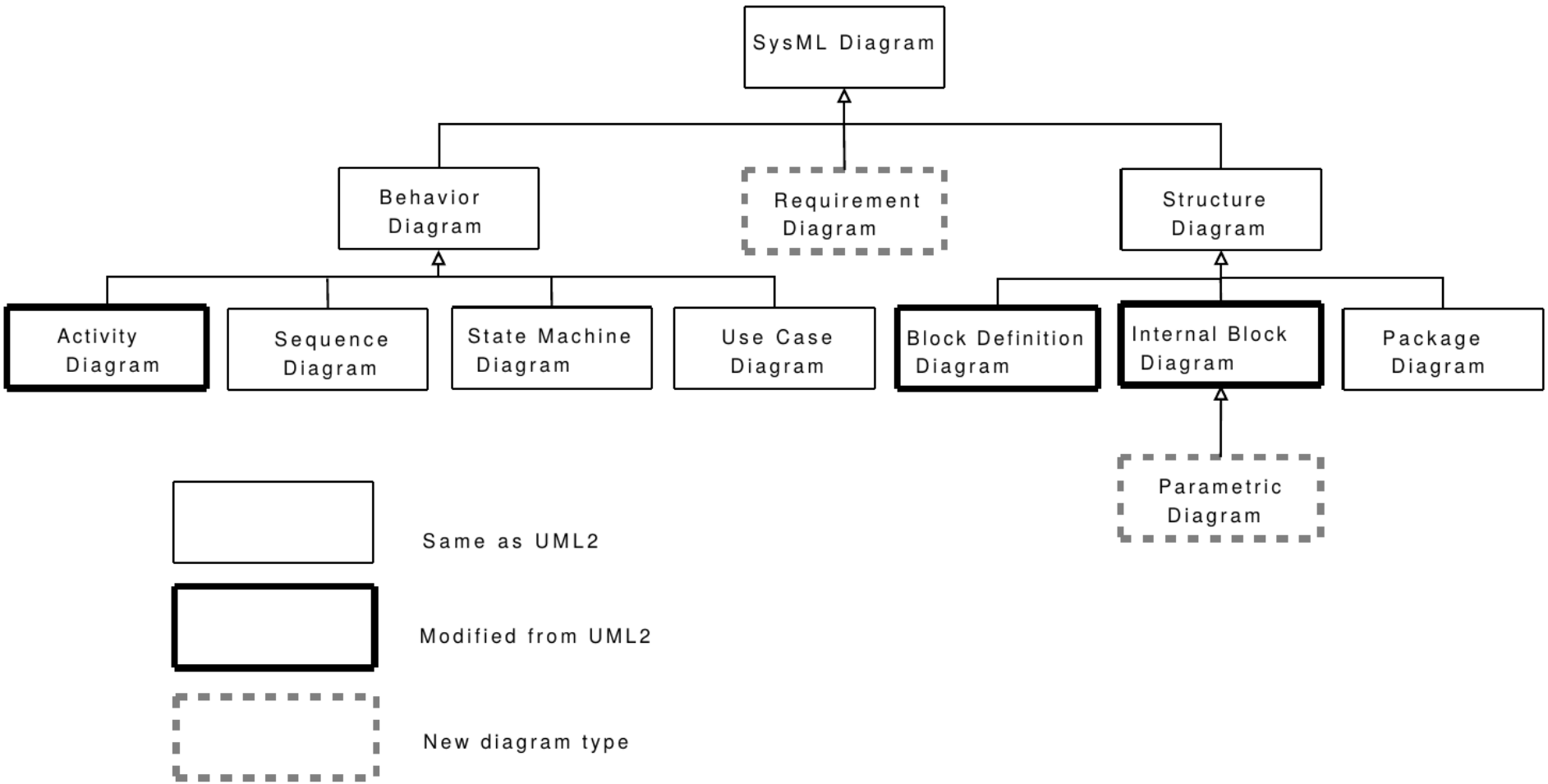
- Shared understanding of system requirements and design
  - Validation of requirements
  - Common basis for analysis and design
  - Facilitates identification of risks
- Assists in managing complex system development
  - Separation of concerns via multiple views of integrated model
  - Supports traceability through hierarchical system models
  - Facilitates impact analysis of requirements and design changes
  - Supports incremental development & evolutionary acquisition
- Improved design quality
  - Reduced errors and ambiguity
  - More complete representation
- Supports early and on-going verification & validation to reduce risk
- Provides value through life cycle (e.g., training)
- Enhances knowledge capture

Systems Modeling Language (SysML)

- Is a graphical modeling language adopted by the OMG in 2006
- Supports specification, analysis, design, verification, and validation of systems that include hardware, software, data, personnel, procedures, and facilities
- Is a visual modeling language that provides
  - Semantics = meaning, connected to a metamodel
    (rules governing the creation and the structure of models)
  - Notation = representation of meaning, graphical or textual
- Is not a methodology or a tool (SysML is methodology and tool independent)
- Extends a subset of UML2
  - The Unified Modeling Language (UML) is a general-purpose, developmental, modeling language for software-intensive applications (**software engineering**)

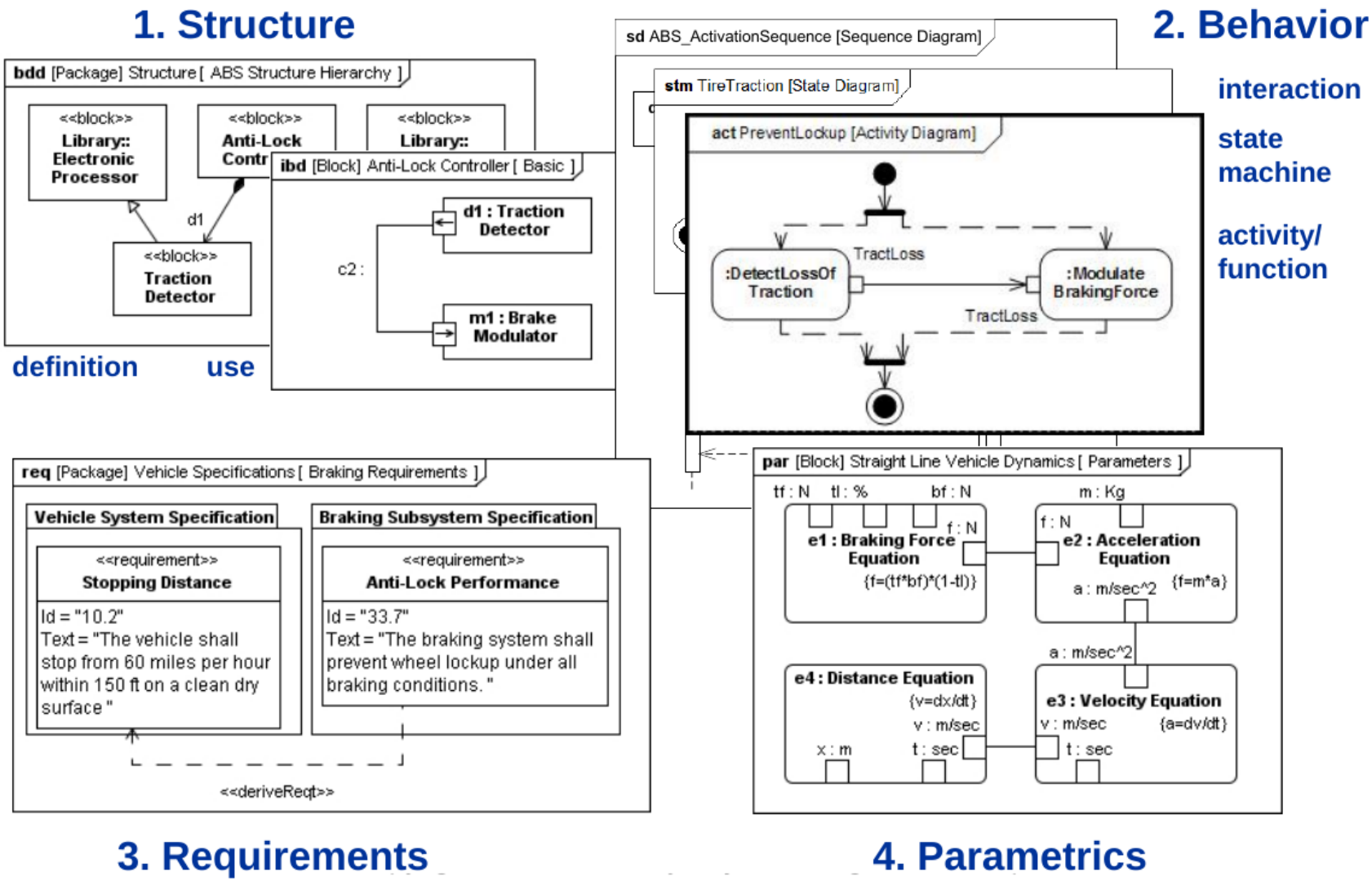
SysML vs UML

- SysML diagrams

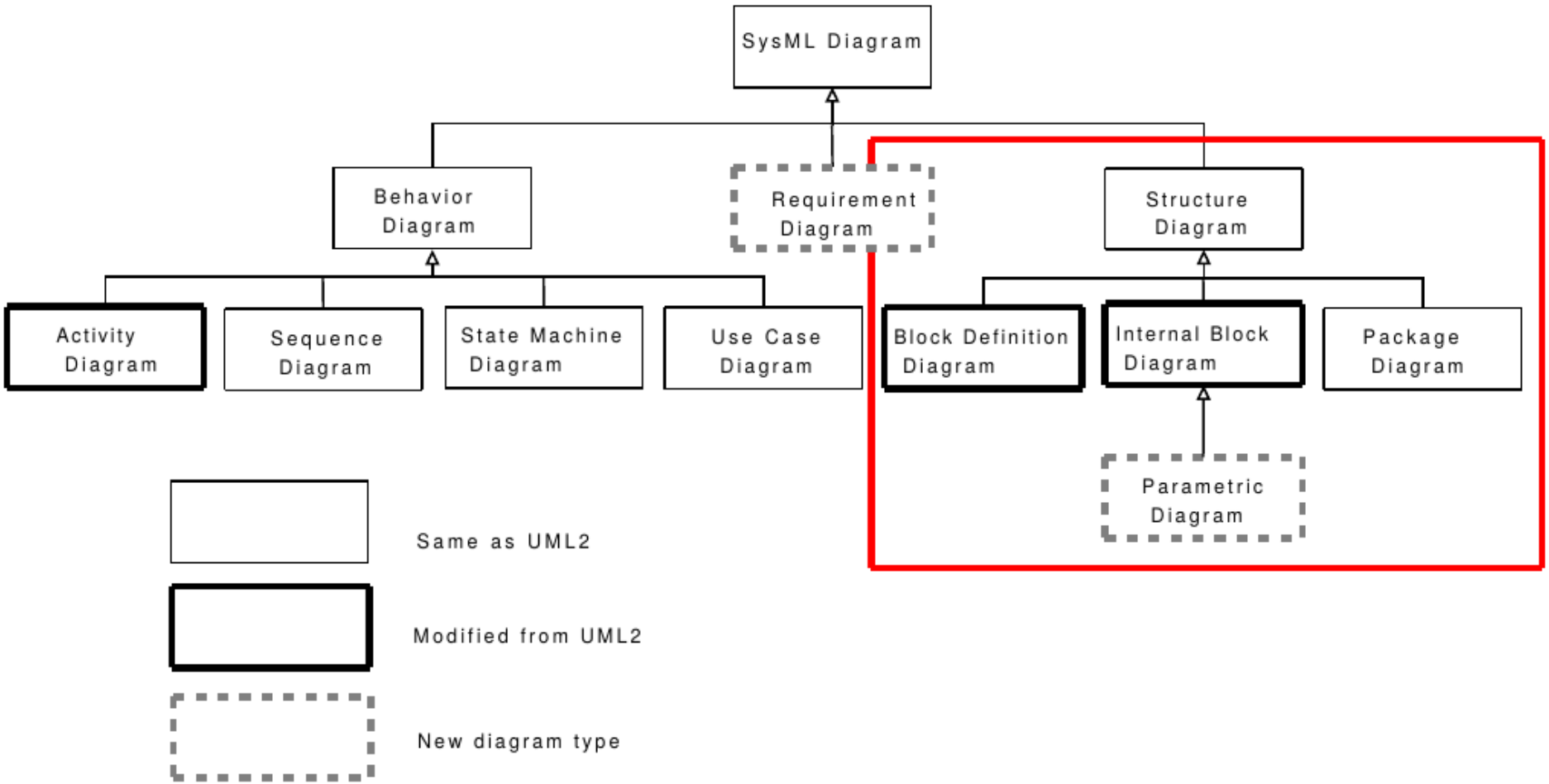


4 pillars of SysML

- Example of modeling of an ABS system



Structural diagrams



Package diagrams

- Same as in UML2
- Used to organize the model
  - Groups model elements into a name space
  - Often represented in tool browser
  - Support model configuration management (check-in/out)
- The model can be organized in multiple ways
  - By System hierarchy (e.g., enterprise, system, component)
  - By diagram kine (e.g., requirements, use cases, behavior)
  - Use viewpoints to augment model organization

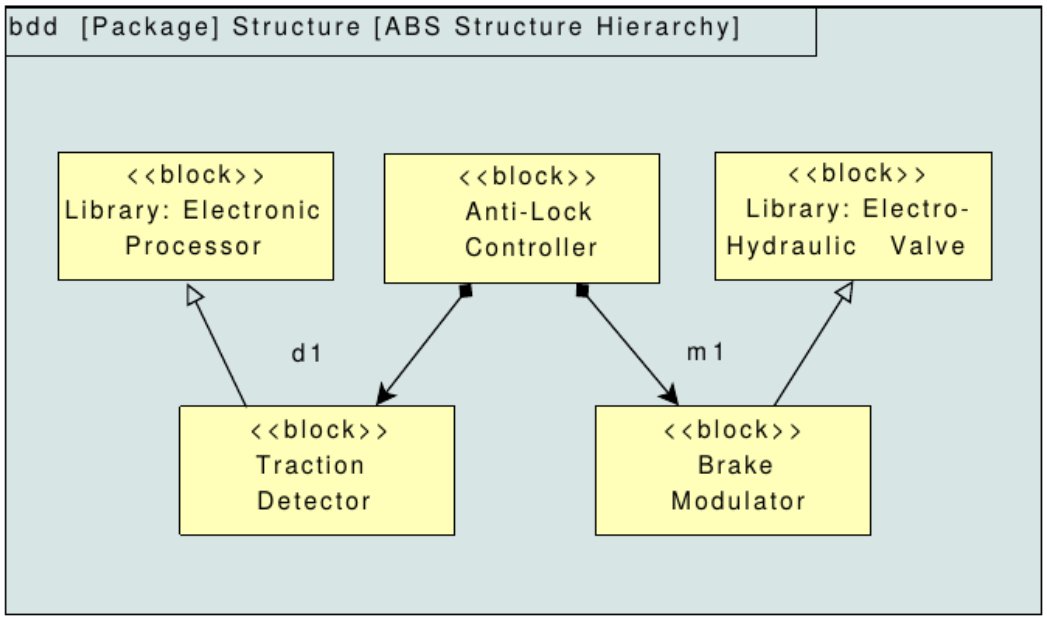
Block

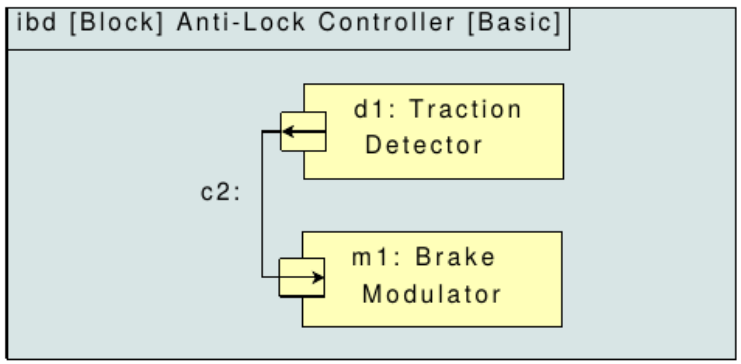
- Is a basic structural element
- Defines a set of structural and functional features to describe an element or system (software, hardware, data, procedures, . . .)
- Used during both physical and logical design
- Its characteristics can be described by multiple standard compartments
  - Properties (parts, references, values, ports)
  - Operations
  - Constraints
  - Allocations from/to other model elements (e.g., activities)
  - Requirements the block satisfies
  - User defined compartments
- Can be used to specify hierarchies and interconnection

Block Definition Diagrams and Internal Block Diagrams (1/2)

- A Block Definition Diagram (BDD) described the relationships among blocks (e.g., composition, association, specialization)
  - Same as a type definition (corresponds to a UML class diagram)
  - Reused in multiple contexts
  - Cannot define completely the communication dependencies and the composition structure (no topology)
- An Internal Block Diagram (IBD) describes the internal structure of a block in terms of its properties and connectors
  - Part is the usage of a block in the context of a composing block (also known as role)
  - Internal structure and communication & signalling topology become explicit
- Ports specify interaction points on blocks and parts
  - Integrate behavior with structure
  - Standard (UML) Ports specify required or provided operations and/or signals (used for software components)
  - Flow Ports specify what can flow in/out of a block/part (used for signals and physical flows)

Block Definition Diagrams and Internal Block Diagrams (2/2)

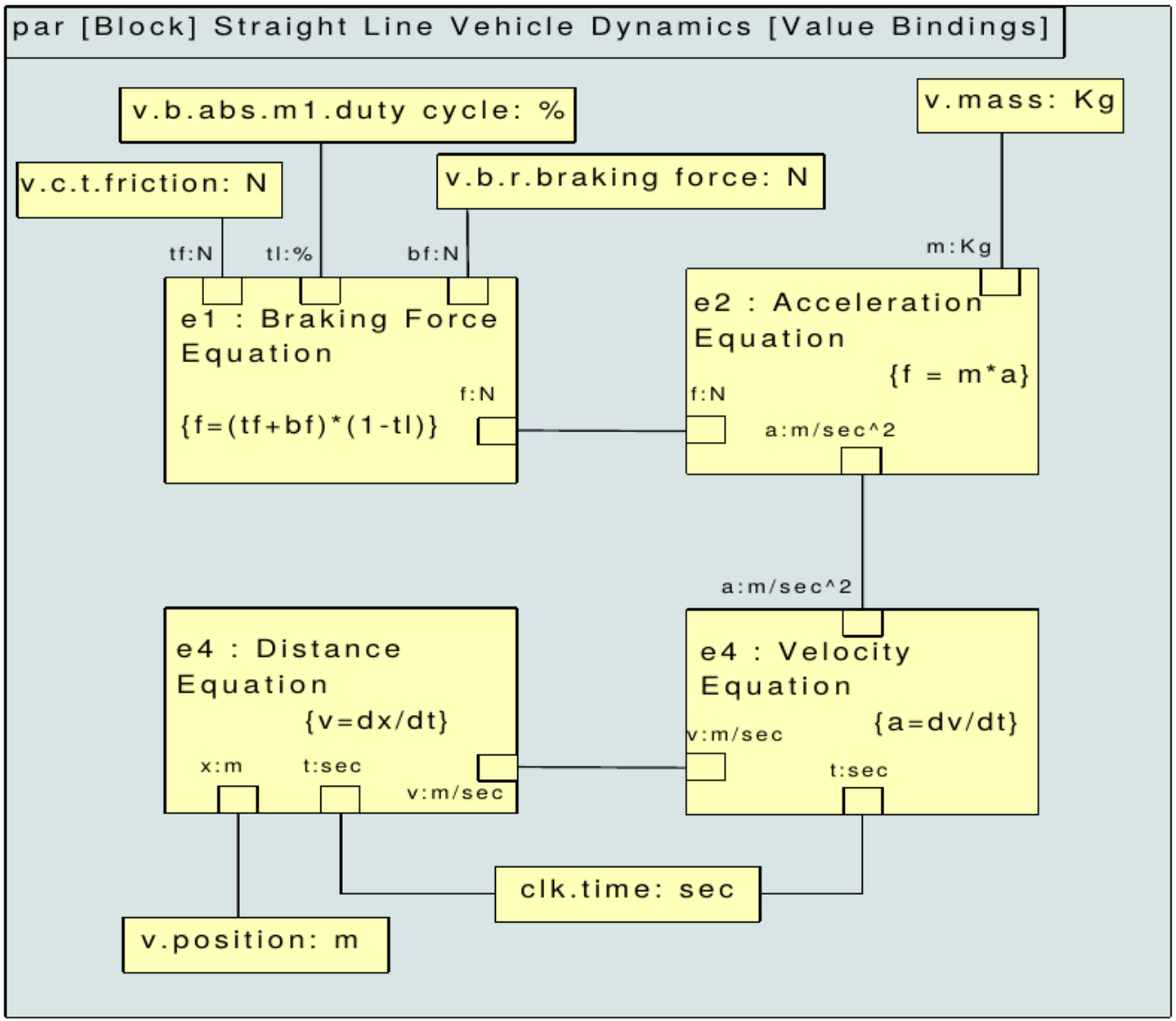




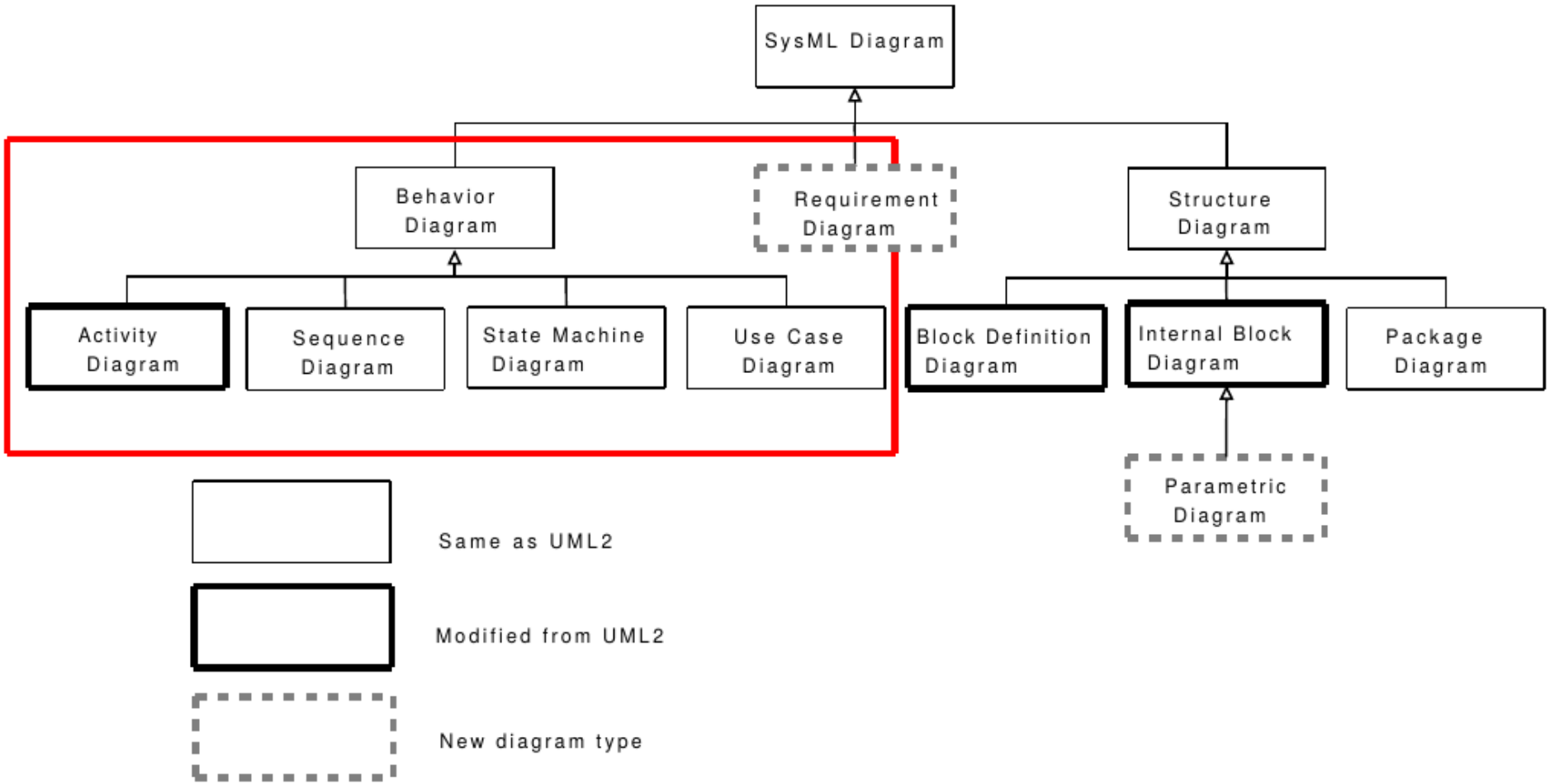
Parametric Diagram (1/2)

- Used to express constraints (equations) between value properties
  - Provides support for engineering analysis (e.g., performance, reliability)
  - Facilitates identification of critical performance properties
- Represents the usage of constraints in an analysis context
  - Constraints are defined by constraint blocks capturing equations
  - Expression language can be formal or informal
  - Computational engine is provided by applicable analysis tool and not by SysML
  - Binding of constraint parameters to value properties of blocks

Parametric Diagram (2/2)



Behavioral diagrams



Activity Diagrams

- Specify transformation of inputs to outputs through a controlled sequence of actions
- Similar to dataflows with enhancements that allow the representation of a more general semantics, including modeling of continuous physical flows
- Secondary constructs show responsibilities for the activities using activity partitions (i.e., swimlanes)
  - SysML adds support for continuous flow modeling (modeling of continuous-time systems)
  - Alignment of activities with classical systems engineering techniques, e.g., Enhanced Functional Flow Block Diagrams (EFFBDs)

Sequence Diagrams

- Provide representations of message based behavior
  - Represent flow of control
  - Describe interactions between parts
- Provide mechanisms for representing complex scenarios
  - Reference sequences
  - Control logic
  - Lifeline decomposition
- Also known as Interaction Diagrams or Interactions

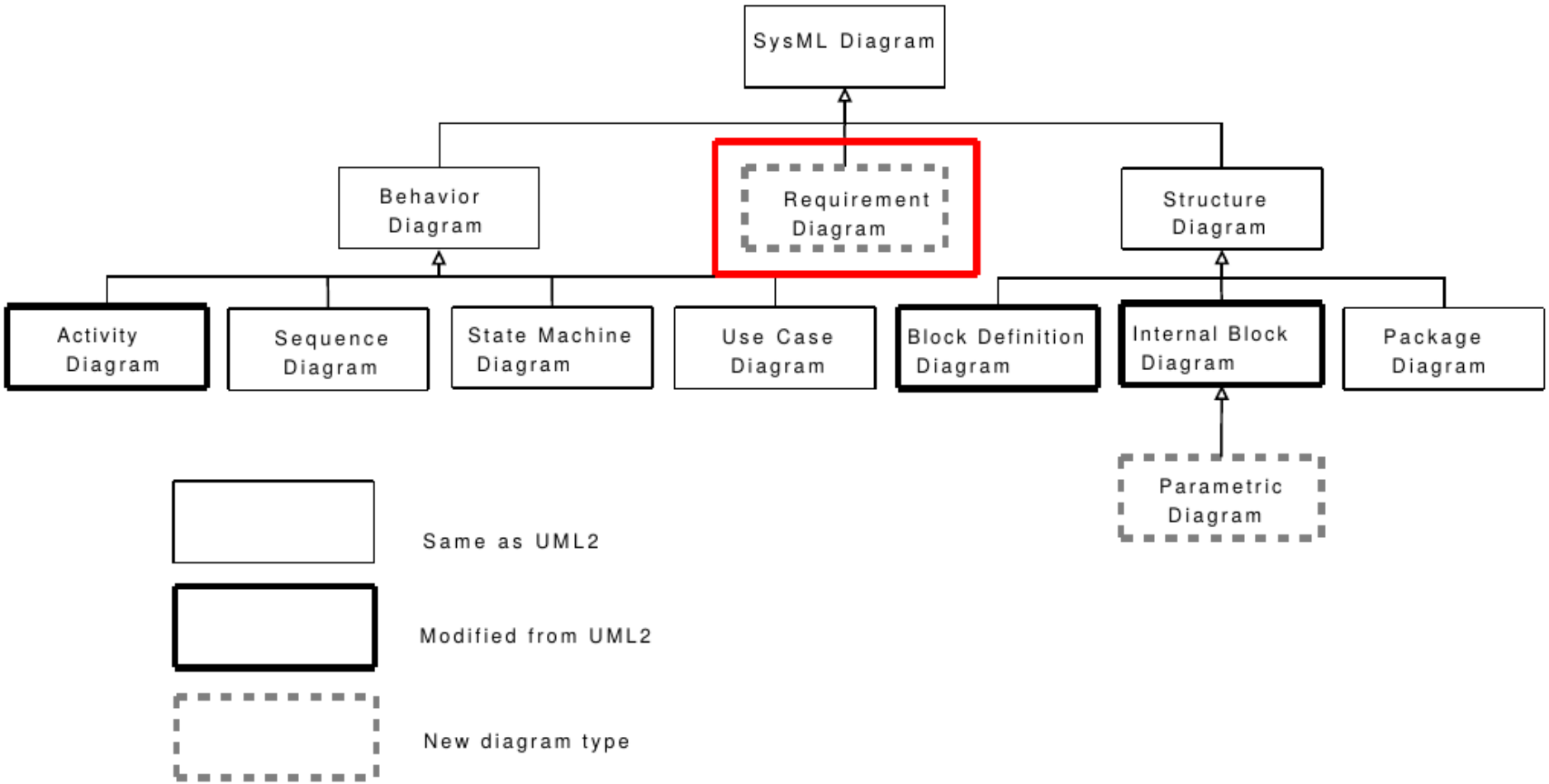
State Machine Diagrams

- State Machines are used to represent the life cycle of a block
- State Machines support the representation of event-based behavior (generally asynchronous behavior)
  - Transition with trigger, guard, action
  - State with entry, exit, and do-activity
  - Can include nested sequential or concurrent states
  - Can send/receive signals to communicate between blocks during state transitions
  - Change events, time events, signal events

Use Case Diagrams

- Provide means for describing basic functionality in terms of usages/goals of the system by actors
  - Use is methodology dependent
  - Often accompanied by use case descriptions
- Common functionality can be factored out via ≪include≫ and ≪extend≫ relationships
- Elaborated via other behavioral representations to describe detailed scenarios
- No changes with respect to UML

Cross-cutting constructs



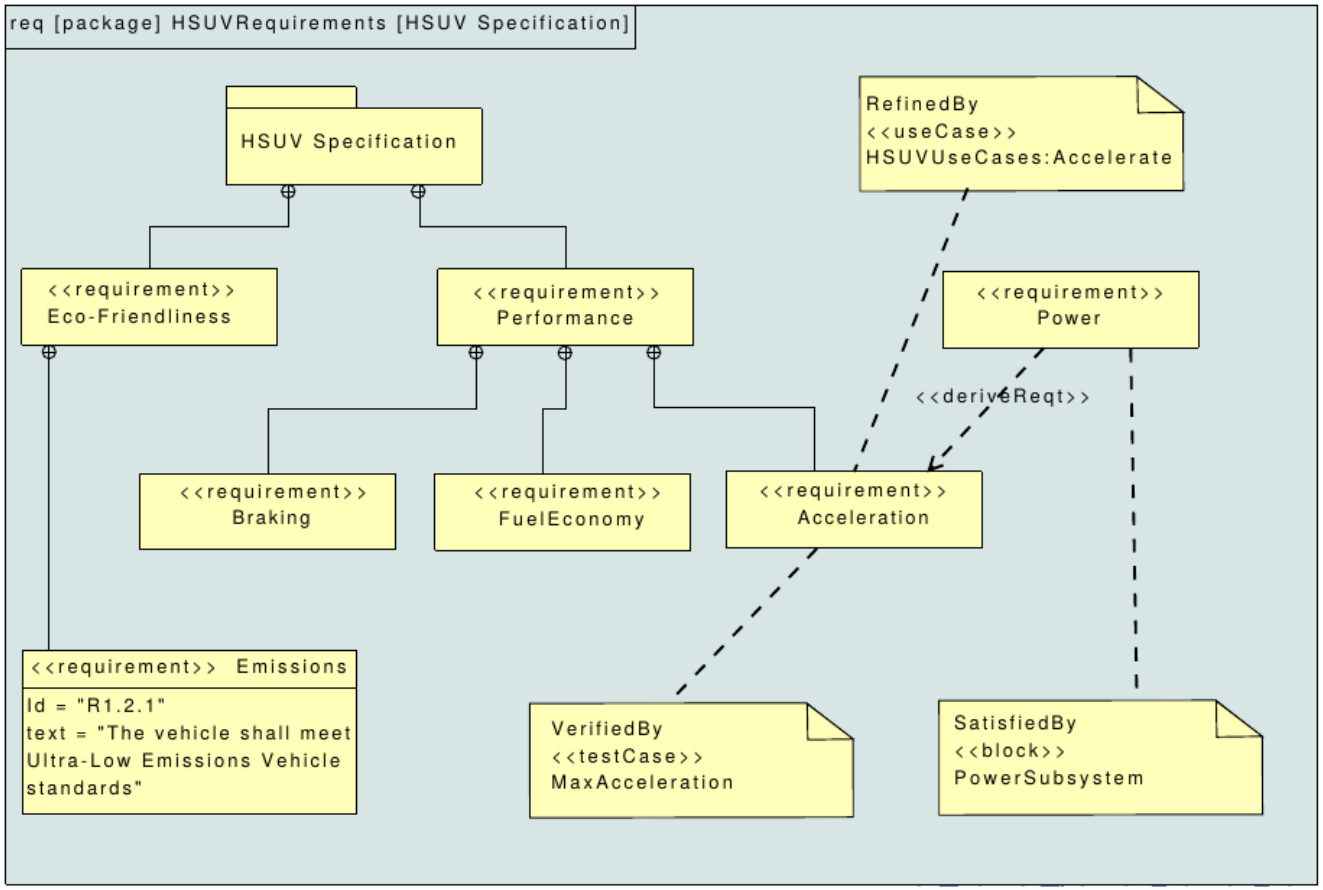
Allocations

- Represent general relationships that map one model element to another
- Different types of allocation
  - Behavioral (i.e., function to component)
  - Structural (i.e., logical to physical)
  - Software to Hardware
  - ...
- Explicit allocation of activities to structure via swimlanes (i.e., activity partitions)
- Both graphical and tabular representations are specified

Requirements

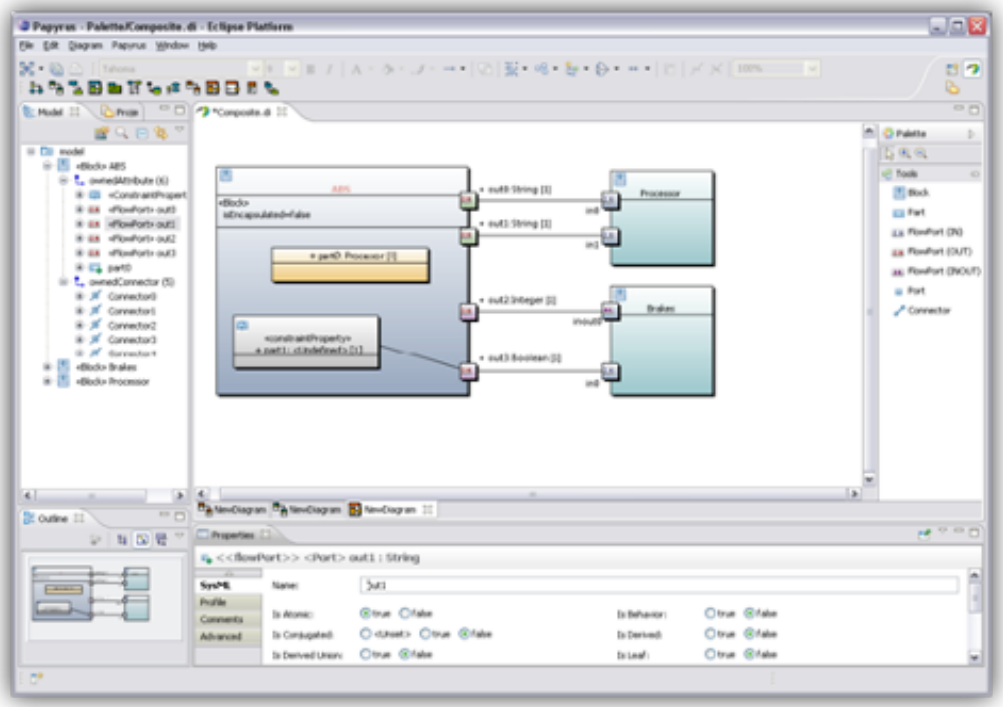
- The ≪requirement≫ stereotype represents a text based requirement
  - Includes id and text properties
  - Can add user defined properties such as verification method
  - Can add user defined requirements categories
    (e.g., functional, interface, performance)
- Requirements hierarchy describes requirements contained in a specification
- Requirements relationships include:
  - `DeriveReqt`
  - `Satisfy`
  - `Verify`
  - `Refine`
  - `Trace`
  - `Copy`

Requirement Diagram



Papyrus

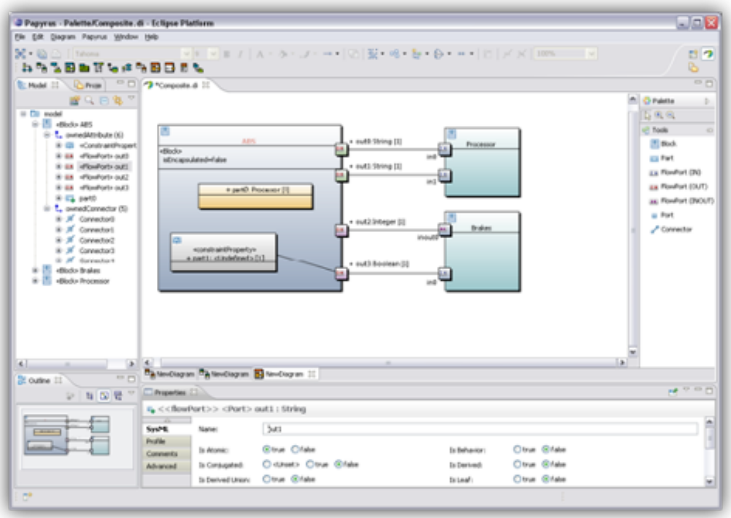
- An open source tool based on Eclipse providing an integrated environment for editing UML and SysML models
  - Developed by the Laboratory of Model Driven Engineering for Embedded Systems of the French Alternative Energies and Atomic Energy Commission
  - Home page: `https://www.eclipse.org/papyrus`
  - Can be used either as a standalone tool or as an Eclipse plugin



Papyrus: how to install the tool

- Select *Help* → *Install New Software*
- In *Work with*, add the following link: `http://download.eclipse.org/modeling/mdt/papyrus/updates/releases/2019-09`
- Select *Add* → *Papyrus for UML* and start the installation

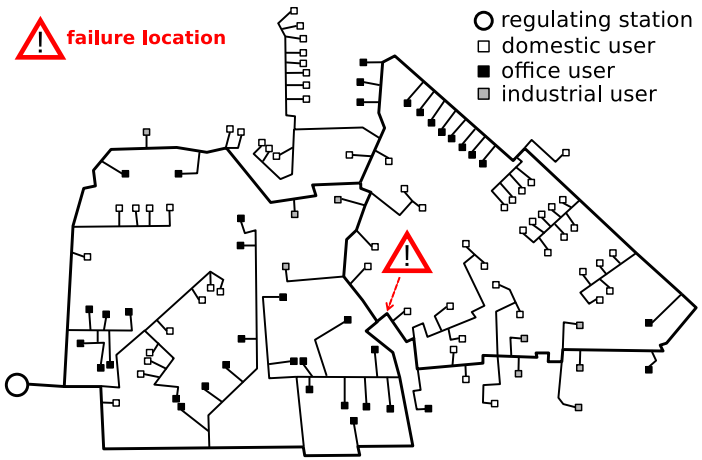
- Select *Help* → *Install New Software*
- In *Work with*, add the following link: `http://download.eclipse.org/modeling/mdt/papyrus/components/sysml14`
- Select *Add* → *SysML* and start the installation

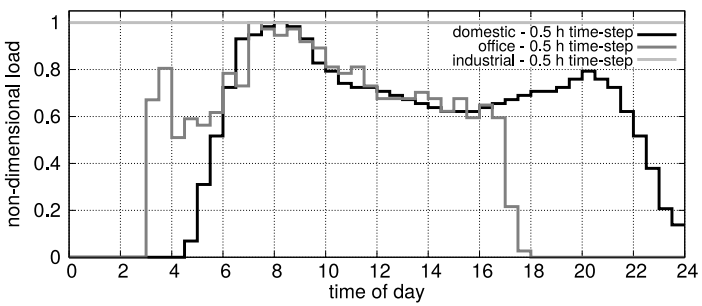


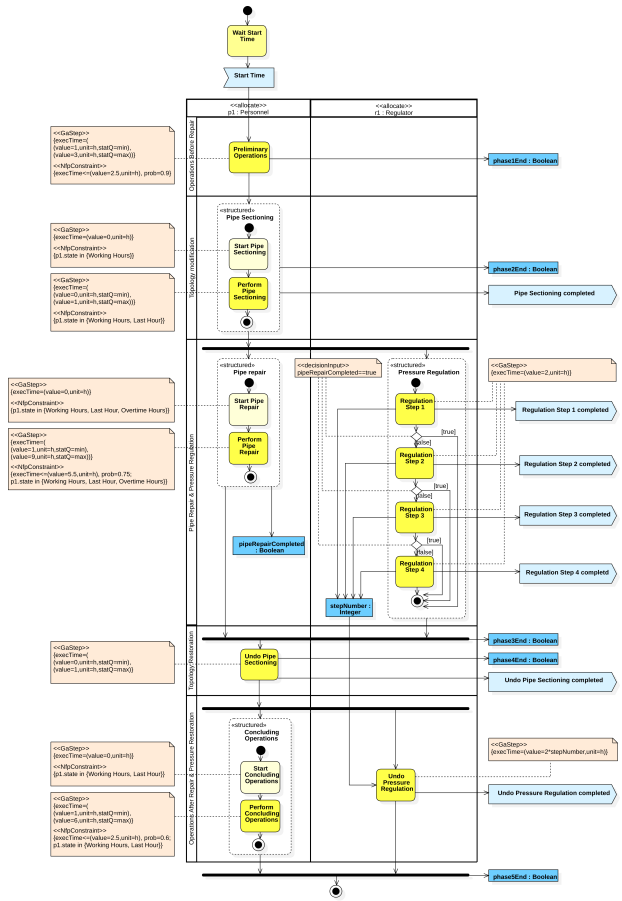
An example in the area of cyber-physical systems

Gas distribution networks

- Couple **physical** fluid-dynamics with **cyber** management procedures
- Goal: evaluate the **low pressure risk** in the transient phase after a repair
  - Fluid-dynamic analysis of gas behavior, stochastic analysis of repair actions



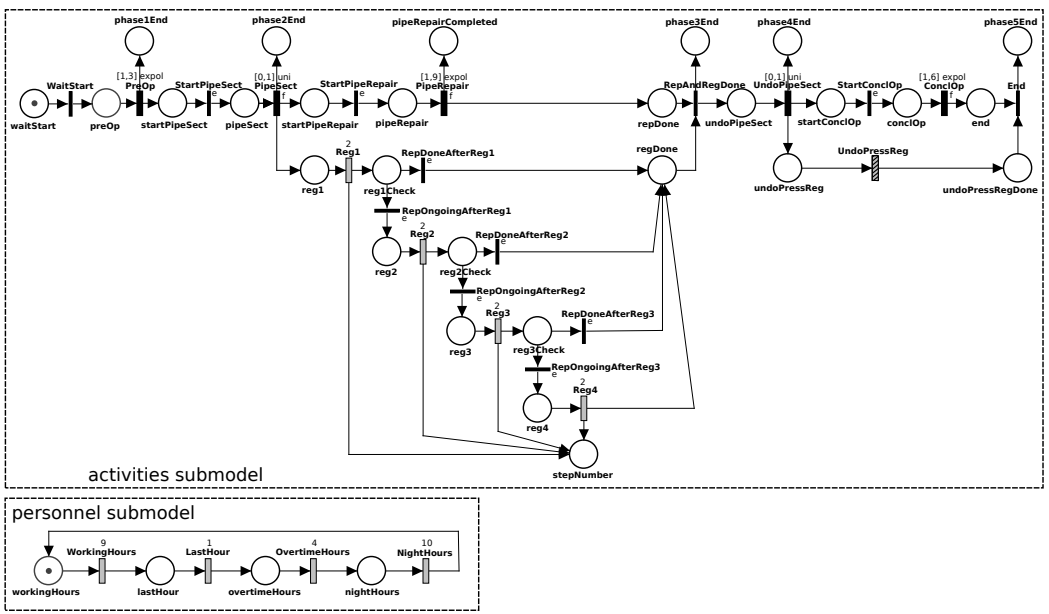




M. Biagi, L. Carnevali, F. Tarani, E. Vicario, "Model-based quantitative evaluation of repair procedures in gas distribution networks," ACM Transactions on Cyber-Physical Systems, 2019.

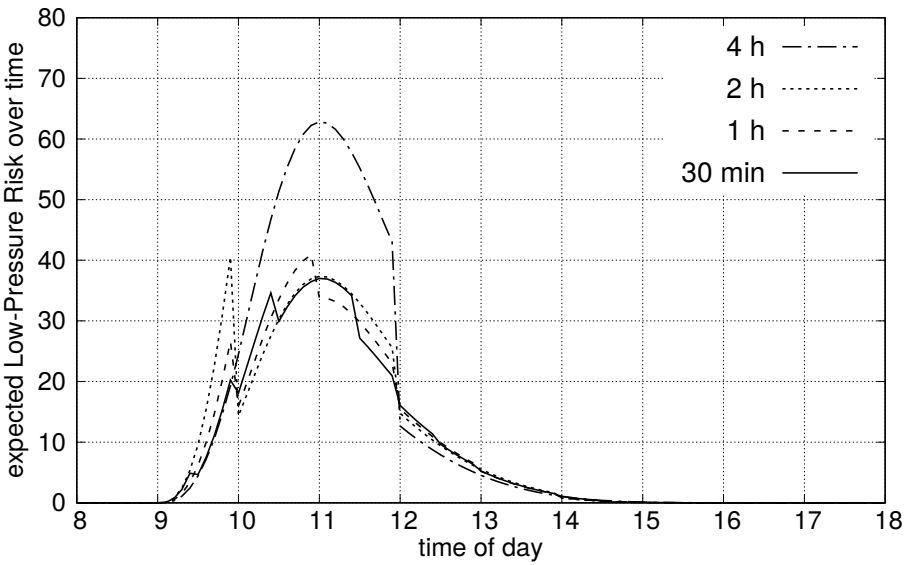
Solution combining fluid-dynamic and stochastic analyses

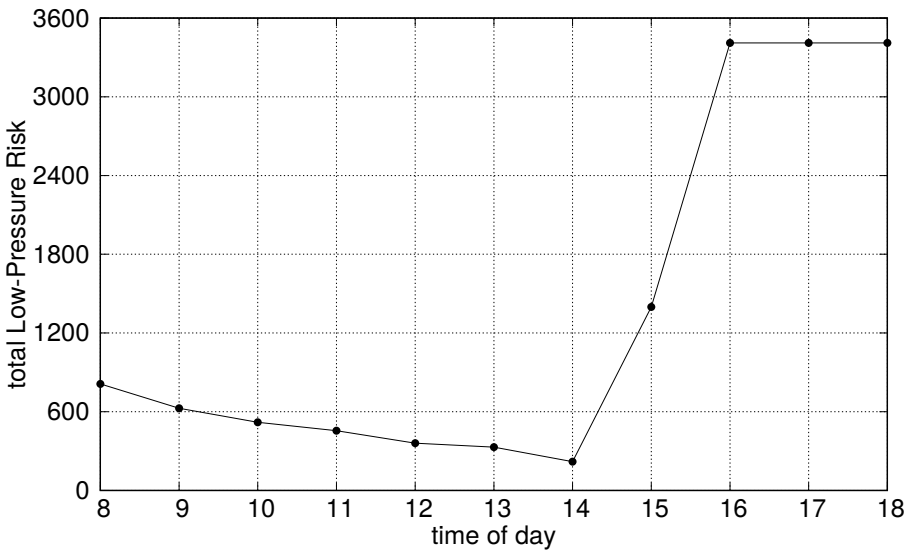
- Fluid-dynamic analysis of gas behavior
  - Derives the Low Pressure Risk (LPR) experienced in each fluid-dynamic state
- Forward transient analysis of the repair procedure
  - Derives the transient probability of each fluid-dynamic state
  - Instantaneous transient reward $\sum_{\gamma \in \Gamma} \texttt{If}\,(e(m,\gamma), \mathrm{LPR}_\gamma, 0)$
    (Γ is the set of fluid-dynamic states, LPR_γ is the LPR measure in $\gamma \in \Gamma$, $e(m,\gamma)$ is TRUE if marking m represents state γ, FALSE otherwise)
- Solution yields the expected **Low Pressure Risk (LPR)** over time



Solution combining fluid-dynamic and stochastic analyses

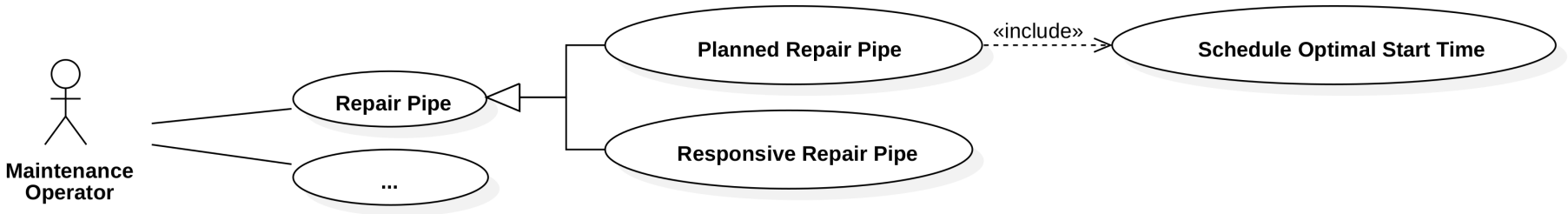
- Fluid-dynamic analysis of gas behavior
  - Derives the Low Pressure Risk (LPR) experienced in each fluid-dynamic state
- Forward transient analysis of the repair procedure
  - Derives the transient probability of each fluid-dynamic state
  - Instantaneous transient reward $\sum_{\gamma \in \Gamma} \texttt{If}\,(e(m,\gamma), \mathrm{LPR}_\gamma, 0)$
    (Γ is the set of fluid-dynamic states, LPR_γ is the LPR measure in $\gamma \in \Gamma$, $e(m,\gamma)$ is TRUE if marking m represents state γ, FALSE otherwise)
- Solution yields the expected **Low Pressure Risk (LPR)** over time





SysML: Use Case Diagram and Requirements

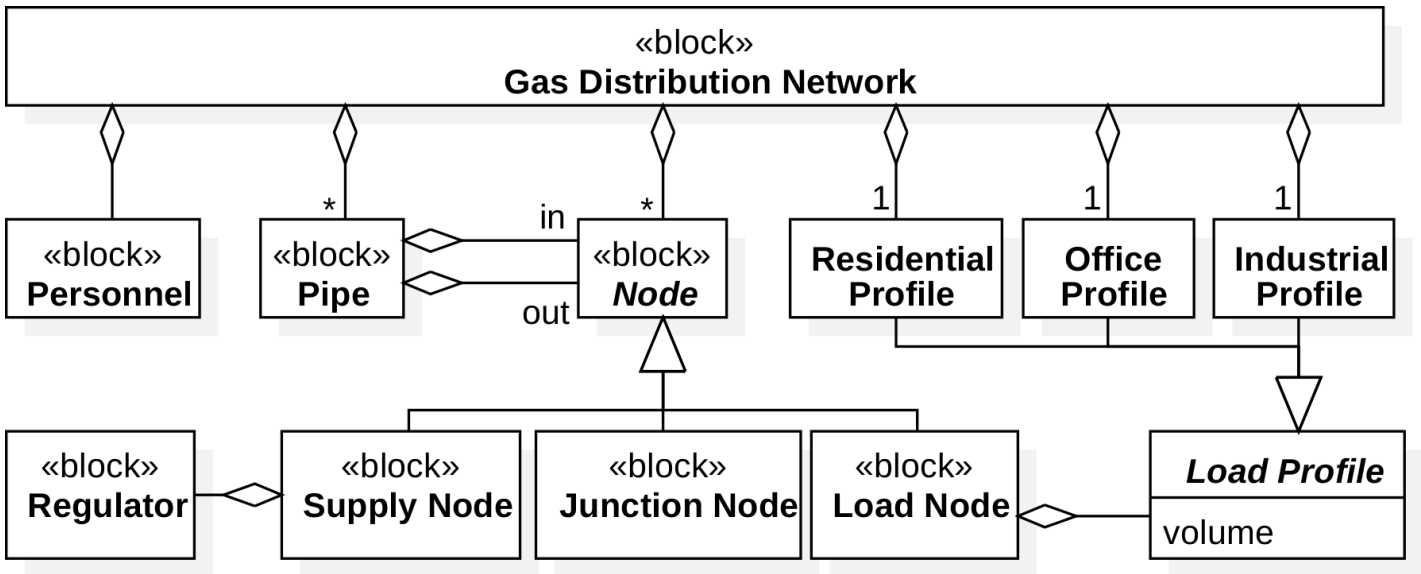
- Use Case Diagram of the pipe repair scenario



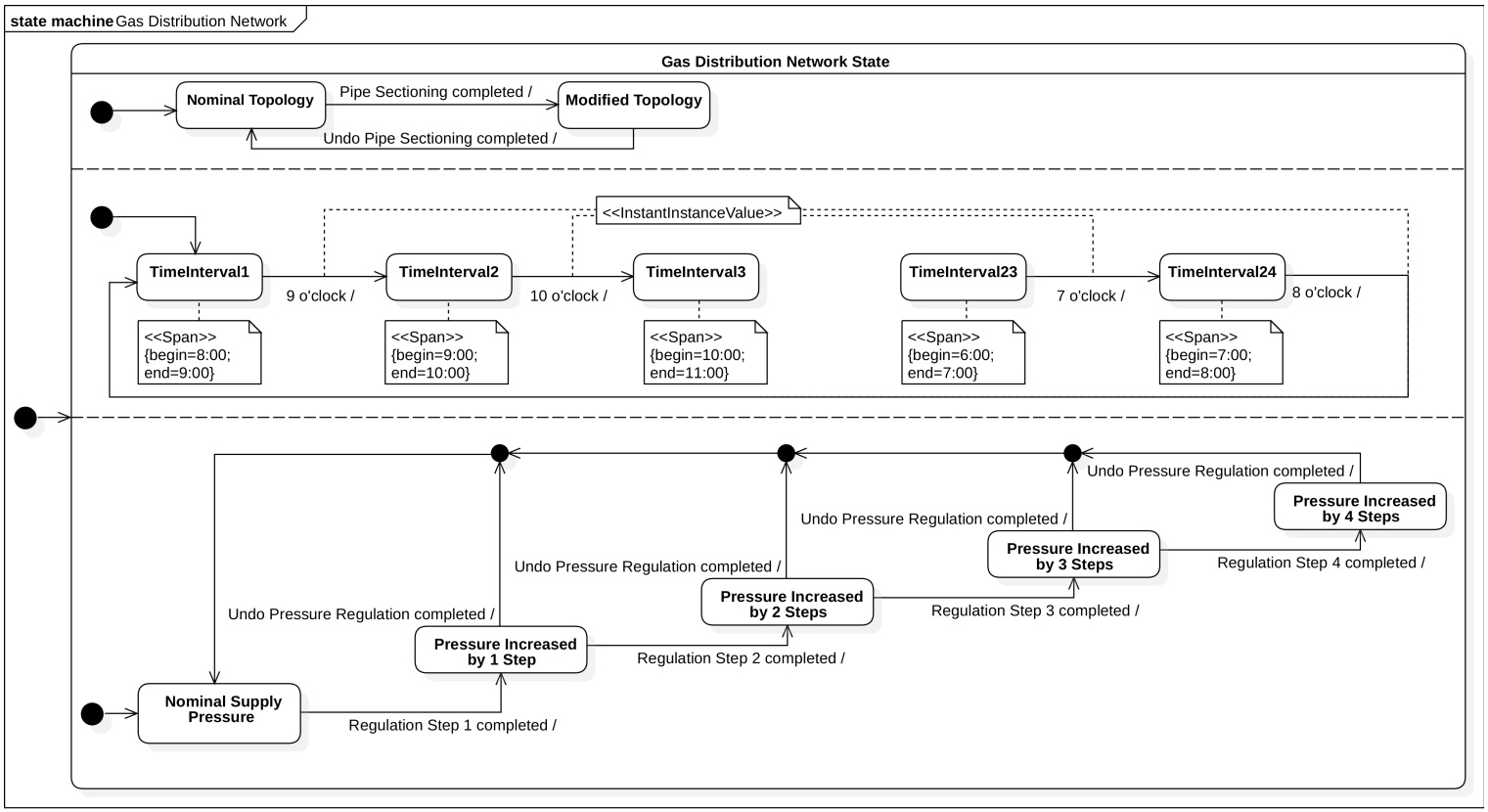
- Requirements Table

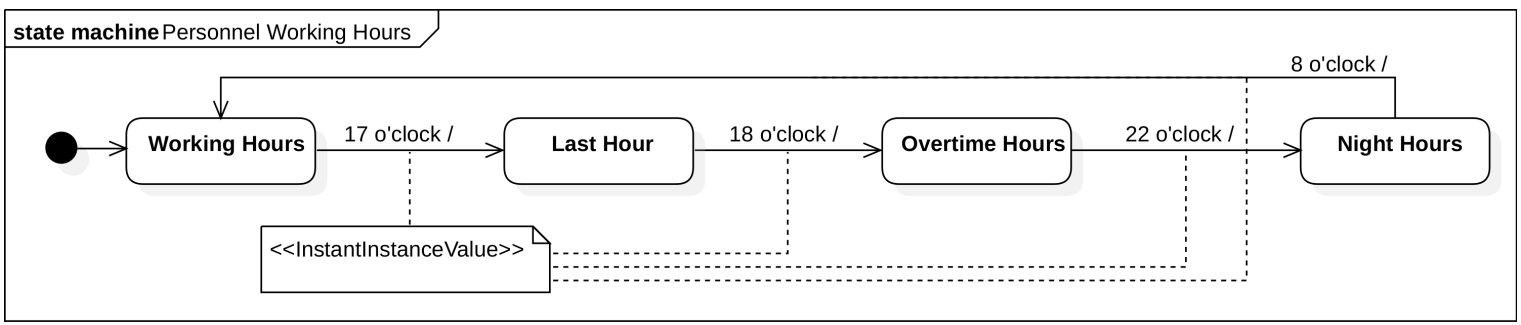
Use Case	Requirements
Planned Repair Pipe	The procedure `Completion Time` must be $<$ 36 h in 75% of the cases, and $<$ 48 h in 99.99% of the cases.
	The `Low-Pressure Risk` at any time must be $<$ 240 $\text{bar}^{1/2}$ on average.
Responsive Repair Pipe	The procedure `Completion Time` must be $<$ 24 h in 75% of the cases, and $<$ 36 h in 99.99% of the cases.
	The `Low-Pressure Risk` at any time must be $<$ 360 $\text{bar}^{1/2}$ on average.

SysML: Block Definition Diagram of the involved resources

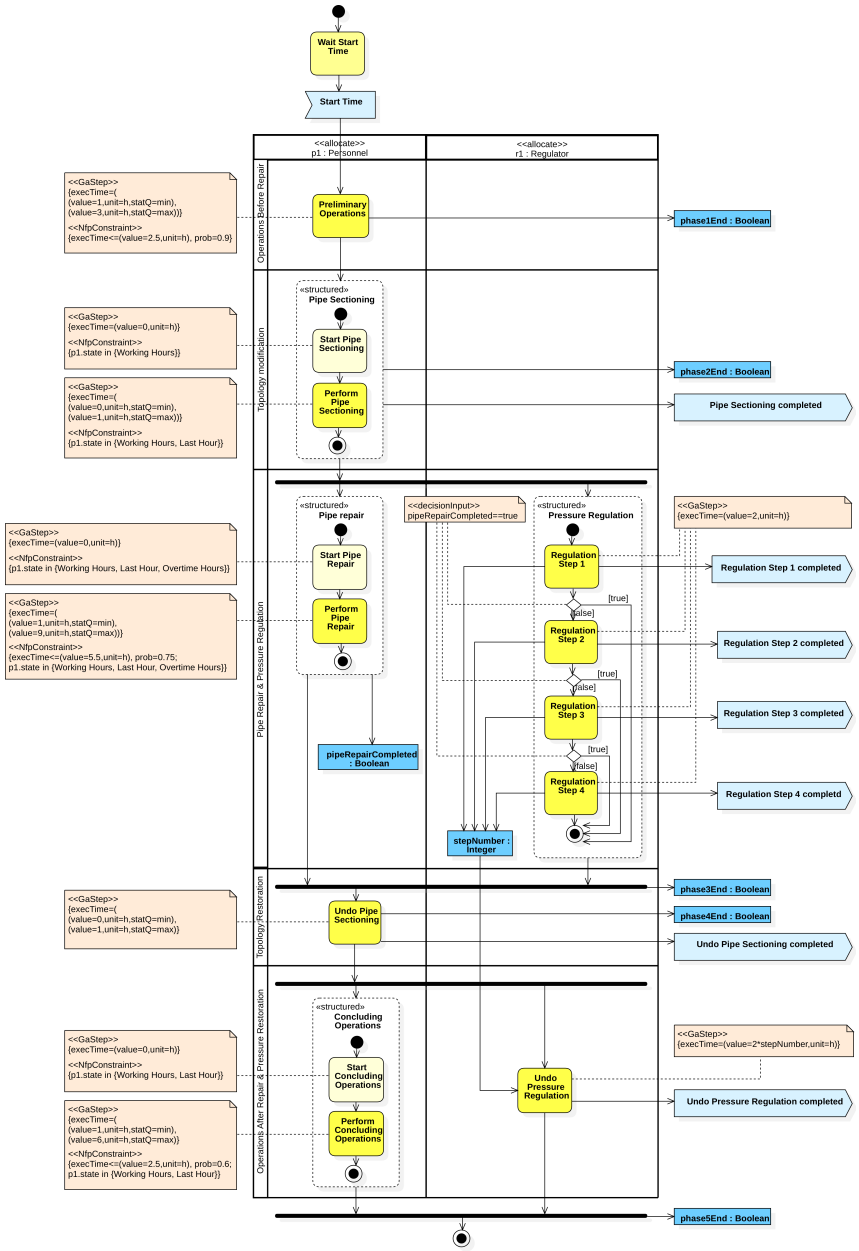


SysML: State Machine Diagrams of the network and personnel





SysML: Activity Diagram of the repair procedure



Credits

Credits

- Most of materials presented in these slides is taken from the OMG SysML tutorial: `http://www.omgsysml.org/INCOSE-OMGSysML-Tutorial-Final-090901.pdf`
- Part of materials presented in these slides is taken from the slides of the course "Embedded Systems - Model-Based Design" given by Prof. Marco Di Natale: `http://retis.sssup.it/~marco/teaching/embeddedsystems`