ECEE 5318
RT Embedded Systems

Design Methods and Notation

Examples

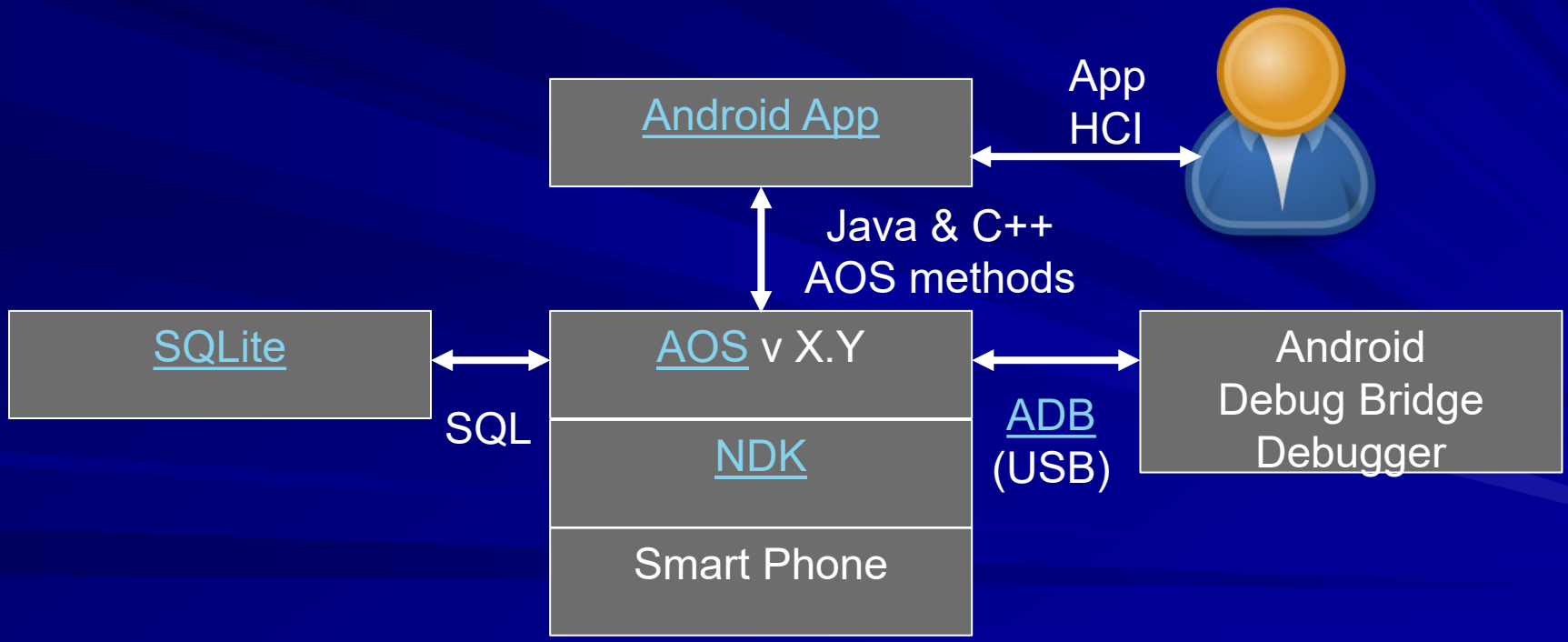
December 6, 2020

Design Elements for RTES Project

- Top N Capability Oriented Requirements
  - State and Explain
  - Hold Q&A and Ask for Reviewer Input on Completeness, Errors and Omissions
- Top N Real-Time Requirements [C_i, T_i, D_i or each S_i]
  - State and Explain Service request frequency drivers and relative deadlines
  - How did you estimate or measure C_i WCET
- Single Page High Level Block Diagram of Software System
  - Show End-to-End Elements and Dataflow
  - Source to Sink (Top Left Corner to Bottom Right)
- CFD/DFD, Flow Charts, State Machines or Other Design Models
- RTES Prototype Time-Stamp Tracing Analysis

Application Block Diagram

- Hardware shows standard platform(s), Software, and User Interaction
- Label Interface Dataflow and Protocols
- E.g. Contacts App on AOS



Block Diagram Resources

- There is no standard for Block Diagrams
- Traditional Engineering Diagram that is Interdisciplinary
- SysML - An Extension to UML for Systems
- http://www.omgsysml.org/
- https://en.wikipedia.org/wiki/Block_diagram

More State Machine Examples

ECEE 5318 – RTES Project

Activity - State Machine & Transition Table

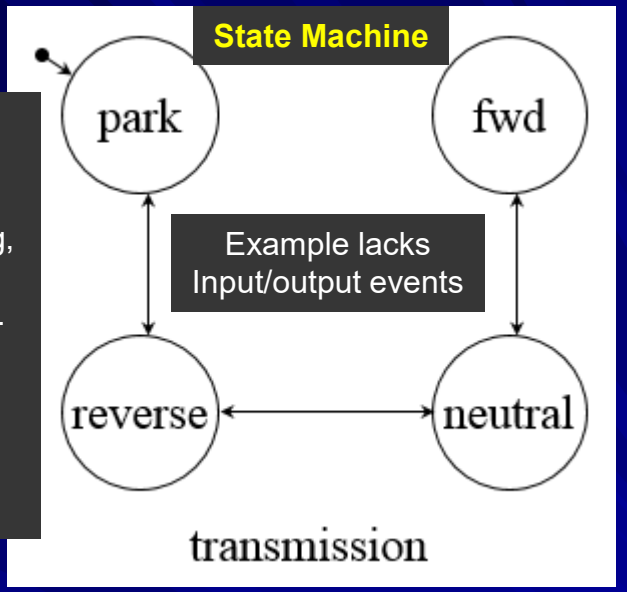
- Transition Table = Current_State X New_State
  - Each cell in table has input / output
  - For N states, table is N^2 in size
- Mealy State Machine
  - State is a labeled circle (state of the software)
  - Transitions are arrows with input / output
- Ignore 4WD, Trailer Pull Options
- Electro-mechanical with software

2017 Chevy Colorado 4WD Automatic – ignore 4WD, Trailer Options
(Has Upshift/Downshift Button – not shown)

State Machine vs. State Transition Table

- State Machine Shows States and events (input/output) we expect in our design
- Format-1: State Transition Table shows ALL States and transitions possible
- State x State with i/o
- 4 States, 6 allowed transitions
- Stay in Current State allowed/assumed

Automatic Common Use Case:
Back out of parking, rev-up engine, then drive forward.
Or
Drive in forward, rev-up engine, reverse and back into parking.



Green states allowed (safe?, necessary?)

Red states not allowed (unsafe?, no implementation?)

Simple State Transition Table (input/output)

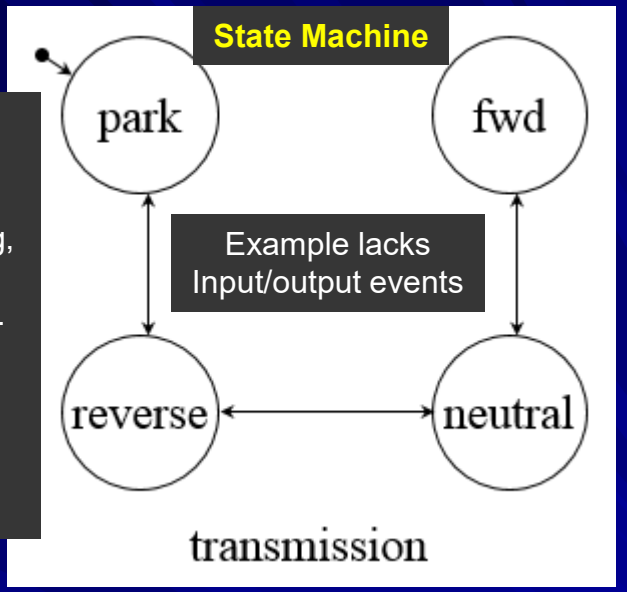
Current State	New State: Park	Reverse	Neutral	Fwd
Park	noop / "P"	shift-down / "R"	NA	NA
Reverse	shift-up / "P"	noop / "R"	shift-down / "N"	NA
Neutral	NA	shift-up / "R"	noop / "N"	shift-down / "F"
Fwd	NA	NA	shift-up / "N"	noop / "F"

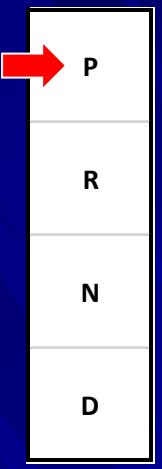
Start state assumed and eliminated

State Machine vs. State Transition Table

- State Machine Shows States and events (input/output) we expect in our design
- Format 2: State Transition Table shows ALL States and transitions possible
- State x Event (input), with condition (guard) and action
- 4 States, 6 allowed transitions
- Stay in Current State allowed/assumed

Automatic Common Use Case:
Back out of parking, rev-up engine, then drive forward.
Or
Drive in forward, rev-up engine, reverse and back into parking.





Green states allowed (safe?, necessary?)

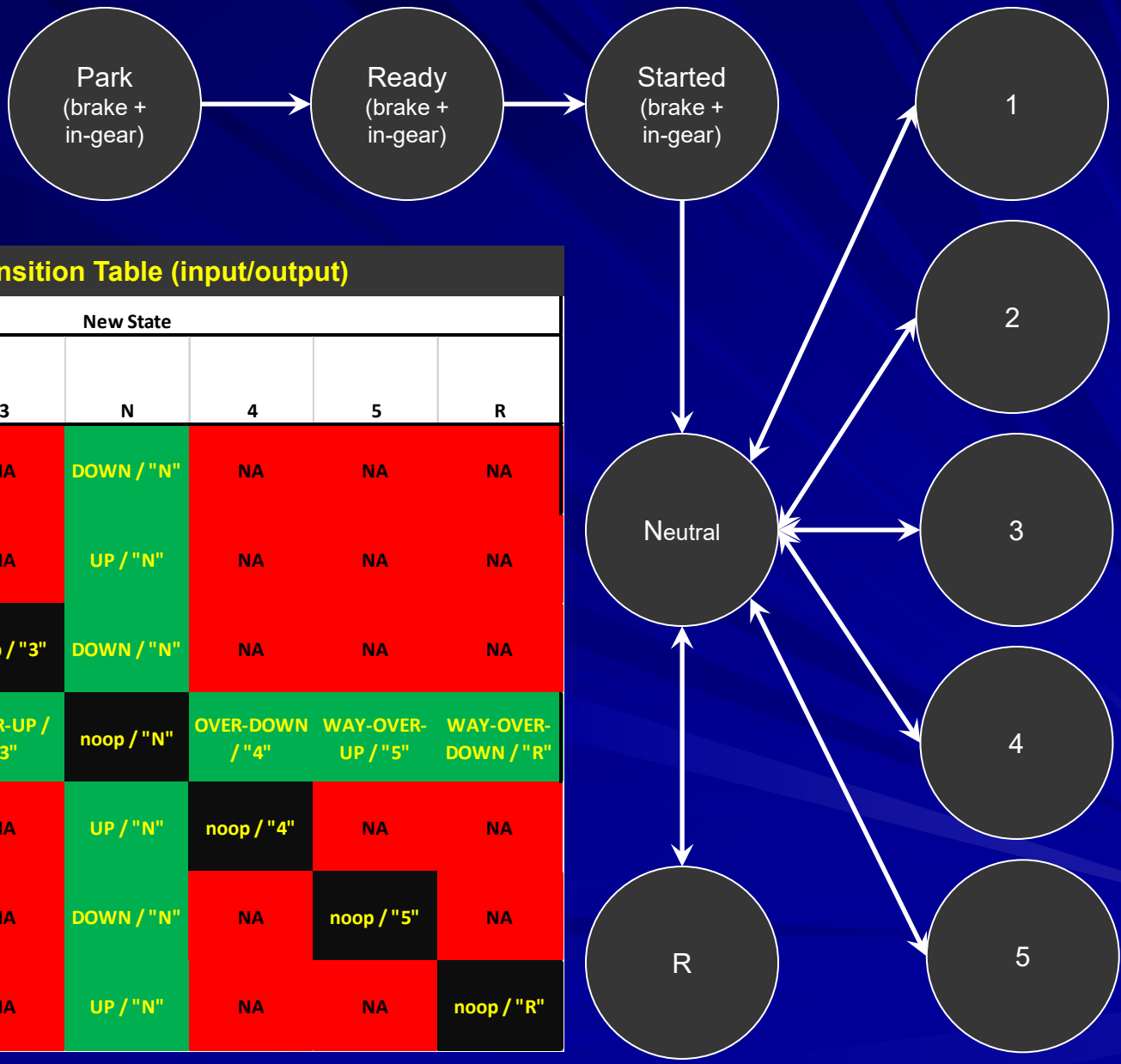
Red states not allowed (unsafe?, no implementation?)

EFSM State Transition Table
(New State, [condition], "action")

State	Event: No action	Event: Shift Down	Event: Shift Up
Park	Park []/"P"	Reverse [enable] / "R"	NA
Reverse	Reverse []/"R"	Neutral [enable] / "N"	Park [enable] / "P"
Neutral	Neutral []/"N"	Fwd [enable] / "D"	Reverse [enable] / "R"
Fwd	Fwd []/"F"	NA	Neutral [enable] / "N"

Start state assume and eliminated here

State Machine vs. State Transition Table



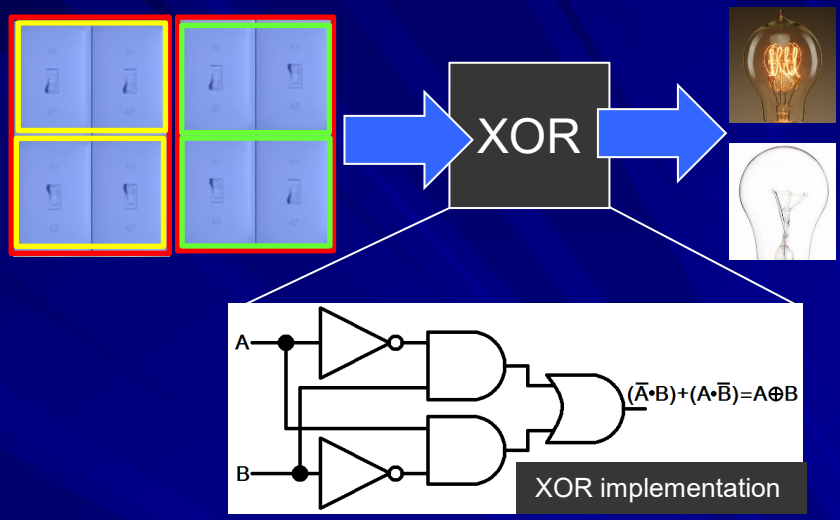
Simple State Transition Table (input/output)

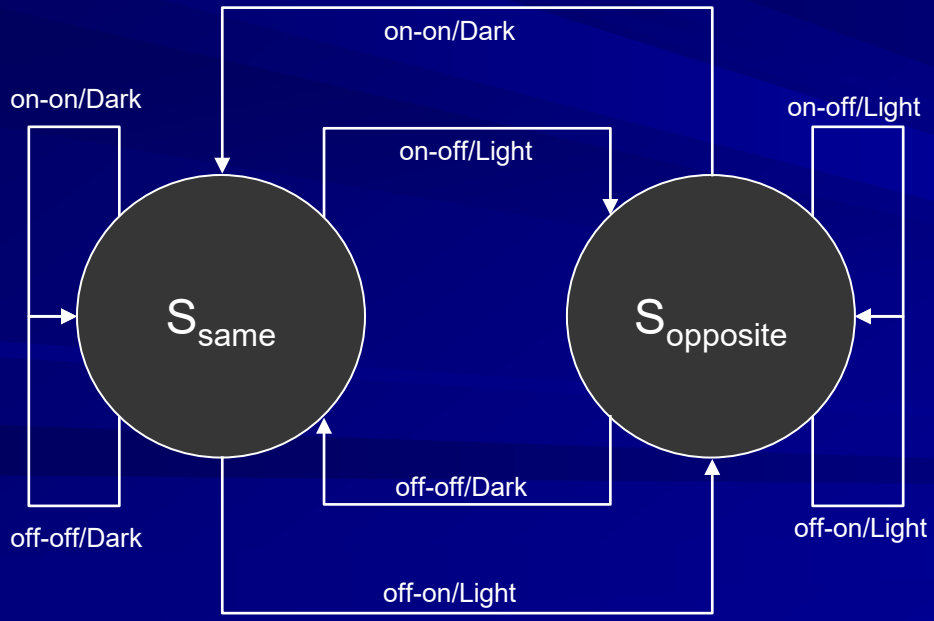
Current State	New State 1	2	3	N	4	5	R
1	noop / "1"	NA	NA	DOWN / "N"	NA	NA	NA
2	NA	noop / "2"	NA	UP / "N"	NA	NA	NA
3	NA	NA	noop / "3"	DOWN / "N"	NA	NA	NA
N	UP / "1"	DOWN / "2"	OVER-UP / "3"	noop / "N"	OVER-DOWN / "4"	WAY-OVER-UP / "5"	WAY-OVER-DOWN / "R"
4	NA	NA	NA	UP / "N"	noop / "4"	NA	NA
5	NA	NA	NA	DOWN / "N"	NA	noop / "5"	NA
R	NA	NA	NA	UP / "N"	NA	NA	noop / "R"

2-Way Switch as State Machine

- Software can model hardware
  e.g. SPICE (analog), HDL (digital)
  Turing > Combinational Logic, State Machine
- 2-Way Circuit Combinational Logic is XOR(S_1, S_2)
- How Many States to Model?
  4 State, Single Input:
  {$S_{same\text{-}up\text{-}up}$, $S_{opposite\text{-}up\text{-}dwn}$, $S_{same\text{-}dwn\text{-}dwn}$, $S_{opposite\text{-}dwn\text{-}up}$}

  2 State, Combined Input:
  {S_{same}, $S_{opposite}$}, {on-on, on-off, off-on, off-off}
  Mealy State Machine is 2 x 2



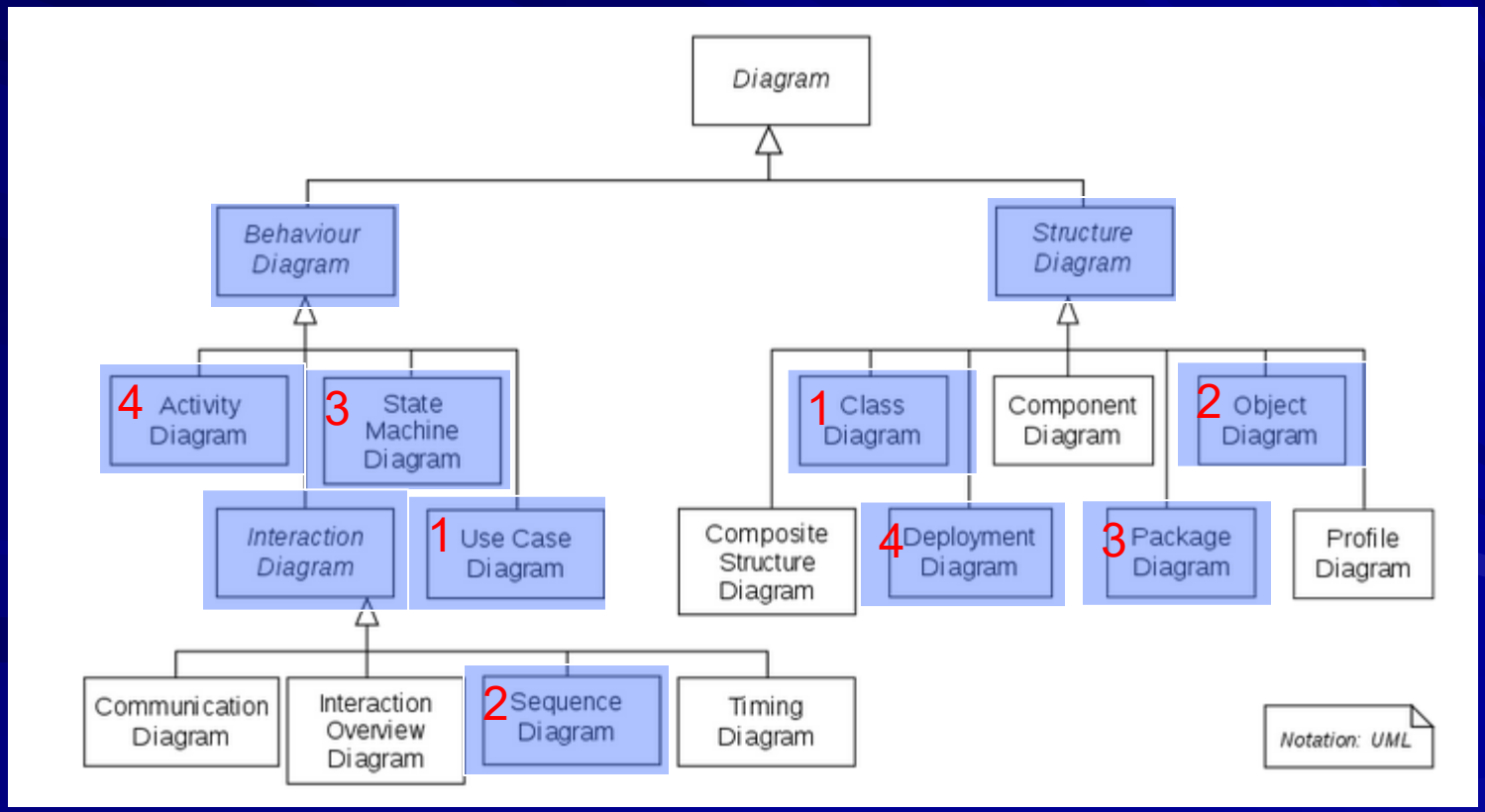


Switch S1	Switch S2	XOR(S1, S2)
0	0	0
0	1	1
1	0	1
1	1	0

Switch S1	Switch S2	Output
off ↓	off ↓	Dark
off ↓	on ↑	Light
on ↑	off ↓	Light
on ↑	on ↑	Dark

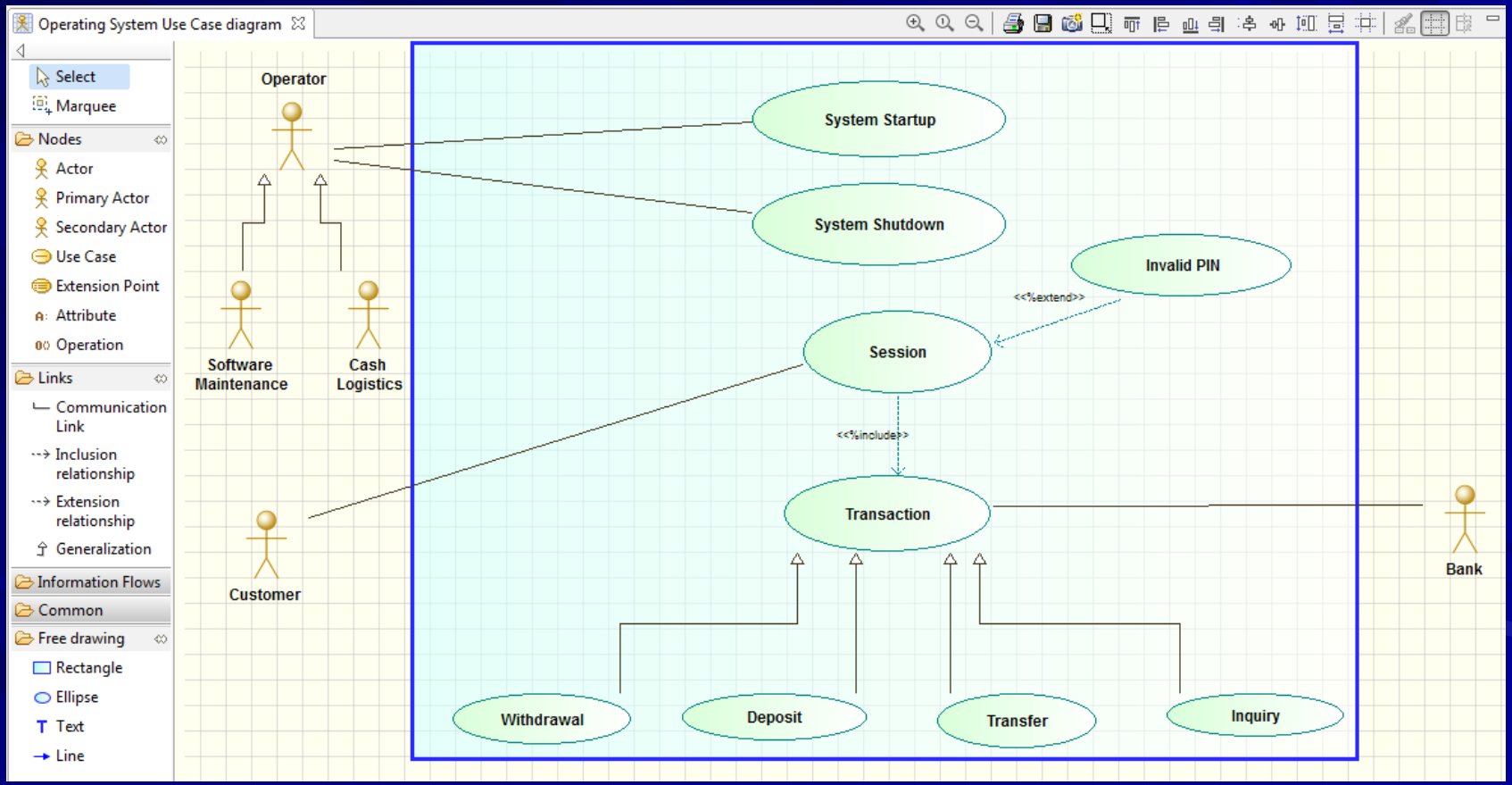
UML (Quick Reference, Visual Paradigm)

- https://www.omg.org/spec/UML/About-UML/
- State with Use Cases and CRC for Classes
- 4 Main Behavioral (UC, OIM sequence, Activity, State)
- 4 Main Structural (Class, Object, Package, Deployment)



UML Use Case - Behavioral, Starting Point

- Concurrent with Goals and Objectives and Block Diagram
- Goes well with requirements analysis (tracing)
- Start here to define project scope & Actors: users, external interfaces



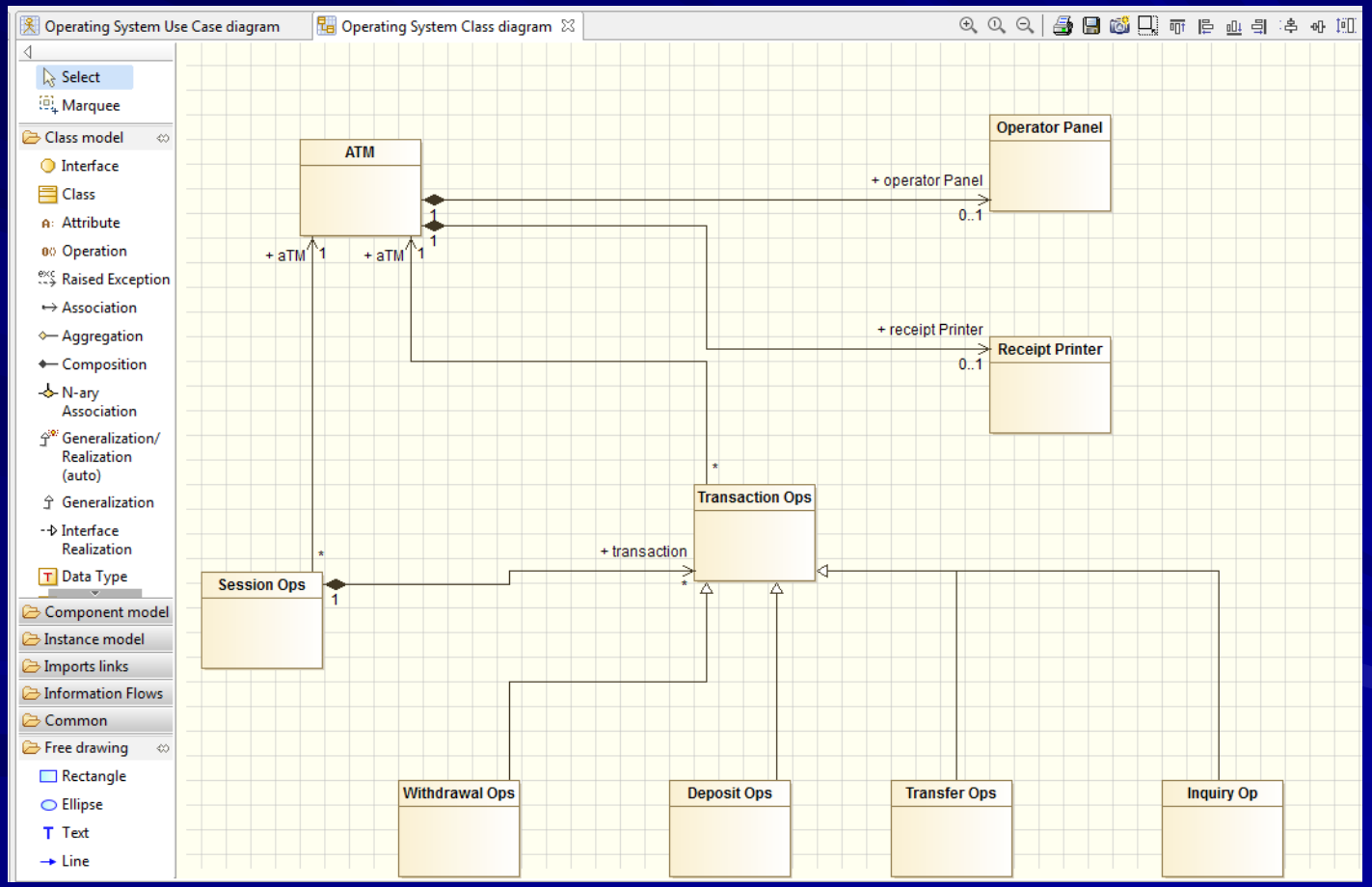
https://www.visual-paradigm.com/guide/uml-unified-modeling-language/what-is-use-case-diagram/

UML Structural Diagrams (Using Modelio)

Class, Object, Package, Deployment (Component)

#1 - UML Class Diagram - CRC level

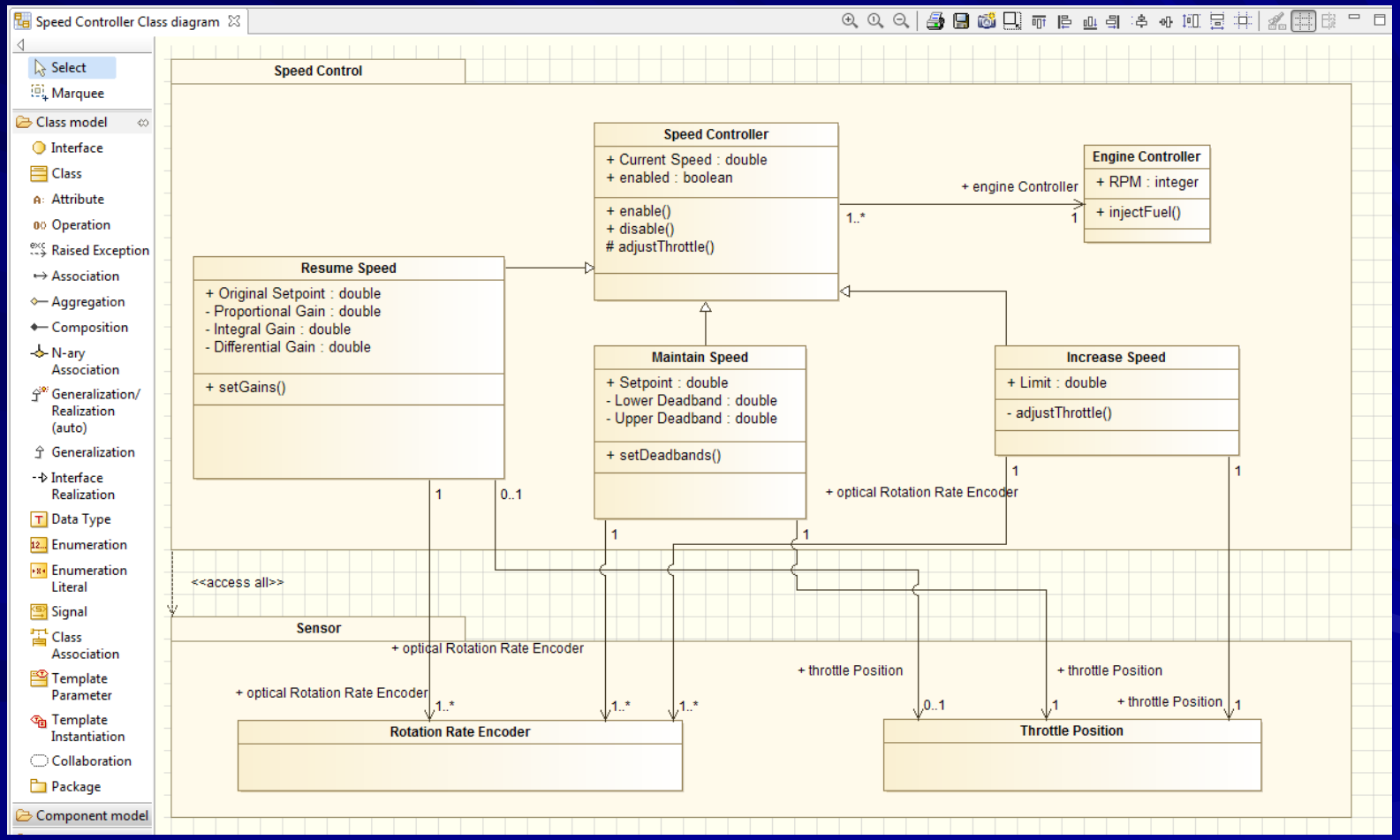
- First, use CRC (Class, Responsibility, Collaborator) analysis and create Class diagram with classes and relationships (attributes and methods can come later)
  1. Association (Cardinality - 1:1, 1:n, 0..1)
  2. Aggregation (may include)
  3. Composition (must include)
  4. Hierarchy (Super class, sub-class - e.g. Withdrawal is a sub-class of Transaction)



https://www.visual-paradigm.com/guide/uml-unified-modeling-language/what-is-class-diagram/

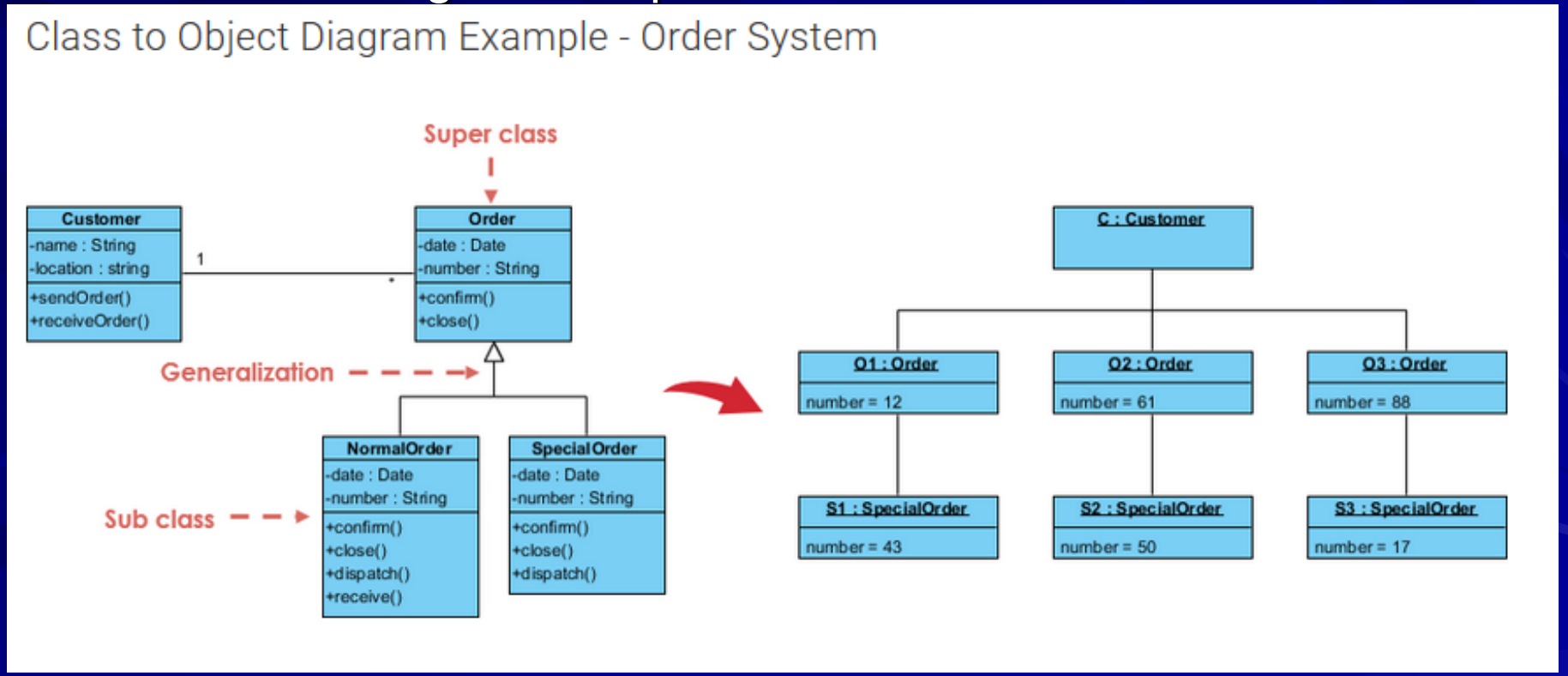
#1 - UML Class Diagram - Domain Model

- Second, refine use CRC analysis along with OIM sequence diagrams and create Class diagram with classes, relationships, attributes and methods, and can combine with packages
- Refinement - additional progress, updates based on Level-0 design walk-throughs, more detail
  1. Association (Cardinality - 1:1, 1:n, 0..1)
  2. Aggregation (may include)
  3. Composition (must include)
  4. Hierarchy (Super class, sub-class - e.g. Withdrawal is a sub-class of Transaction)



#2 - UML Object Diagram

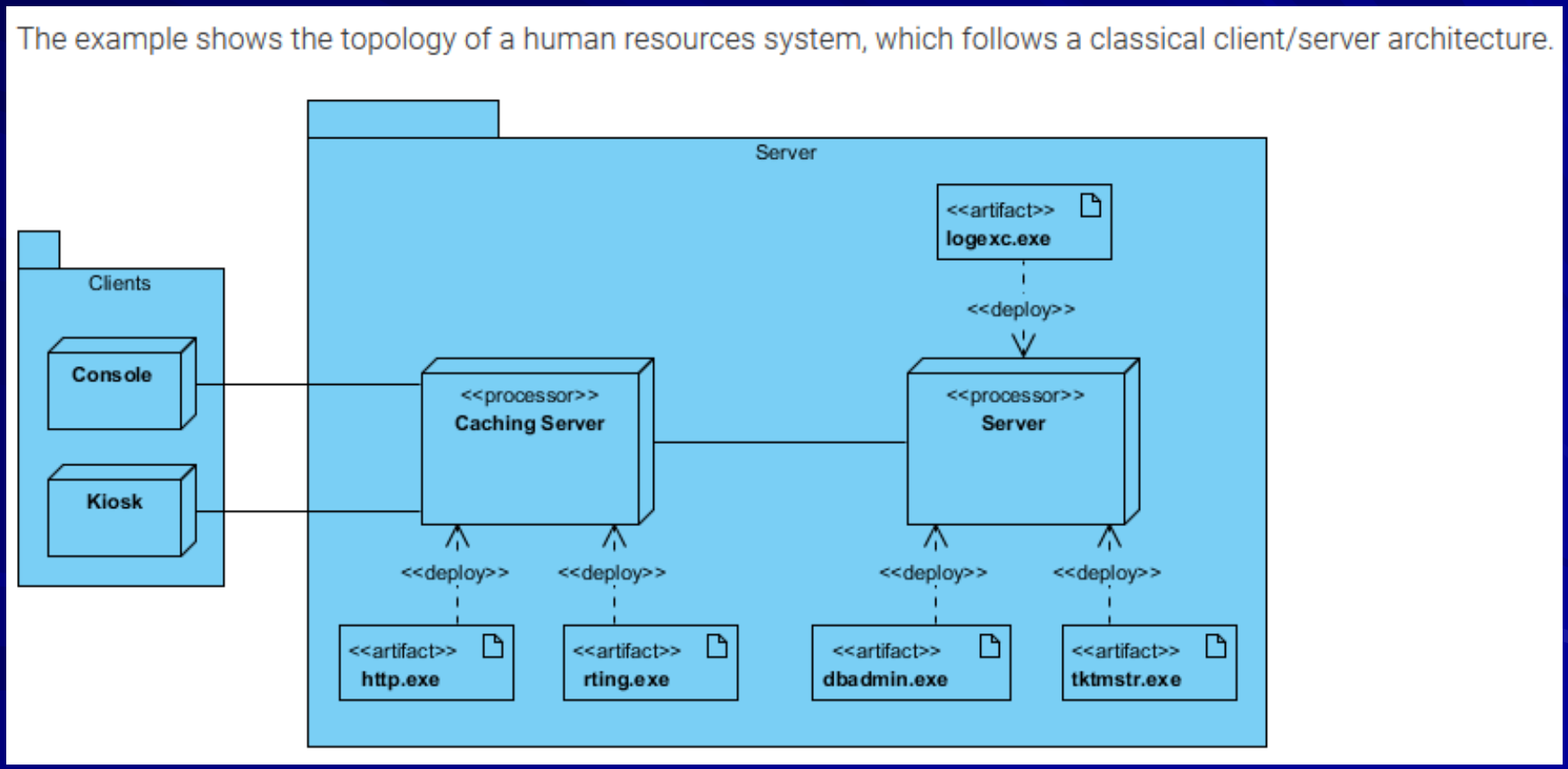
- Simple Goal - Show Class instantiation as objects and how objects relate (test instantiation, containers, constructor/destructor, etc.)
- https://www.visual-paradigm.com/guide/uml-unified-modeling-language/what-is-object-diagram/
- Test Class Diagram completed in Level-1 Domain Model



https://www.visual-paradigm.com/guide/uml-unified-modeling-language/what-is-object-diagram/

#3 - UML Deployment Diagram

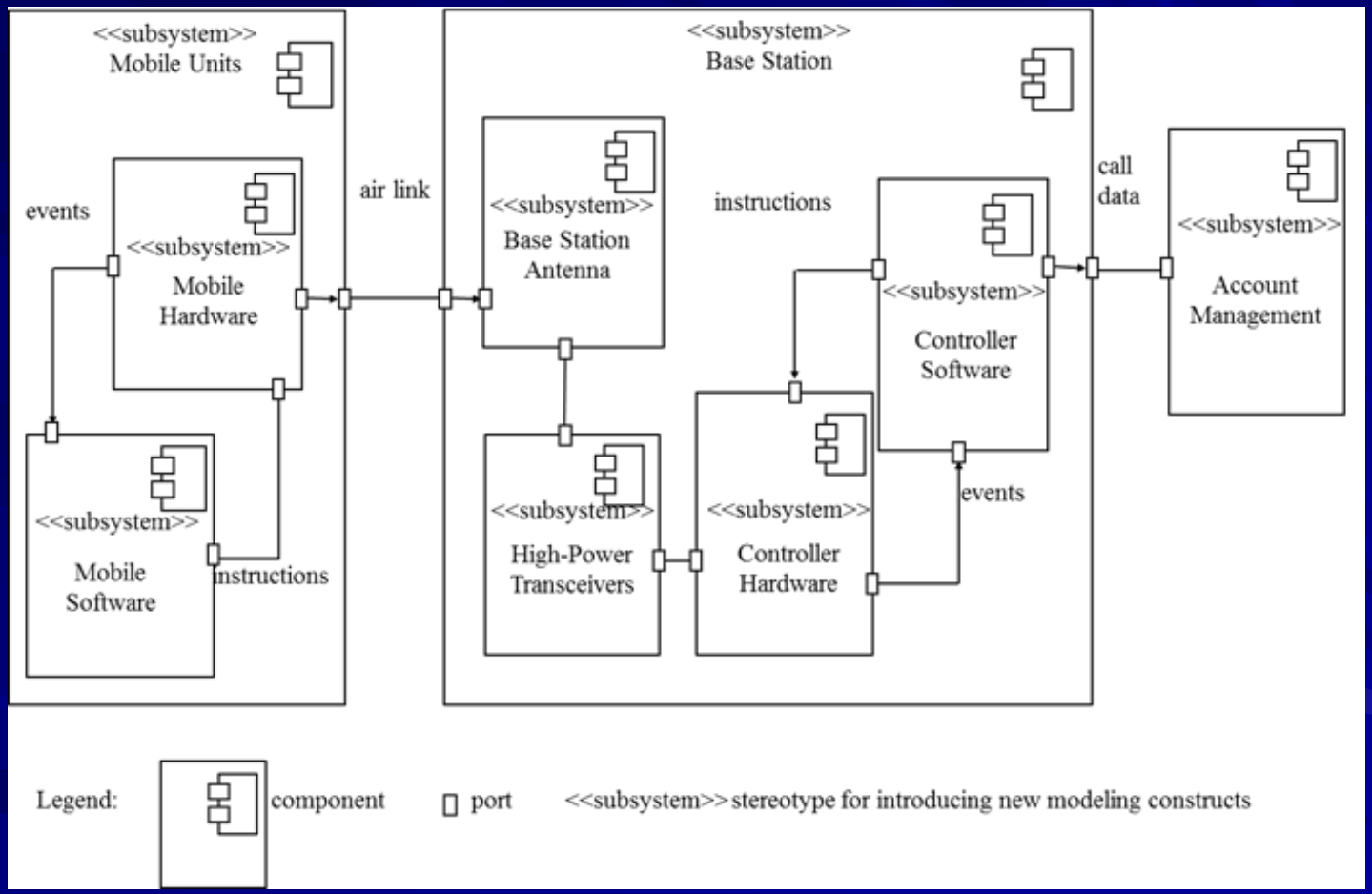
- Useful for Systems with Hardware platforms and Multiple stand-alone software applications or services
- What you need to ship and configure for a system test



https://www.visual-paradigm.com/guide/uml-unified-modeling-language/what-is-deployment-diagram/

#3 - Deployment Diagram - Example

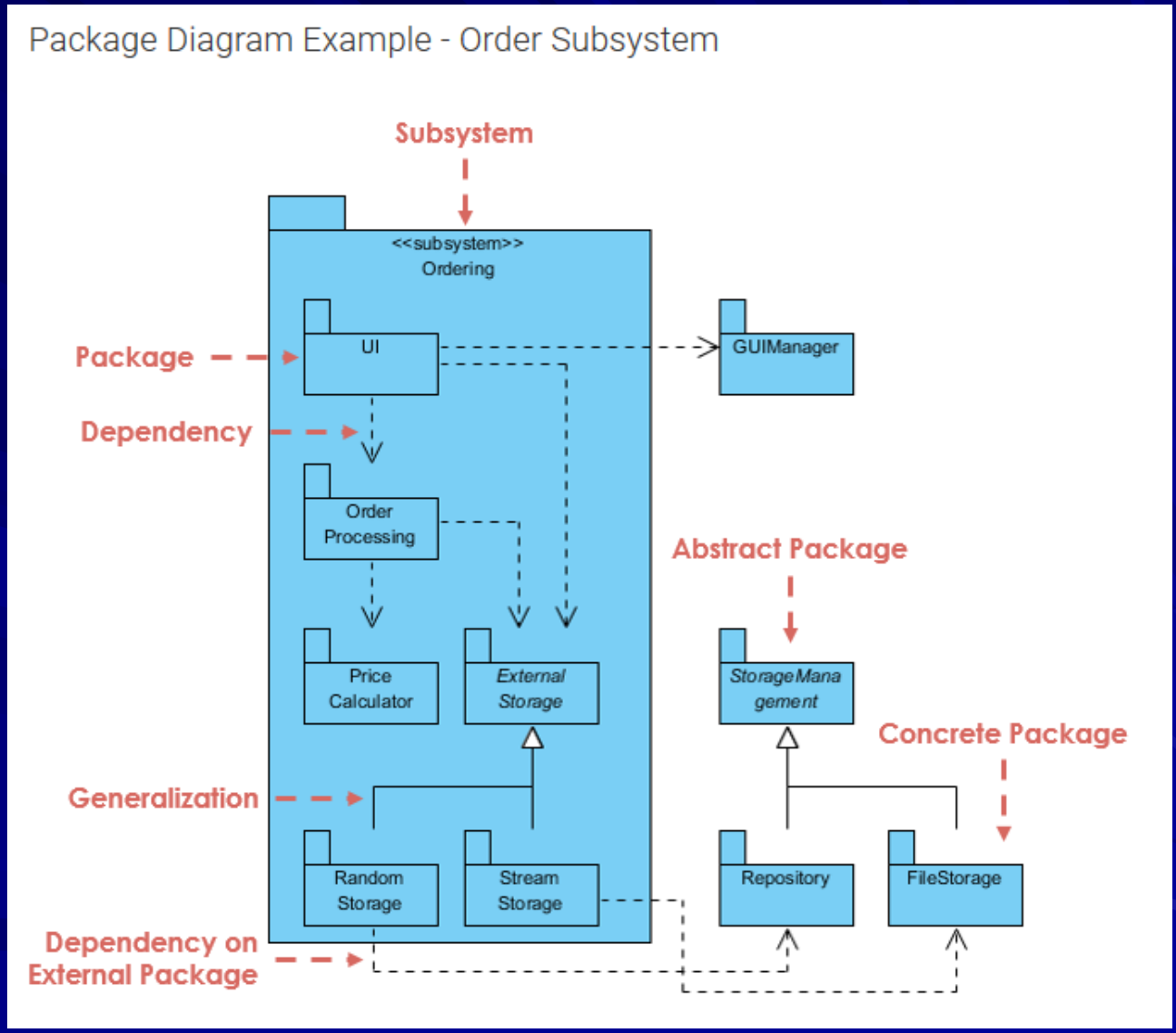
- Shows mixture of software sub-systems and hardware
- Similar to block diagram with hardware and software blocks



From Kung, OOSE - Example of Deployment diagram for cellular system

#4 - UML Package Diagram

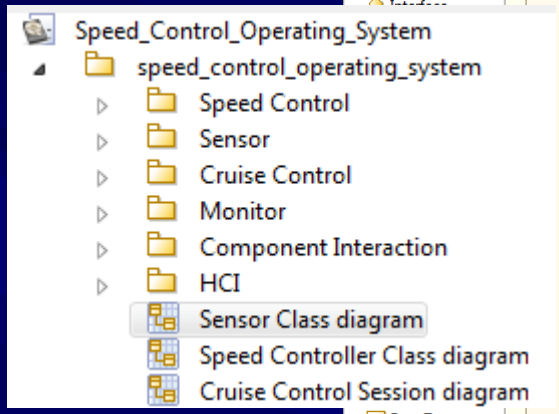
- Can be done alone (typically level-2) or combined with a Class Diagram
- Module strong cohesion and interface loose coupling
- Define directory structure for CMVC

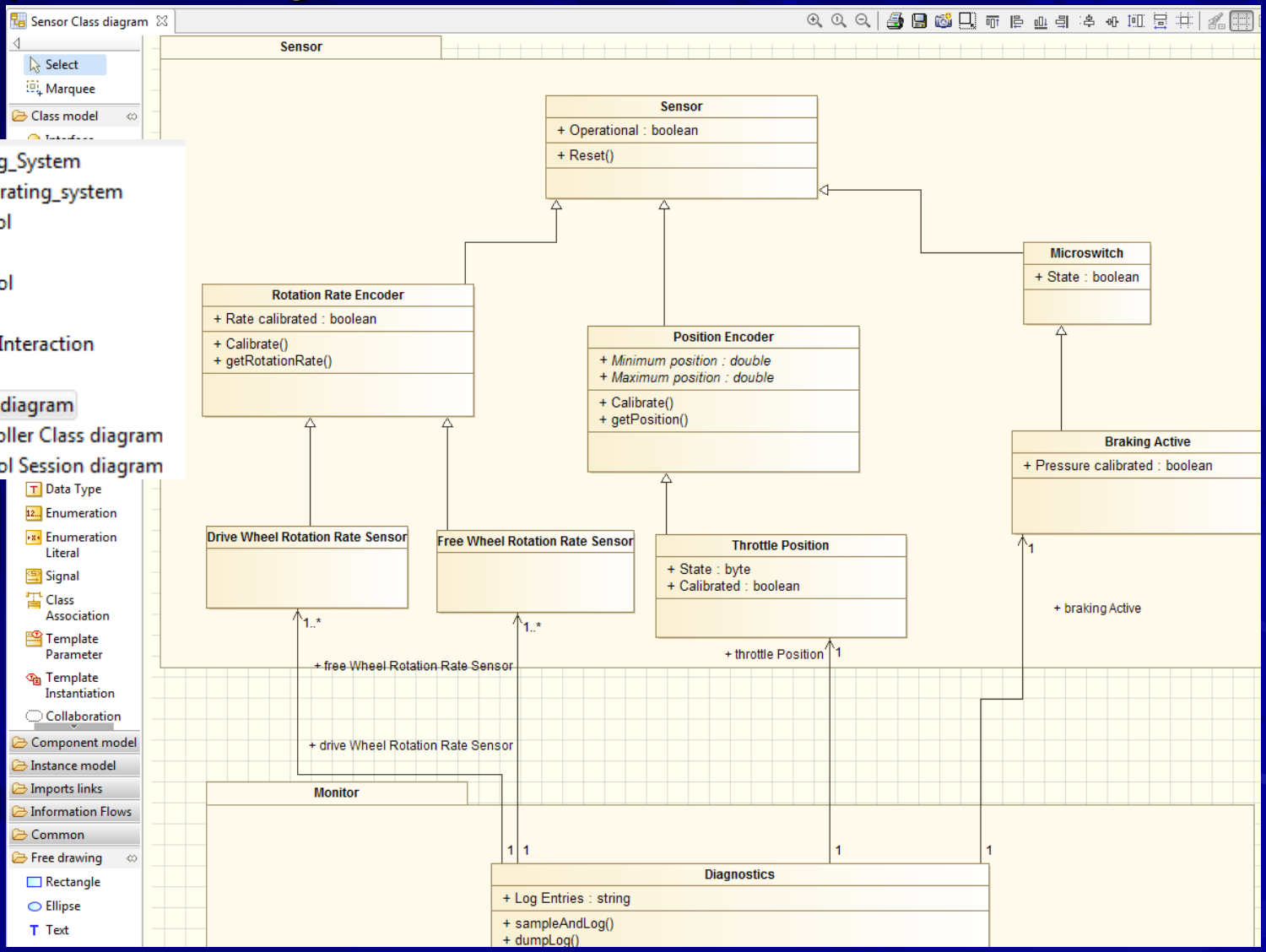


https://www.visual-paradigm.com/guide/uml-unified-modeling-language/what-is-package-diagram/

UML Packages with Class Diagram

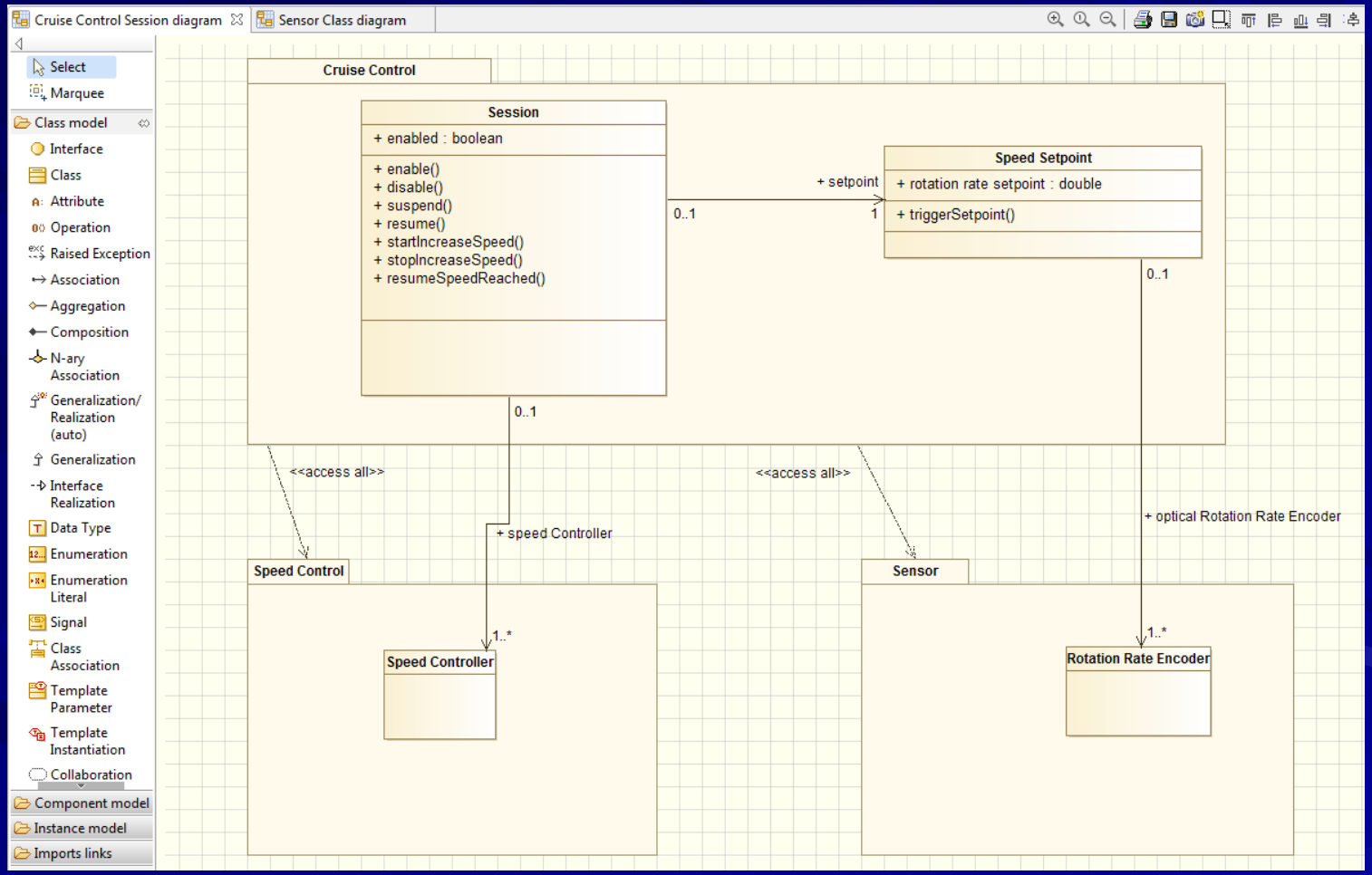
- Break up Class Diagram and abstract





Class Diagram with Packages

- Show detail of cruise control session and interfaces to packages containing related classes

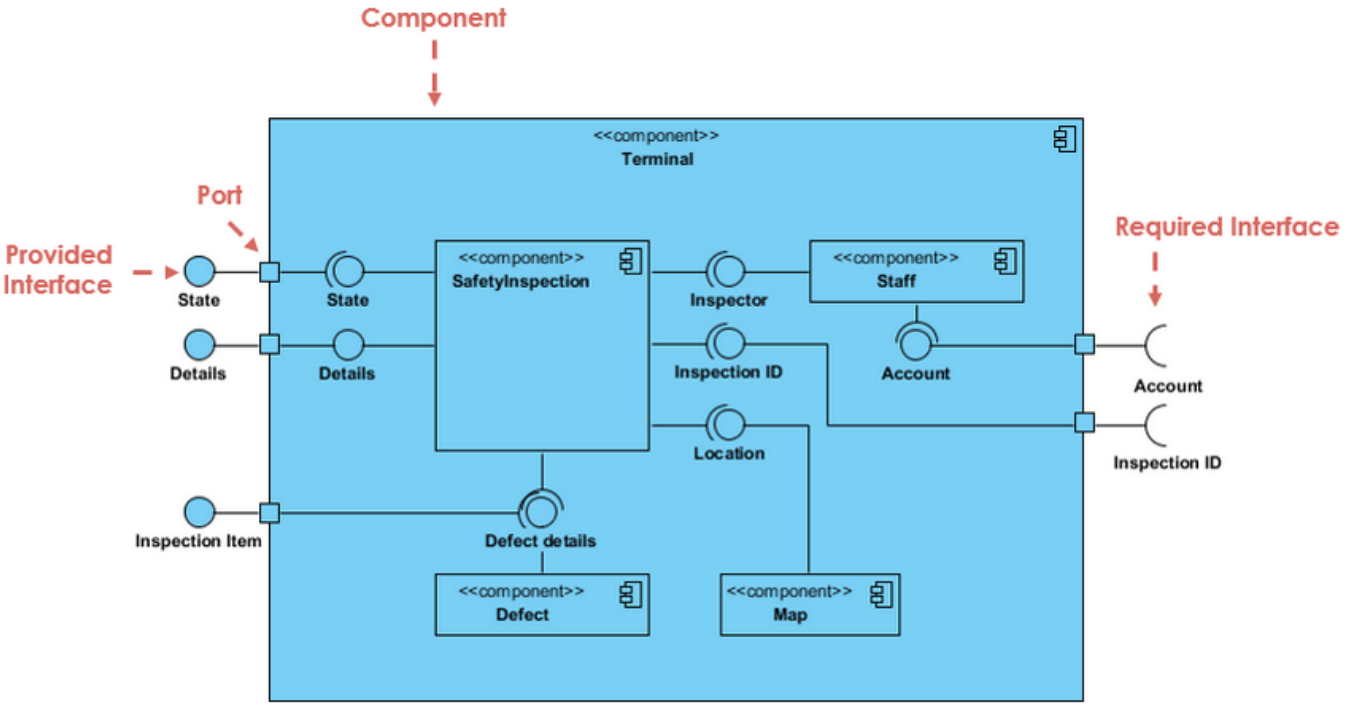


UML Component Diagram - FYI

- Models well concepts of 1) Unit (Module, CSU), 2) Subsystem (CSCI), 3) System (CSC) and Interface Requirements Specificaiton (DoD-2167-A, MIL-STD-498)

Component Diagram at a Glance

A component diagram breaks down the actual system under development into various high levels of functionality. Each component is responsible for one clear aim within the entire system and only interacts with other essential elements on a need-to-know basis.



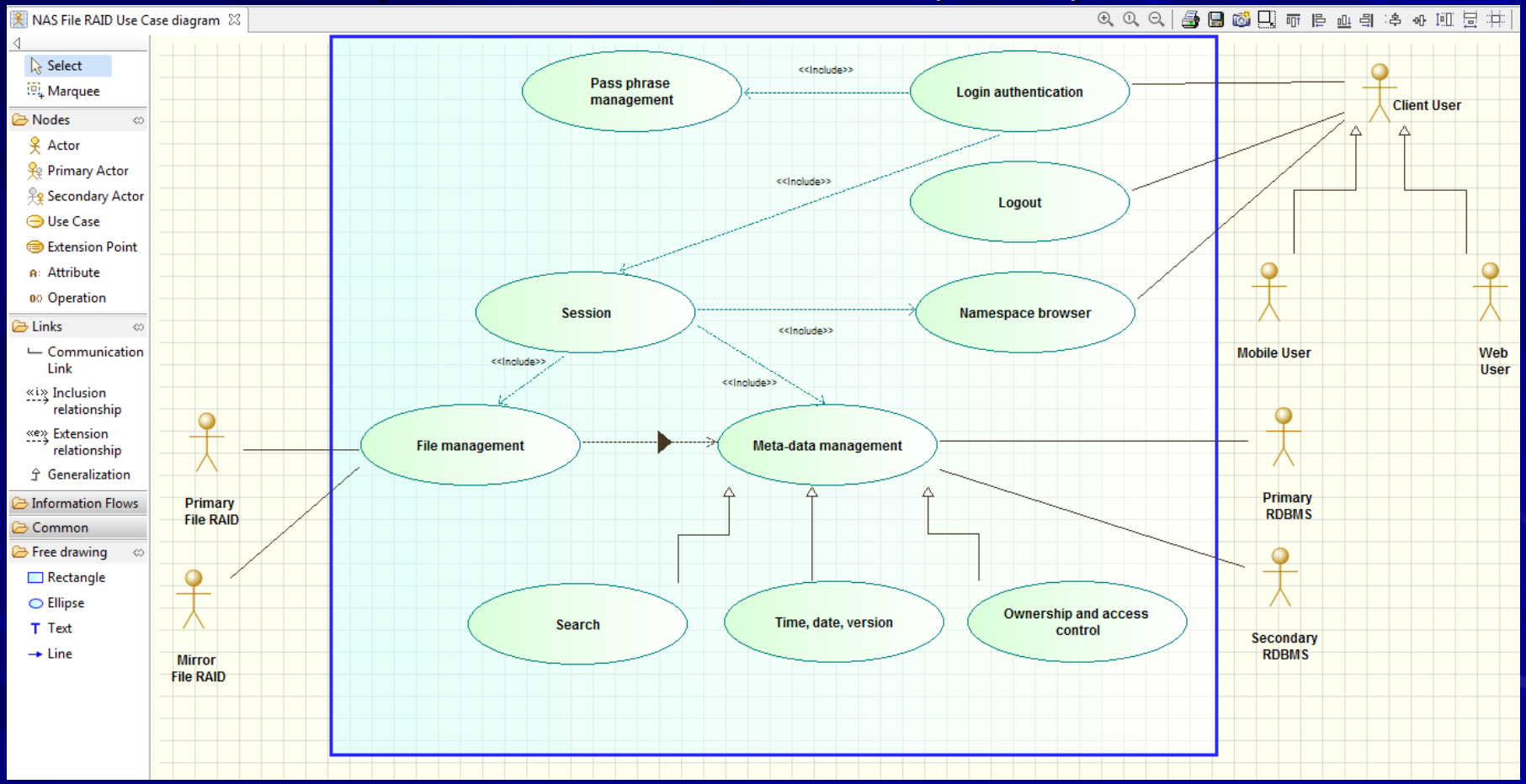
https://www.visual-paradigm.com/guide/uml-unified-modeling-language/what-is-component-diagram/

UML Behavioral Diagrams

Use Case, State, Activity Diagram, OIM Interaction Sequence

#1 - UML Detailed Use Cases

- Show hierarchy, information flow, actors
- Relationships between use cases (ovals)



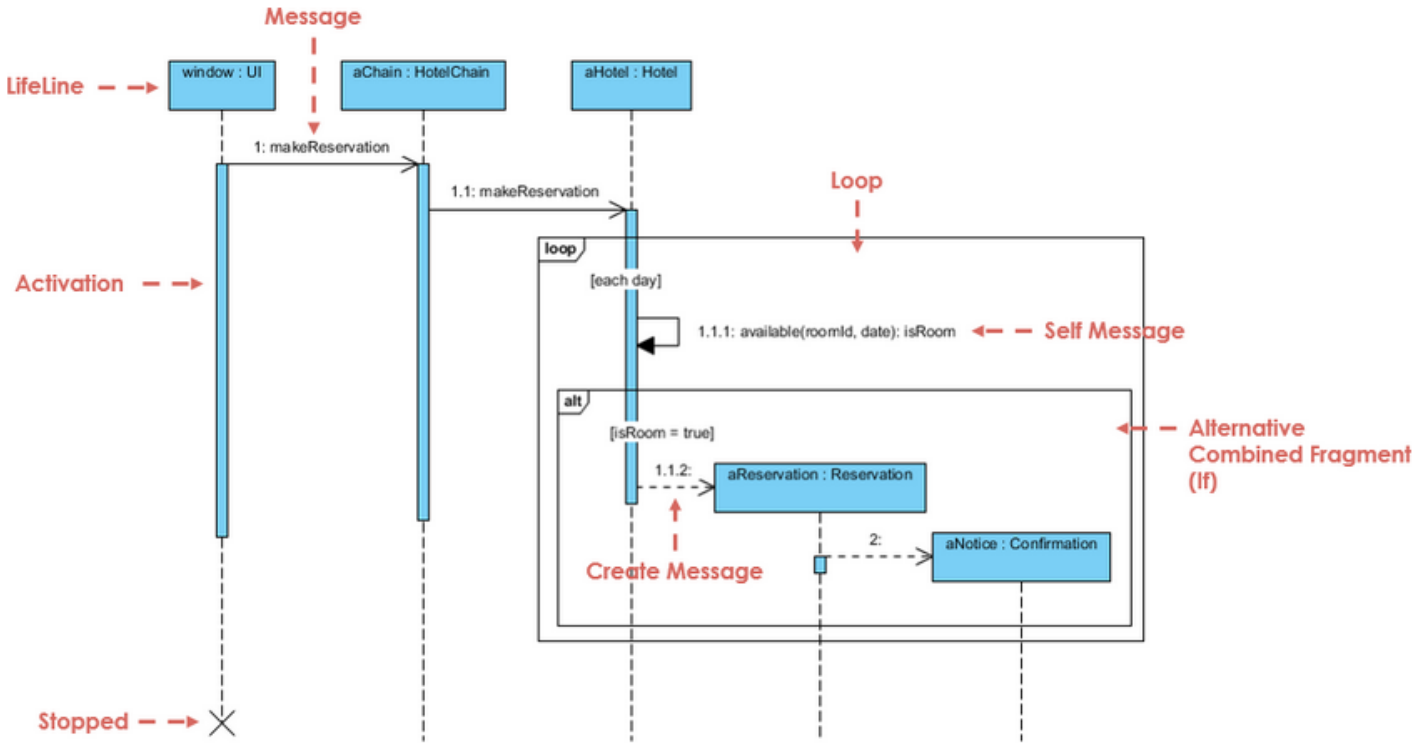
https://www.visual-paradigm.com/guide/uml-unified-modeling-language/what-is-use-case-diagram/

#2 - UML OIM Message Sequence Diagram

- Class only
- Object only
- Class with Object instance
- Level-1 Domain Model with Classes and messages
- Level-2+ with Objects and Methods

Sequence Diagram is an interaction diagram that details how operations are carried out -- what messages are sent and when. Sequence diagrams are organized according to time. The time progresses as you go down the page. The objects involved in the operation are listed from left to right according to when they take part in the message sequence.

Below is a sequence diagram for making a hotel reservation. The object initiating the sequence of messages is a Reservation window.

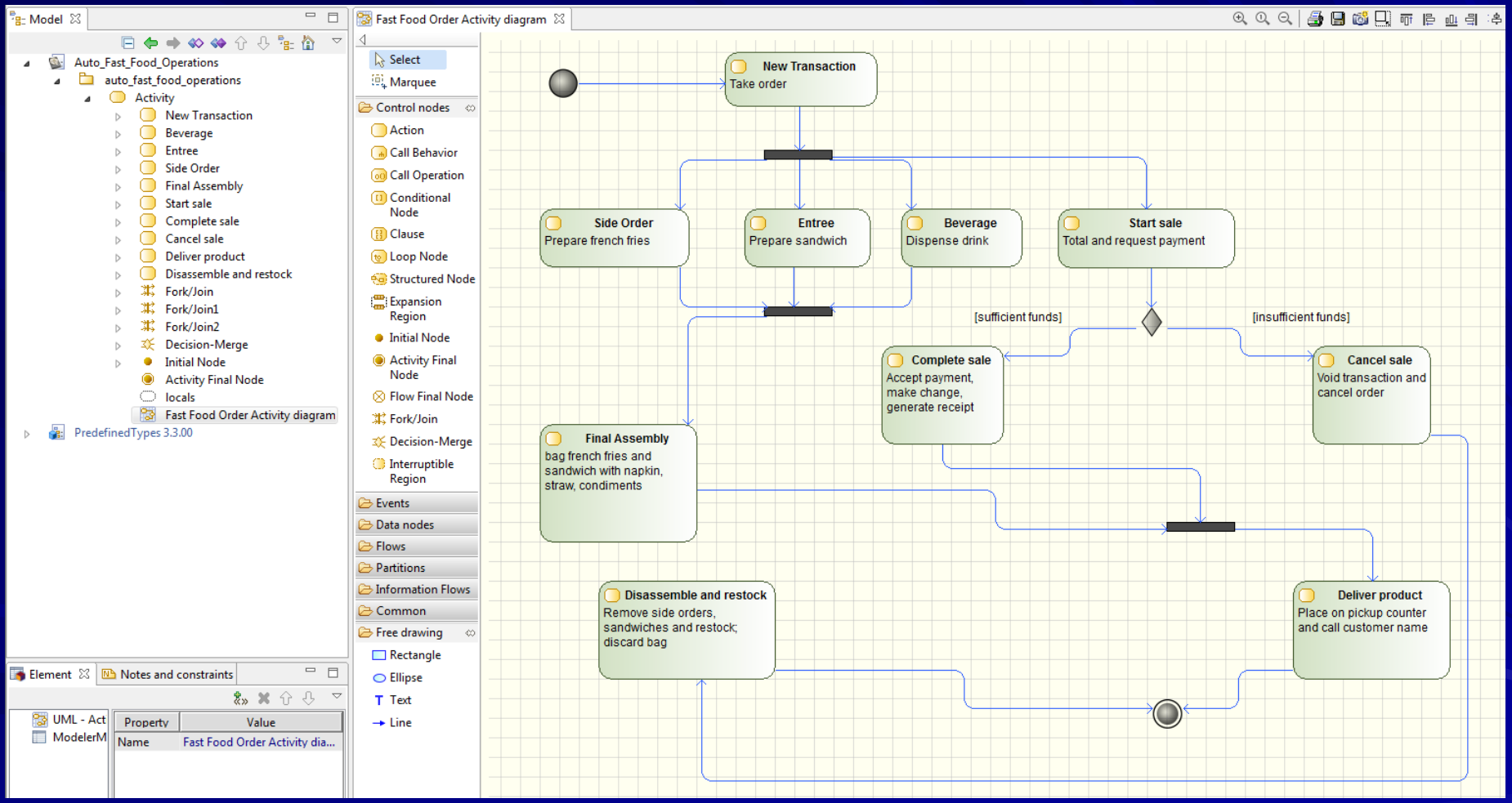


Note That: Class and object diagrams are static model views. Interaction diagrams are dynamic. They describe how objects collaborate.

https://www.visual-paradigm.com/guide/uml-unified-modeling-language/what-is-sequence-diagram/

#3 - UML Activity Diagram

- Flowchart with fork and join for concurrency, Start, Finish
- Use to model process bars in OIM sequence diagram



https://www.visual-paradigm.com/guide/uml-unified-modeling-language/what-is-activity-diagram/

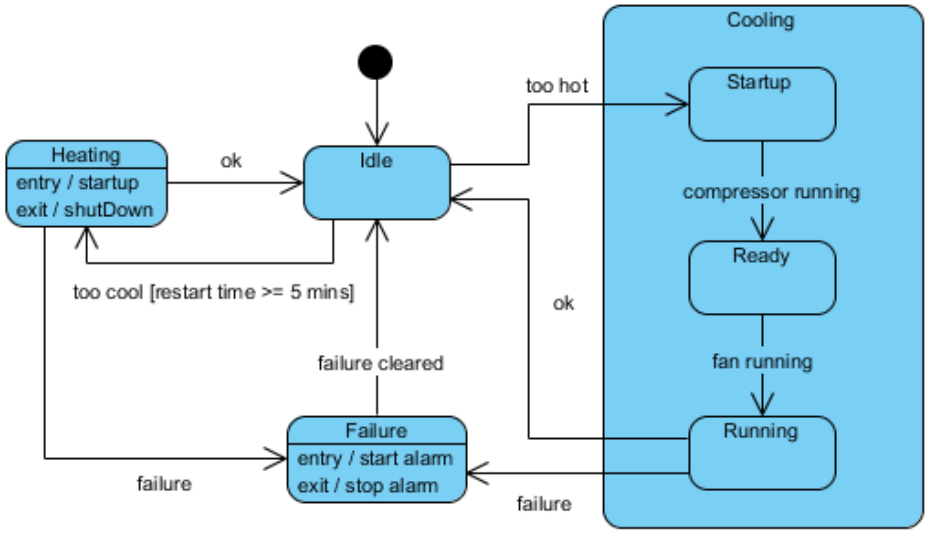
#4 - UML State Machine (Extended)

- Useful at any Level (0,1,2, 2+) for Event Driven Architectures and Hybrid Architectures
- State Machine Hierarchy of States with sub-states

Substates

A simple state is one which has no substructure. A state which has substates (nested states) is called a composite state. Substates may be nested to any level. A nested state machine may have at most one initial state and one final state. Substates are used to simplify complex flat state machines by showing that some states are only possible within a particular context (the enclosing state).

Substate Example - Heater



https://www.visual-paradigm.com/guide/uml-unified-modeling-language/what-is-state-machine-diagram/

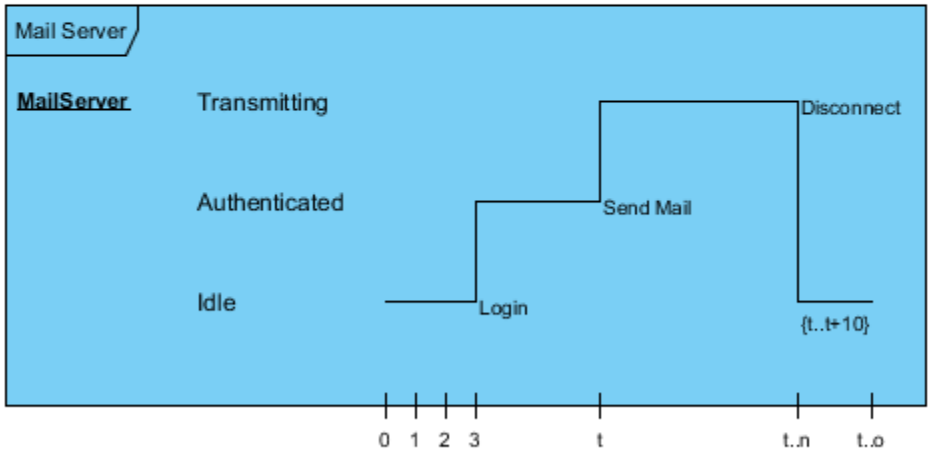
UML Timing Diagram - FYI

- Use for time constraints (if you have them)
- Rate Monotonic Analysis - CEC450 (Example Timing Diagrams)

Timeline and Constraints

We can use the length of a timeline to indicate how long the object remains in a particular state by reading it from left to right. To associate time measurements, you show tick marks online the bottom part of the frame.

The example below shows that the Login event is received three time units after the start of the sequence. To show relative times, you can mark a specific instance in time using a variable name. The figure marks the time the sendMail event is received as time



You can use relative time marks in constraints to indicate that a message must be received within a specified amount of time.

https://www.visual-paradigm.com/guide/uml-unified-modeling-language/what-is-timing-diagram/

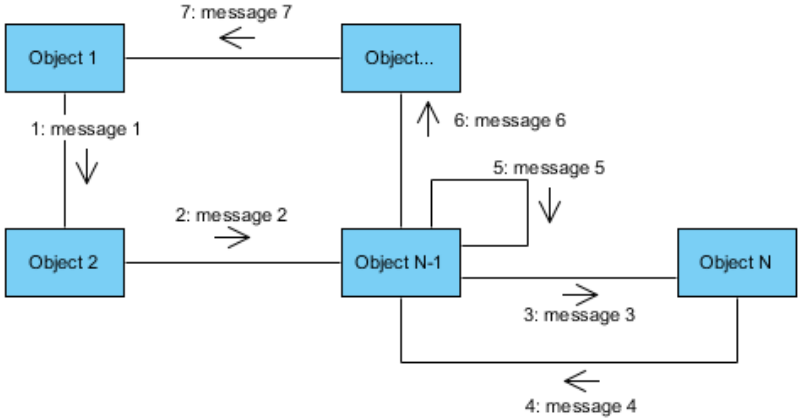
UML Communication Diagram - FYI

- Use instead of or in addition to OIM Sequence Diagram
- Use with Object Model
- Analysis for Domain Model

Communication Diagram at a Glance

In the example of the notation for a communication diagram, objects (actors in use cases) are represented by rectangles. In the example (generic communication diagram):

- The objects are Object1, Object2, Object..., ObjectN-1 ..., and ObjectN.
- Messages passed between objects are represented by labeled arrows that start with the sending object (actor) and end with the receiving object.
- The sample messages passed between objects are labeled 1: message1, 2: message2, 3: message3, etc., where the numerical prefix to the message name indicates its order in the sequence.
- Object1 first sends Object2 the message message1, Object2 in turn sends ObjectN-1 the message message2, and so on.
- Messages that objects send to themselves are indicated as loops (e.g., message message5).



https://www.visual-paradigm.com/guide/uml-unified-modeling-language/what-is-communication-diagram/