Capella

python

Google Antigravity

3 Day training for
Functional architecture design and validation

Sejong Oh (sejong5.kr@gmail.com)

Training Contents Overview

1st Day :

- What is state machine and how to make
  - ✓CountDown example
  - ✓Washing Machine example
- How to define SW requirements based on state machine
  - ✓Requirement Viewpoint of Capella
  - ✓Extract requirement traceability from Capella model using Python

Training Contents Overview

2nd Day :

- What is a system and how to design using SysML
  - ✓Make use case scenario
  - ✓Derive state machines from the sequence diagrams of use case
- Publish your model via Web dashboard
  - ✓How to work as a team to make robust design using Capella
    - ➢ Share and collaborate based on model using Sourcetree (Git Server)
  - ✓How to use HTML5 generator of Capella
    - ➢ Publish your model via web pages and learn how to navigate
- Report your model via MS word document
  - ✓Prepare your own docx template using M2Doc (Obeo)
  - ✓Generate your MS word reports automatically

Training Contents Overview

3rd Day :

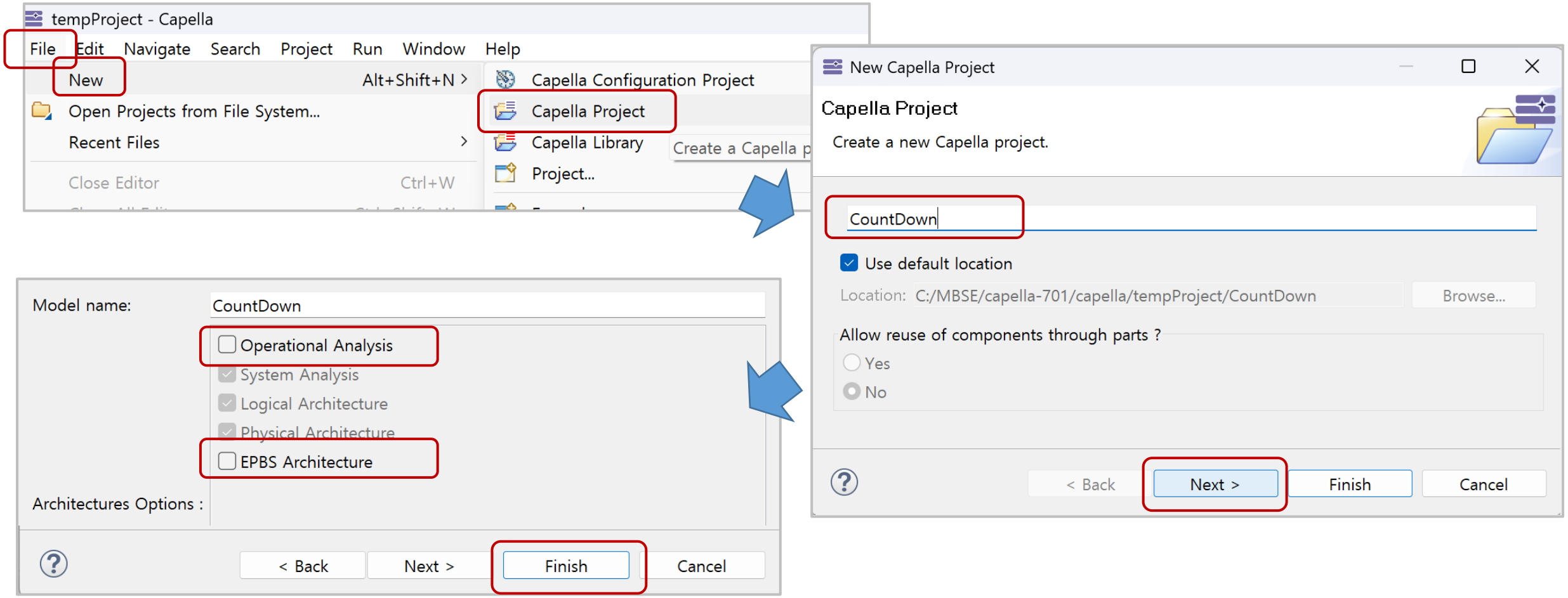
- How to make your own application on Raspberry-Pi
  - ✓ Using python interpreter
  - ✓ Translate and make C application
- Team project
  - ✓ Select a system and define goals of your model
    - ➢ Build a team and design together
    - ➢ Define system and software requirements based on your model
    - ➢ Import legacy design artifacts into your Capella model using Python4Capella
      - ➢ Requirements, CAN DB, SW functions, Interfaces and exchange elements
    - ➢ Design functional architecture of your system
    - ➢ Validate and automatically publish into web dashboard and MS word

1st Day training

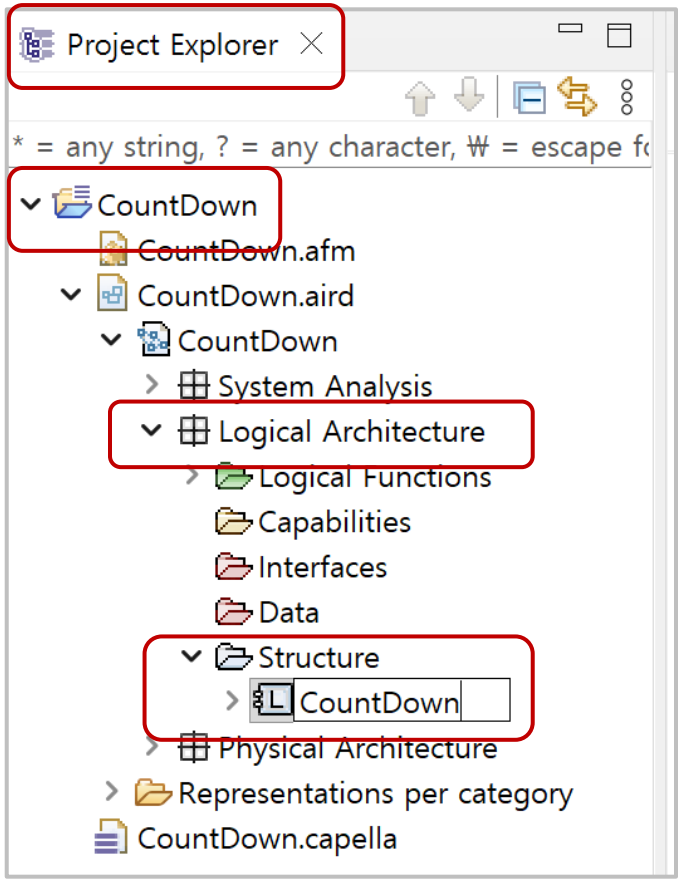
CountDown
Washing Machine

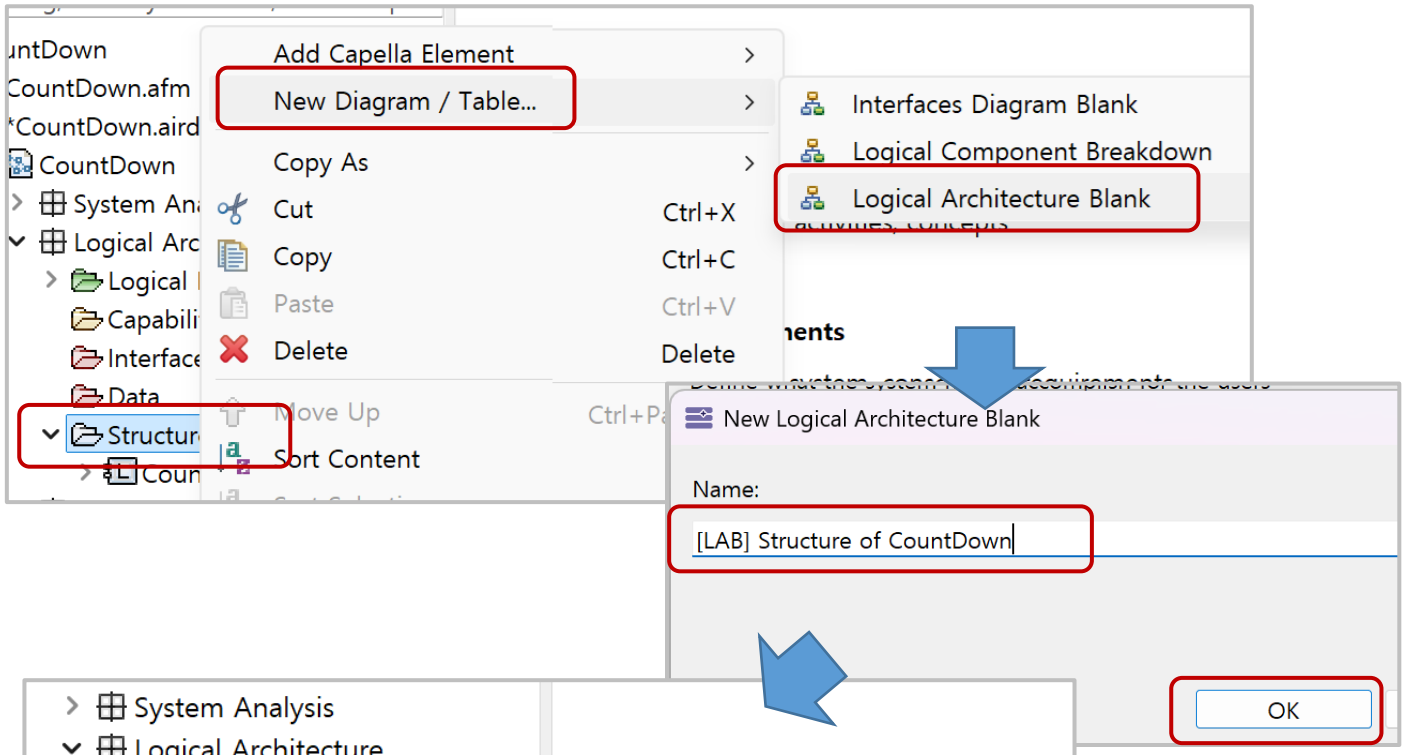
What is state machine and how to make

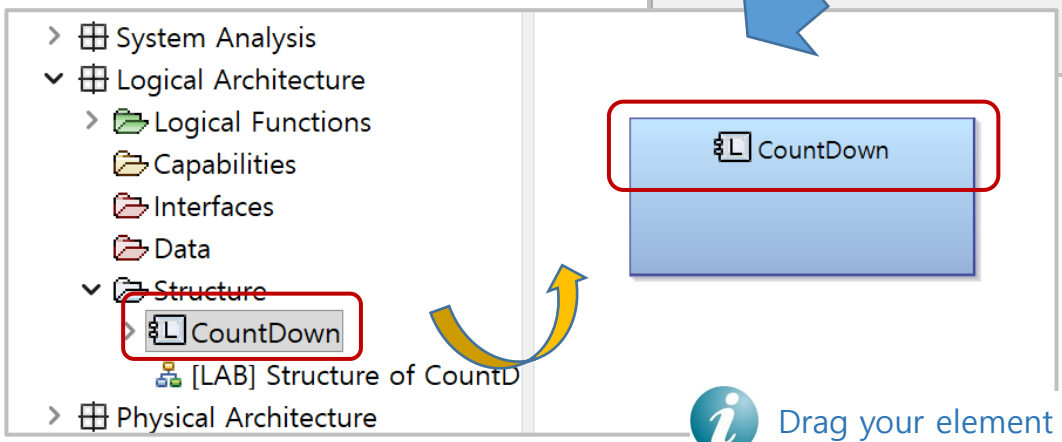
- Launch a new project in Capella



Define structural architecture





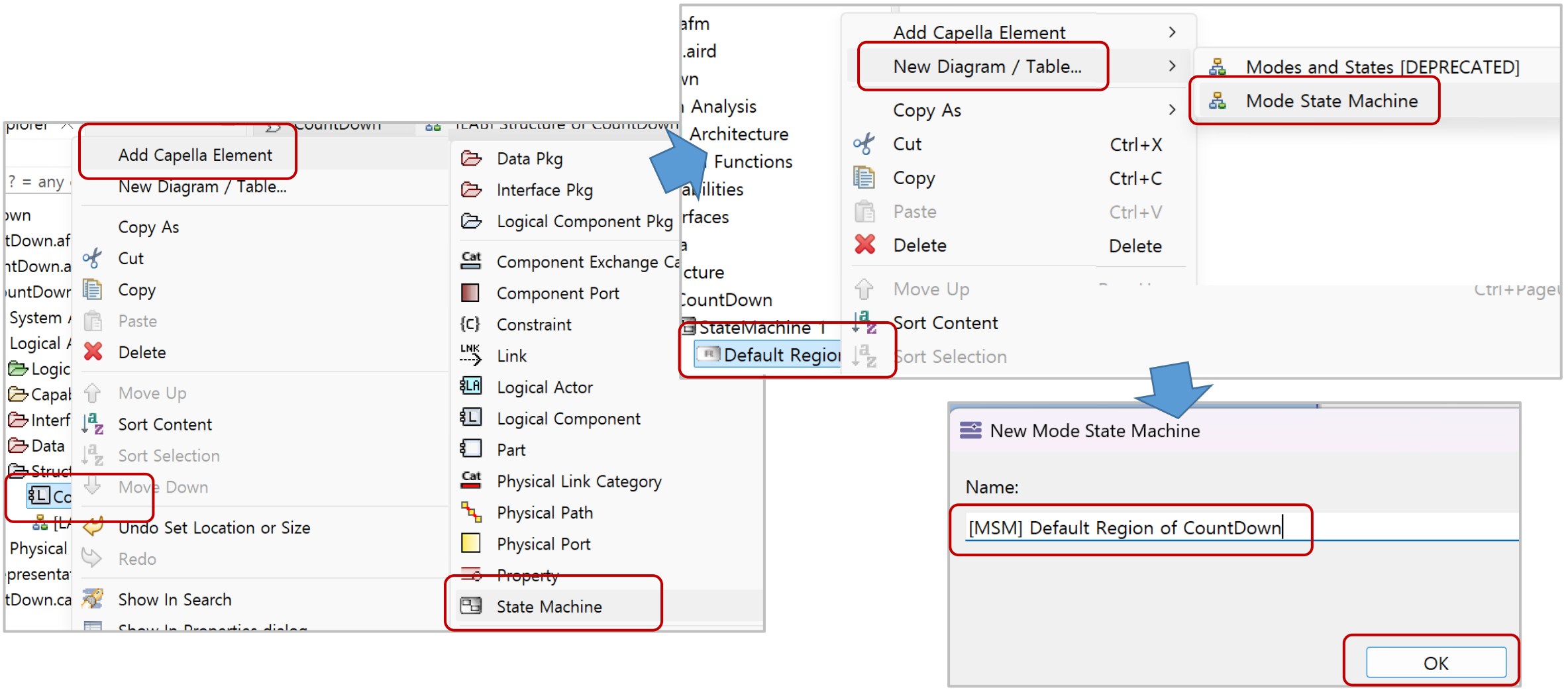


Press F2 button for editing element's name

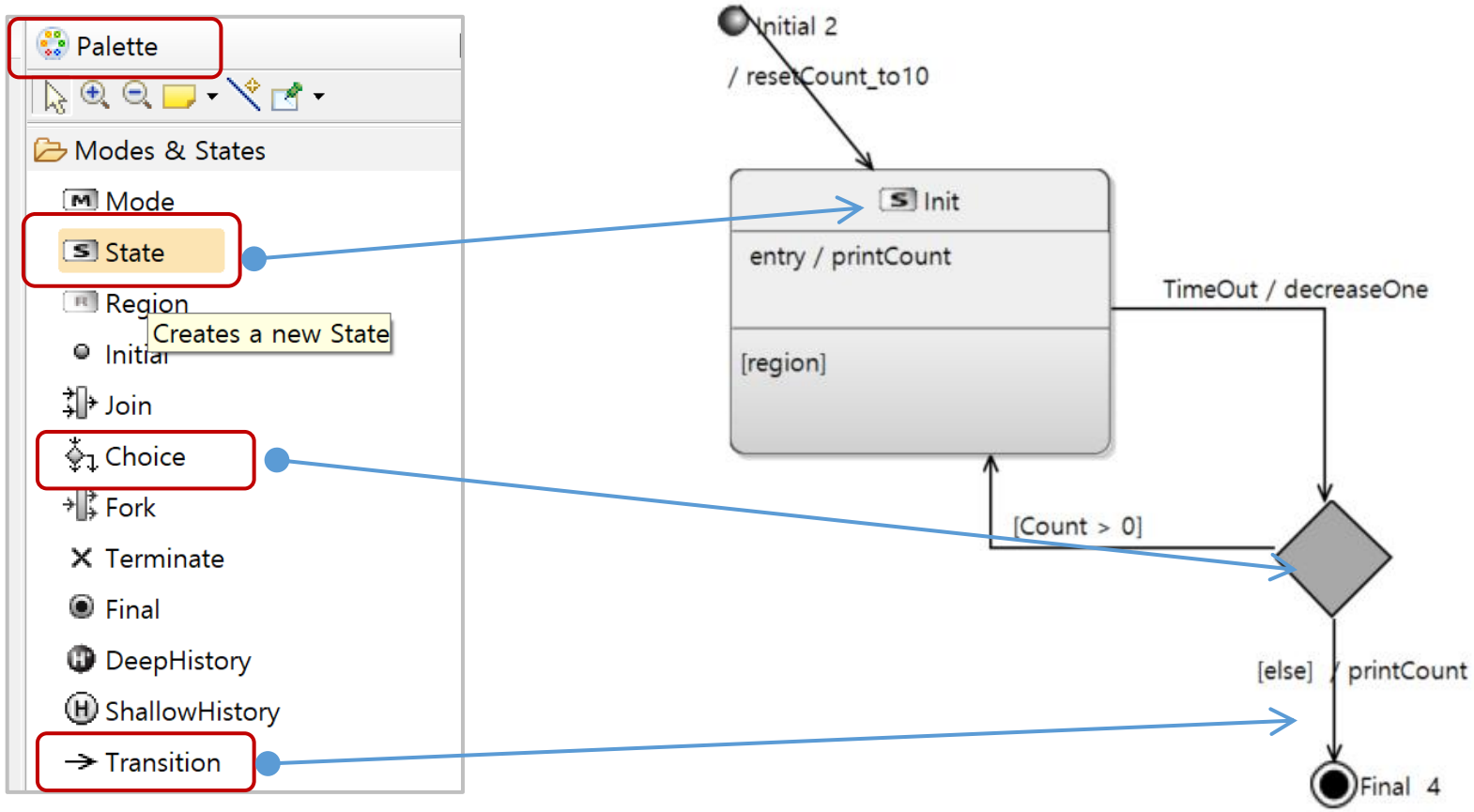
Press Ctrl+z button to undo the last editing

Drag your element into the diagram editor

Add a state machine to a SW component



Build behavioral models in a state machine



Mode and State are same but not be used at the same time

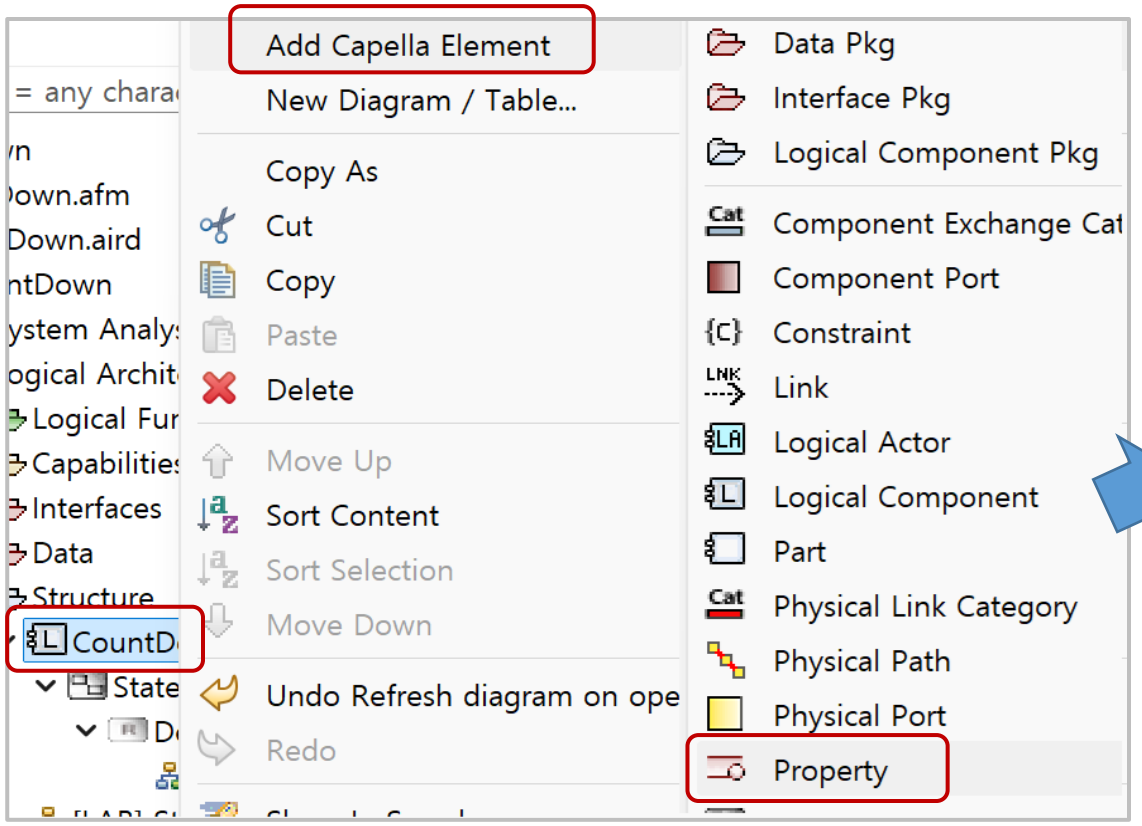
Tips for state machine model

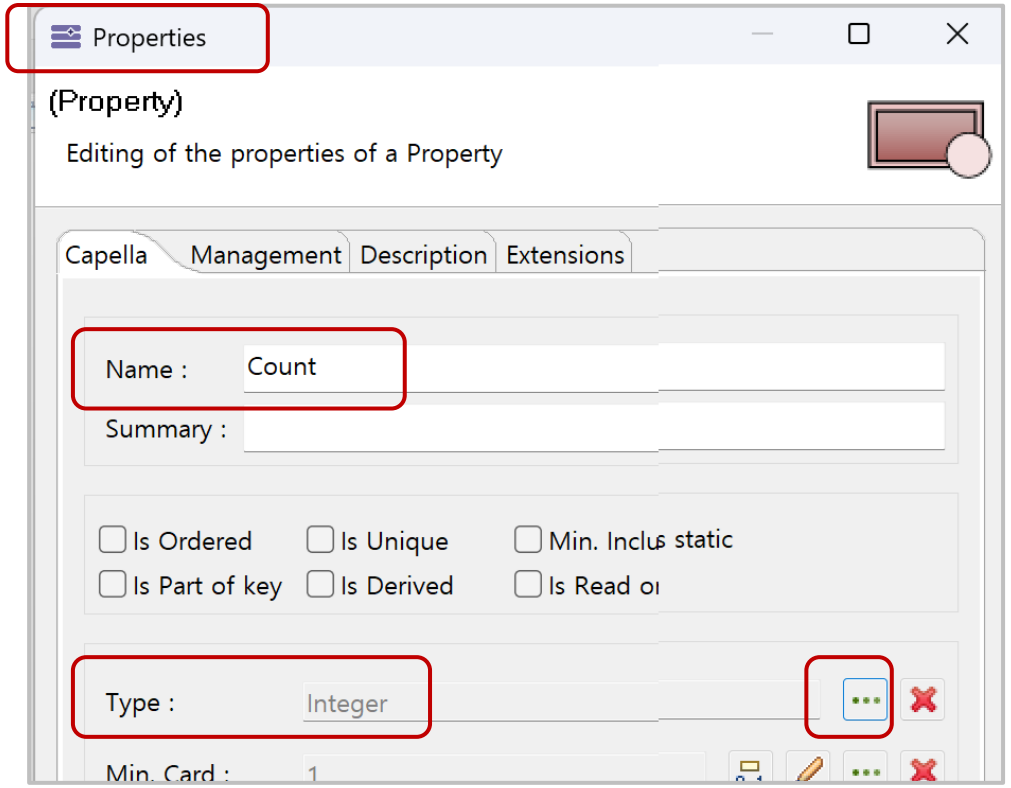
- 3 major information that we can get from state machine :
  - ✓ **Trigger :** A type of message that actively moves an object from state to state
  - ✓ **Guard condition :** A Boolean condition that allows or stops a transition
  - ✓ **Action :** A behavior that results when a state transitions
    - ➢ 4 types of actions : On transition from one state to others, On entry to a state, Do in a state, On exit out of a state

- Benefits of state machine in functional specifications :
  - ✓ **Modular development :** Suppliers can read and understand ECU's complex behavior in easier way of state by state
  - ✓ **Modular validation :** OEM can validate the complex behavior of ECU from suppliers in a clearer way of state by state
  - ✓ **Automation :** Some advanced tools (e.g. Google AntiGravity) can automatically generate runnable codes or executable test cases from this state machine

Add operational parameters into a module

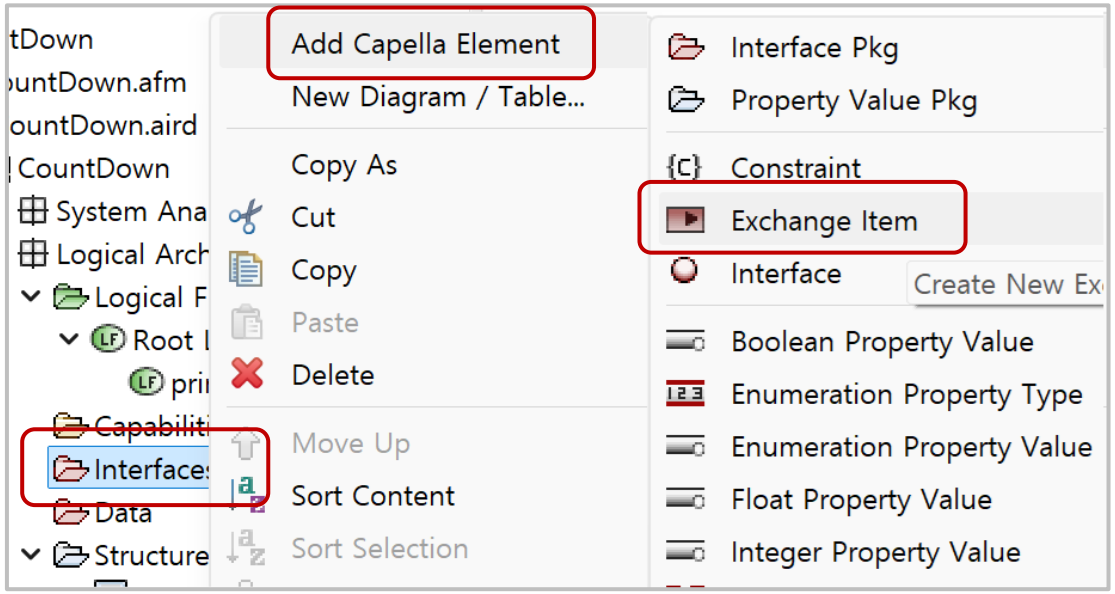
- You can add member variables to ***a module*** or static variables in ***Data package***

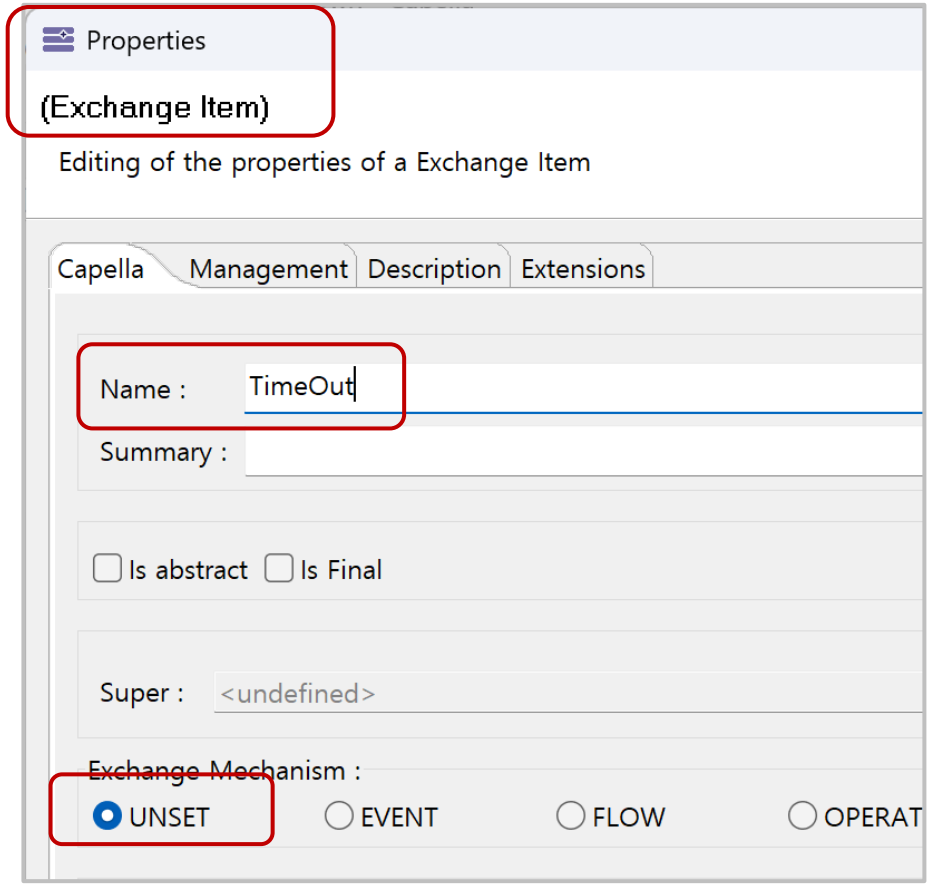




Add triggers and interface in a model

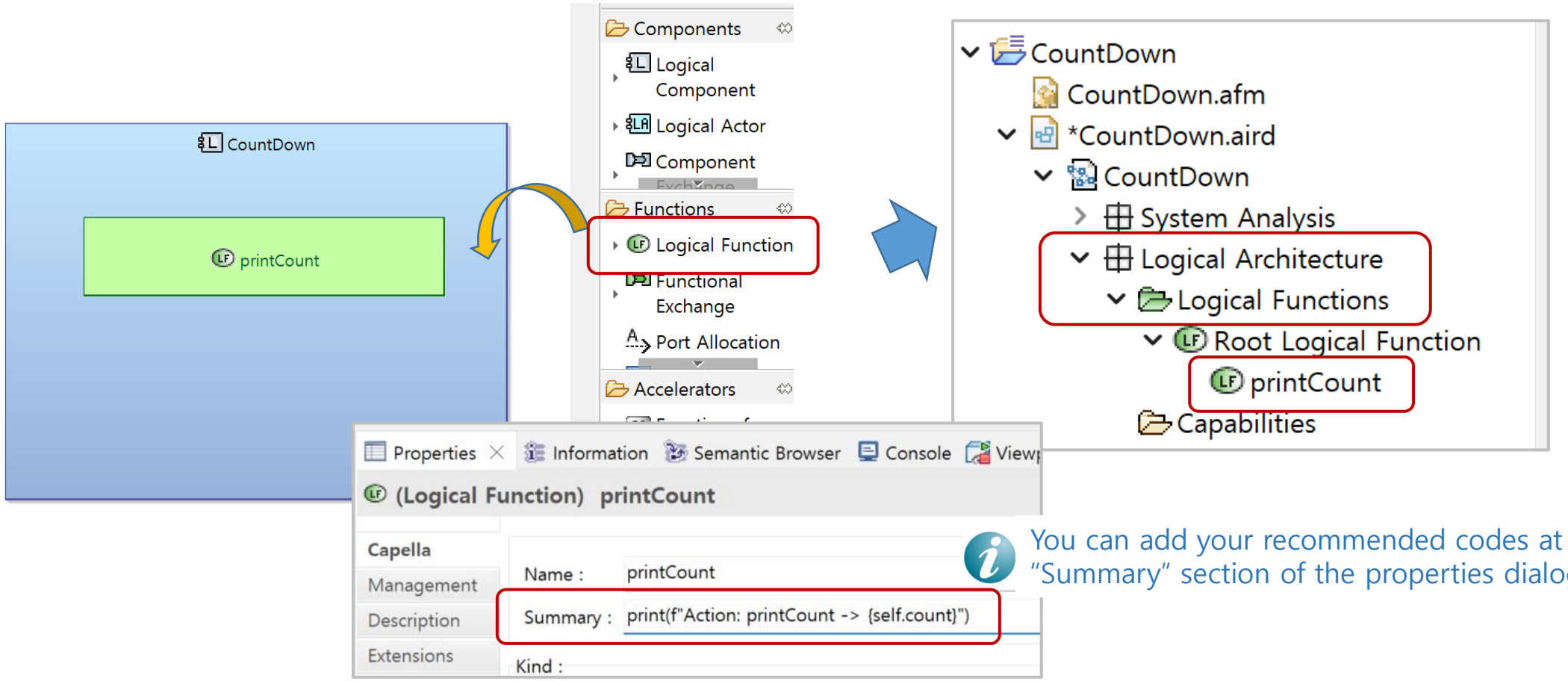
- Triggers can be assigned to a specific interface or not
- Triggers will be used for state transition





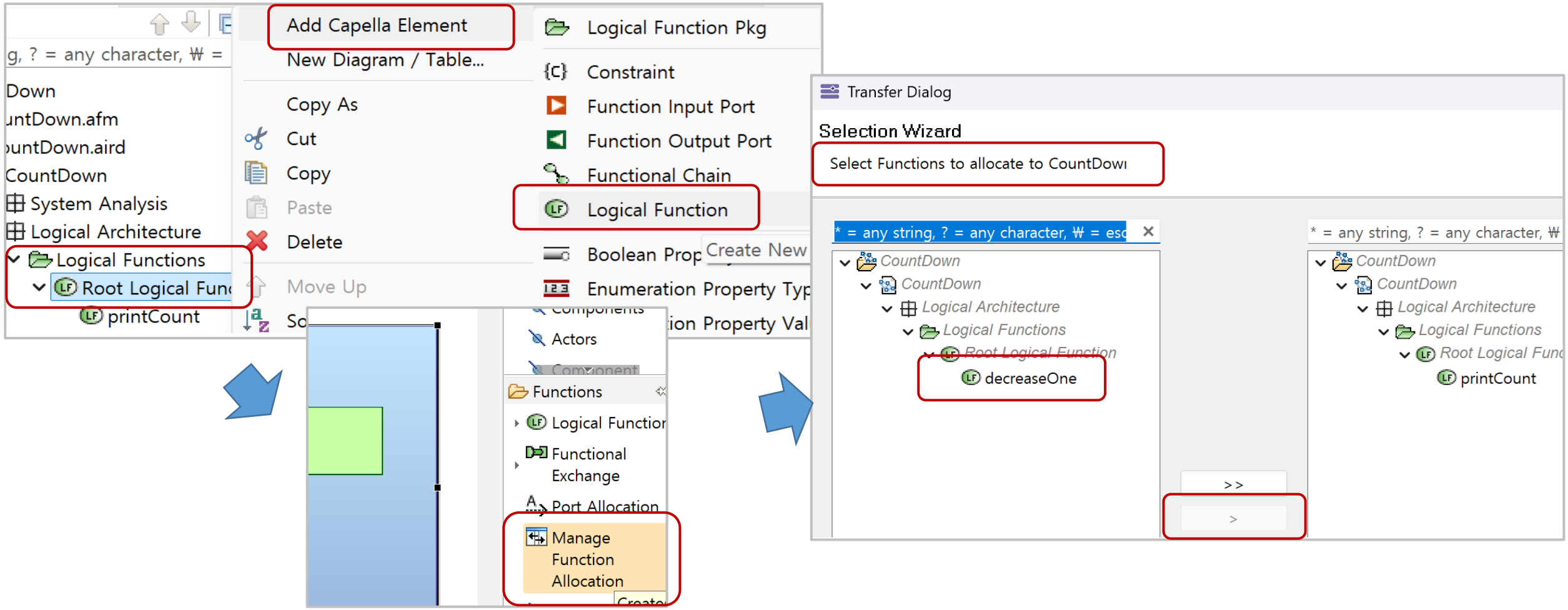
Add functions into a module (1/2)

- In Capella, all functions are shown in “Logical Functions” package but all of them are allocated into specific modules



Add functions into a module (2/2)

- Another way to add functions into a module



Validate your state machine with AntiGravity

- Let’s complete the other functions of module
- Use this prompt to make a python code for CountDown

Prompt to AntiGravity >
The CountDown project contains a single logic component named "CountDown" and a state machine. Please write the Python code for this component and state machine.
You can refer to the recommended codes provided in the summary section for each component.
In this model, change the TimeOut event to an actual 1 second timeout.

End of Exercise 1

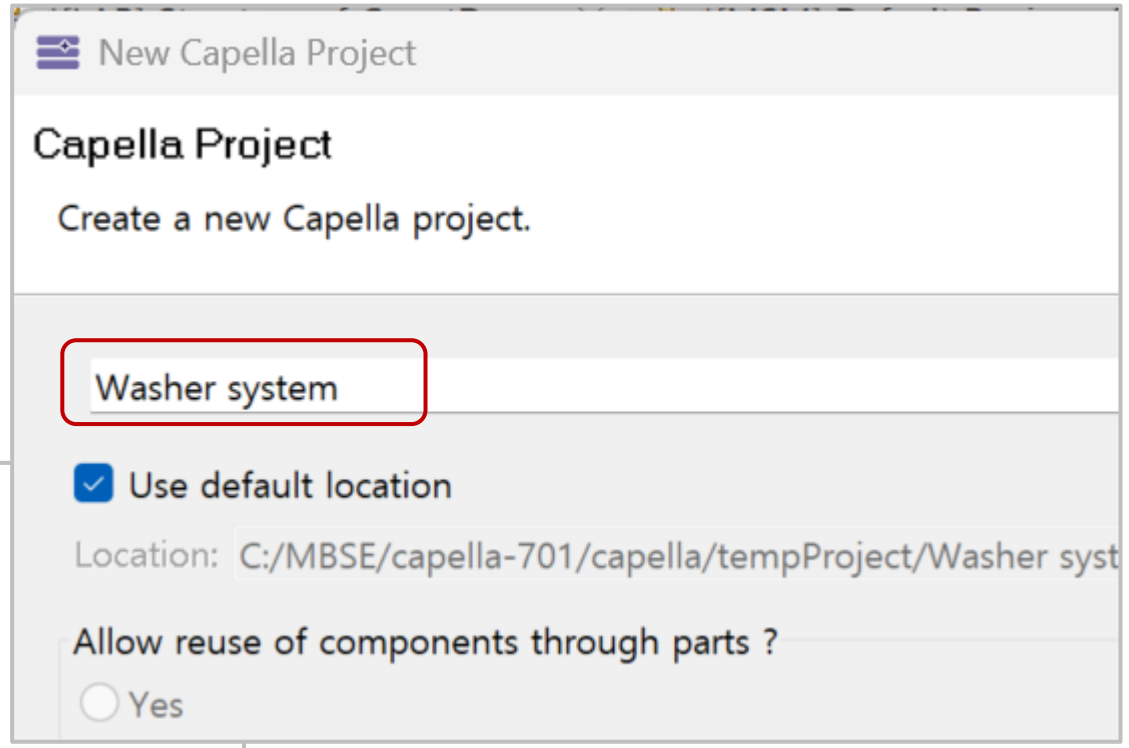
System Requirements

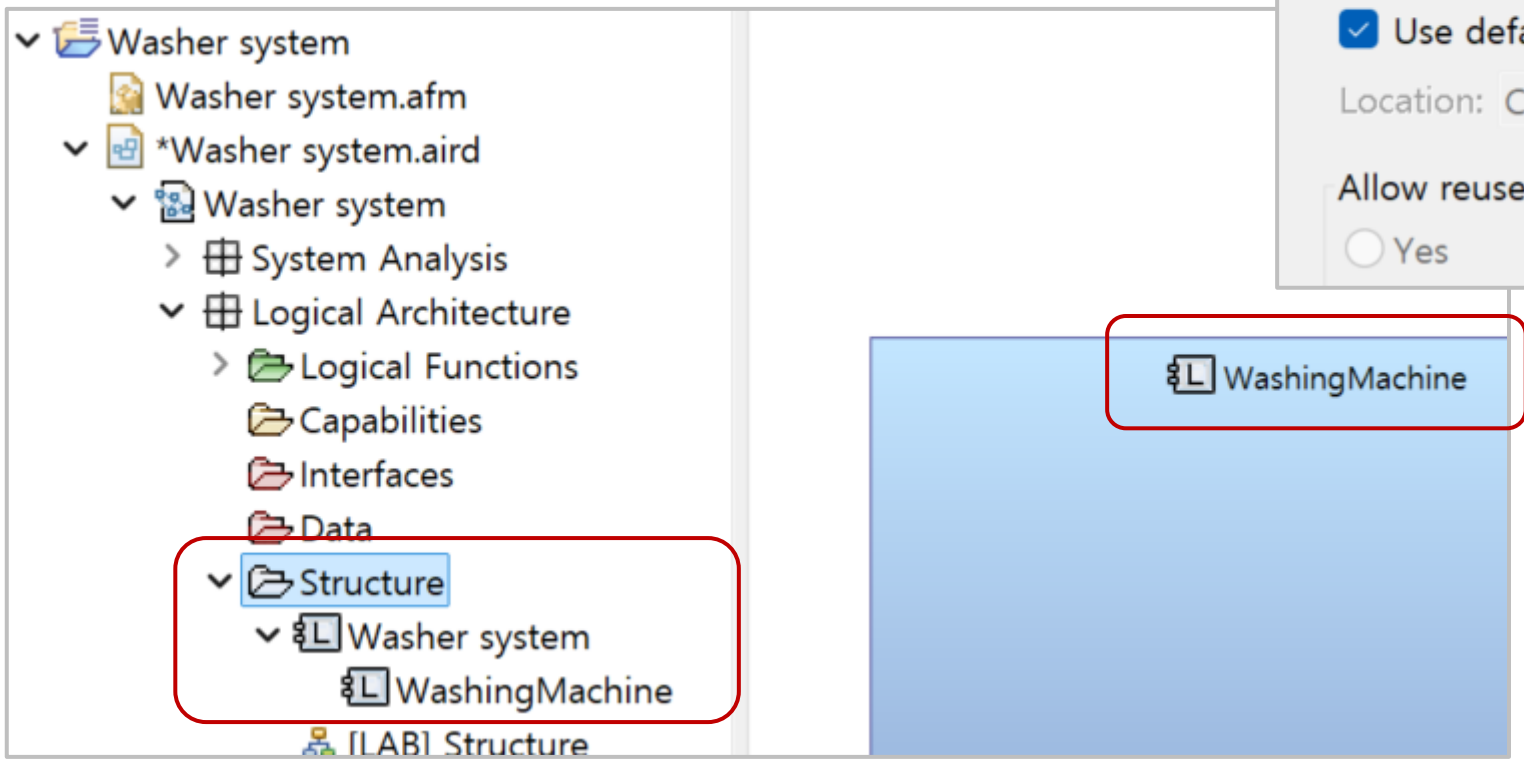
■ Functional requirements for a simple washing machine

[1] The washing machine can be set a specific number of shampoo, rinse, and dehydrate cycles.
[2] When the top door is opened, the machine stops operating and a warning light illuminates.
[3] When the top door is closed, the machine resumes operation and continues to run.
[4] Once all washing cycles are complete, the machine will wait for a set amount of time and then automatically turn off.

Setup a new project

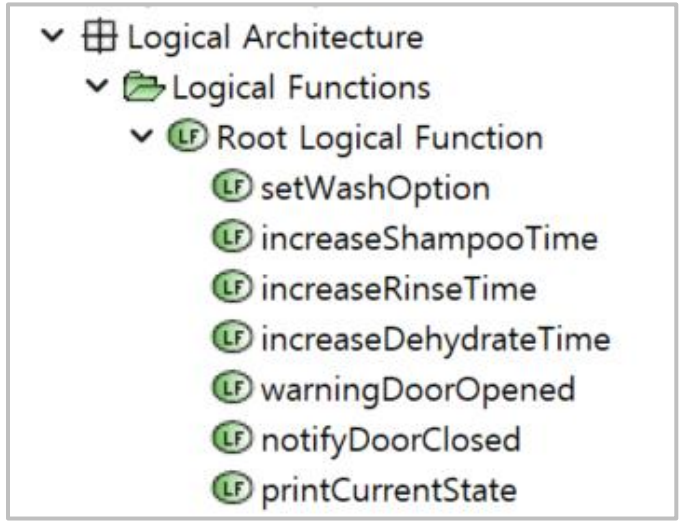
- In Washer system project
- Make WashingMachine module

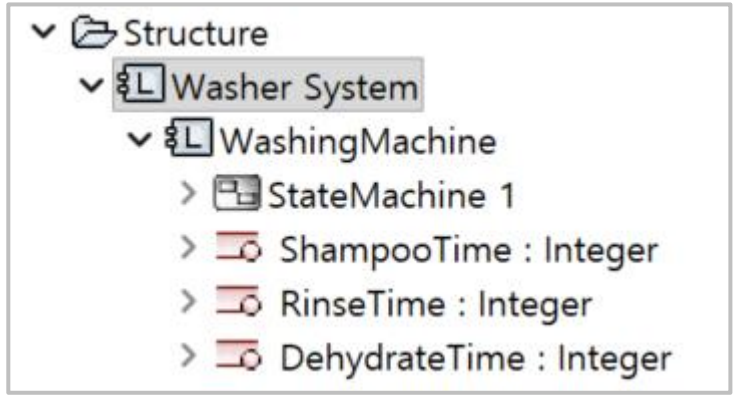




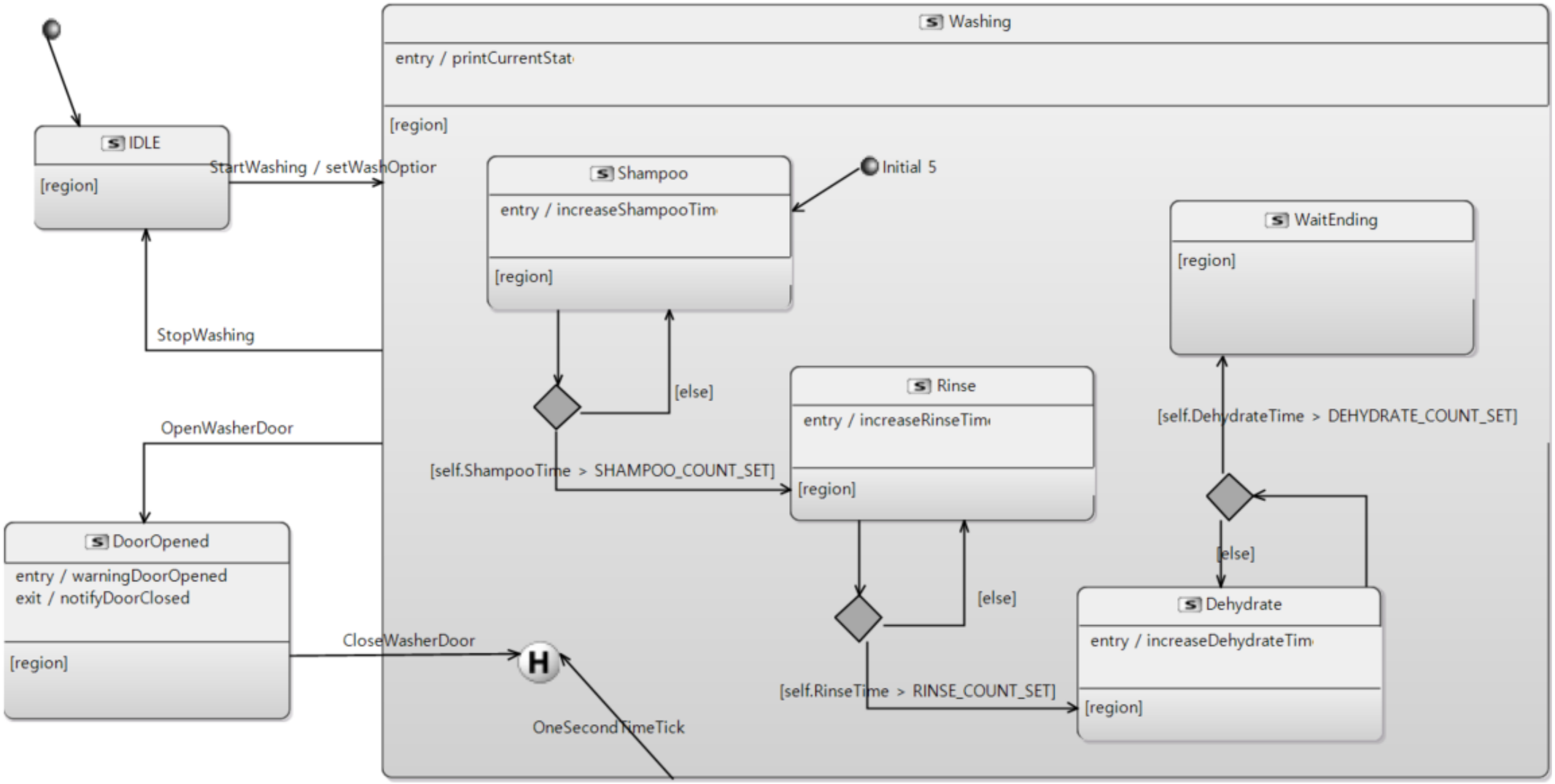
Define elements for state machine

- Define (1)triggers (2)guard conditions and (3)actions
  - StartWashing, StopWashing, OpenWasherDoor, CloseWasherDoor
  - OneSecondTimeTick
  - setWashOption(), warningDoorOpened(), notifyDoorClosed(), printCurrentState(), increaseShampooTime(), increaseRinseTime(), increaseDehydrateTime()
- Define operational parameters
  - ShampooTime, RinseTime, DehydrateTime

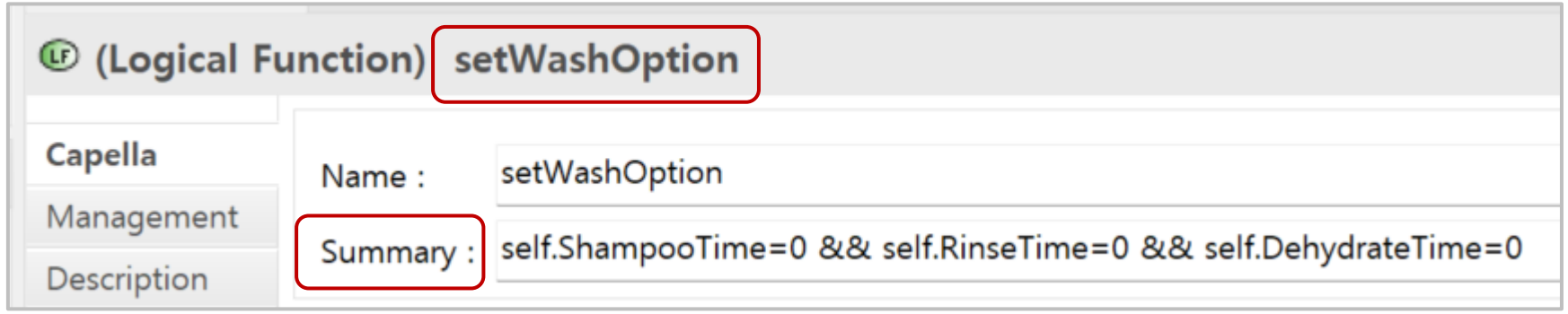


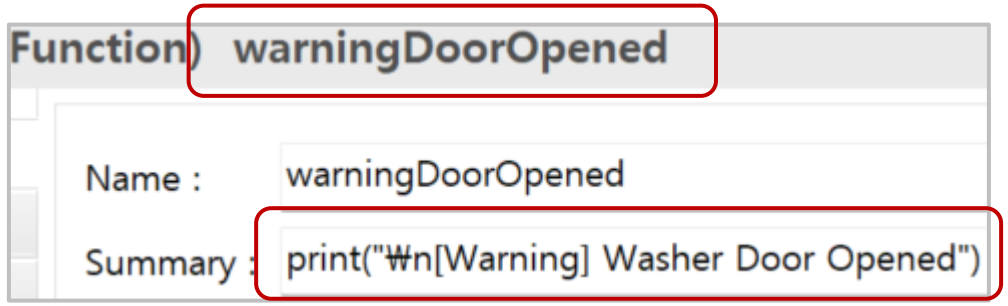


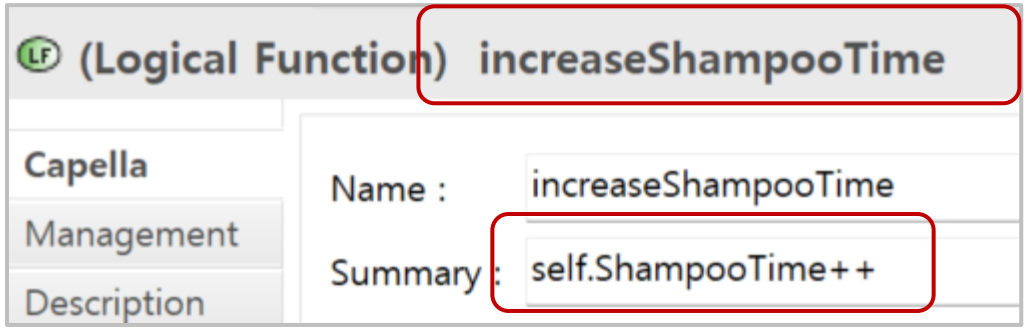
Make a state machine



Fill recommended codes into functions



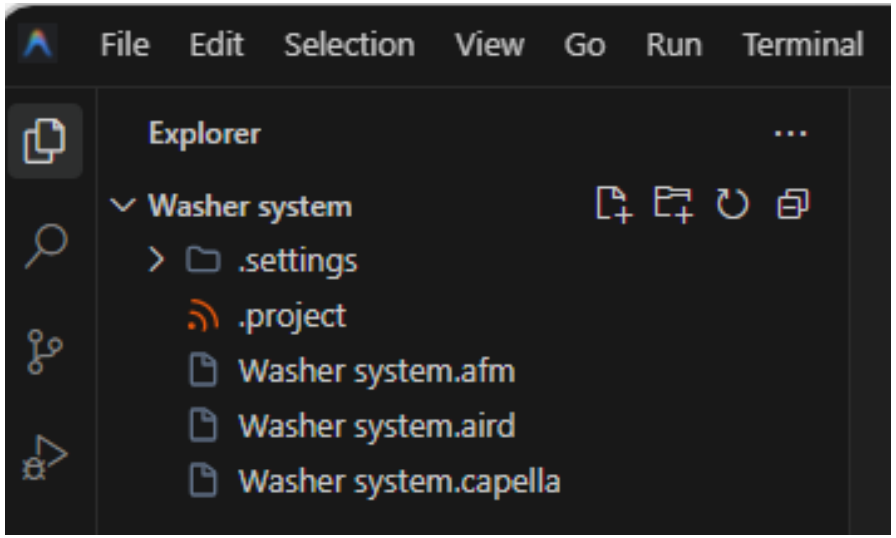




AntiGravity takes into account your recommendations while generating Python code

Generate codes and run the model

■ Open your workspace in AntiGravity and prompt like this >



Prompt to AntiGravity >
The "***Washer system***" project contains a single logic component named "***WashingMachine***" and a state machine. Please write the Python code for this component and state machine. You can refer to the recommended codes provided in the summary section for each component.
In this model, change the ***OneSecondTimeTick*** event to an actual 1 second timeout.

Validate model with your scenario

- What happens if the door opens during shampooing?
- What happens if the door closes after two seconds?
- Will the machine return to its previous shampooing state?
- What happens when the washing cycle is complete?
- What happens if you forcefully stop the washing cycle during the rinse cycle?
- Write test code in Python and verify that the model works as intended

End of Exercise 2

Requirement Traceability

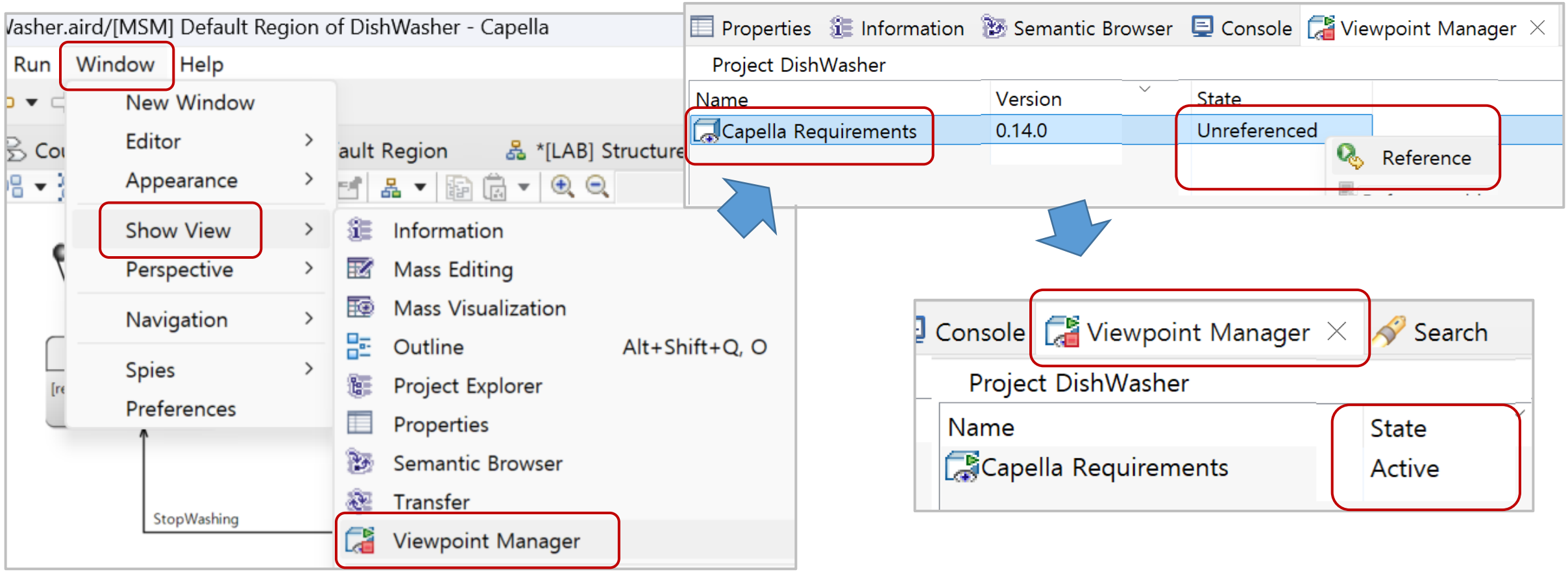
Using Requirement ViewPoint and Python4Capella

Add Requirement Viewpoint to Capella

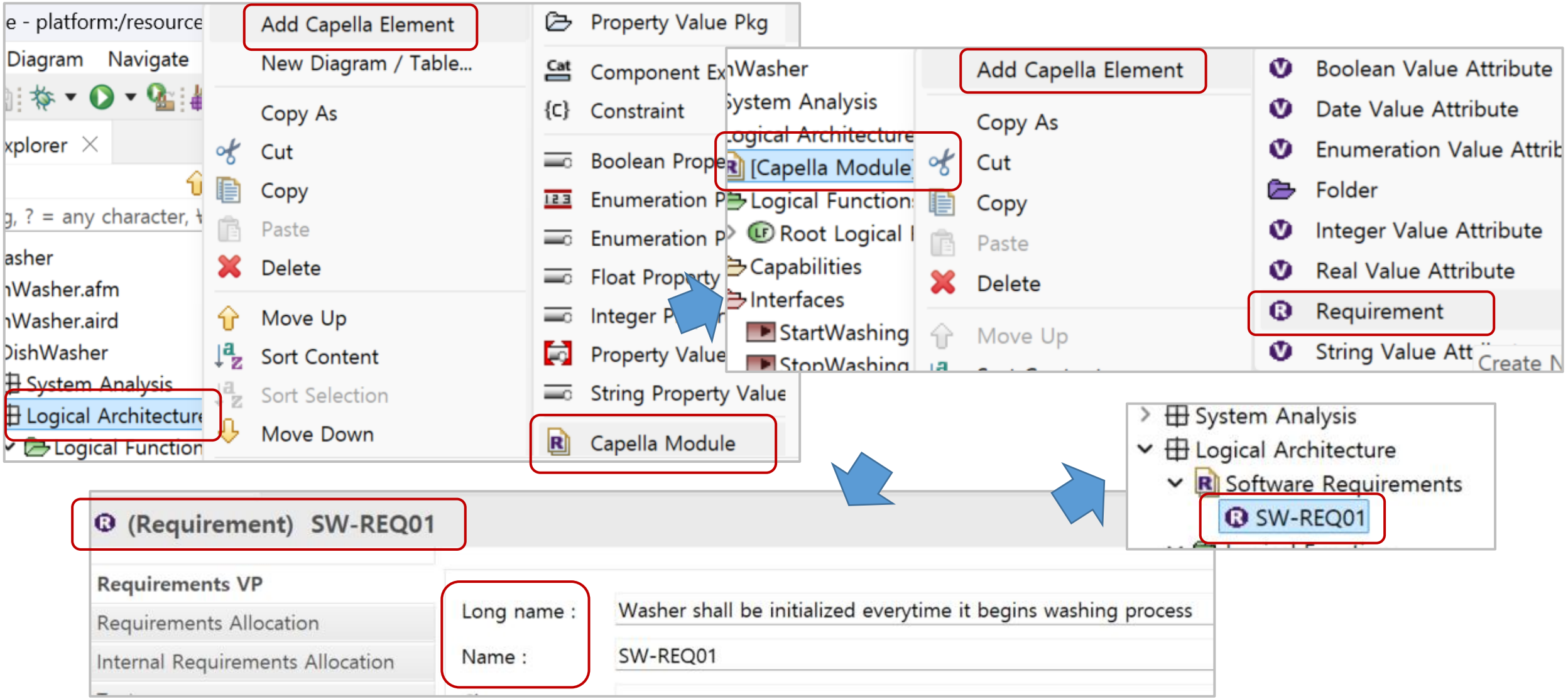
- Go to Github eclipse-Capella site to download the Dropins for requirement viewpoint (R.V)
- After installing R.V, you can add requirements and setup links to and from elements of model.
- Go to Github python4Capella site to install the Python API script project to manipulate data to and from Capella model
- After installing the Python4Capella project in Eclipse, you can modify the sample Python scripts to get or set the information you want from the Capella model

Get ready to add requirements

- Activate the Capella Requirements V.P



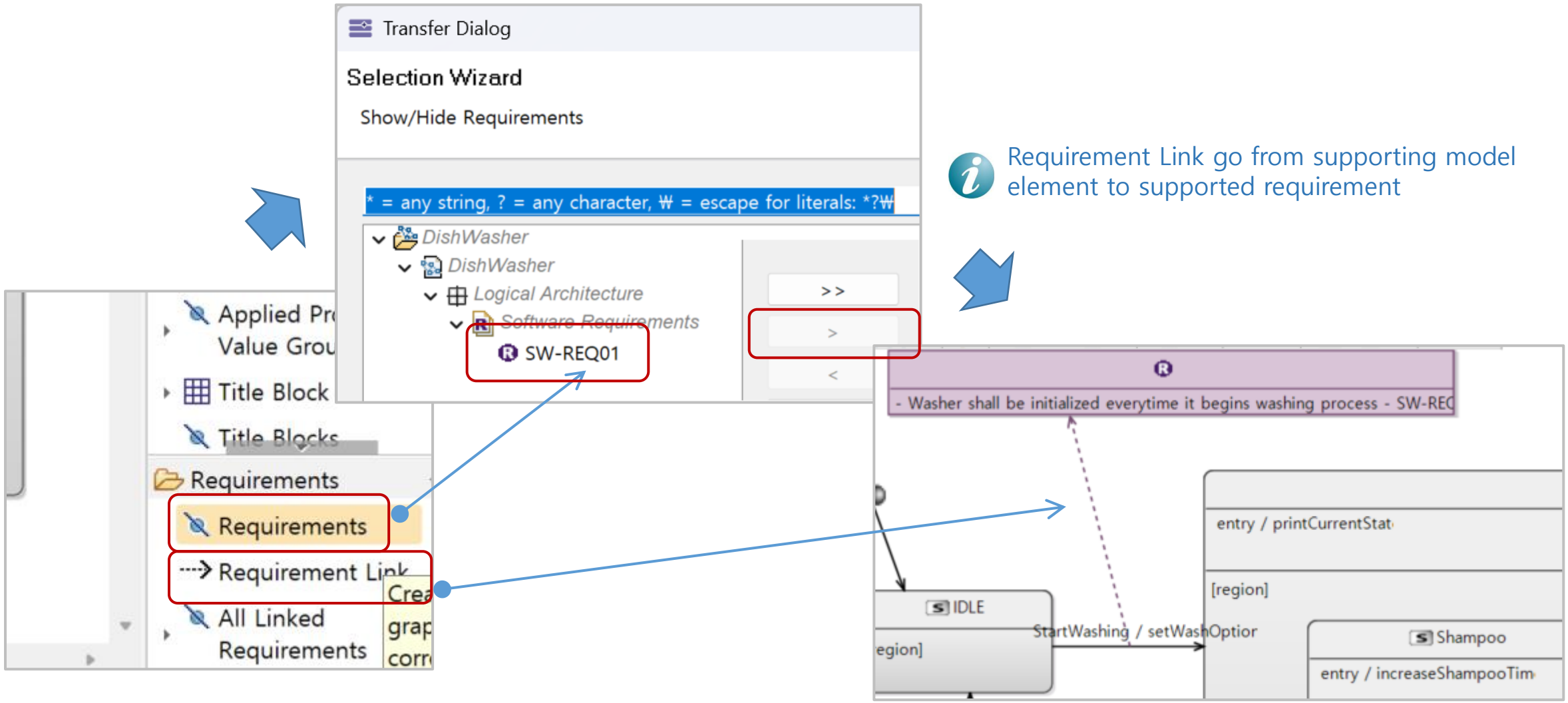
Add your 1st requirement of SW module



What is requirements in SW engineering

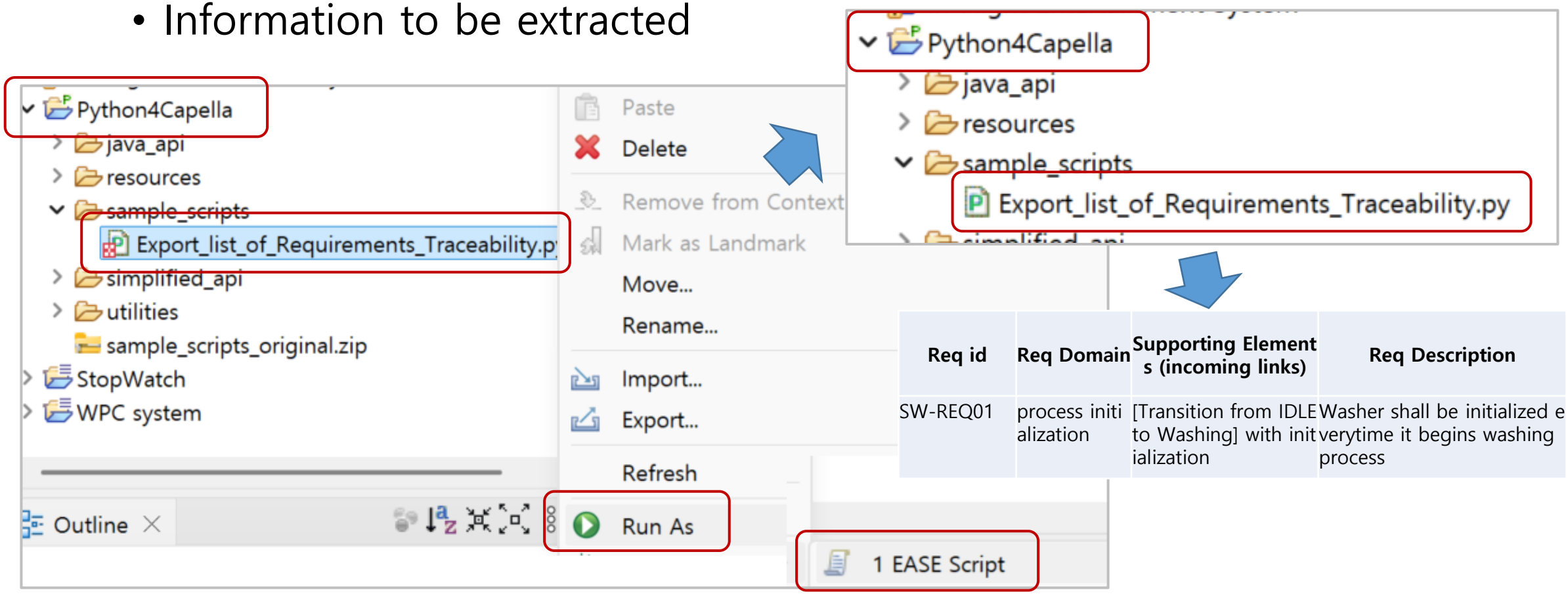
- The ***E***asy ***A***pproach to ***R***equirements ***S***yntax (***EARS***)
  - Official web site for detail information
  - Use "complex type" requirements for normal path scenarios
  - Use "unwanted behavior" requirements for abnormal path scenarios
  - All software requirements discovered during design activities should be linked to model elements such as triggers, parameters, functions, transitions, and states
  - Ensure that model elements are named so they can be displayed in the traceability matrix

Setup trace link btw req and models



Extract traceability matrix using Python

- Adapt the sample python code before running “EASE script”
  - Project path
  - Information to be extracted



Req id	Req Domain	Supporting Elements (incoming links)	Req Description
SW-REQ01	process initialization	[Transition from IDLE to Washing] with initialization	Washer shall be initialized everytime it begins washing process

End of 1st day training

2nd Day training

StopWatch system
Publish and share your model via Web

What is sequence diagram in SysML

- Visual Paradigm page to learn sequence diagram
  - For brief introduction >
    - ✓ We can use sequence diagram to describe the "***Use case scenario***"
    - ✓ In Capella, we call the "Use case scenario" as "***Capability Realization***"
    - ✓ In Capella, we can use ***3 types*** of sequence diagram
      - **Interface scenario :** most commonly used for describing interactions btw modules
      - **Exchange scenario :** same purpose of "Interface scenario" but it is not a standard SysML
      - **Function scenario :** used before specific modules are not yet defined (in system analysis phase)
  - Quick tips for drawing in Capella >
    - ✓ If you need a LOOP, draw ordinary sequence first and add it at last steps
    - ✓ If your scenario is too complex and large, break it down into small enough chunks to use as reference diagrams later.
    - ✓ After your scenario is done, you can find the exchange items automatically generated in Interface package. You can use these while you're designing state machines

System Requirements

■ Functional requirements for a simple Stop Watch system

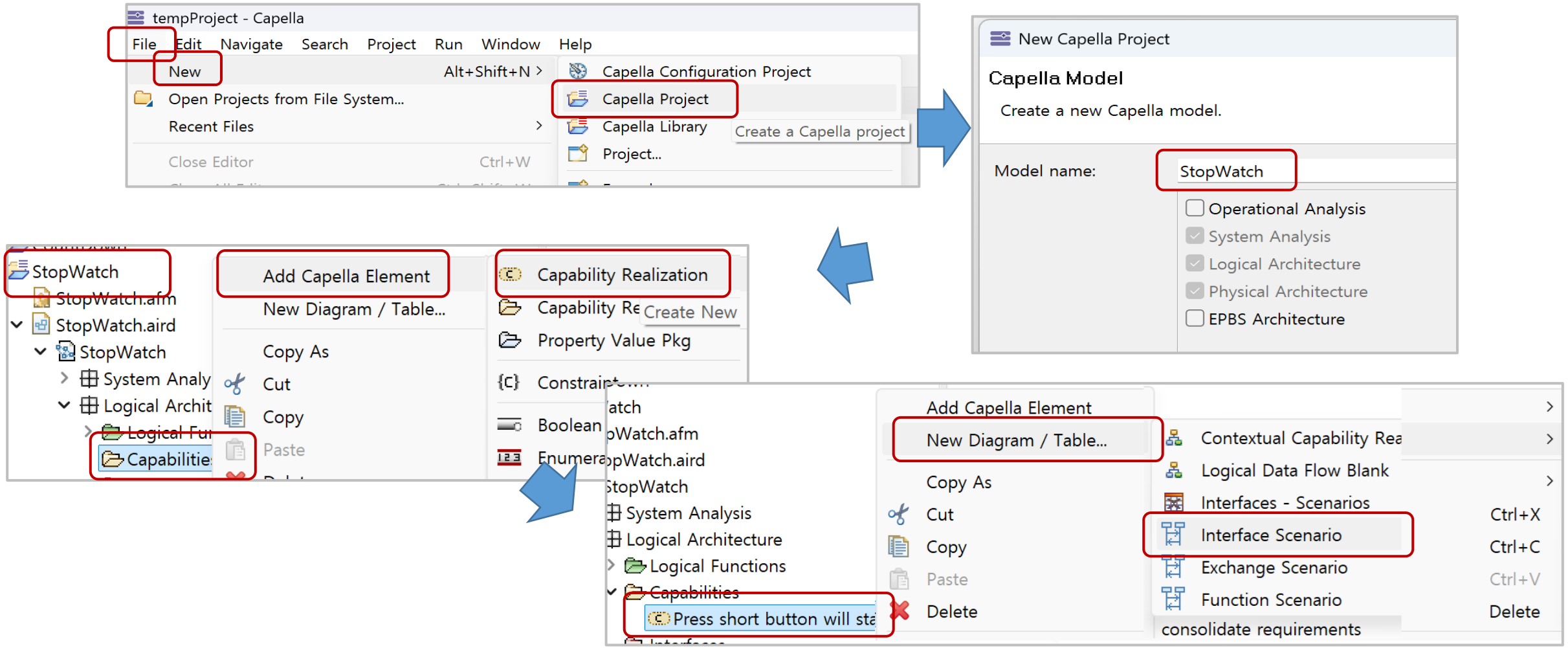
[1] The stopwatch system consists of one button for user operation, one display module (needle) and one timer that controls the timing within the system

[2] When the timer is stopped, a short press for less than 2 seconds will start the timer. When the timer is running, a short press will stop the timer

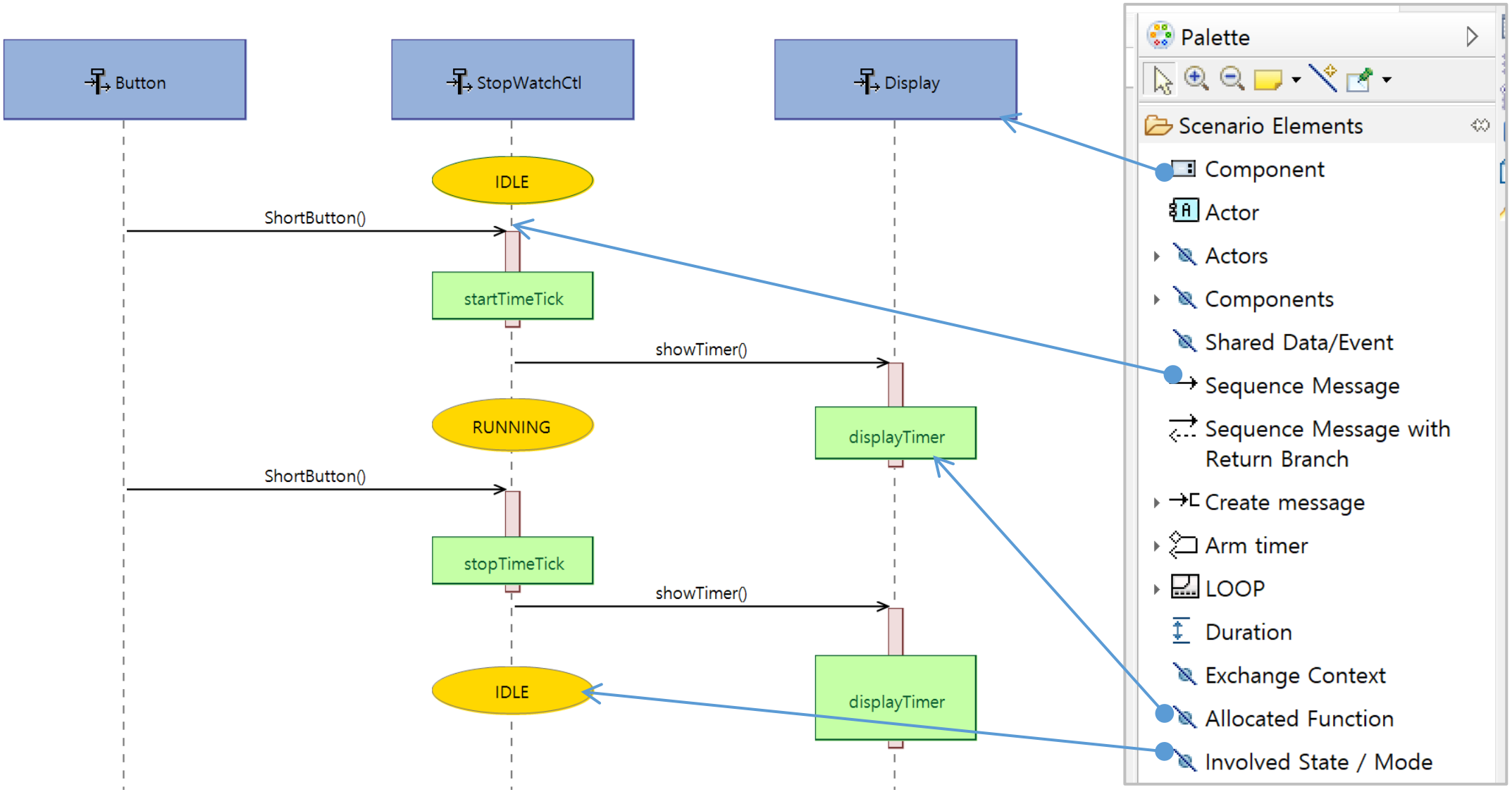
[3] In any state, a long press for more than 2 seconds will reset the timer to zero.

Let’s make a principal use cases

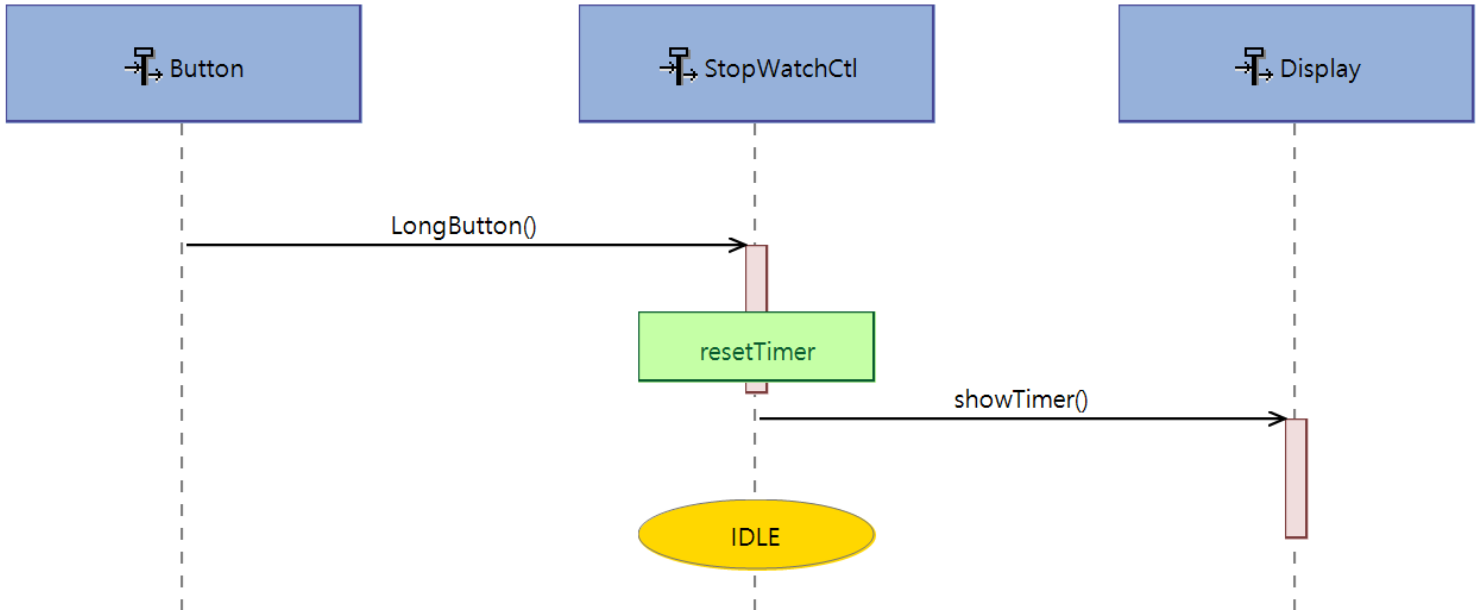
- Based on the system requirements, we can make some principal use cases



Information displayed in a seq. diagram



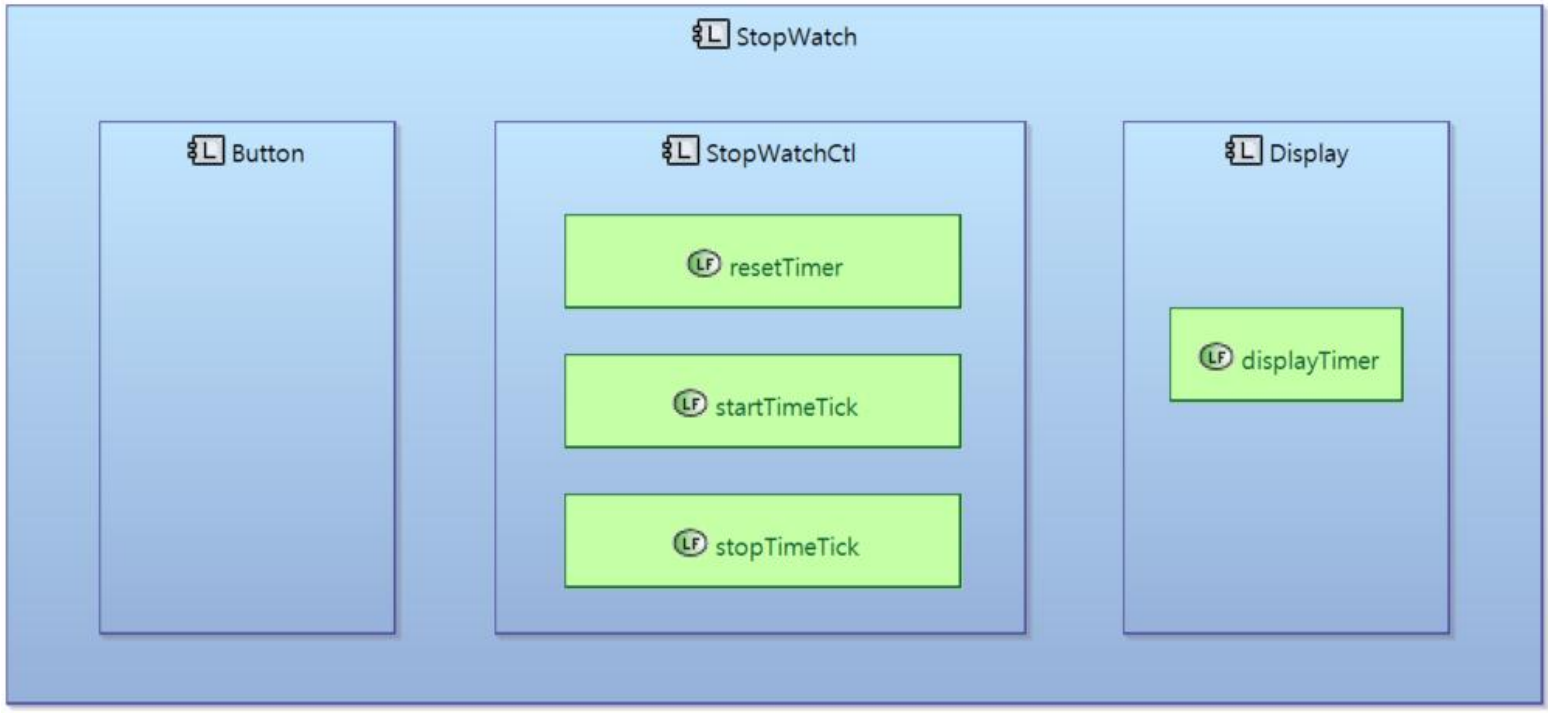
Seq. Diagram will be updated with more information



- After you draw a sequence diagram :
  - Check what are automatically created in your model
  - Check how to add functions in the lifeline of a module later
  - Check how to add states in the lifeline of a module later

Add functions of modules

- Define structural architecture first
  - Check 2 different ways to add functions in a module



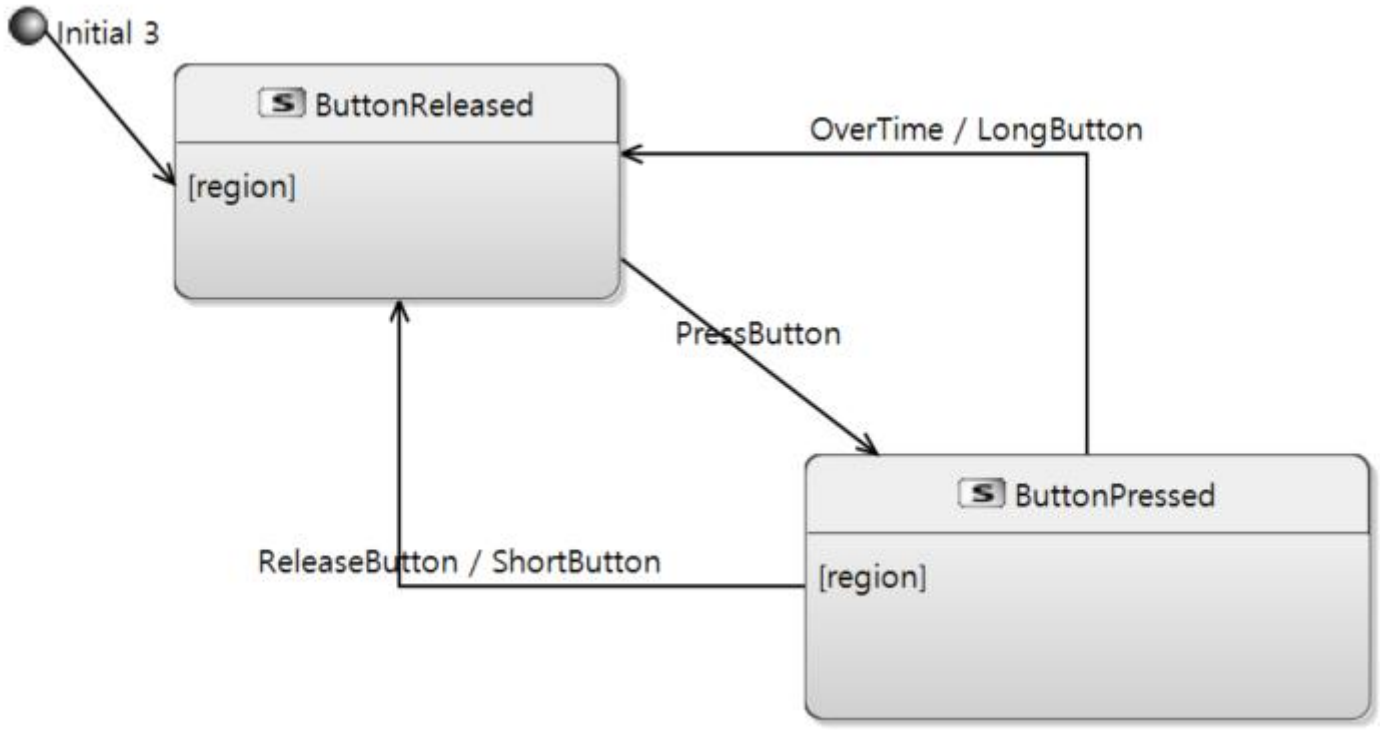
Exchange item and interfaces

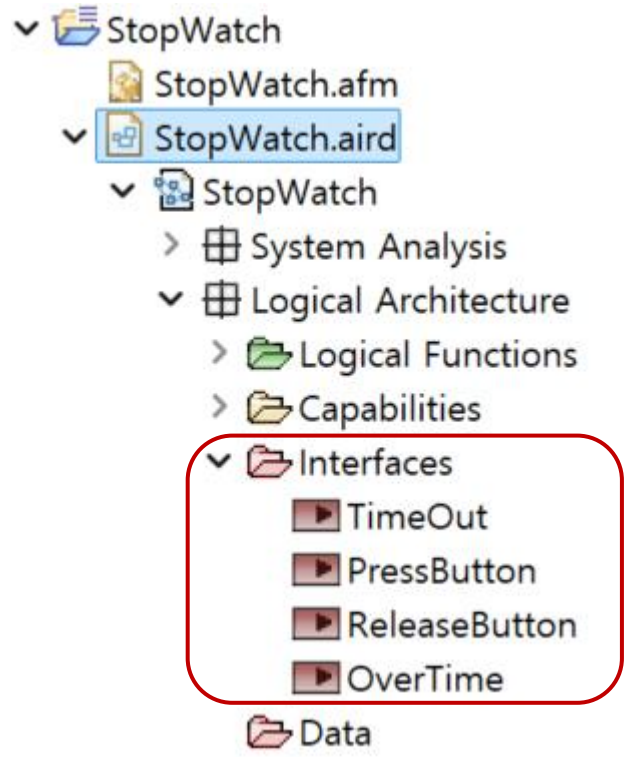
- Exchange item can be “event”, “operation” or “data flow”
  - At initial phase of design, just set it as “UNSET” and modify later
  - Exchange item(E.I) can have multiple “Exchange Item Elements(E.I.E)” that is a group of information to be delivered by this E.I.
    - Leave this E.I.E empty and fill it as you design information data structure

- Interface declares contracts btw modules
  - In Capella, an interface is a group of E.I.s assigned to a module for a specific actions of that module
  - In Capella, interfaces with external actors and interfaces btw internal modules are defined in separate packages
  - In Capella, 2 different ways of adding E.I. to an interface
    - Using “Interface Scenario”
    - Using “Interface Diagram Blank”

Add states of modules (1/2)

■ For **Button** component (or module)

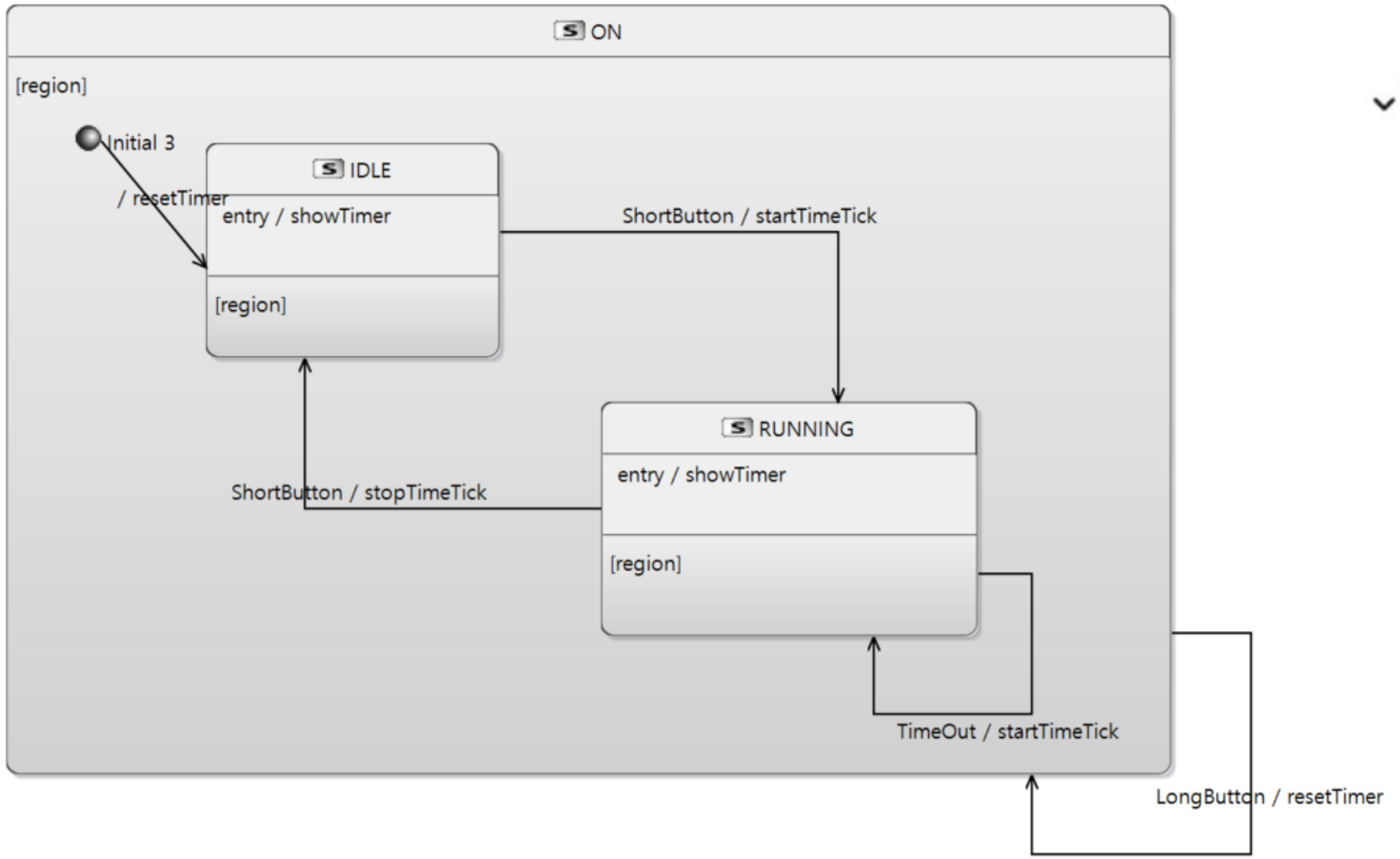


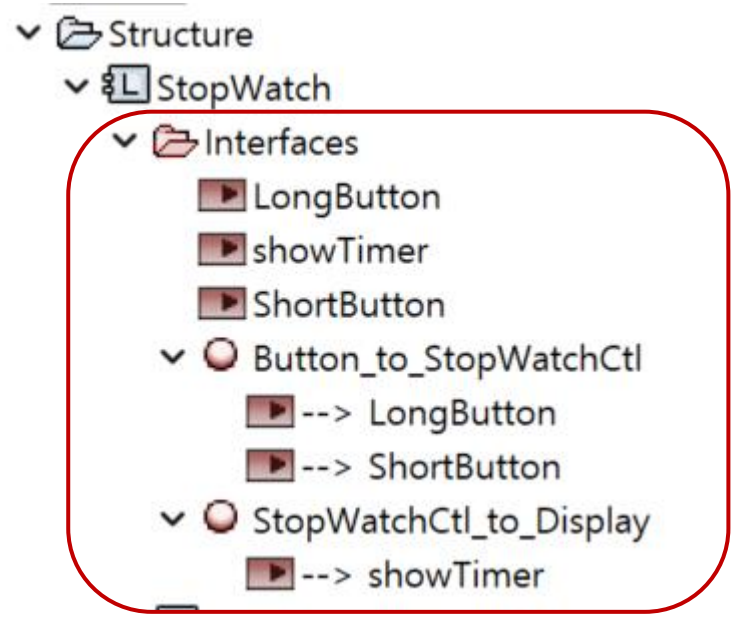


E.I with external actors will be created in a different interface package

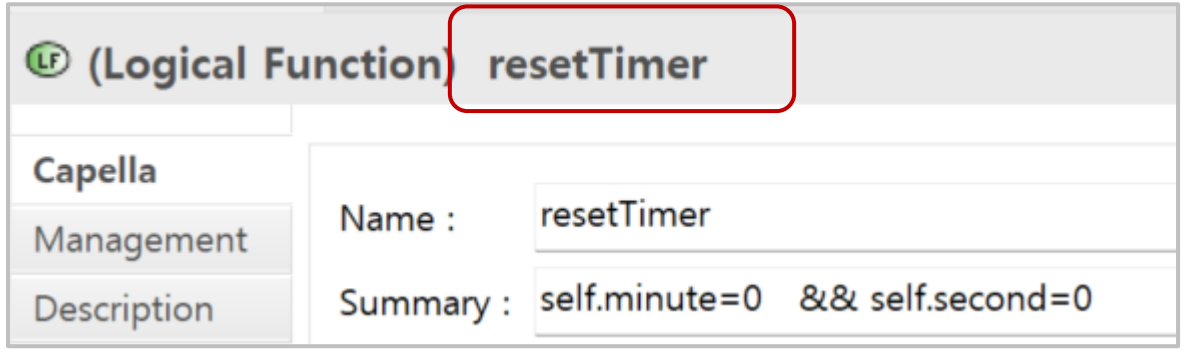
Add states of modules (2/2)

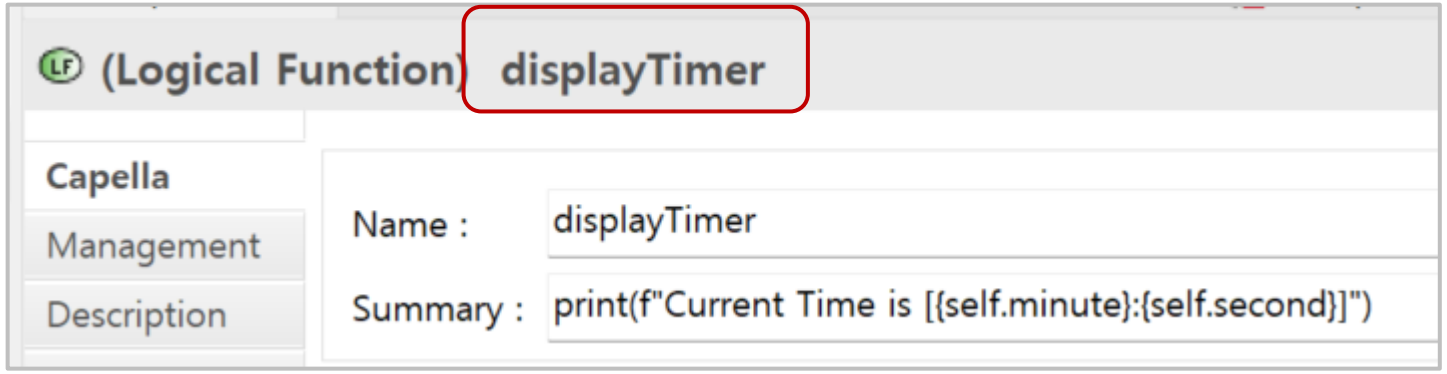
■ For **StopWatchCtl** component (or module)





Fill recommended codes into functions





Generate codes and run the model

■ Open your workspace in AntiGravity and prompt like this >

Prompt to AntiGravity >
The "***Stopwatch***" project contains 3 logical components: "***Button***" "***Display***" and "***StopWatchCtl***" Of these 3 components, "***Button***" and "***StopWatchCtl***" each have their own state machines.
Please write the Python code for these components and their state machines.
You can refer to the recommended codes provided in the summary section for each component and function.
In this model, change the ***OverTime*** event to a real 2 second timeout and the ***TimeOut*** event to a real 1 second timeout.

End of Exercise 3

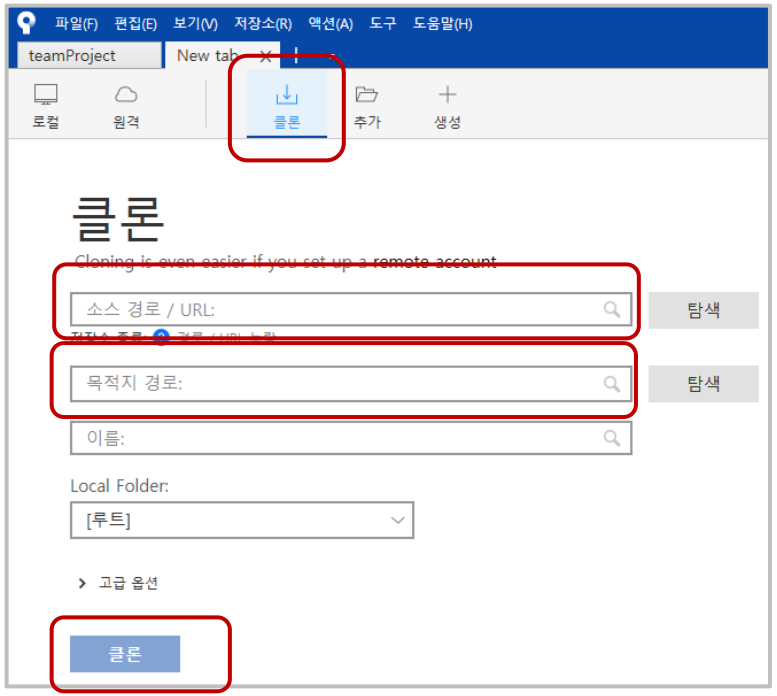
Work as a team using SourceTree

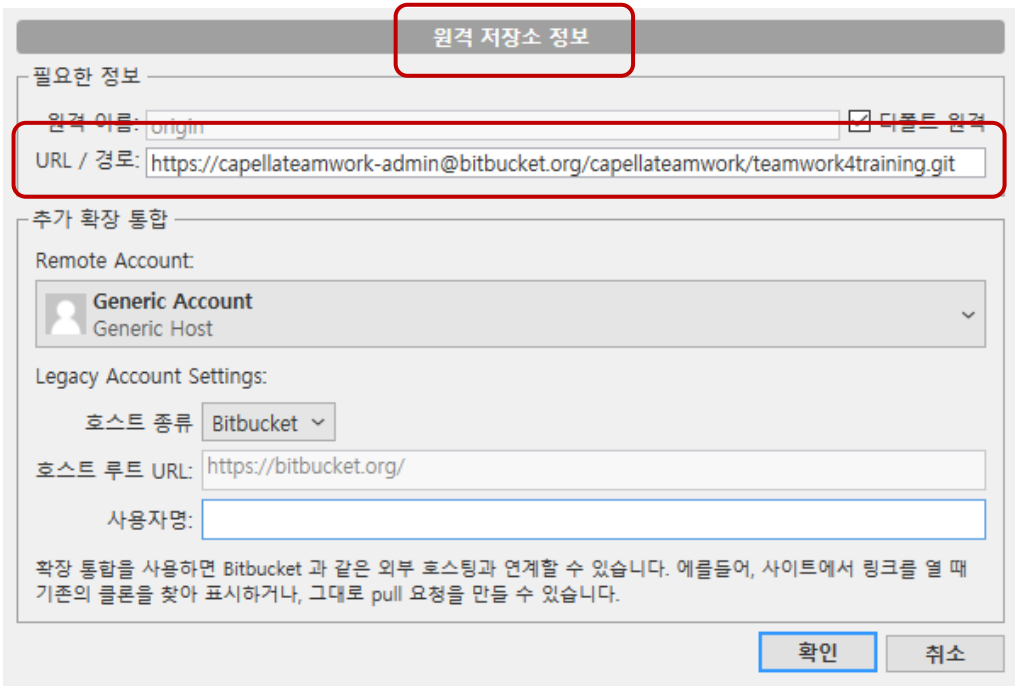
Review and edit together for a real digital twin

Clone git repository using Sourcetree

[1] Install Sourcetree

- Download Sourcetree installer from : https://www.sourcetreeapp.com/
- By skipping all the setup, you don't install the server, but only the Sourcetree terminal
- In your Sourcetree terminal, clone the remote repository of ours below :
  - ✓ Repository Path : **https://capellateamwork-admin@bitbucket.org/capellateamwork/teamwork4training.git**





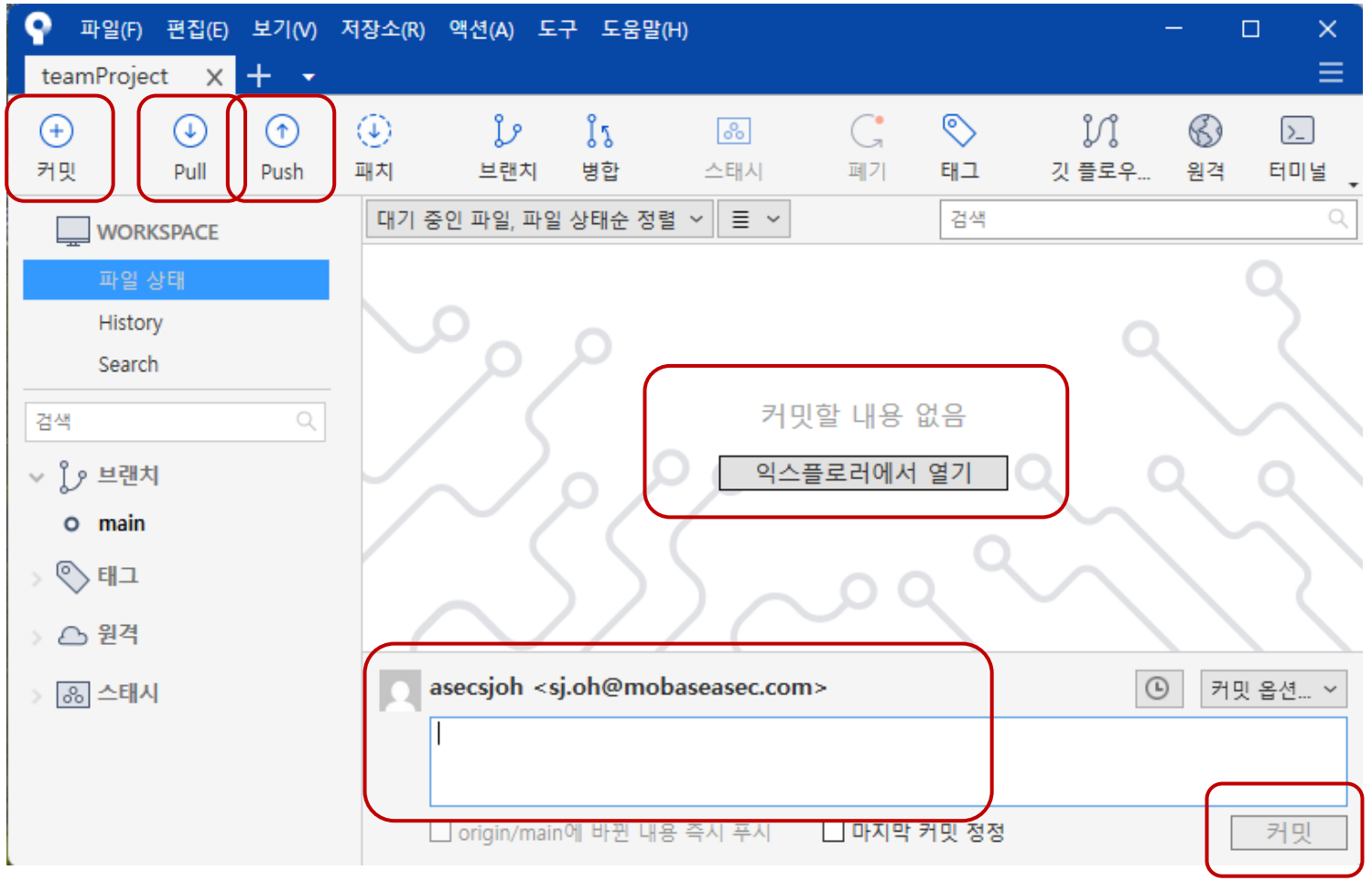
Share your model within a team

■ Use Commit, Pull and Push to share and update your model

[1] When modifying a model in a local folder registered in Sourcetree, you can commit the changes with comments

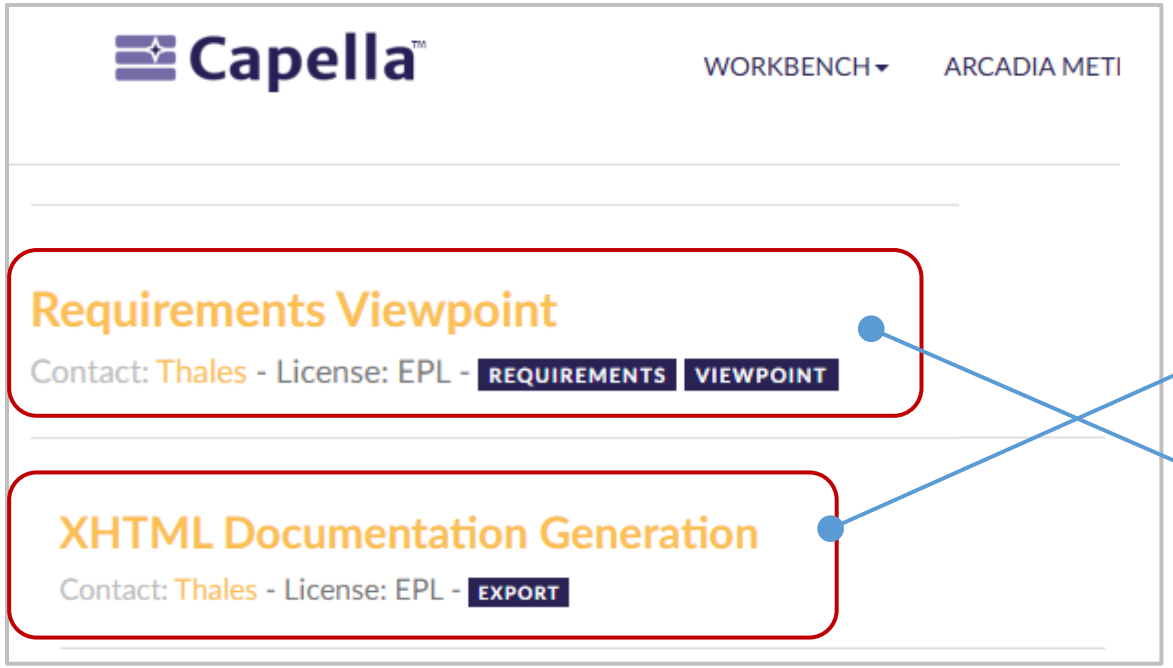
[2] Once the commit is complete, execute "Push" to share the model with the remote repository

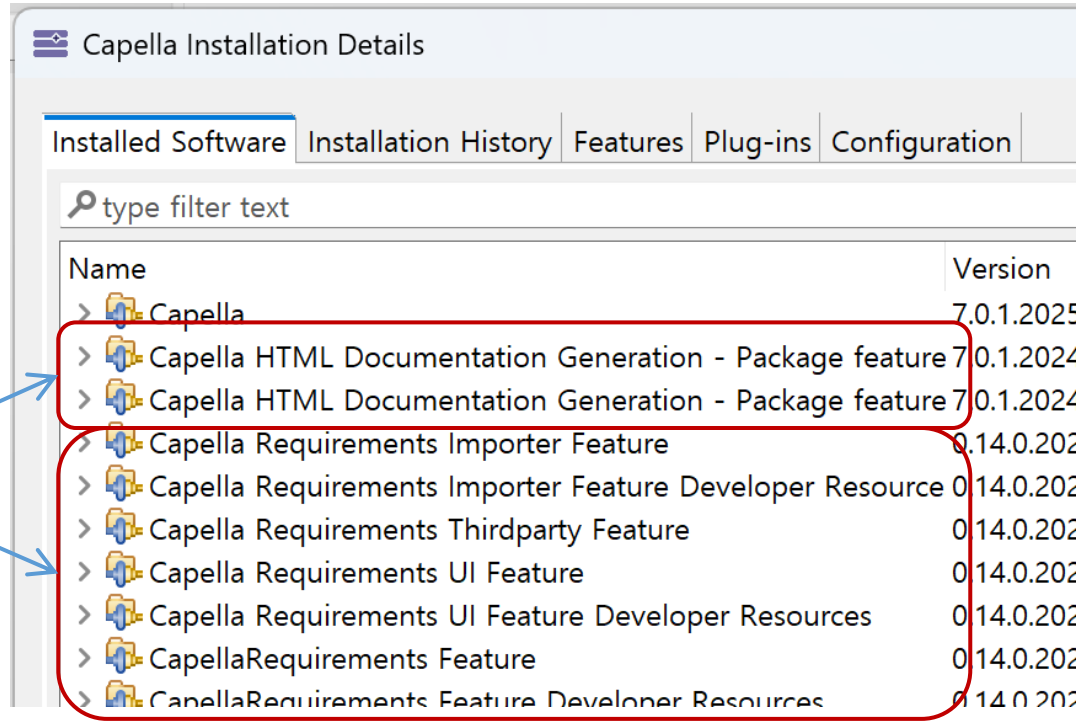
[3] If another user has updated the remote repository, execute "Pull" to retrieve the changes to your local folder.



Install XHTML Document generator to Capella

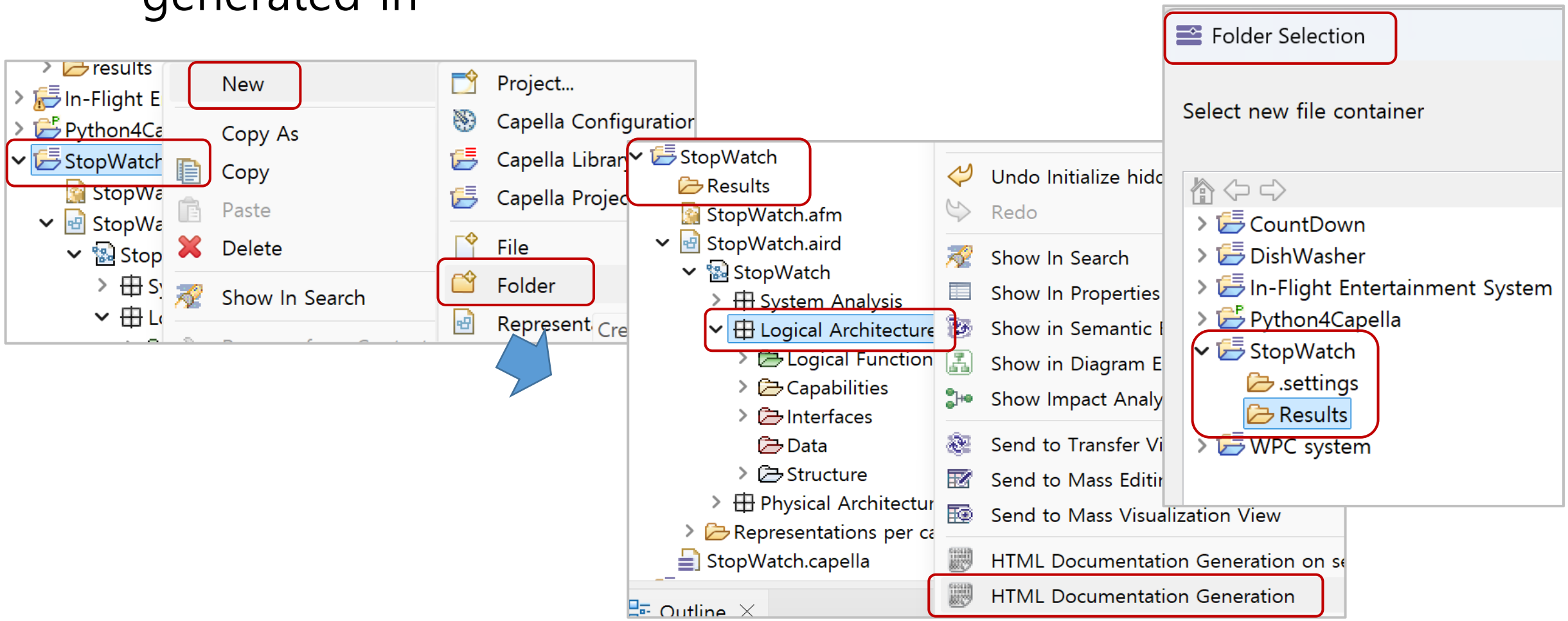
- Once you've finished reviewing your team model, it's time to publish and share with anyone who has the access link
- Go to the download site for Capella add-ons





Publish the functional architecture via Web

- Make a new folder named “Results” for the html to be generated in



"index.html" generated from Capella

파일 C:/MBSE/capella-701/capella/workspace/StopWatch/Results/output/StopWatch/index.html

암호 입력

StopWatch

Capella

Search Index | Back to Index

- StopWatch
  - StopWatch
    - Logical Architecture
      - Logical Functions
        - startTimeTick
        - stopTimeTick
        - displayTimer
      - Capabilities
        - Press Long button will reset stopwatch
        - Press short button will start and stop the stopwatch
      - Interfaces
        - TimeOut
        - PressButton
        - ReleaseButton
        - OverTime
      - Data
      - Structure

Model navigator in left frame

Default Region

Region

StopWatch > StopWatch > Logical Architecture > Structure > StopWatch > StopWatchCtl > StateMachine 1 > Default Region

No description.

Diagram navigator in right frame

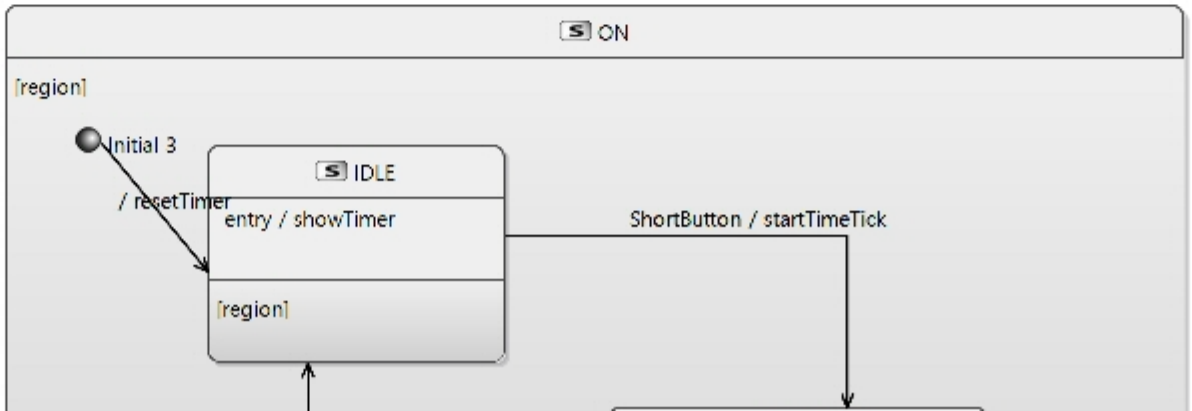
Modes and States

ON

Owned diagrams

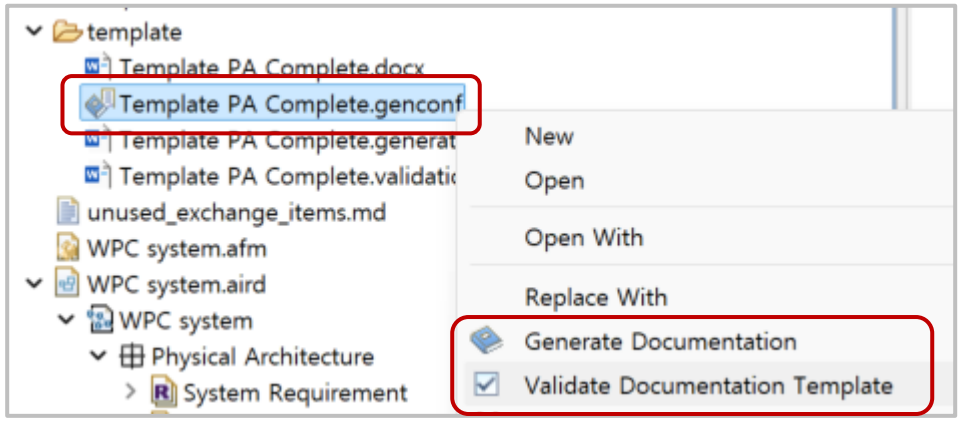
By installing the Apache Web Server and configuring it to display this page, you can maintain a ***functional architecture dashboard*** for ongoing review by stakeholders.

MSM Default Region of StopWatchCtl



Report your model via MS word

- Download and install M2Doc solution of Obeo
  - Download and install guide
- Report all of your model elements via MS word like this
  - Ch 1> System and Software requirement description
  - Ch 2> Software components and their functions
  - Ch 3> Interfaces between Software components
  - Ch 4> State machines of all software components
  - Ch 5> Use case scenario based on the software components in this product
  - Appendix > Traceability between requirements and SW component and interfaces



- ✓ From generation configuration, run *"Validate"* first
- ✓ You will get Validation report if you have errors in your template (AQL language)
- ✓ Once your template passes validation, click *"Generate"* to get your report automatically from your model

End of 2nd day training

3rd Day training

Wireless Power Charging system
Let’s play as a team for robust design

Goal of 3rd day training

- Understand the system requirement of WPC product
- Play as a team to

  [1] Build ideal structure of application software (apply design patterns)

  [2] Derive user scenario as many as possible from the system requirements

  ✓ Derive input triggers (e.g. CAN DB) and module functions

  [3] Define the ideal state machines for each software component

  [4] Generate python codes for state machine simulation (Google AntiGravity)

  [5] Convert the codes into C and compile with gcc to run on Windows terminal

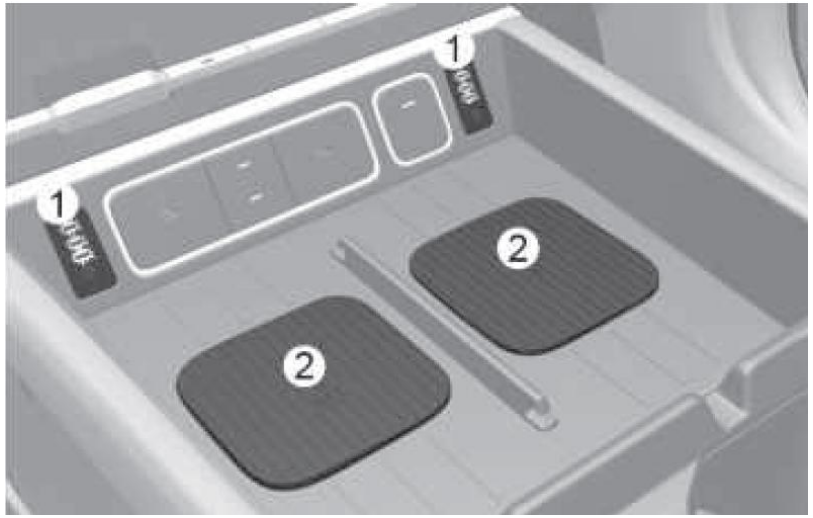
System Requirements

■ Example of functional requirements for WPC

[1] After turning off the engine and leaving your smartphone on the charging pad, if you open the driver's or passenger's door, a warning message will appear on the cluster and a warning tone (in vehicles with voice guidance) will sound, indicating that your phone is on the wireless charger

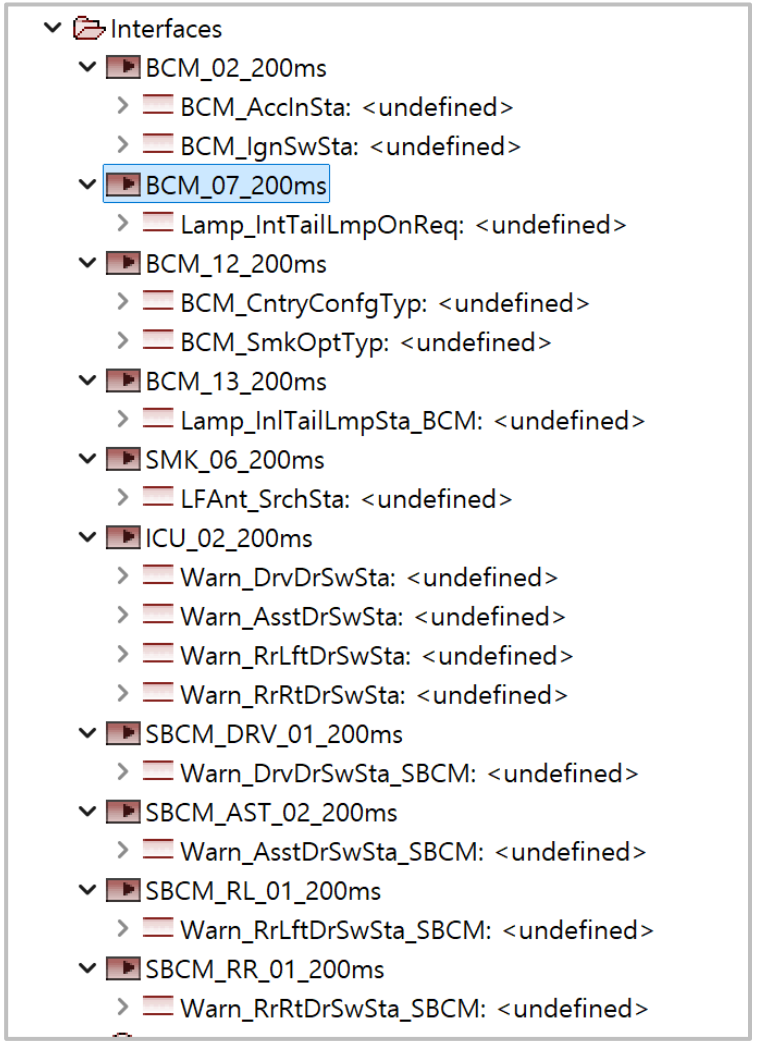
[2] If the temperature inside the wireless charging system exceeds a certain level, charging will be interrupted to protect your smartphone. Charging will resume when the temperature drops below a certain level

[3] Place your smartphone in the center of the charging pad. Charging may not occur at the edge, and charging at the edge may generate significant heat.



Import CAN DB to your Capella model

- Modify "***Import_physical_components_from_xlsx.py***" to import sample CAN messages into your Interface package

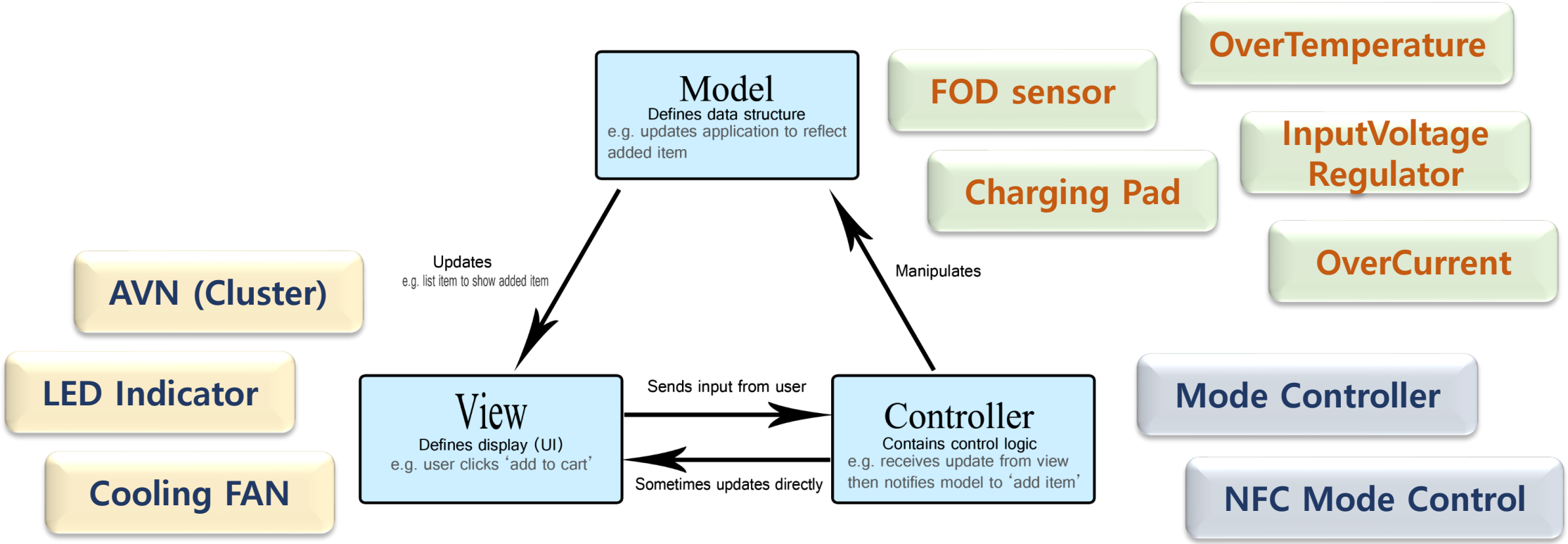


- Group exercise :
  - Import CAN signals into the Exchange Item Elements by ***modifying the python code*** as well
  - Import System requirement delivered from your product team as well

Build your own structure of SW

■ What is MVC design pattern? Reference site

- separates an application into three interconnected components to isolate business logic from the user interface, improving modularity, maintainability, and scalability (e.g. StopWatch system)



[Step1] physical architecture of WPC

[Step2] software components @ MCU

[Step3] Define and import SW functions

[Step4] Define interfaces btw SW components

[Step5] Define SW component’s state machine

[Step6] Derive product use cases

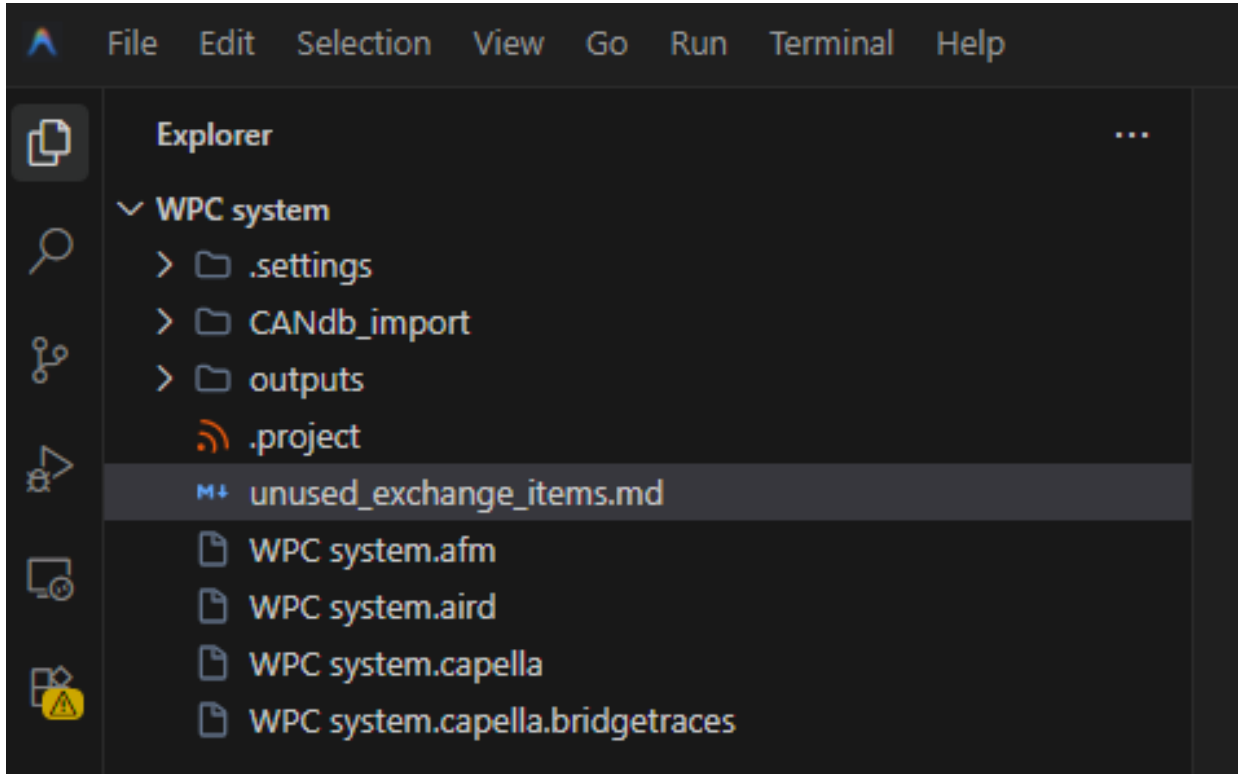
Consult with your instructor

- Detail level of training materials and python scripts for the exercise 4 will be provided from your instructor

Generate codes and run the model

■ Open your workspace in AntiGravity and prompt like this >

Prompt to AntiGravity >
In this Capella model, there are 27 requirements in the "Software Requirements" Capella module. Can you make a test scenario for the requirement "SwFR-09002"?
Please use the names of exchange items and physical functions in WPC system Capella model to write detail scenario.



Let's see and evaluate the results of Google AntiGravity !!

Validate the model

- Use python as a default and convert the codes into “C”
  - Install “gcc” compiler in your Windows PC first.
  - Request the “AntiGravity” to convert the python codes into “C” so that we can compile via “gcc” and run on Windows terminal
  - Some features dependent OS such as “timer” or “task” will be migrated but most of the state machine logics will remain unchanged.

End of Exercise 4

End of 3rd day training

Congratulations!

You’re now a master of Capella functional architecture