Tutorial: Programming in UML Using Activities

Asma Smaoui, Ansgar Radermacher, Jérémie Tatibouët, Shuai Li,
Patrick Tessier and François Terrier
{first-name}.{last-name}@cea.fr

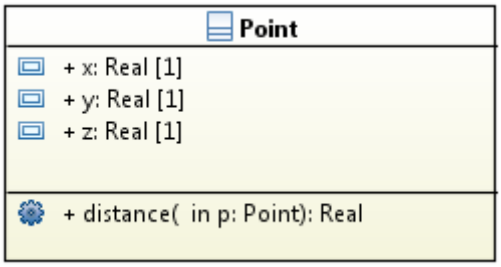
Lab 1 – The basics of UML activities

Objectives:

- Use basic actions provided by activities
- Use activity and actions input and output values
- Use nodes to coordinate the execution flow

Exercise:

1. Import the project "**Lab1-QuickStart**"
2. Open the class diagram that is at the root of the model



3. Provide the implementation of the "**distance**" operation as an activity. This operation enables the computation of the distance between two points (i.e., the current point and the one given as parameter) placed in a 3D plan and returns the distance between. The computation of the distance is based on coordinates x, y and z.

   A. Associate the activity "**distanceImpl**" with the operation. Explain how you achieved this association.

   B. Make sure the activity and the operation have their parameters aligned (i.e., parameters of the same type). Add the required activity parameters. Explain why we need the activity parameters nodes.

   C. Complete the activity implementing the operation "distance".

Advices:

- **Syntactic elements required for the exercise**
  - Control flow
  - Object flow
  - Activity parameter node

- Action input and output pins
- Fork node
- Read structural feature action
- Call behavior action
- Initial node

- **Required functions**
  - You will need to use function like **"square"** and **"squareroot"**. Such functions are available in the library **"MathsLib"** which is already imported by the model.

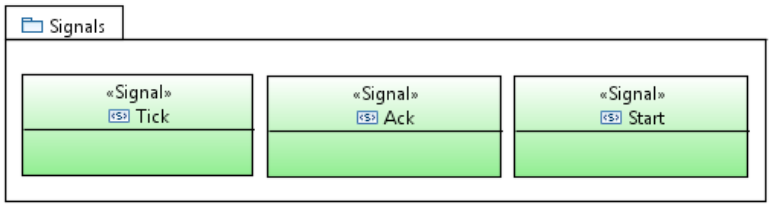
Lab 2 – Activities used in the context of communicating active objects

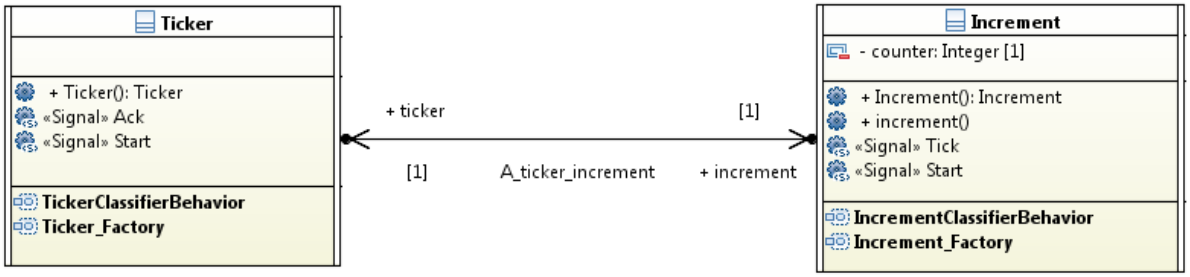
Objectives:

- Use basic actions provided by activities
- Use some of the nodes to coordinate the execution flow
- Synchronize active objects using actions to send and receive asynchronous communications.

Exercise:

1. Import the project **"Lab2-AsynchronousCommunications"**
2. Open the class diagram available at the root of the "System" package





3. There are two active classes **"Ticker"** and **"Increment"**. The **"Ticker"** communicates with **"Increment"** via asynchronous communications. These asynchronous communications are materialized by signals (cf. the package **"Signals"**).

   A. **Role of the "Ticker"**: it stimulates the **"Increment"** with **"Tick"** signal occurrences. Before any another **"Tick"** signal occurrence is sent, the **"Increment"** waits until it receives an **"Ack"** signal.

B. **Role of the "Increment"**: it waits until the reception of a **"Tick"** signal occurrence. When such occurrence is received it increments the value of its **"counter"** attribute. As soon as this sequence of action is realized it sends an **"Ack"** signal occurrence to the **"Ticker"** and waits for another occurrence of **"Tick"**.

4. What you have to do is:

   A. Identify the type of actions that are required to perform asynchronous communications in the context of activities.

   B. Implement the classifier behavior of **"Ticker"** as an activity.

   C. Implement the classifier behavior of **"Increment"** as an activity.

   D. Try to run your model using Moka (model execution framework of Papyrus). Fix it in case it does not execute.

   E. **Bonus**: Instead of an infinite execution make sure that when the counter value reach 10, both execution of **"Tick"** and **"Increment"** properly terminate.

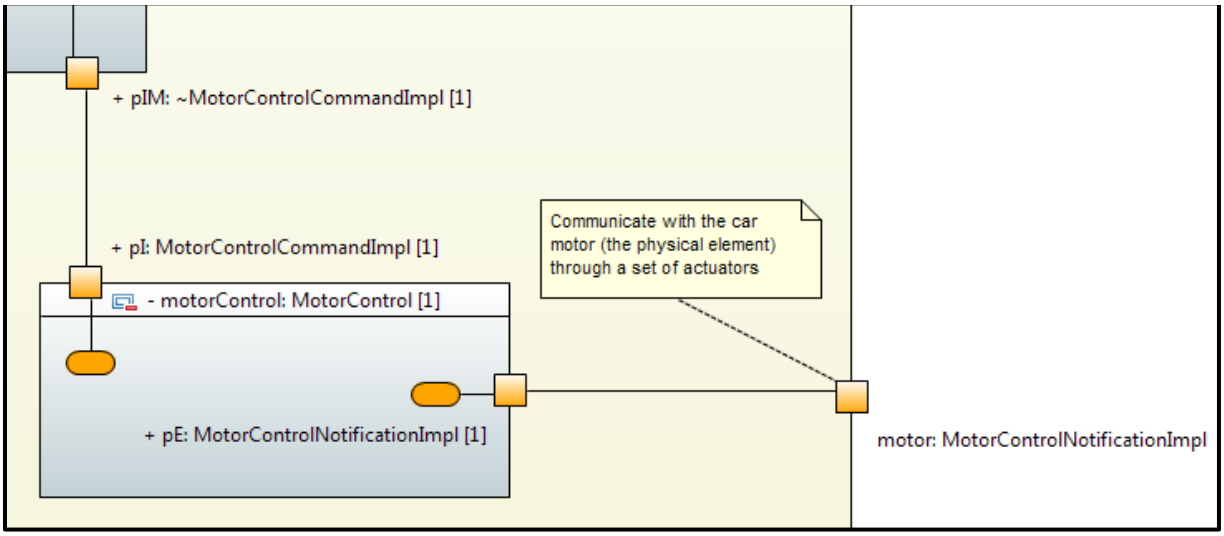
Lab 3 – Apply you knowledges about activities in the context of the elevator

Objectives:

- Reuse what your learnt to locally complete the model of the elevator controller you built

Exercise:

1. Open the last version of your elevator model.



When the **"RequestProcessor"** wants the elevator car to move at a specific floor it notifies the part of the system named **"MotorControl"**. This notification takes the form of an operation call emitted through the port **"pIM"**. When the notification gets computed by **"MotorControl"** it makes the state of this part of the system to move from **"ElevatorCarStopped"** to **"ElevatorCarMoving"**. To transmit the movement order to the real motor (the one that physically exists), the **"MotorControl"** sends a signal from its **"pE"** port.

2. What you have to do is:

   A. Choose in your model the state that represents the situation where the **"RequestProcessor"** interacts with the **"MotorControl"**.

      - Attach an entry behavior to this state.

      - Implement this entry as an activity. This latter handles the emission of the operation call **"move(instruction)"** through the port **"pIM"**

   B. Choose in your model the transition enabling to switch from **"ElevatorCarStopped"** to **"ElevatorCarMoving"**.

      - Attach an effect to this transition. This latter is an activity.

      - This effect must have an activity parameter node whose type is **"MovementRequest"** (cf. data types package).

      - Implements the activity to emit a signal of type **"Move"** to the real motor. This signal ships the destination floor value. The signal is emitted through the port **"pE"**.

Advices: resolution of B requires that you add a new signal in your model. In addition the contract on the port **"pE"** must be updated.