Lesson 3

Diagrams and Models including Sysml

Case 3

This system was developed to handle case files for the FBI. It was eventually abandoned while still in the development stage, after costing 170 million dollars. It was considered so poorly designed and inadequate as to be completely unusable in real world conditions. It failed even the most basic systems and failed to meet basic requirements. A detailed report regarding the project's failure listed several problems including:

1. Repeated changes to specifications.
2. Poor architectural decisions.
3. Scope creep.
4. Managers of the project with little or no computer science training.

What is SysML?

- A graphical modelling language in response to the UML for Systems Engineering RFP developed by the OMG, INCOSE, and AP233
  - a UML Profile that represents a subset of UML 2 with extensions
- Supports the specification, analysis, design, verification and validation of systems that include hardware, software, data, personnel, procedures, and facilities
- Supports model and data interchange via XMI and the evolving AP233 standard (in-process)

SYSML Summary

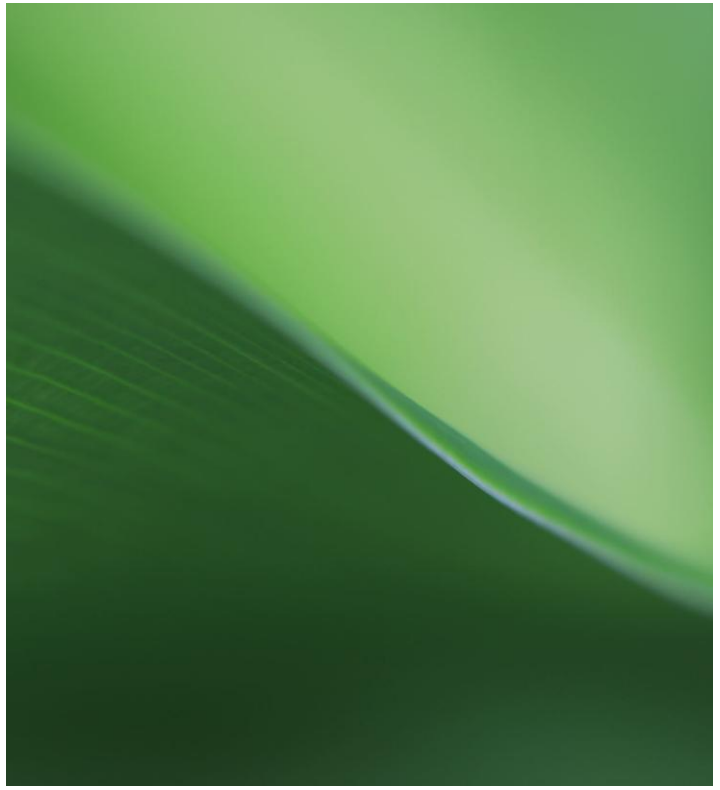
Structure
system hierarchy, interconnection

Behavior
function-based behavior, state-based behavior

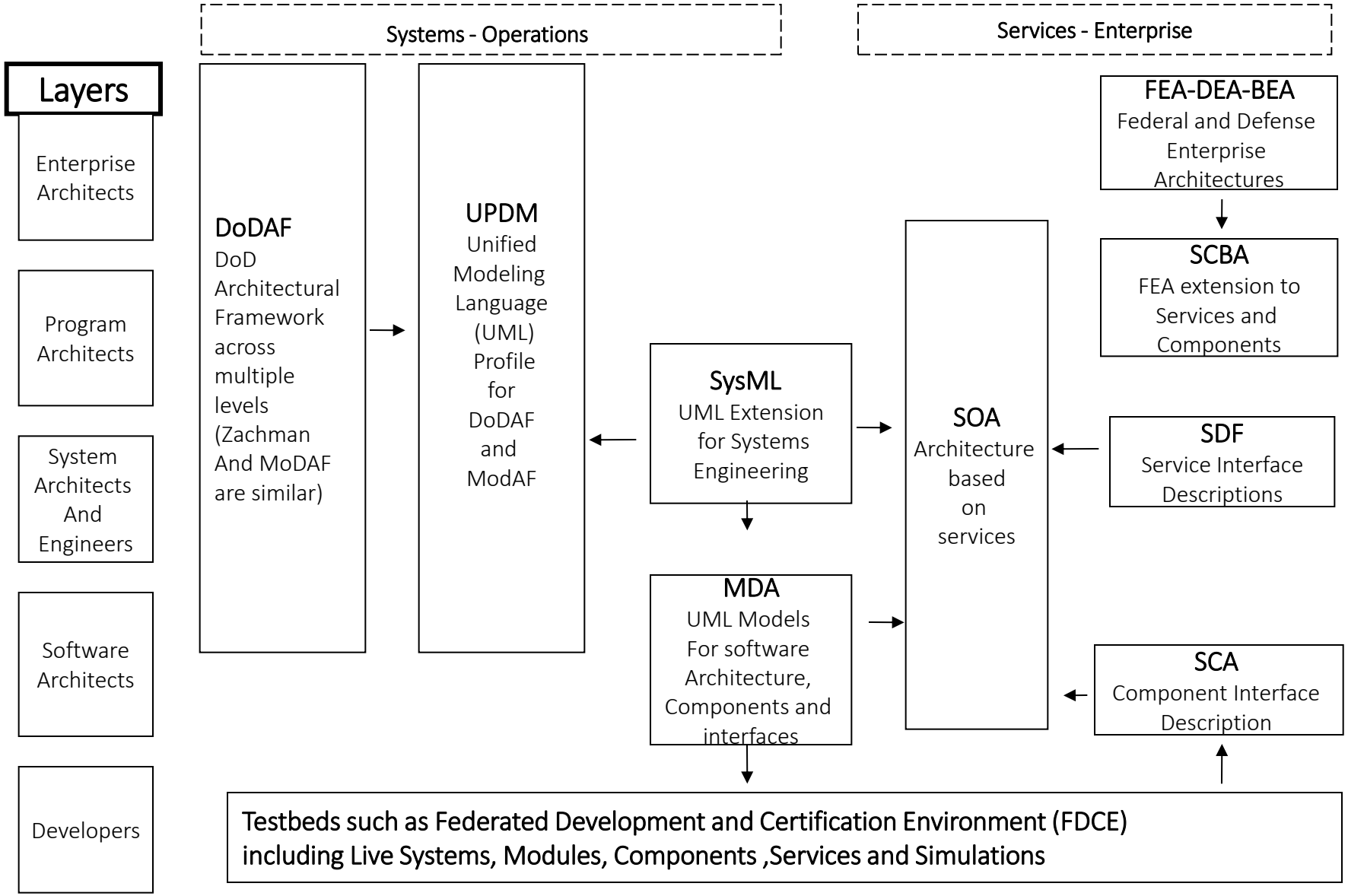
Properties
parametric models, time property

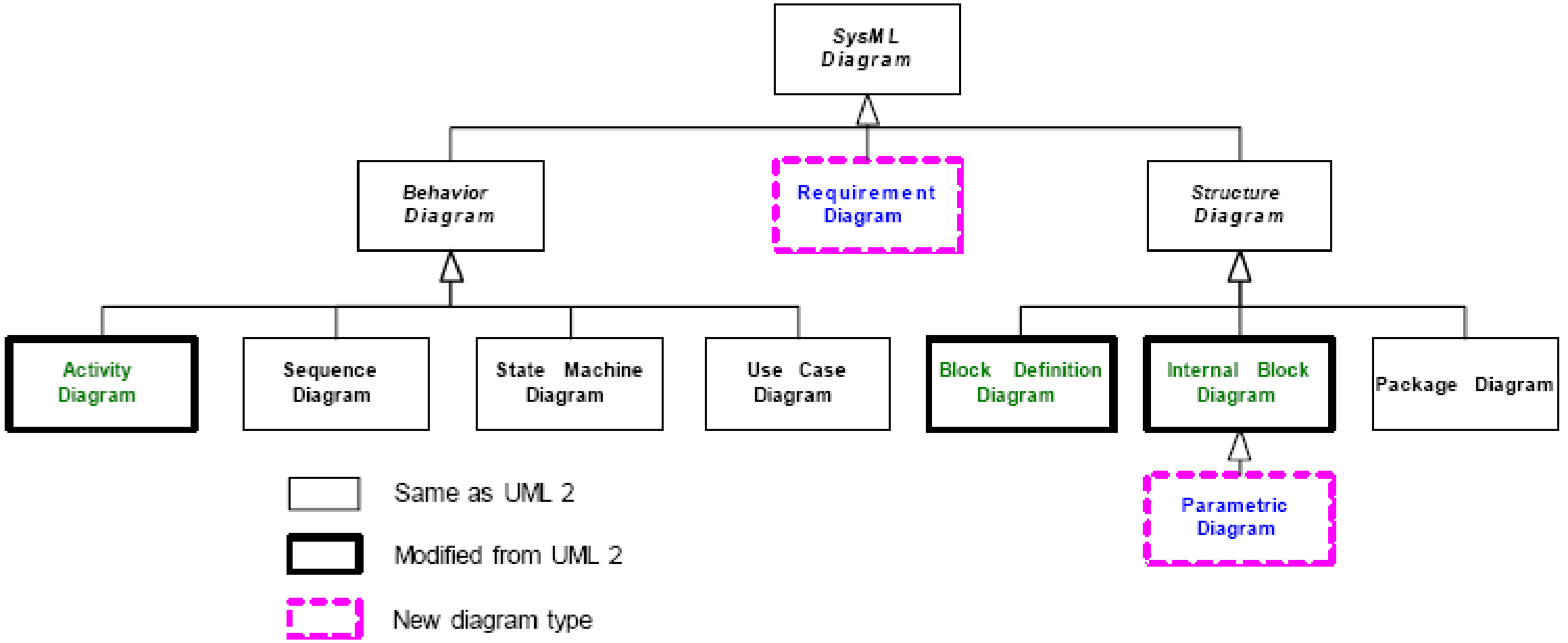
Requirements
requirements hierarchy, traceability

Verification
test cases, verification results



SysML Context





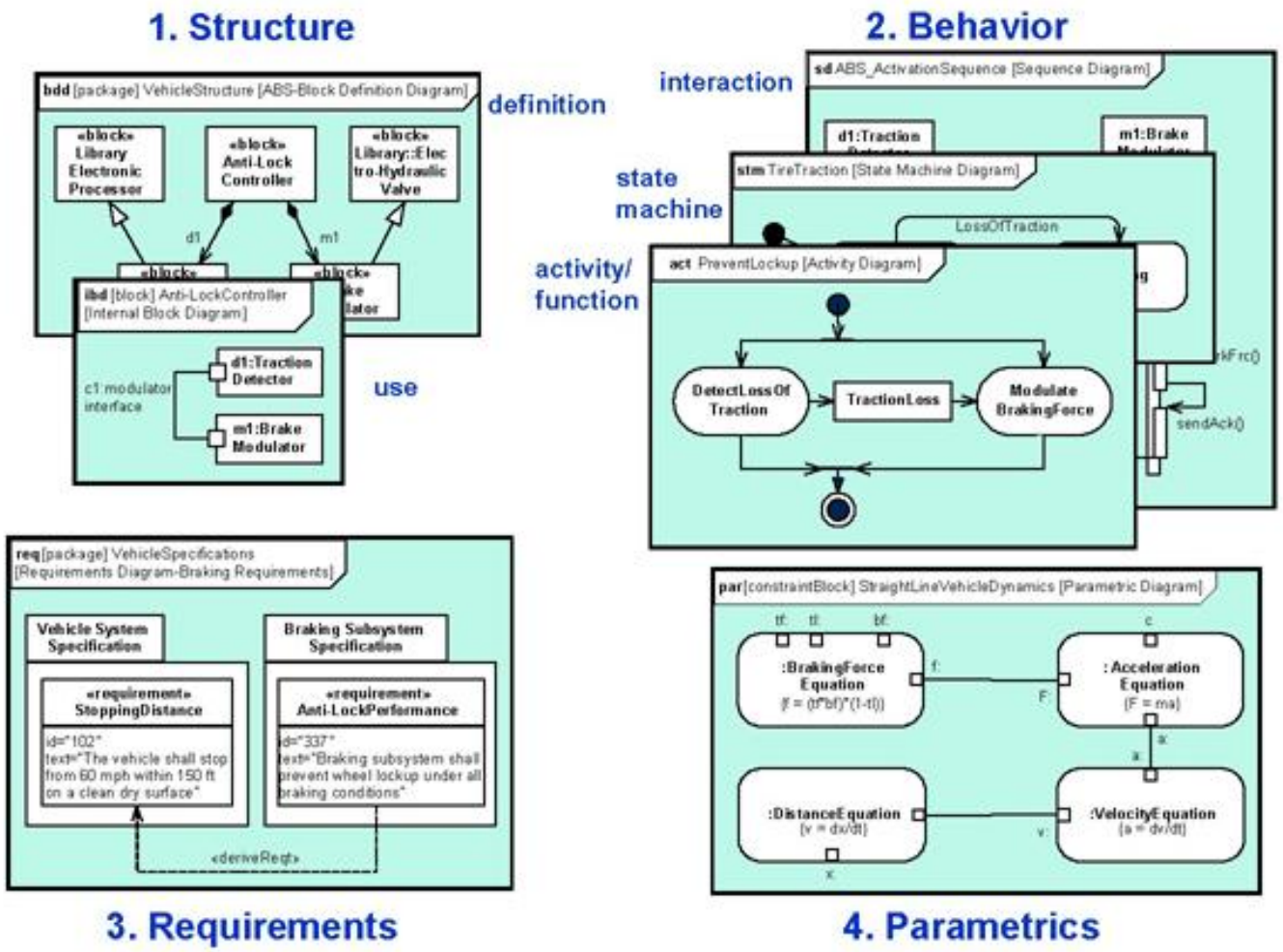
SYSML DIAGRAM TAXONOMY

SYSML DIAGRAMS

- Requirement diagram represents text-based requirements and their relationship with other requirements, design elements, and test cases to support requirements traceability (not in UML)
- Activity diagram represents behavior in terms of the order in which actions execute based on the availability of their inputs, outputs, and control, and how the actions transform the inputs to outputs (modification of UML activity diagram)
- Sequence diagram represents behavior in terms of a sequence of messages exchanged between systems, or between parts of systems (same as UML sequence diagram)
- State machine diagram represents behavior of an entity in terms of its transitions between states triggered by events (same as UML state machine diagram)
- Use case diagram represents functionality in terms of how a system is used by external entities (i.e., actors) to accomplish a set of goals (same as UML use case diagram)
- Block definition diagram represents structural elements called blocks, and their composition and classification (modification of UML class diagram)
- Internal block diagram represents interconnection and interfaces between the parts of a block
- (modification of UML composite structure diagram)
- Parametric diagram represents constraints on property values, such as F [m * a, used to support engineering analysis (not in UML)

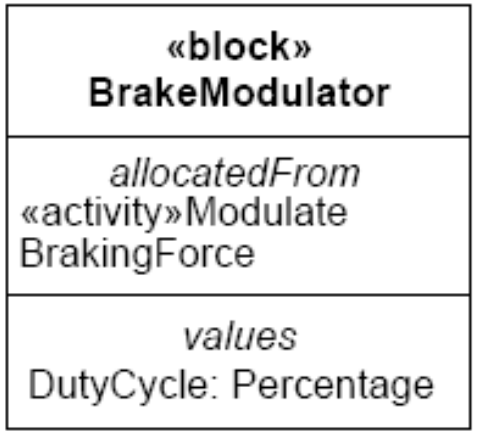
The four pillars of sysml

https://www.omgsysml.org/what-is-sysml.htm



Blocks are Basic Structural Elements

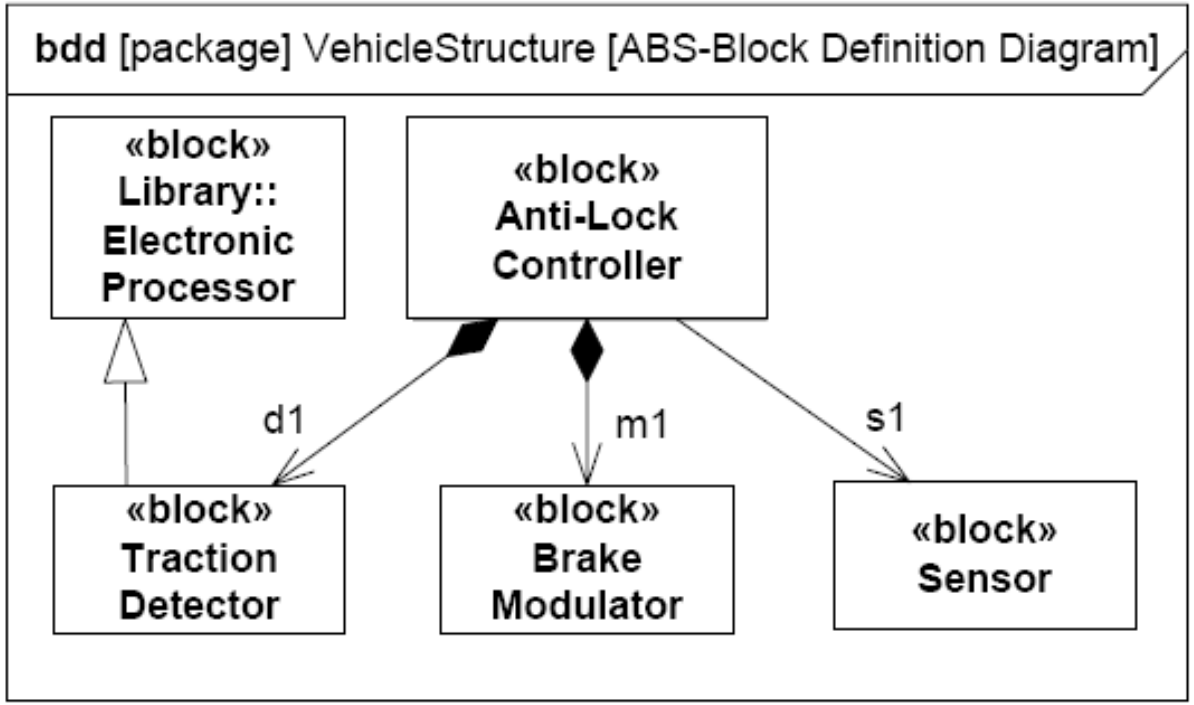
- Provides a unifying concept to describe the structure of an element or system
  - Hardware
  - Software
  - Data
  - Procedure
  - Facility
  - Person



- Multiple compartments can describe the block characteristics
  - Properties (parts, references, values)
  - Operations
  - Constraints
  - Allocations to the block (e.g. activities)
  - Requirements the block satisfies

Block Definition Diagram

Block Definition Diagram



Definition

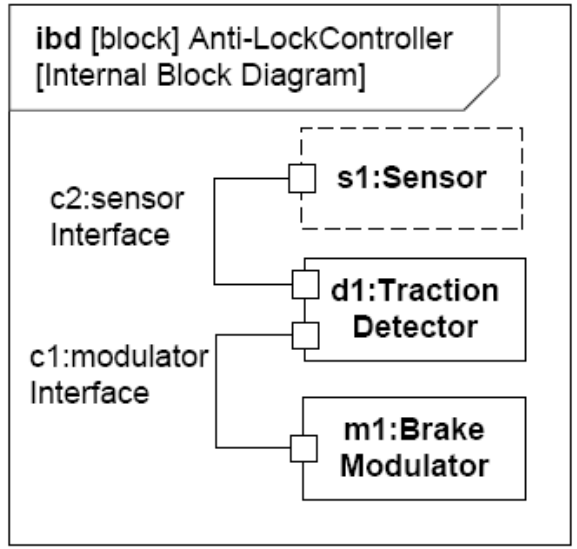
- Block is a definition/type
- Captures properties, etc.
- Reused in multiple contexts

Internal Block Diagram

Usage

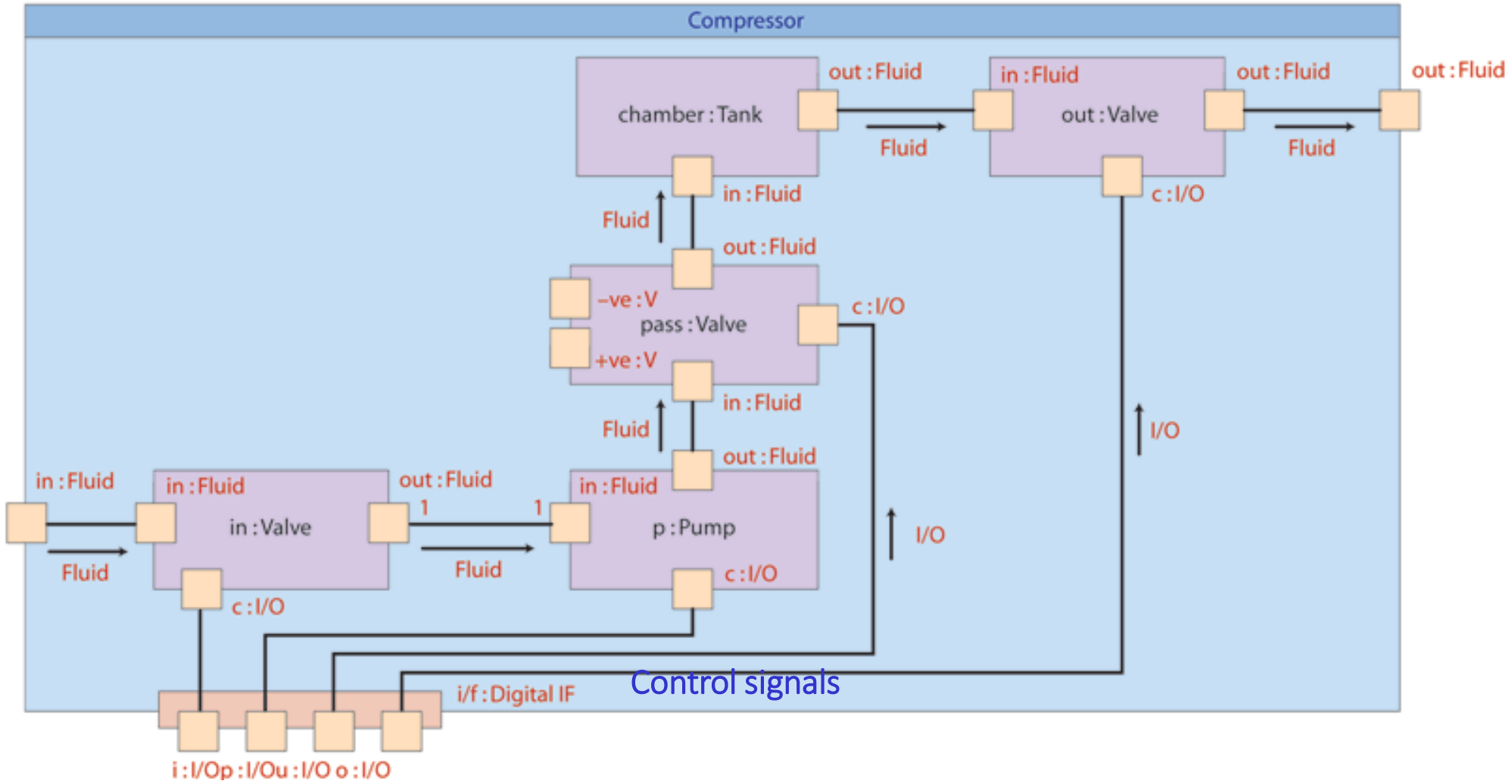
- Part is the usage in a particular context
- Typed by a block
- Also known as a role

Internal Block Diagram

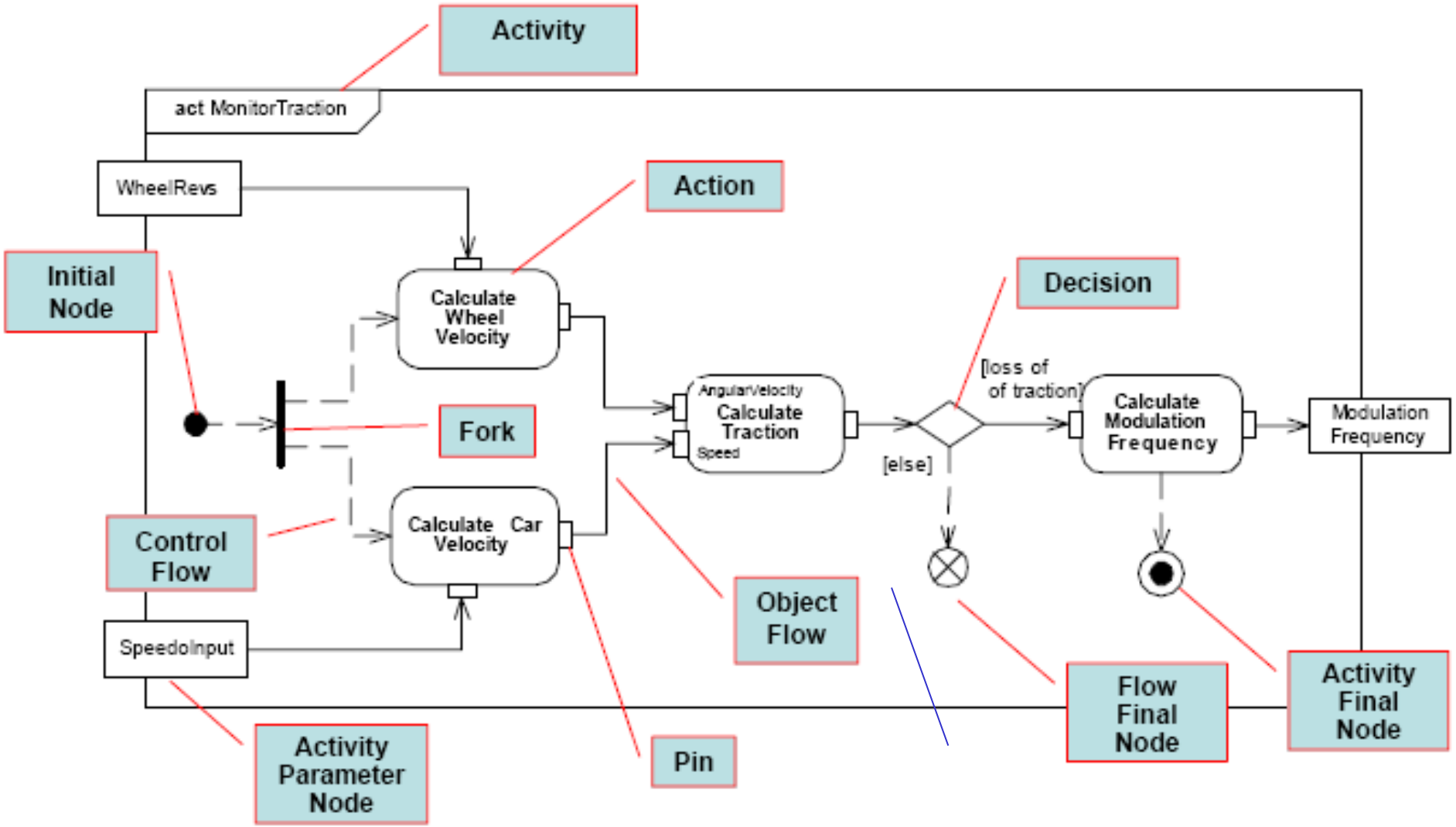


Internal Block Diagram Example

Compressor



Activity Diagram Notation



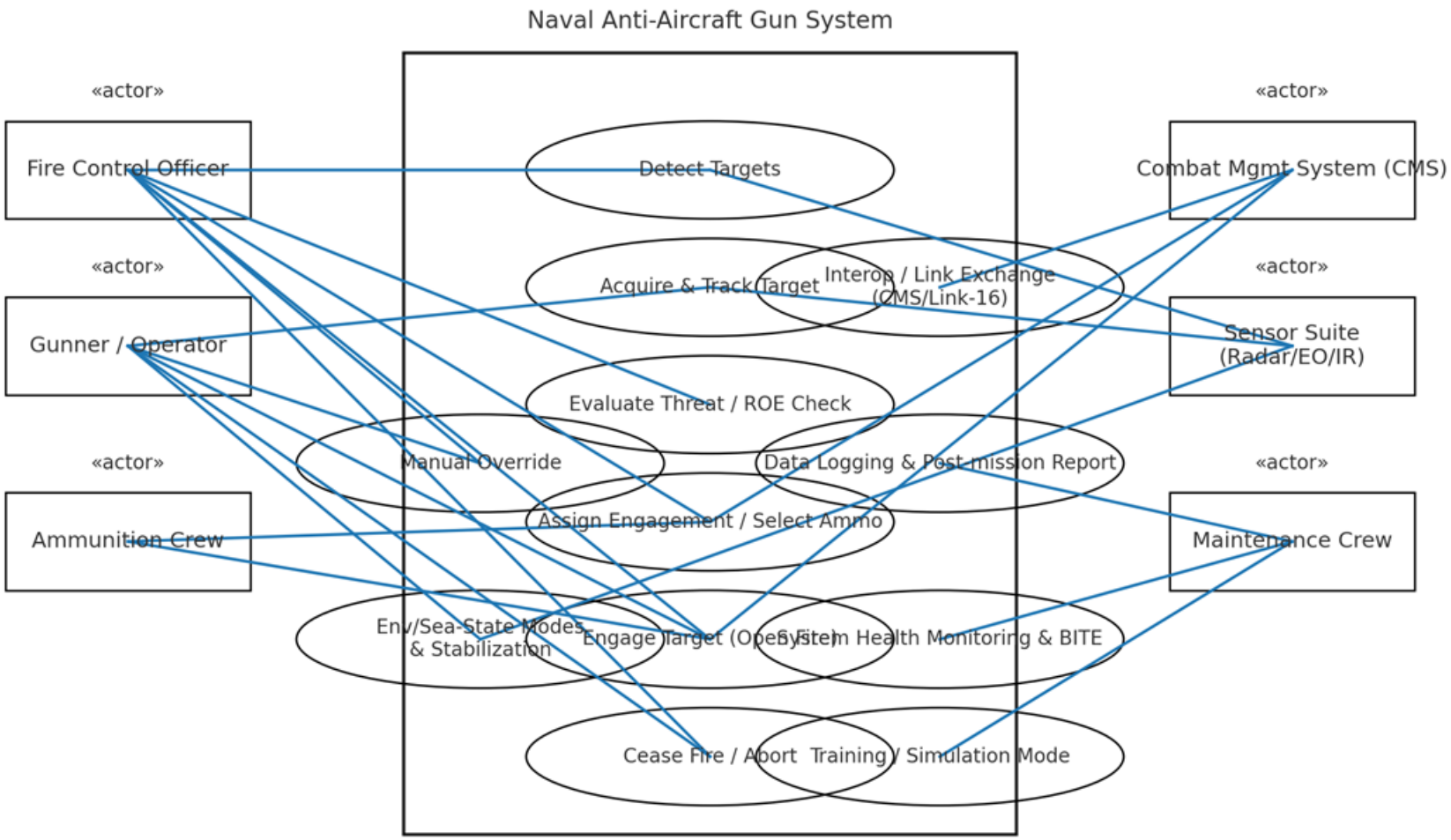
Modeling methodology

Modeling starts at the highest level of the system by modeling the different top-level components.

Next, a behavioral SysML use case diagram is presented, to determine the different case scenarios present in the system.

Following that, low level components and illustrate their relative aspects, such as behavior expressed via SysML diagrams such as state and sequence diagrams.

Use Case Diagram (Naval based anti aircraft gun)



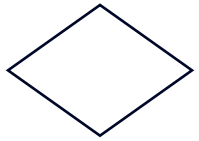
Activities

- Activity diagrams are graphical representations of workflows of stepwise activities and actions
- Activity used to specify the flow of inputs/outputs and control, including sequence and conditions for coordinating activities
- Secondary constructs show responsibilities for the activities using swim lanes
- SysML extensions to Activities
  - Support for continuous flow modeling
  - Support probabilistic choice
  - Alignment of activities with Enhanced Functional Flow Block Diagram

Activity Diagram

Rounded Rectangles represent *actions*;

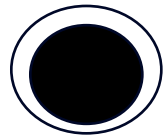
diamonds represent *decisions*;

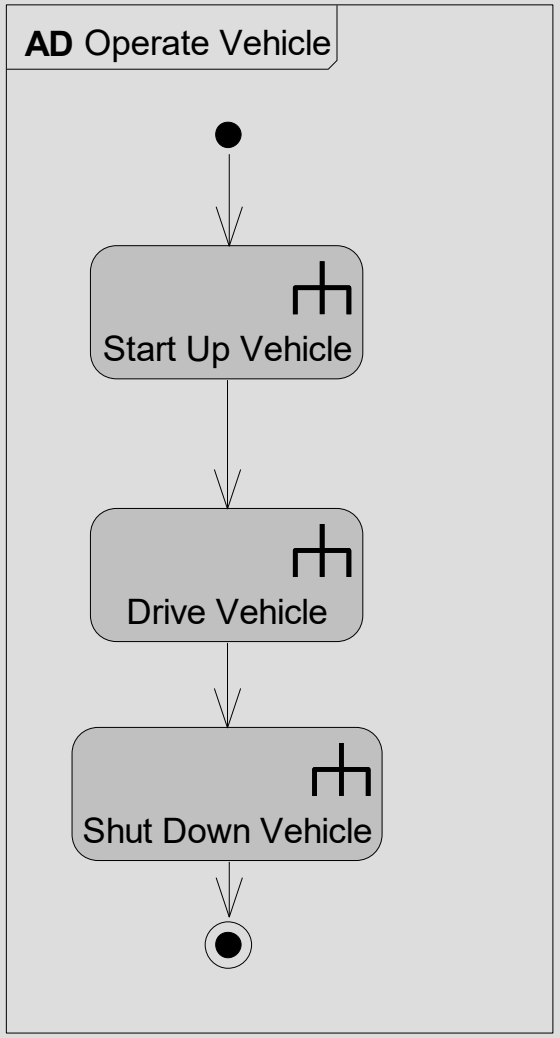


bars represent the start (*split*) or end (*join*) of concurrent activities;

a *black circle* represents the start (*initial node*) of the workflow;

an *encircled black circle* represents the end (*final node*)



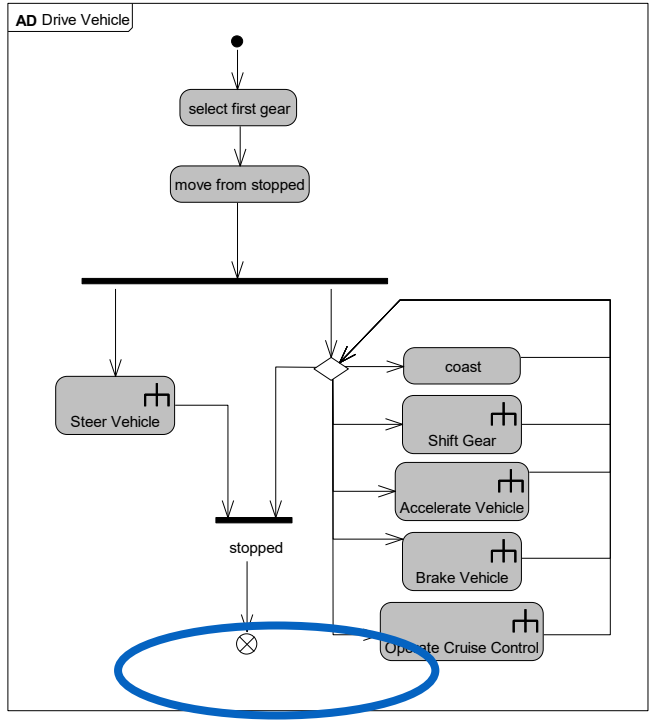


Analysis Model of Vehicle

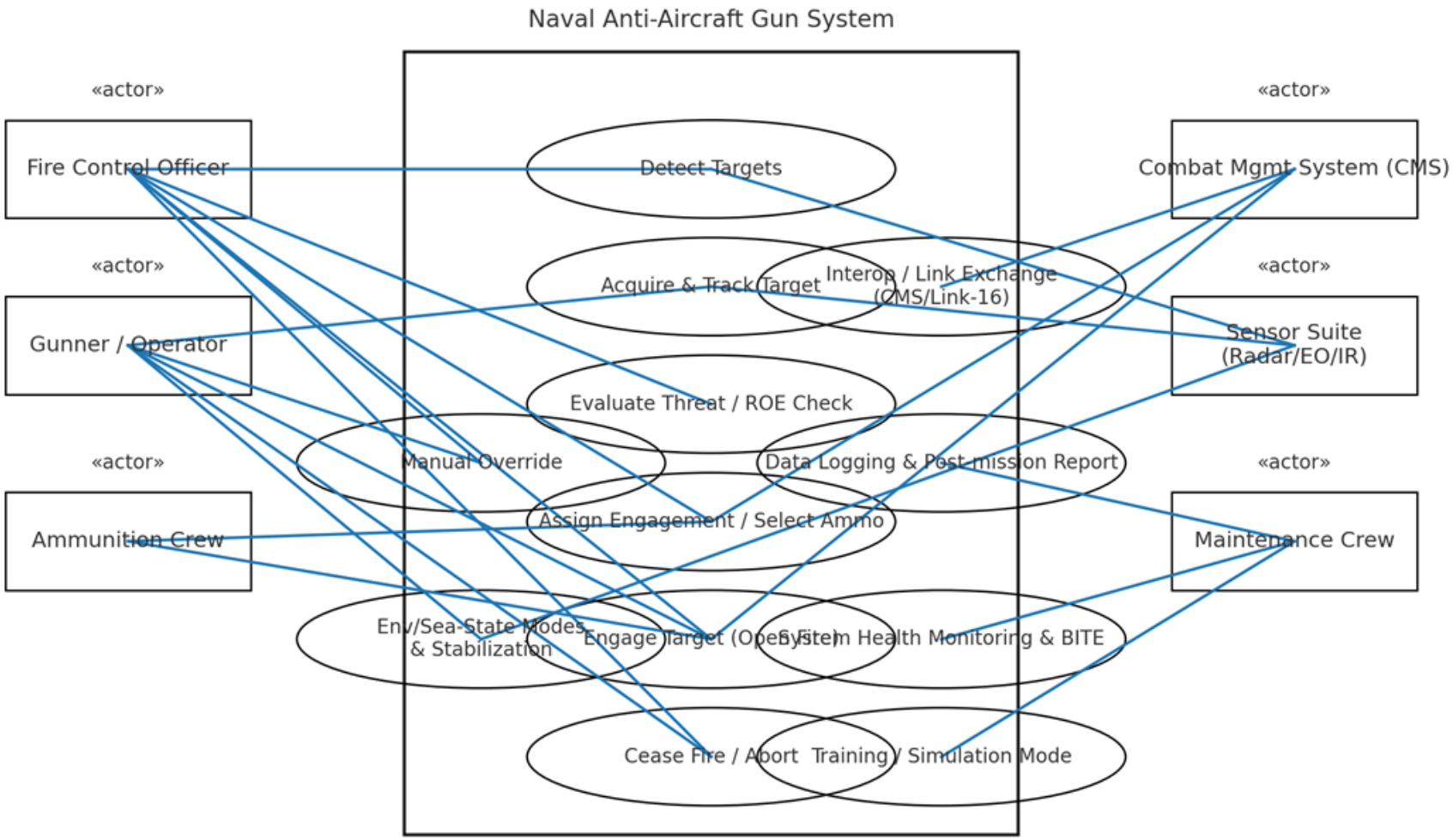
- SysML additions on this chart
  - «streaming» activities consume inputs after initialization
  - «continuous» flows

Analysis Model of Vehicle

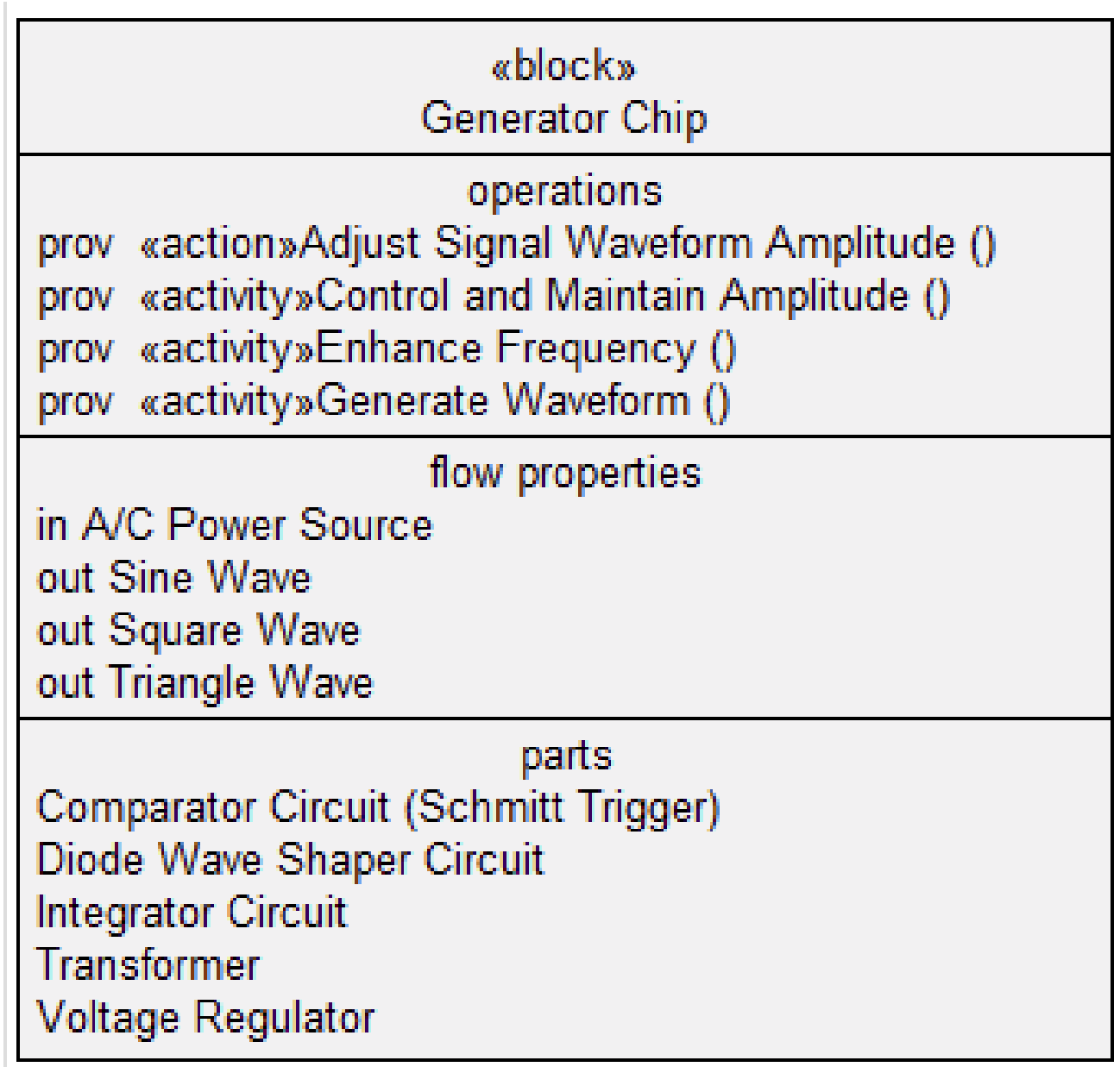
- SysML additions on this chart
  - «streaming» activities consume inputs after initialization
  - «continuous» flows



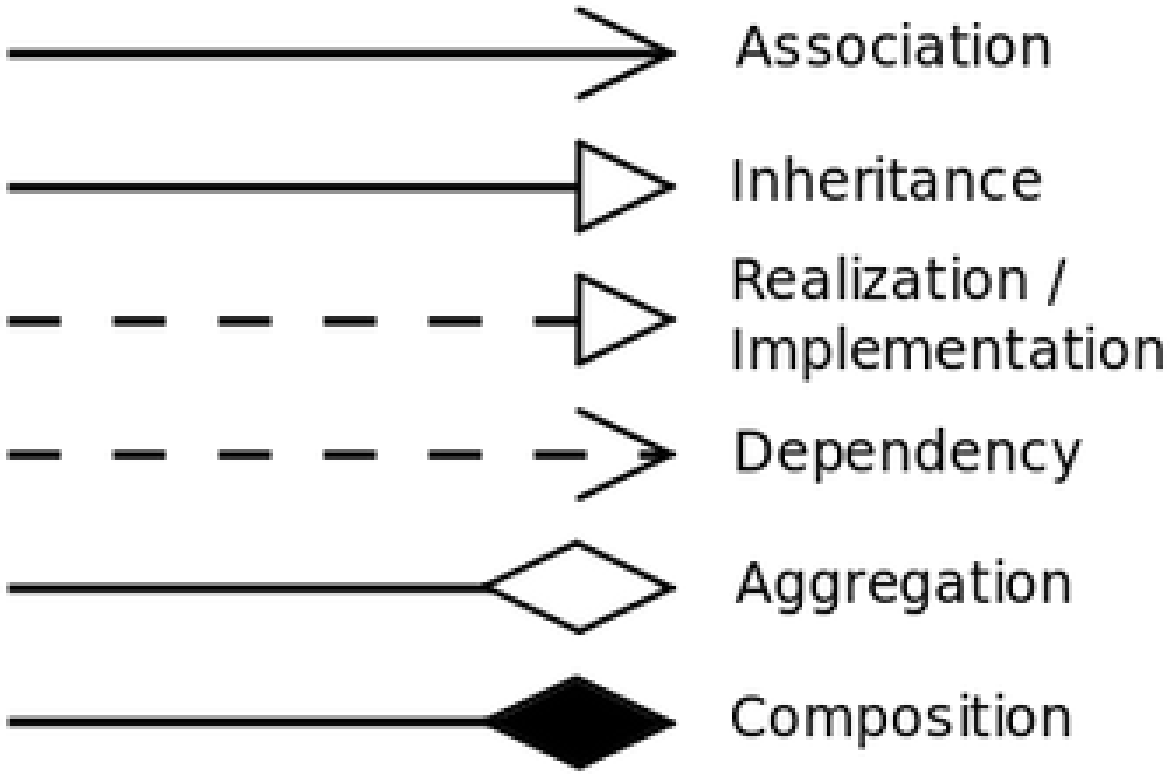
Block definition diagram



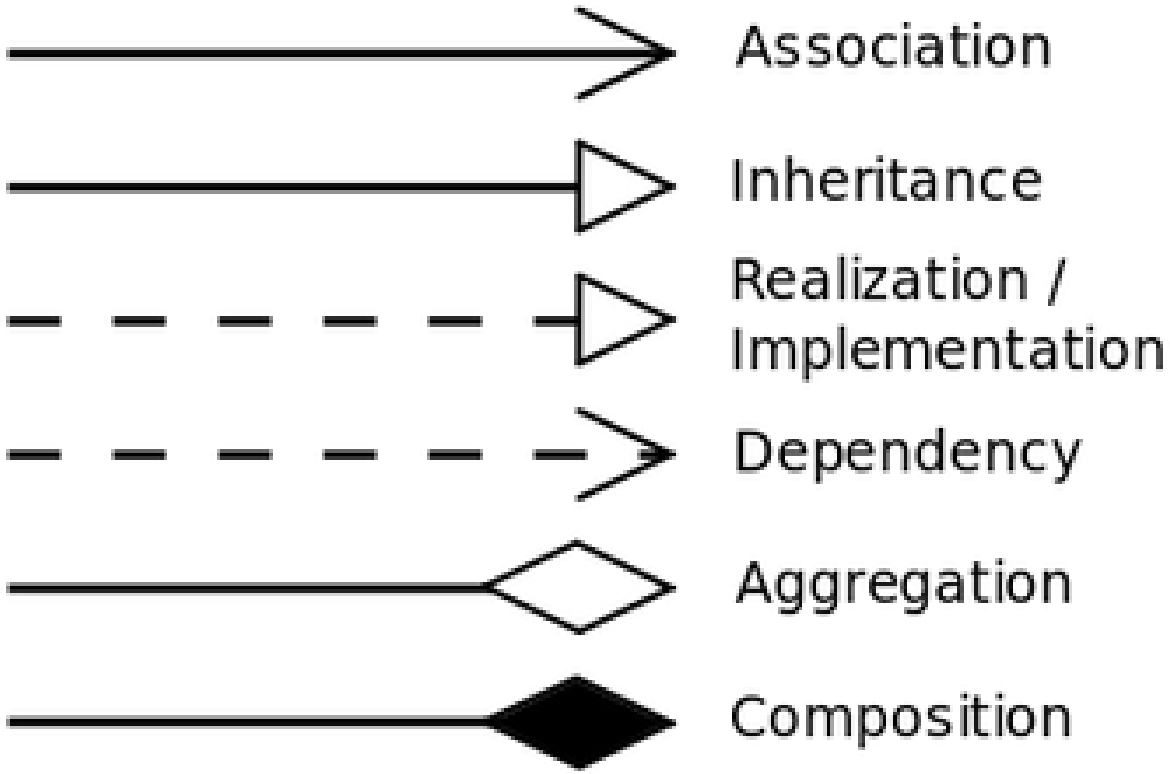
BLOCK DEFINITION DIAGRAM



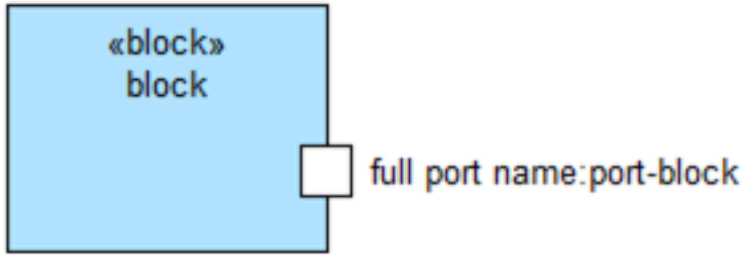
ELEMENTS


Association
Inheritance
Realization /
Implementation
Dependency
Aggregation
Composition

ELEMENTS

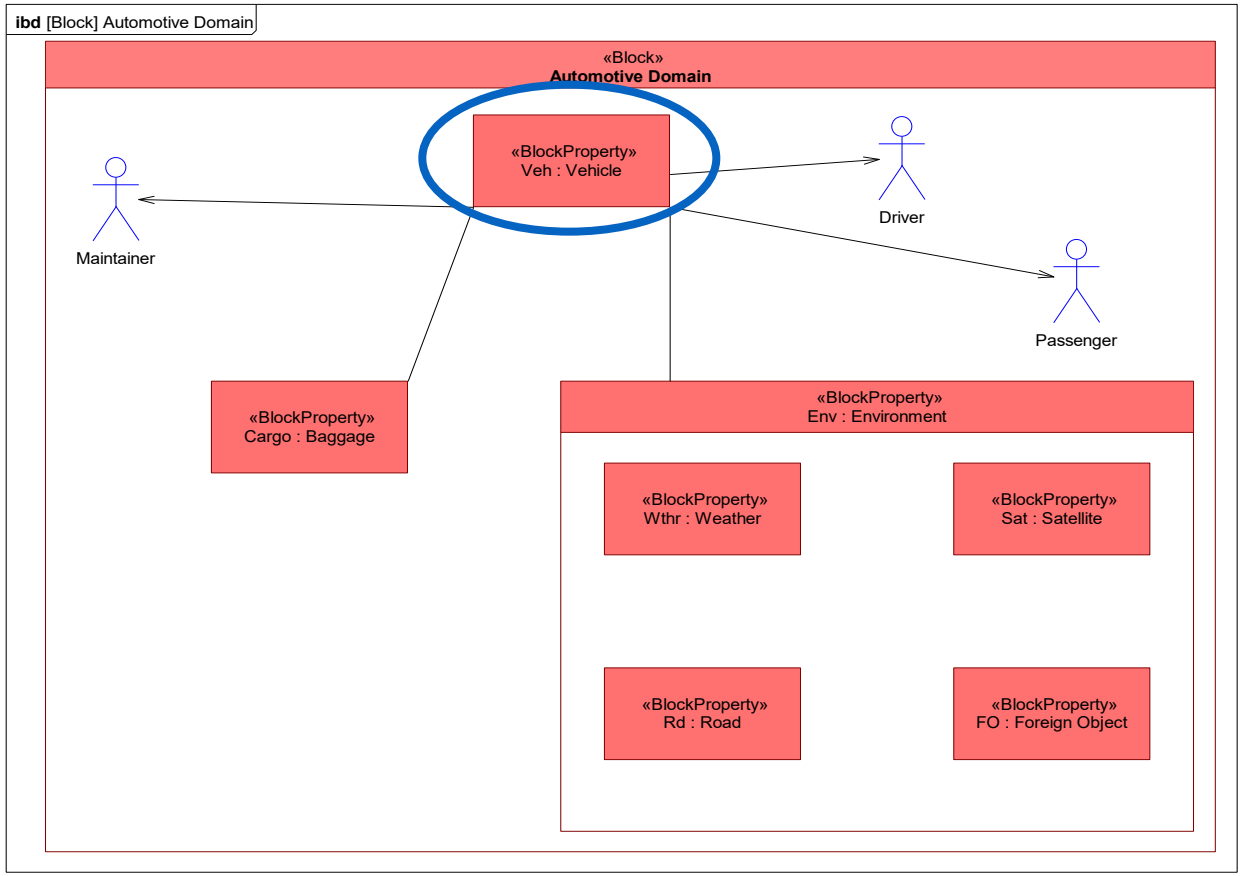

Association
Inheritance
Realization /
Implementation
Dependency
Aggregation
Composition

PORTS



Block	«block»
Full Port	«fullPort»
Proxy Port	«proxyPort»
Interface Block	«interfaceBlock»

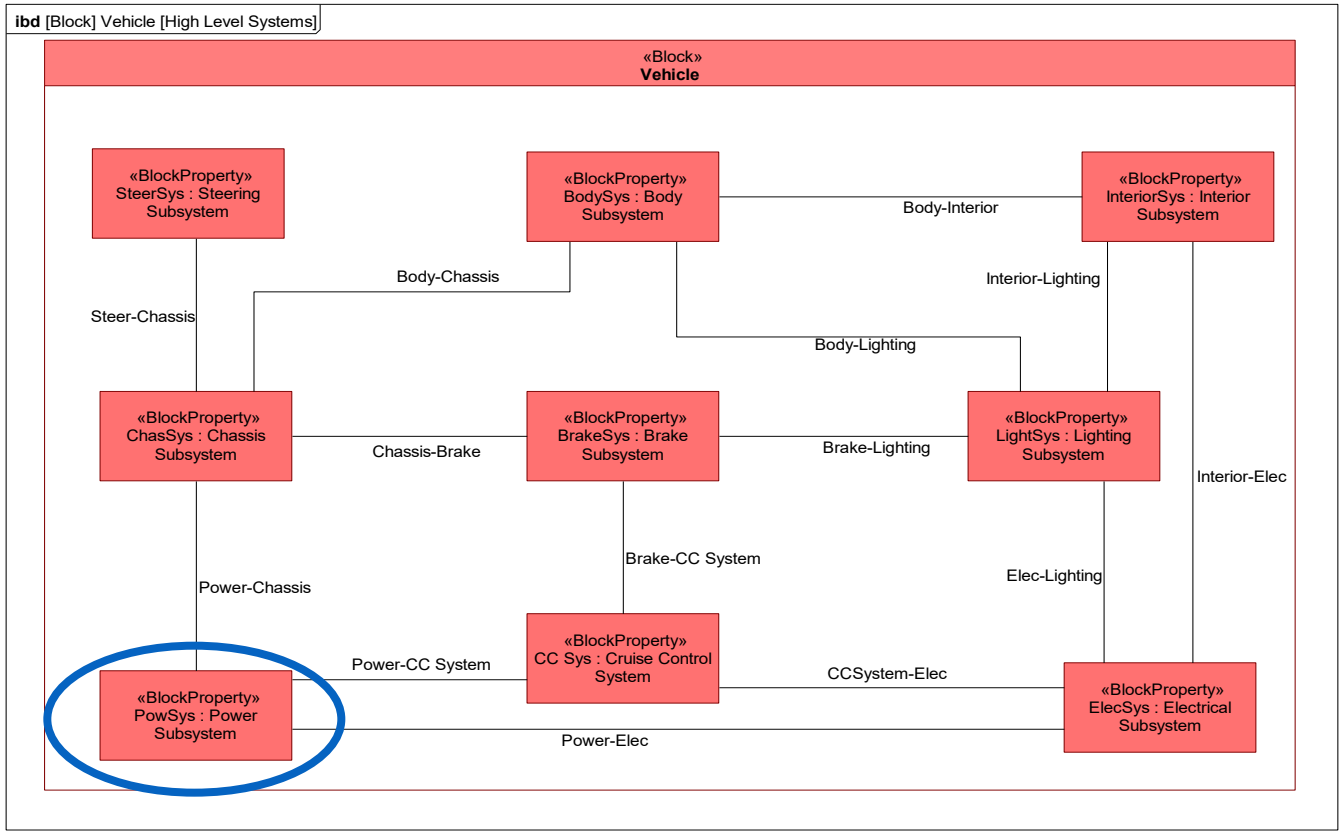
Internal Block Diagram for Vehicle



- Non-Atomic Ports
  - I/O is specified using FlowSpecification
  - FlowSpecification consists of properties stereotyped «FlowProperty»
  - isConjugate promotes reuse of flowSpecifications

- Atomic FlowPorts
  - In this case the port is directly typed by the item type (Block or ValueType)
  - Direction property specify the direction of flow

IBD for Vehicle



- Non-Atomic Ports
  - I/O is specified using FlowSpecification
  - FlowSpecification consists of properties stereotyped «FlowProperty»
  - isConjugate promotes reuse of flowSpecifications

- Atomic FlowPorts
  - In this case the port is directly typed by the item type (Block or ValueType)
  - Direction property specify the direction of flow

IBD COMPONENTS

Comment


Picture

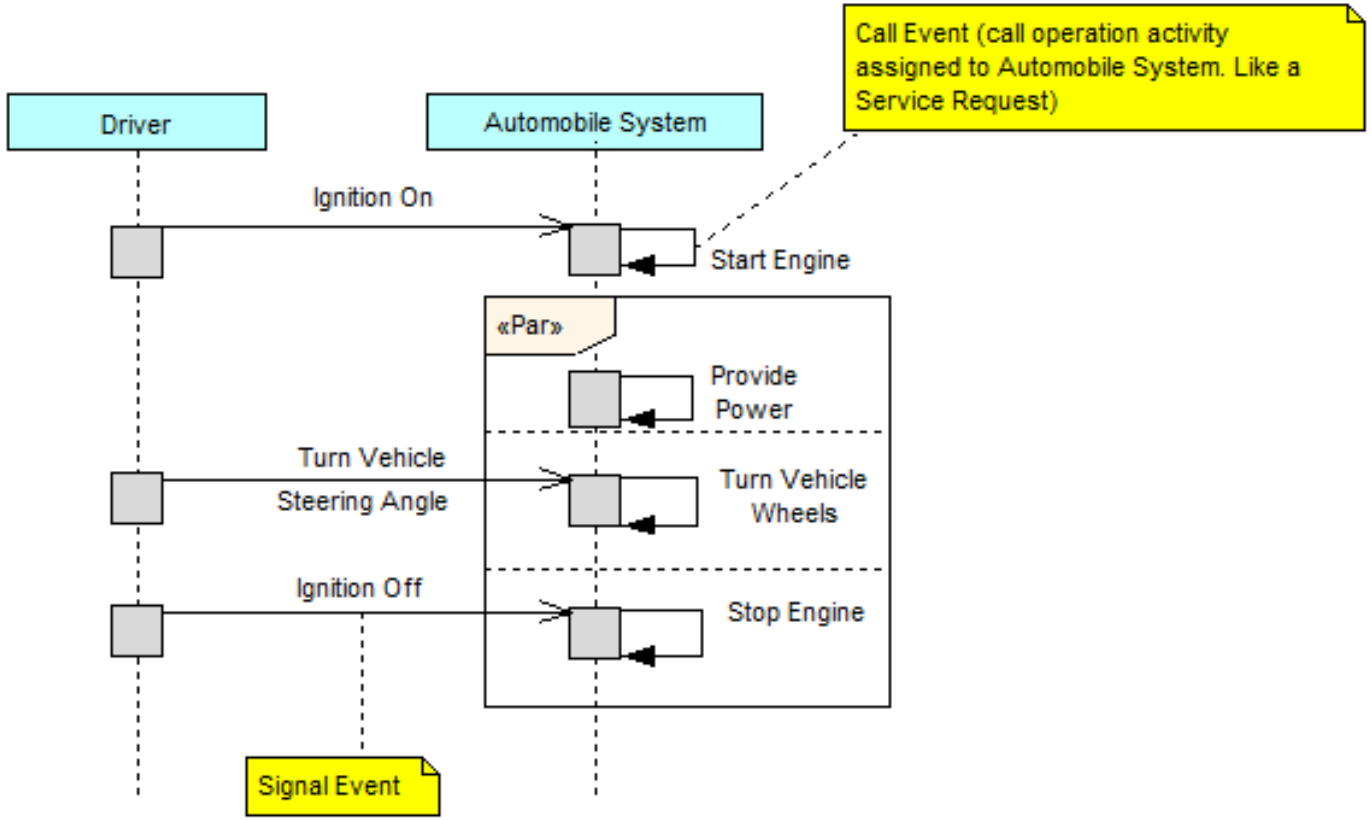

Note Callout


Tag

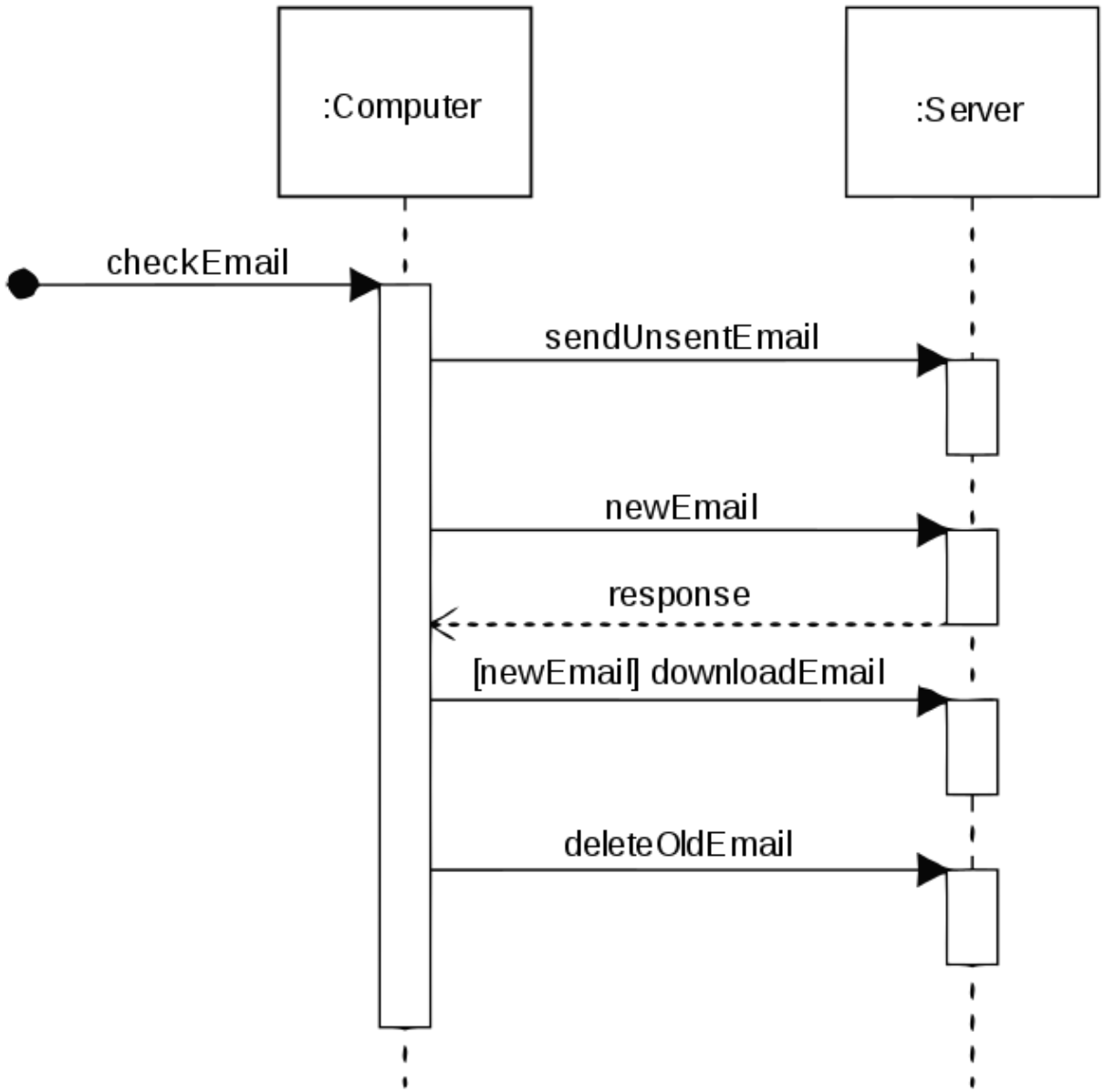

Part


Reference

SEQUENCE DIAGRAM



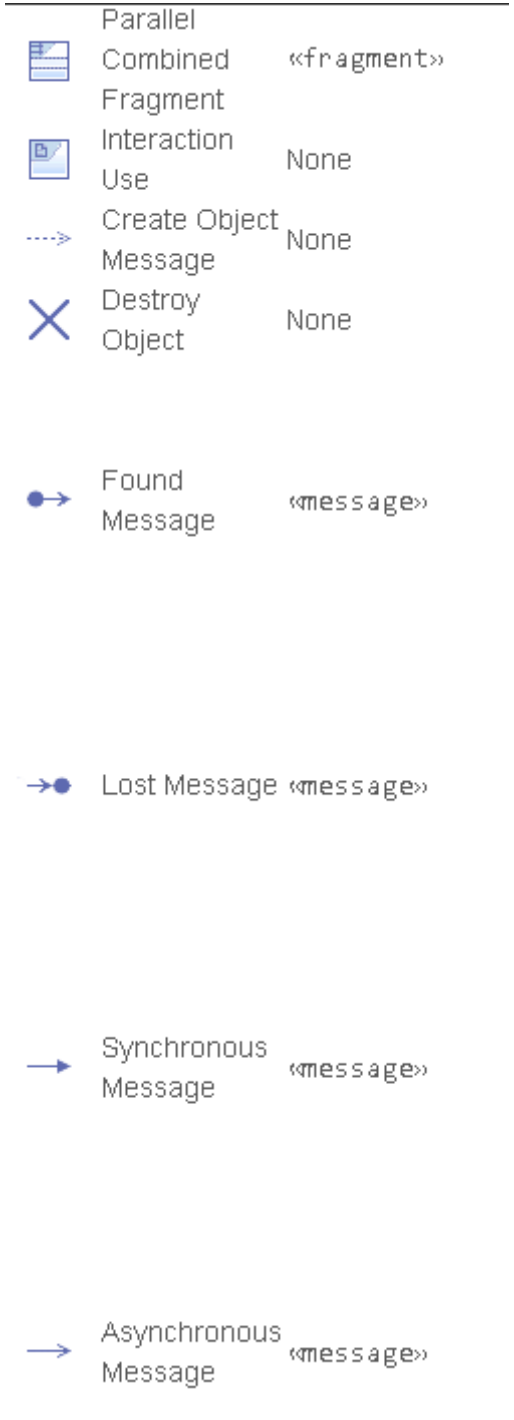
SEQUENCE DIAGRAM



SEQUENCE DIAGRAM

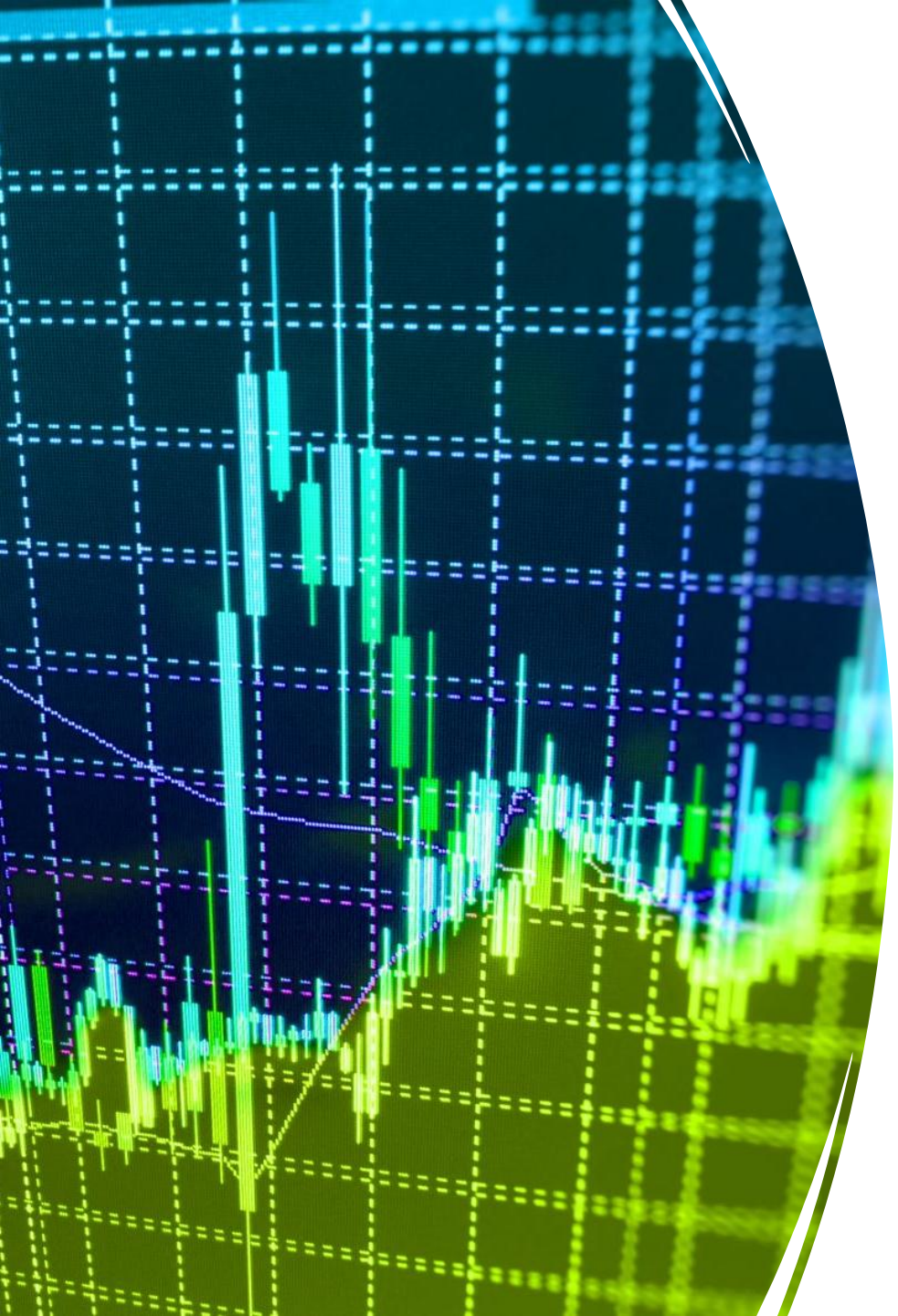
 Lifeline	«lifeLine»
 Execution Specification	None
 Alternative Combined Fragment	«fragment»
 Loop Combined Fragment	«fragment»
 Optional Combined Fragment	«fragment»
 Parallel Combined Fragment	«fragment»

SEQUENCE DIAGRAM



Modeling Approaches & Methods

- Data Modeling
  - Entity-relationship diagrams (ERDs)
  - Higraphs
- Process Modeling
  - Data flow diagrams (DFDs)
  - IDEF0
  - N^2 charts
- Behavior Modeling
  - Function flow block diagrams (FFBDs)
  - Behavior diagrams (BDs)
  - State-transition diagrams (STDs)
  - Statecharts
  - Control flow diagrams (CFDs)
  - Petri nets (PNs)
- Object-oriented Modeling
  - Object modeling technique (OMT)
  - Real-time object-oriented modeling (ROOM)



Data Flow Diagrams

visual representations of the data that moves through the organization, the paths through which the data moves, and the processes that produce, use, and transform data.

Differences Between DFDs and Flowcharts

Processes on DFDs can operate in parallel (at-the-same-time)

- Processes on flowcharts execute one at a time

DFDs show the flow of data through a system

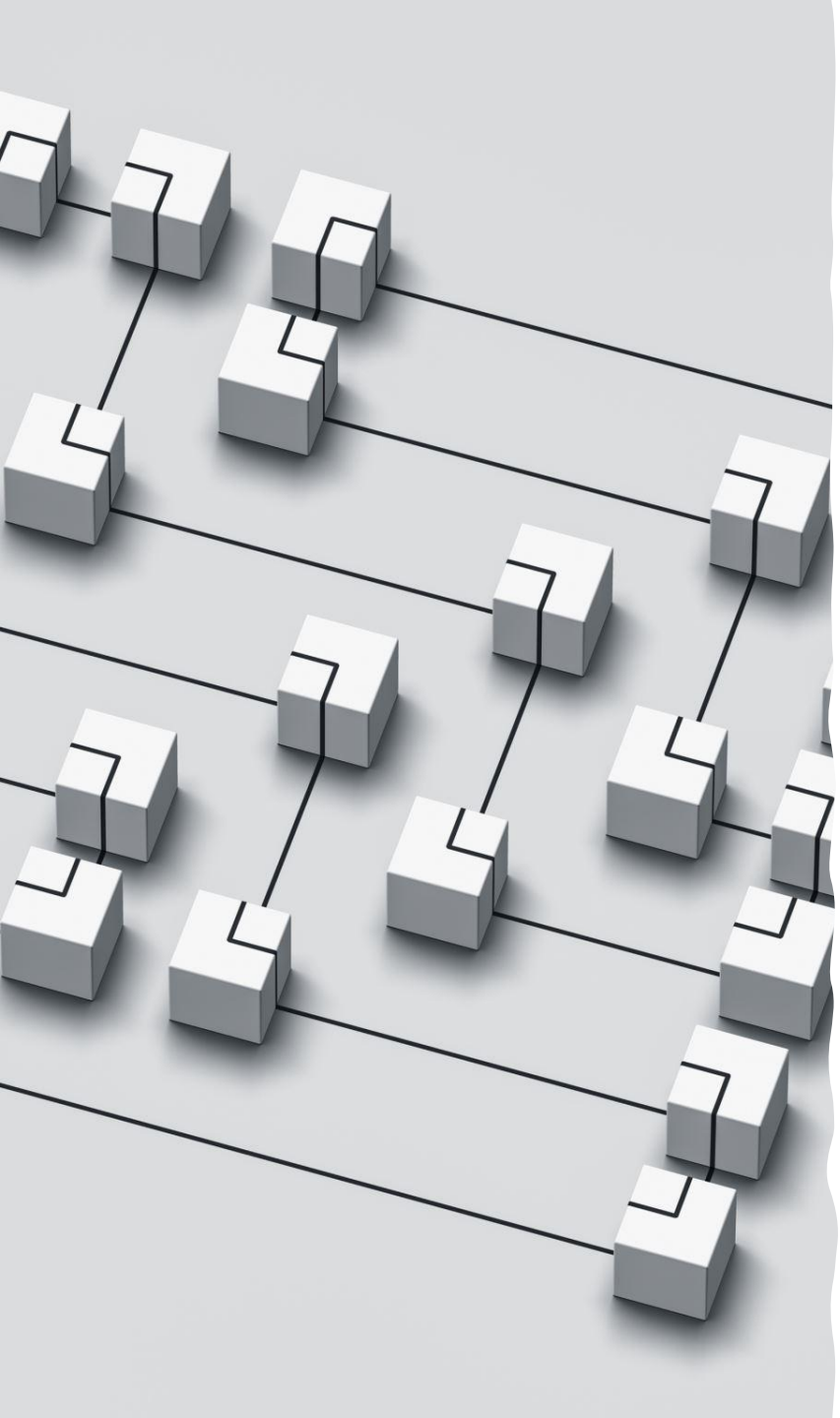
- Flowcharts show the flow of control (sequence and transfer of control)

Processes on a DFD can have dramatically different timing (daily, weekly, on demand)

- Processes on flowcharts are part of a single program with consistent timing

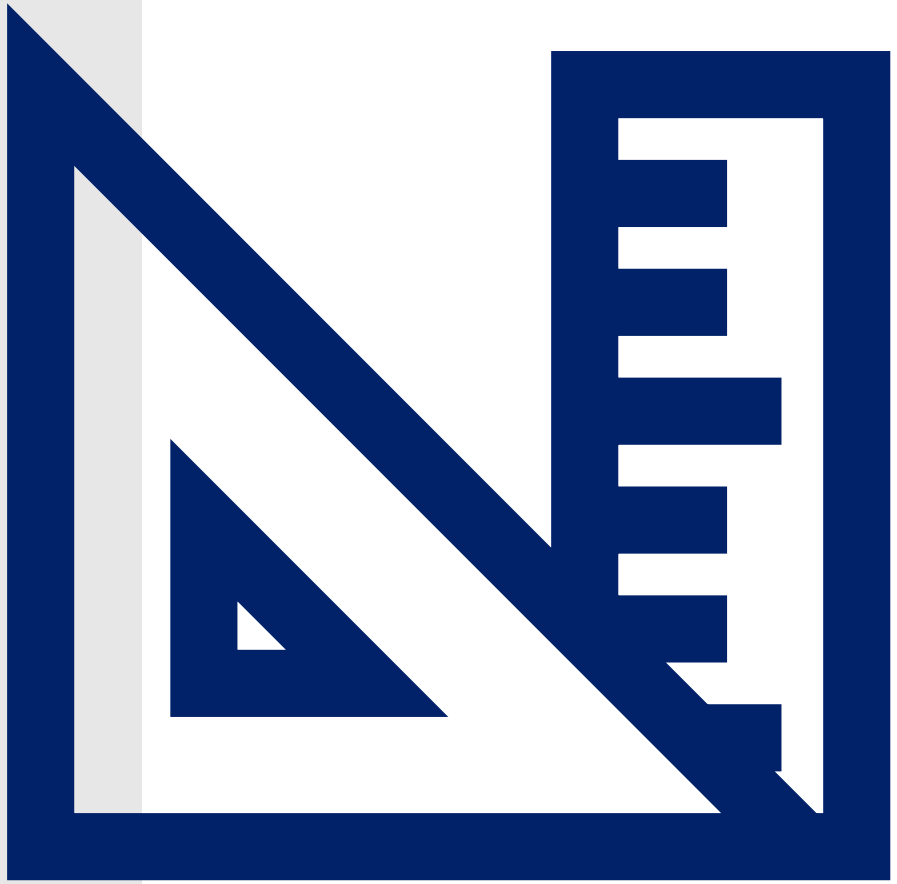
Types of DFDs

- **Current** - how data flows now
- **Proposed** - how it is intended to flow in the future
- **Logical** – The flow of the data
- **Physical** – The flow of the physical system
- **Partitioned physical** - system architecture or high-level design



Levels of Detail

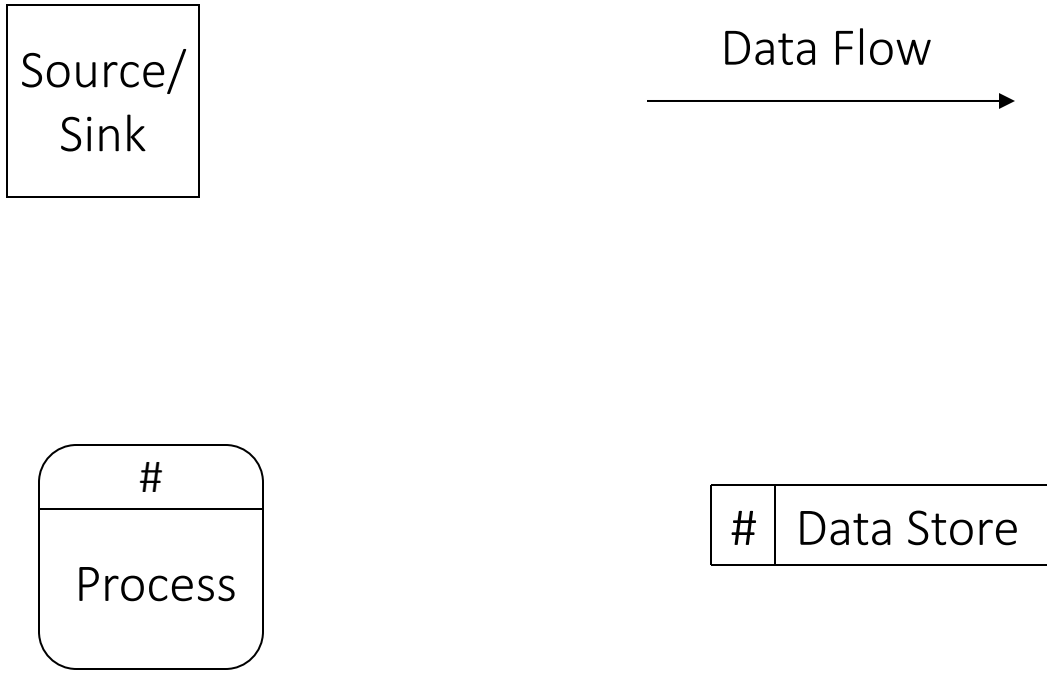
- **Context level diagram** - shows just the inputs and outputs of the system
- **Level 0 diagram** - decomposes the process into the major subprocesses and identifies what data flows between them
- **Child diagrams** - increasing levels of detail
- **Primitive diagrams** - lowest level of decomposition



Recommended Progression

- Current logical diagrams
  - start with context level
  - decompose as needed for understanding
- Proposed logical diagrams
  - start at level where change takes place
  - decompose as far as possible
- Current physical diagrams
  - at level of change
- Proposed physical diagrams
  - same levels as proposed logical
  - lower levels become design

Four Basic Symbols



DFD Mechanics

- Data Flow
  - Depicts data that are in motion and moving as a unit from one place to another in the system.
  - Drawn as an arrow
  - Select a meaningful name to represent the data

DFD Mechanics

- Data Store
  - Depicts data at rest
  - May represent data in
    - File folder
    - Computer-based file
    - Notebook
  - Drawn as two horizontal parallel lines
  - The name of the store as well as the number are recorded in between lines

DFD Mechanics

- Process
  - Depicts work or action performed on data so that they are transformed, stored or distributed
  - Drawn as a circle
  - Number of process as well as name are recorded

DFD Mechanics

- Depicts the origin and/or destination of the data
- Sometimes referred to as an external entity
- Drawn as a square symbol
- Name states what the external agent is
- Because they are external, many characteristics are not of interest to us

DFD Syntax

- Context Diagram
  - A data flow diagram (DFD) of the scope of an organizational system that shows the system boundaries, external entities that interact with the system and the major information flows between the entities and the system
- Level-O Diagram
  - A data flow diagram (DFD) that represents a system's major processes, data flows and data stores at a higher level

Context Level Diagram

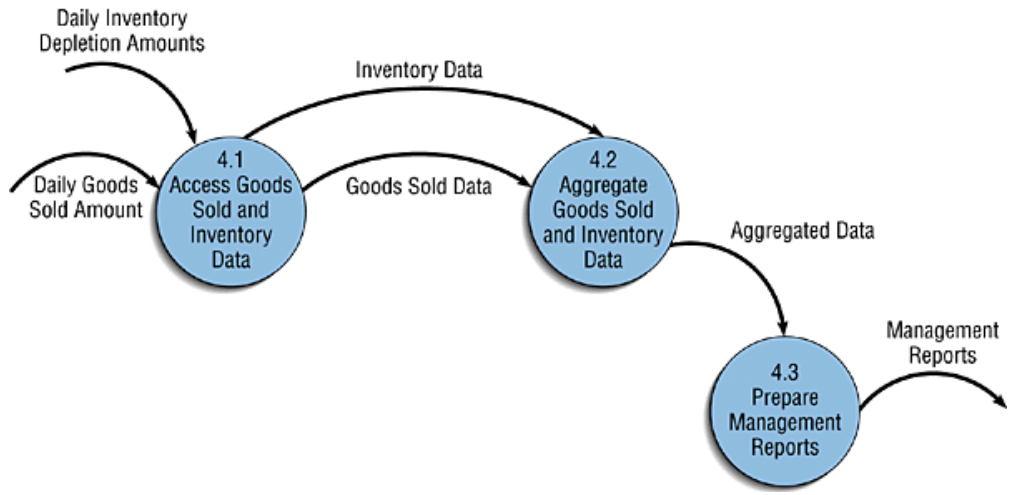
- Just one process
- All sources and sinks that provide data to or receive data from the process
- Major data flows between the process and all sources/sinks
- No data stores

Decomposition of DFDs

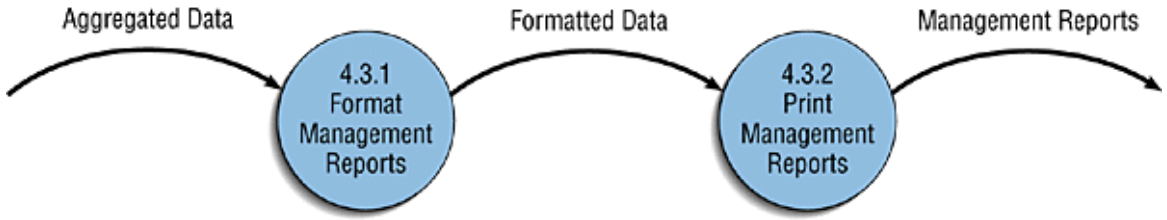
- Functional decomposition
  - Act of going from one single system to many component processes
  - Repetitive procedure
  - Lowest level is called a primitive DFD
- Level-N Diagrams
  - A DFD that is the result of *n* nested decompositions of a series of sub processes from a process on a level-0 diagram

Levels

Level 1



Level 2

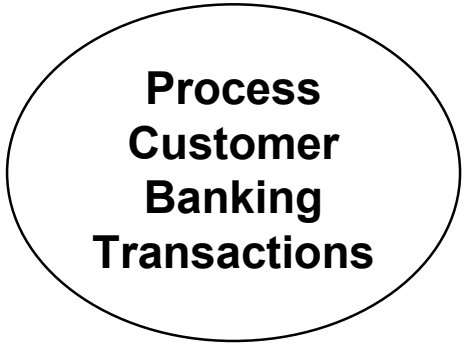


Data Flow Diagram

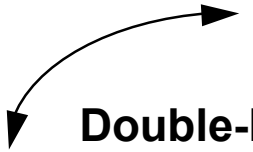
- https://docs.microsoft.com/en-us/archive/msdn-magazine/2006/november/uncover-security-design-flaws-using-the-stride-approach

Item	Symbol
Data flow	One way arrow
Data store	Two parallel horizontal lines
Process	Circle
Multi-process	Two concentric circles
Interactors	Rectangle
Trust boundary	Dotted line

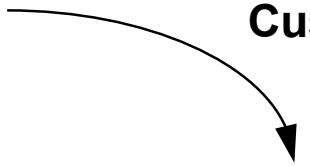
Data Flow Diagram Semantics



These are three equally valid representations of a process. Note a process begins with a verb, just as functions or activities do in IDEF0.



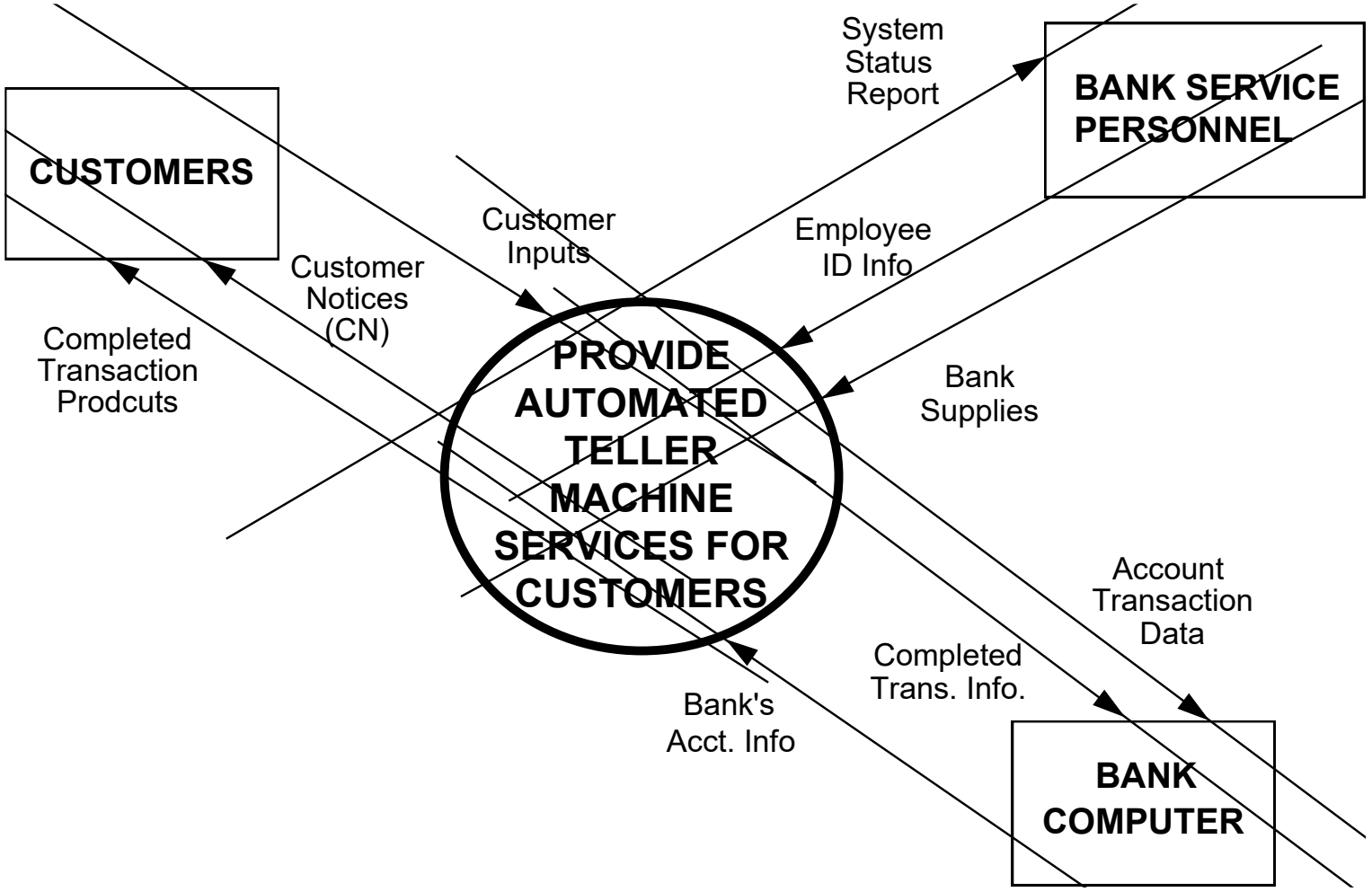
Double-headed arcs signify dialog between functions



Customer Notice: Main Menu Selection

This is an example of a "data flow". Note, it is a noun phrase and attached to an arc.

Context (External Systems) Diagram in DFD



Data Flow Diagramming Rules

- Process
  - A. No process can have only outputs (a miracle)
  - B. No process can have only inputs (black hole)
  - C. A process has a verb phrase label

- Data Store
  - D. Data cannot be moved from one store to another.
  - E. Data cannot move from an outside source to a data store
  - F. Data cannot move directly from a data store to a data sink
  - G. Data store has a noun phrase label

Data Flow Diagramming Rules

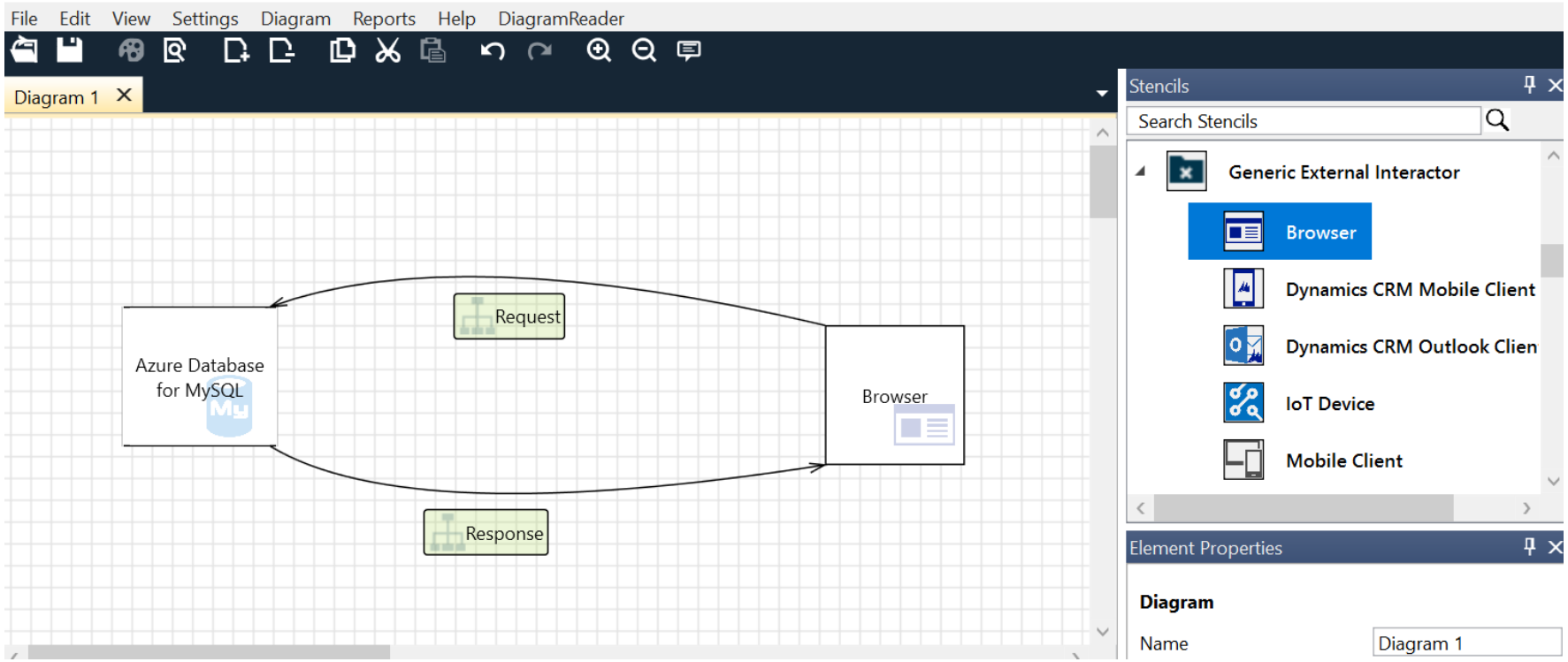
- Source/Sink
  H. Data cannot move directly from a source to a sink
  I. A source/sink has a noun phrase label

- Data Flow
  J. A data flow has only one direction of flow between symbols.
  K. A fork means that exactly the same data go from a common location to two or more processes, data stores or sources/sinks

Data Flow Diagramming Rules

- Data Flow (Continued)
  L. A join means that exactly the same data come from any two or more different processes, data stores or sources/sinks to a common location
  M. A data flow cannot go directly back to the same process it leaves
  N. A data flow to a data store means update
  O. A data flow from a data store means retrieve or use
  P. A data flow has a noun phrase label

Microsoft's Tool



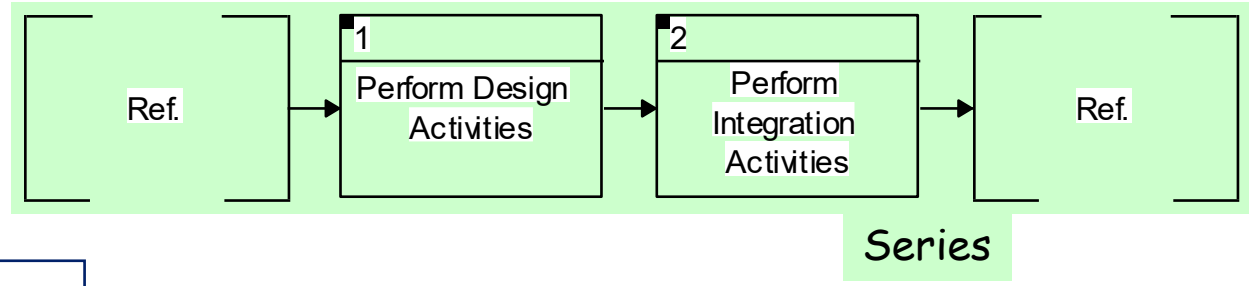
Function Flow Block Diagrams

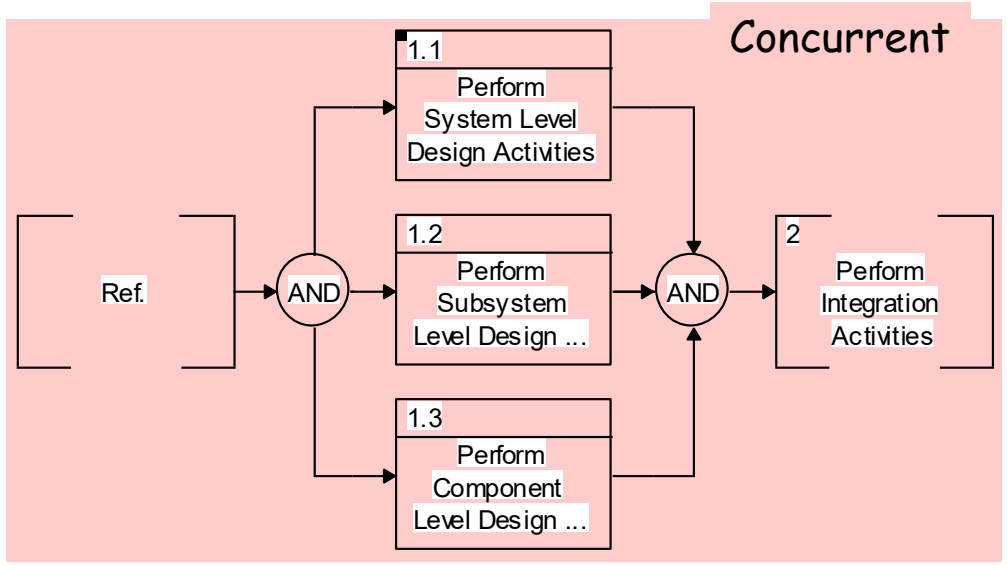
Basic

- Series
- Concurrent
- Selection
- Multiple-exit function

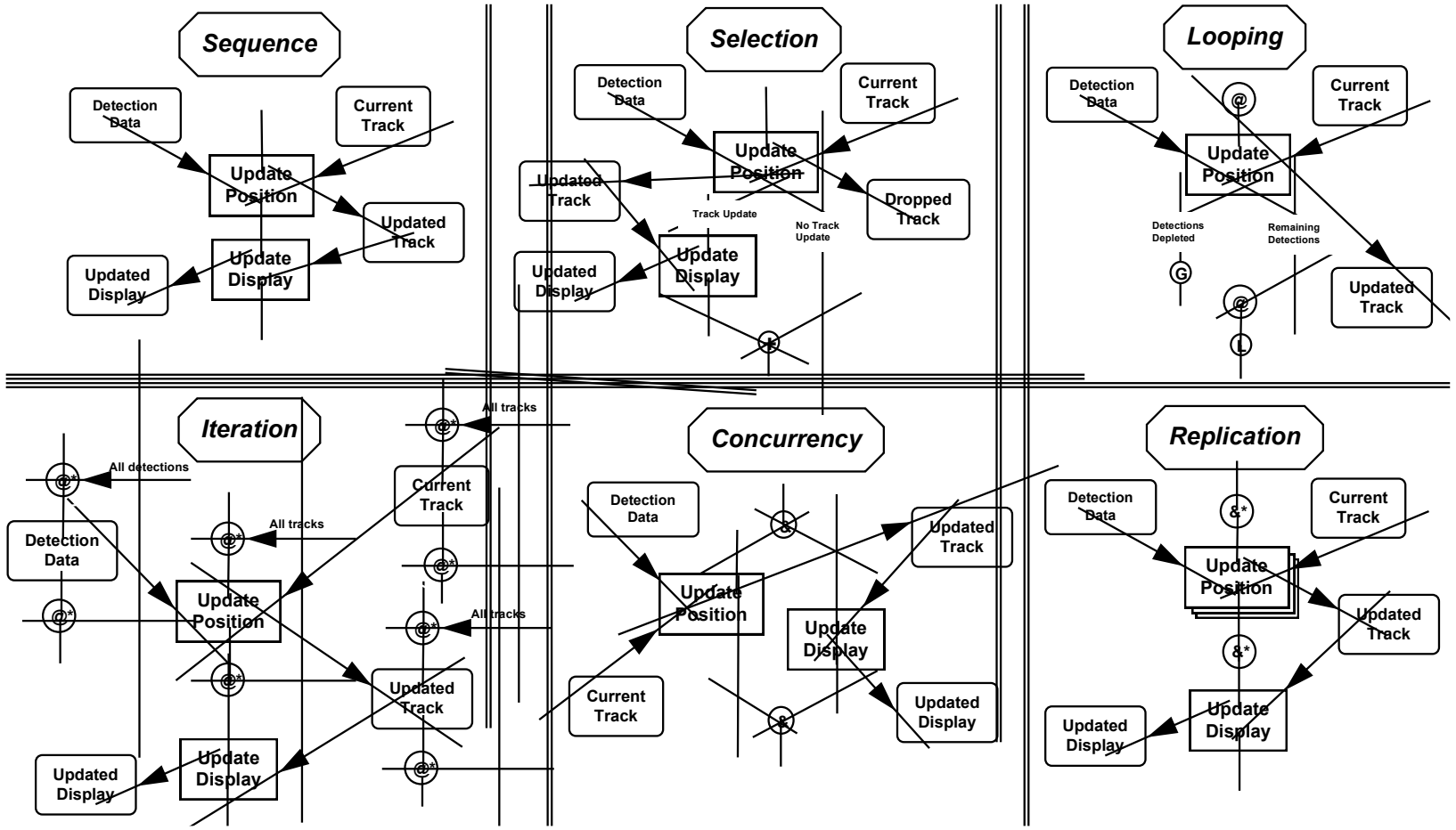
Enhanced

- Iteration
- Looping
- Replication



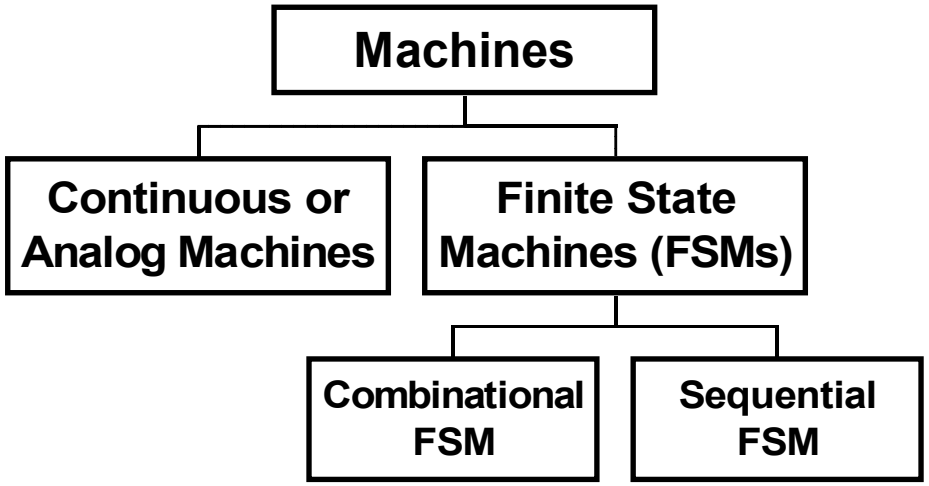


Behavior Diagrams

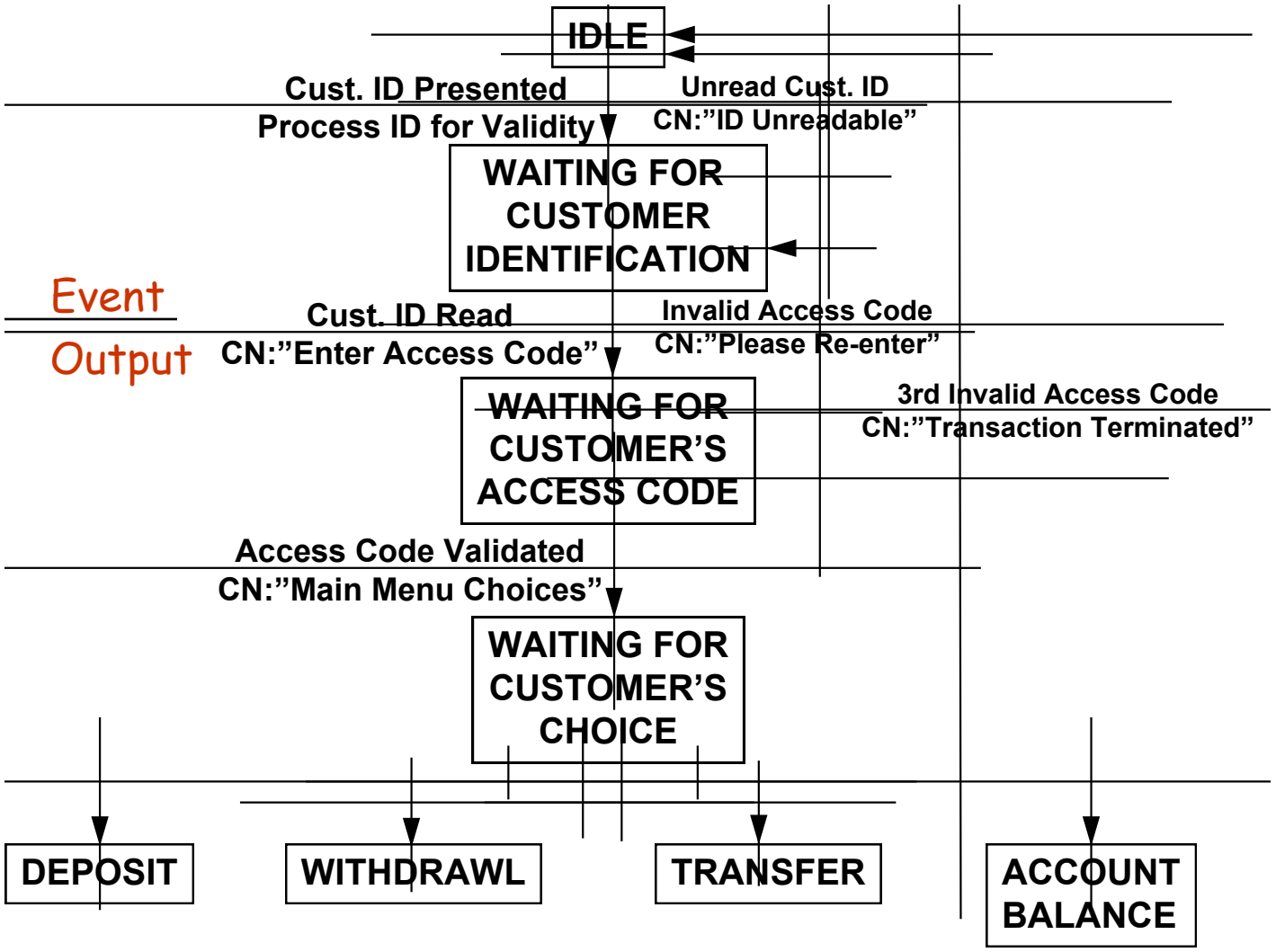


Finite State Machines

- Finite state machines: discrete valued inputs, outputs and internal items
  - Sequential: past inputs impact current outputs (e.g., state-transition diagram)
  - Combinational: current outputs characterized only current inputs
- Continuous machines: continuous and discrete inputs, outputs and internal items



State-transition Diagram for ATM



Other Modeling Languages

- We have already mentioned UML, DODAF and UAF will be discussed in the next lesson. But there are other modeling languages:

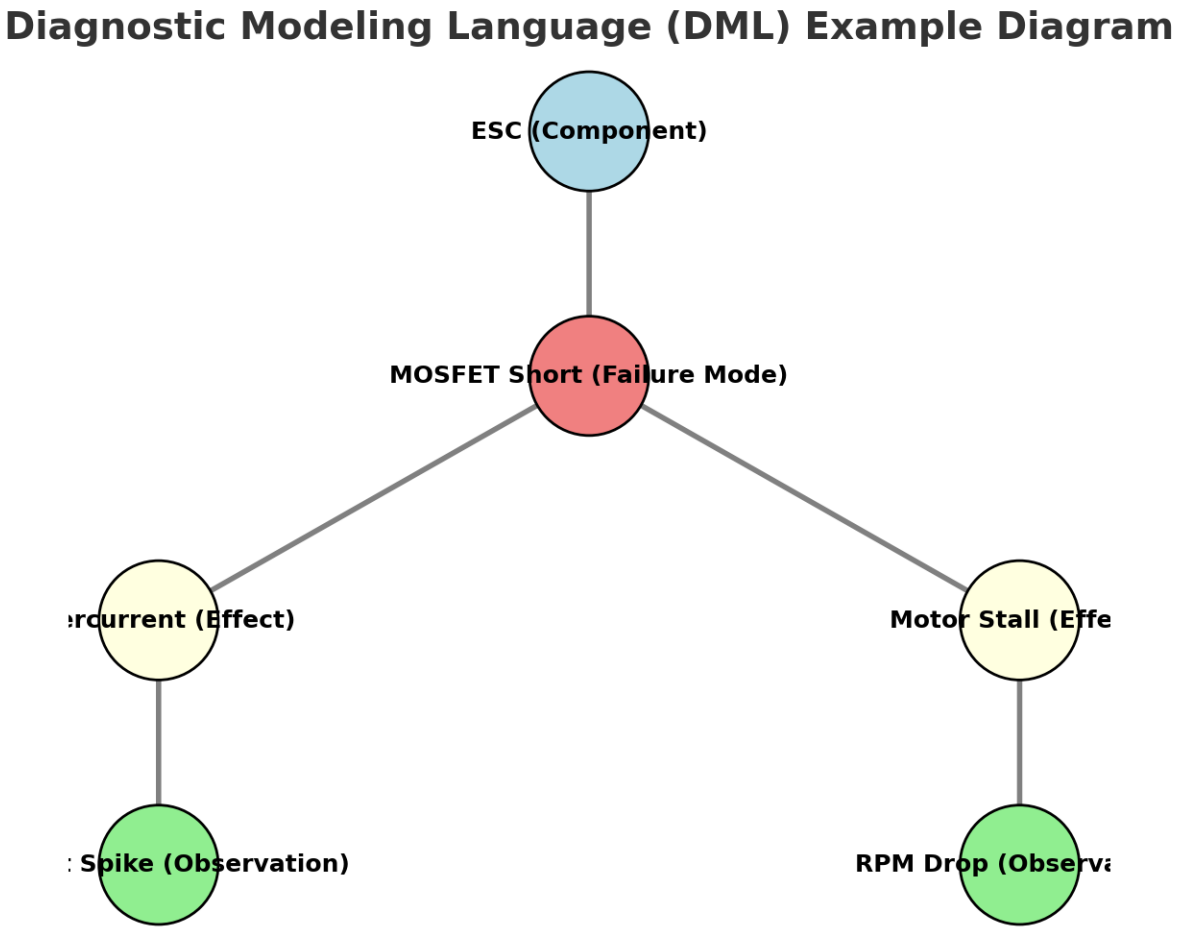
- Diagnostic Modeling Language
- Architecture Analysis & Design Language
- MARTE (Modeling and Analysis of Real-Time and Embedded systems
- LML (Lifecycle Modeling Language) — Lightweight MBSE language for full lifecycle modeling.
- SDL (Specification and Description Language) — Telecommunications system modeling.
- SysADL (System Architecture Description Language) — Architecture modeling for complex systems

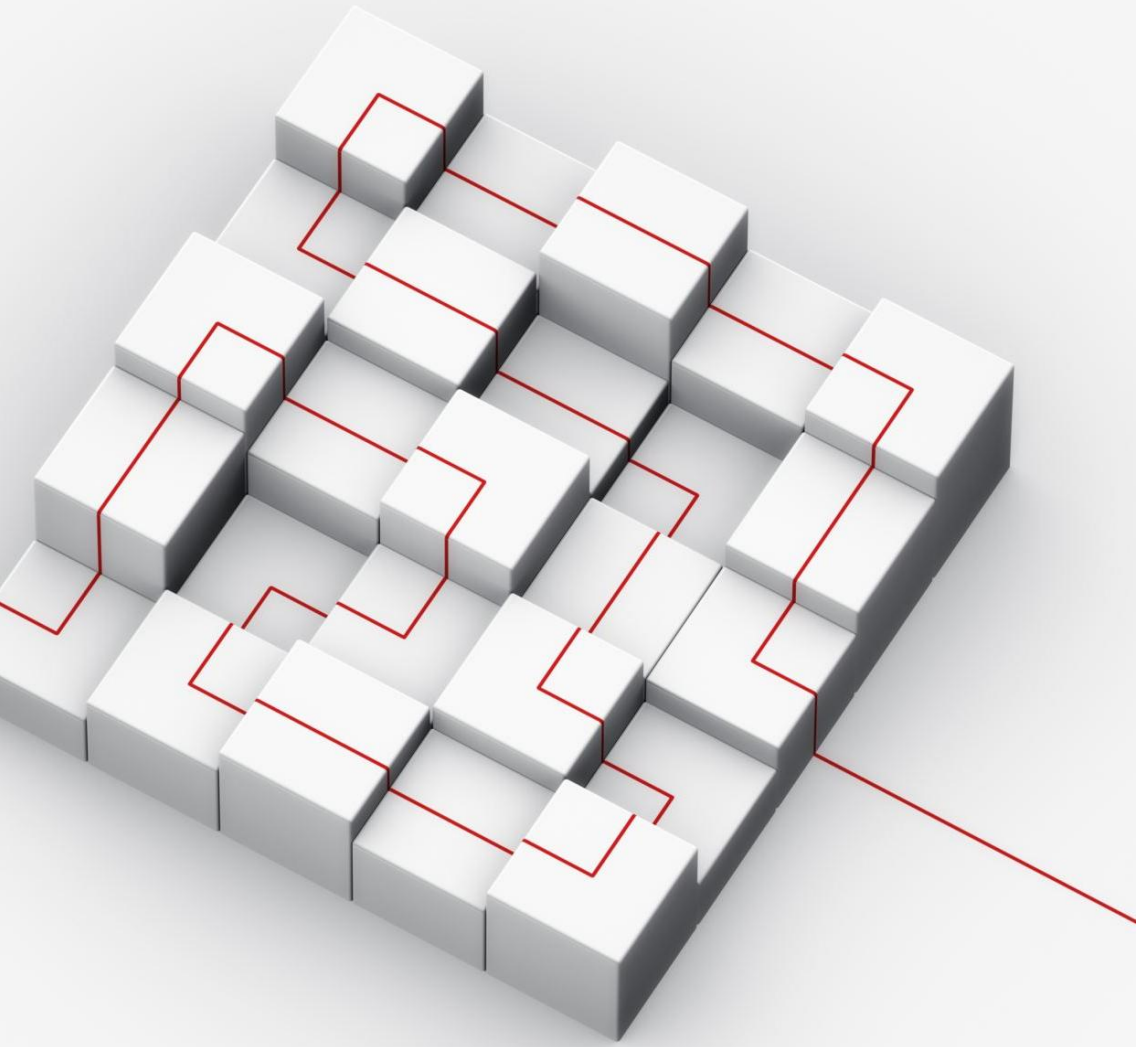
Diagnostic Modeling Language

Diagnostic Modeling Language (DML) is a specialized modeling language used in systems engineering—particularly in model-based diagnostics—to formally describe how to detect, isolate, and identify faults in complex systems. It's essentially a structured way to encode diagnostic knowledge so that fault detection and troubleshooting can be automated or semi-automated. This is an XML-based format designed for interoperability between diagnostic software and test execution environments from different vendors.

- Key Features
- Component Modeling
  - Models physical or logical parts of the system and their relationships.
- Fault Propagation Modeling
  - Defines how a fault in one component can cause changes in other components' behavior.
- Observation Mapping
  - Connects measurable signals (sensor readings, alerts, performance metrics) to potential failure modes.
- Test and Procedure Modeling
  - Specifies what diagnostic tests are available, what they measure, and what outcomes mean.
- Supports Multiple Diagnostic Strategies
  - Can be used for model-based reasoning, rule-based systems, or hybrid diagnostic frameworks.

Diagnostic Modeling Language





Architecture Analysis & Design Language (AADL)

- Architecture Analysis & Design Language (AADL) is a standardized modeling language (defined by SAE AS5506) used to describe and analyze the architecture of complex, embedded, real-time systems—particularly where hardware, software, and their interactions must be modeled together.
- It's especially valuable in avionics, aerospace, automotive, and defense because it supports early verification of performance, timing, safety, and reliability before the system is built.

Architecture Analysis & Design Language (AADL)

- Component-Based
- AADL models systems as hierarchies of components:
  - Software: threads, processes, subprograms, data.
  - Hardware: processors, memory, devices, buses.
- Formal Semantics
  - Unambiguous meaning for each construct, enabling automated analysis.
- Annex Mechanism
- Allows extensions for specific analyses (e.g., safety, behavior, fault modeling). Examples: Error Model Annex (for reliability/safety), Behavior Annex (for state machines).
- Property Sets
  - Attach quantitative and qualitative attributes to components (e.g., CPU speed, WCET — Worst-Case Execution Time, failure rate).
- Tool Support
  - Tools like OSATE (Open Source AADL Tool Environment) allow model creation, consistency checking, and analysis.

- https://www.sei.cmu.edu/projects/architecture-analysis-and-design-language-aadl/

MARTE (Modeling and Analysis of Real-Time and Embedded systems

MARTE — the Modeling and Analysis of Real-Time and Embedded systems profile — is an extension of UML (Unified Modeling Language) specifically designed for modeling, analyzing, and specifying real-time, embedded, and performance-critical systems. It's standardized by the Object Management Group (OMG) and is widely used in MBSE (Model-Based Systems Engineering) for domains like aerospace, automotive, industrial automation, and telecommunications.

MARTE (Modeling and Analysis of Real-Time and Embedded systems

MARTE is organized into several sub-profiles, including:

Time — modeling and reasoning about time.

NFP (Non-Functional Properties) — expressing measurable properties.

HRM (Hardware Resource Modeling) — describing hardware architecture.

SRM (Software Resource Modeling) — describing software execution and services.

GCM (Generic Component Modeling) — reusable component structures.

SAM (Schedulability Analysis Modeling) — linking models to performance analysis.

Lifecycle Modeling Language (LML)

Lifecycle Modeling Language (LML) is a simplified, standardized modeling language created to support the entire system lifecycle—from concept and requirements through design, verification, operations, and disposal—while being easier to learn and apply than traditional modeling languages like SysML. It's intended for Model-Based Systems Engineering (MBSE), program management, and decision support, with a focus on clarity, usability, and integration.

Lifecycle Modeling Language (LML)

Key Characteristics

- Core Ontology
  - LML defines a small, consistent set of **entity types** (about a dozen core ones) such as:
    - *Action* (tasks, activities, functions)
    - *Asset* (hardware, software, facilities)
    - *Requirement*
    - *Risk*
    - *Decision*
    - *Verification*
    - *Concern* (issues, opportunities)
    - *Interface*
  - This core vocabulary applies across the entire lifecycle.
- Integrated Views
  - **Functional**: what the system does (*Actions*).
  - **Physical**: what the system is (*Assets*).
  - **Parametric**: performance, cost, schedule data.
  - **Traceability**: links between requirements, risks, designs, and verifications.
- Consistency
  - All diagrams and tables come from the same underlying model—no duplication.
  - Ensures updates in one view are reflected everywhere.
- Ease of Use
  - Designed to reduce learning curve.
  - Works well with tools like **Innoslate** (which natively implements LML).

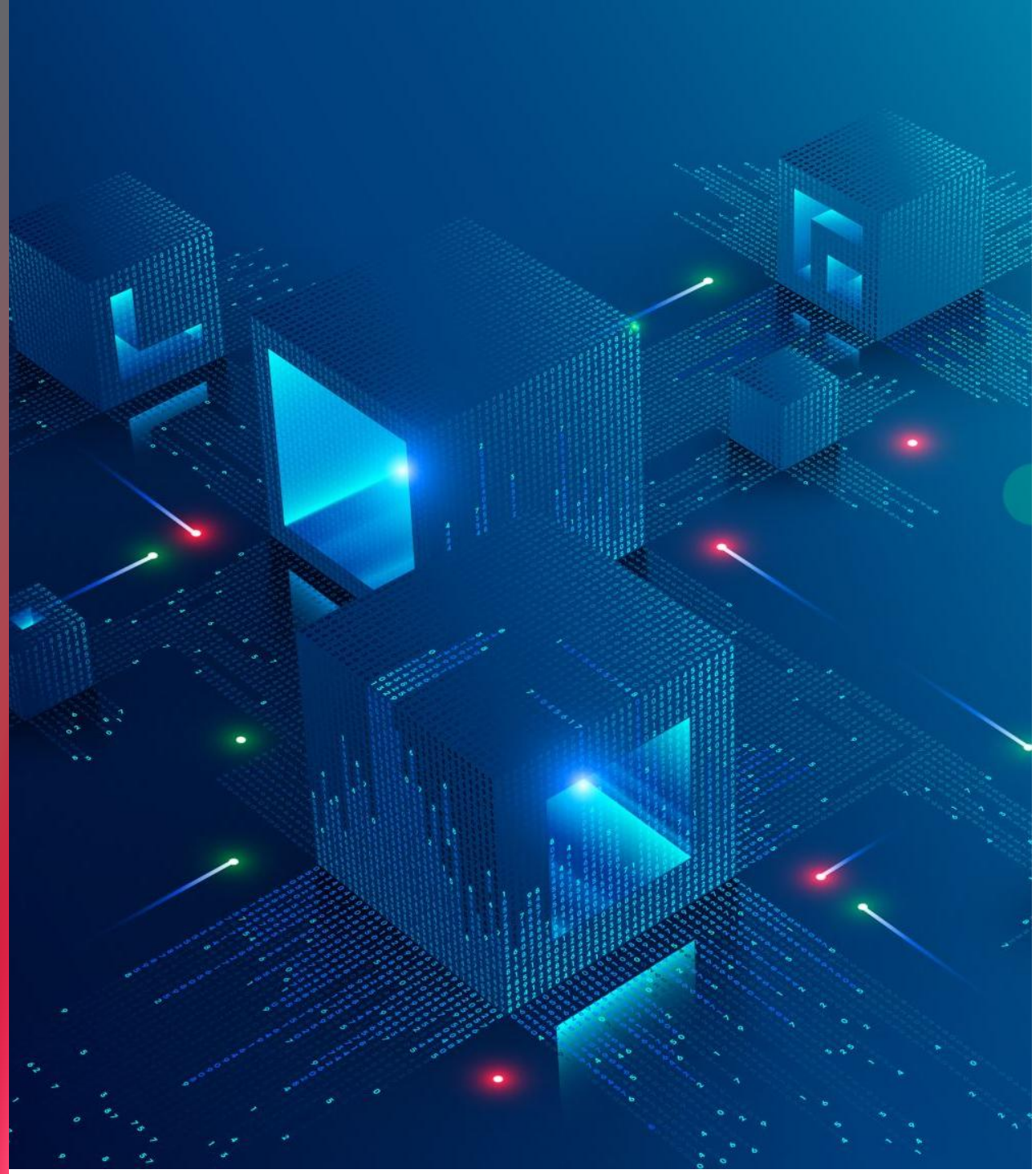
SDL (Specification and Description Language)

- Specification and Description Language (SDL) is a formal, standardized modeling language (ITU-T Z.100 standard) used primarily for specifying, designing, simulating, and validating real-time, event-driven, and distributed systems—especially in telecommunications, networking, and embedded control systems. It combines a formal mathematical basis with a graphical notation so that models can be both precise and human-readable.

SDL (Specification and Description Language)

Basic Concepts in SDL

- **System**: The top-level entity being modeled.
- **Block**: A subsystem within the system.
- **Process**: A sequential thread of control, modeled as an EFSM.
- **Signal**: A message sent between processes or from the environment.
- **Channel**: The communication path for signals.
- **Procedure**: Reusable sequence of actions within a process.
- **Timer**: Built-in concept for modeling time-based event



SysADL (System Architecture Description Language)

SysADL is a Software Architecture Description Language (ADL) implemented as a SysML profile. It extends the modeling capabilities of the Systems Modeling Language (SysML) to provide robust, standardized support for architectural modeling in software-intensive systems.