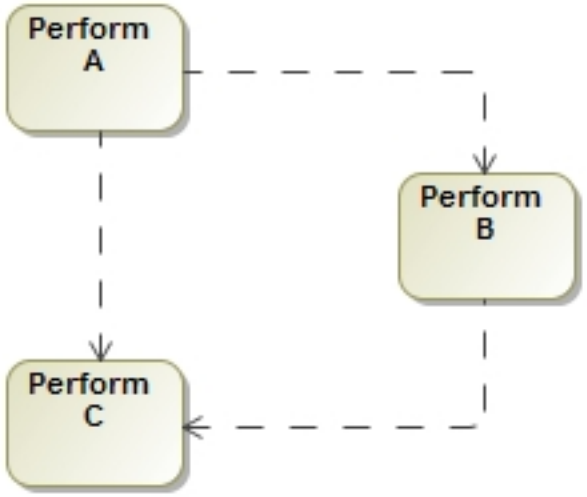
SysML Activity Diagram

Some Basic Semantics

theodore.kahn@engility.com



1. Does this diagram follow all SysML rules, is it semantically correct?
2. If it is correct, is it good SysML, is the modeler's intent clear?
3. What is the modeler's intent?

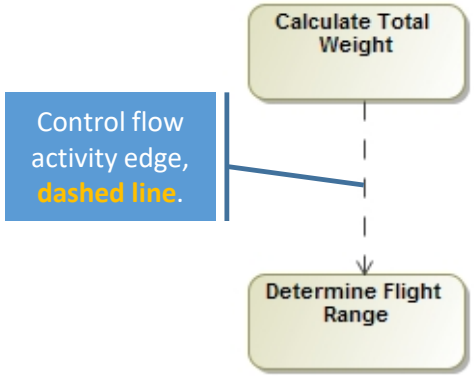
The semantics discussed are not comprehensive; rather, they are meant to cover only some of the most common usages. Readers are encouraged to consult additional documentation to obtain a fuller understanding of the SysML grammar.

Activity Edges

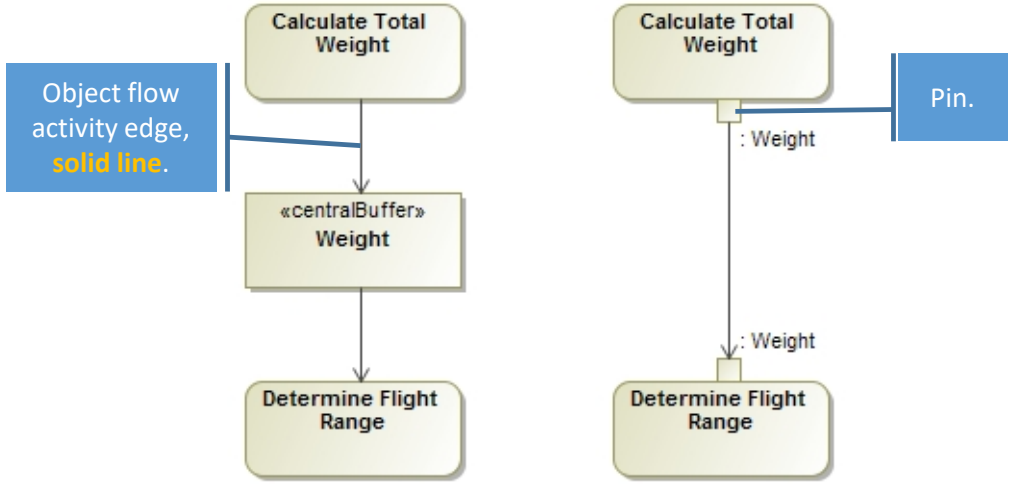
Control and Object Flows

The most basic function of Activity Diagrams is the commutation of information among Actions. This is accomplished by using either or both types of activity edges, as required:

Control Flow



Object Flow



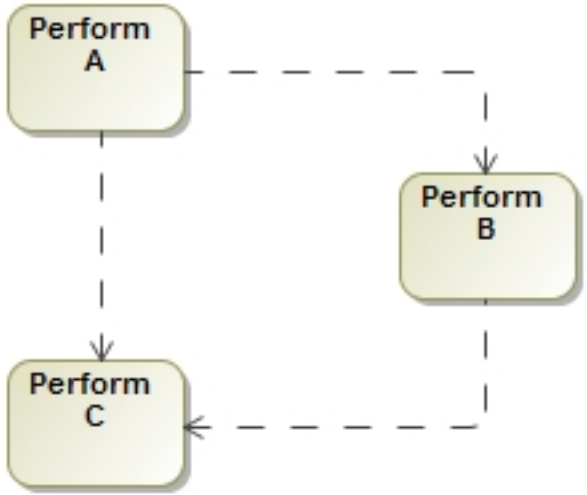
Best Practices

1. Use Object Flows whenever possible as they explicitly specify what is being communicated between Actions.
2. The two object flow constructs on the right are semantically identical. In general, the use of pins is preferred as it is more concise and allows for typecasting, should that be required.

Activity Edge Semantics

1. An Action cannot start until all Control and Object tokens are received at the Action edge.
2. When an Action finishes, it provides all Control and Object tokens to their respective flows.

Diagram Flow



1. When **A** finishes, it provides Control tokens to **B** and **C** at the same time.

2. **B** then starts. When **B** finishes, it provides a Control token to **C**.

3. **C** now starts as both Control tokens are available at the Action edge.

What is the problem with this diagram?

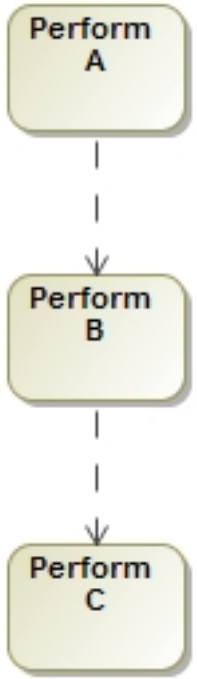
Discussion

The Activity Diagram on the left is grammatically correct.

However, the Control Flow between **A** and **B** is unnecessary as **C** cannot start until **B** finishes.

Perhaps the modeler had a different intent in mind?

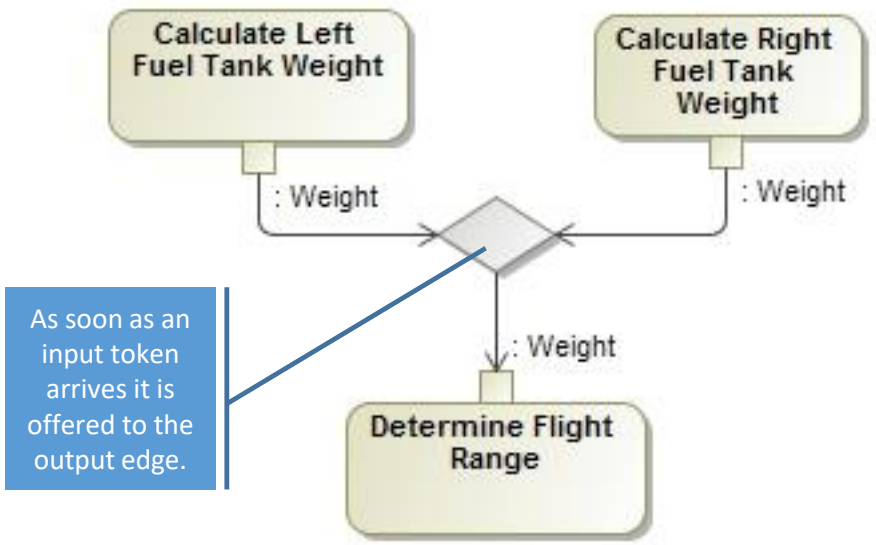
As it is, the left diagram can be simplified as shown in the diagram on the right.



Merge and Decision Nodes

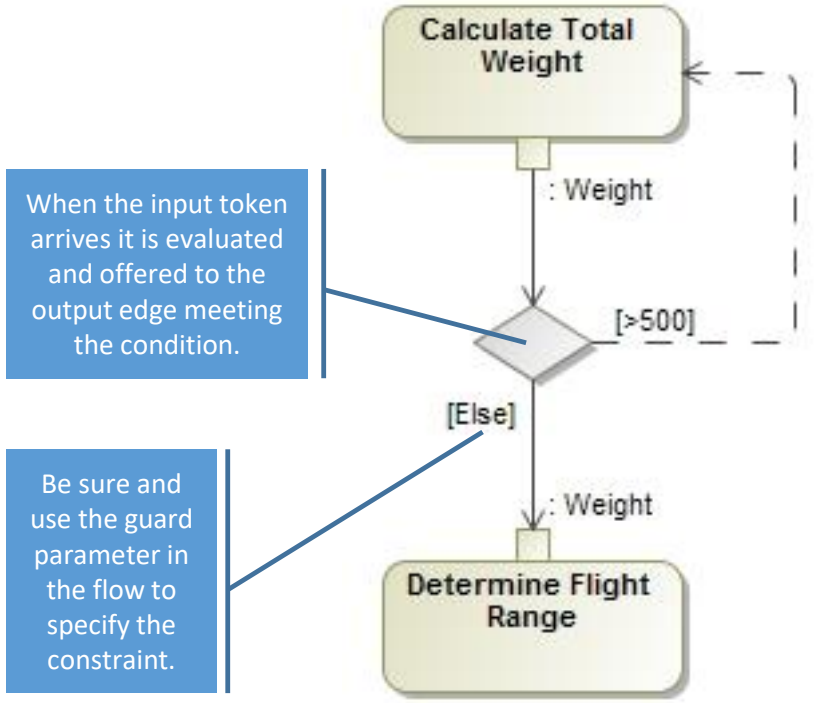
These nodes address **OR** conditions

Merge Node



Multiple inputs, one output.

Decision Node

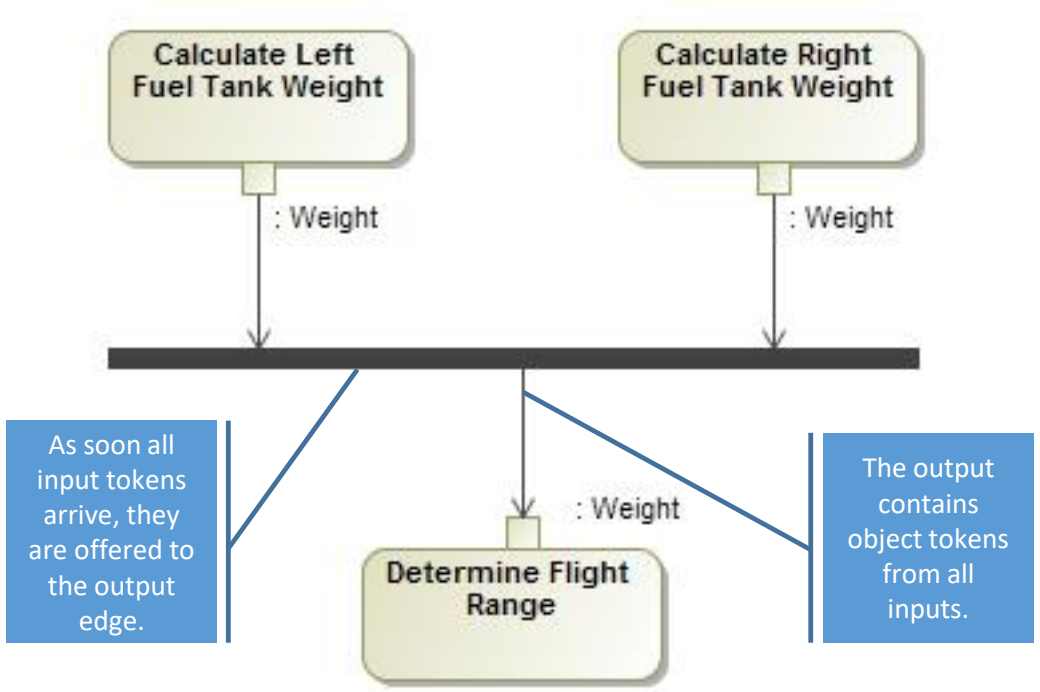


One input, multiple outputs, but only one at a time.

Join and Fork Nodes

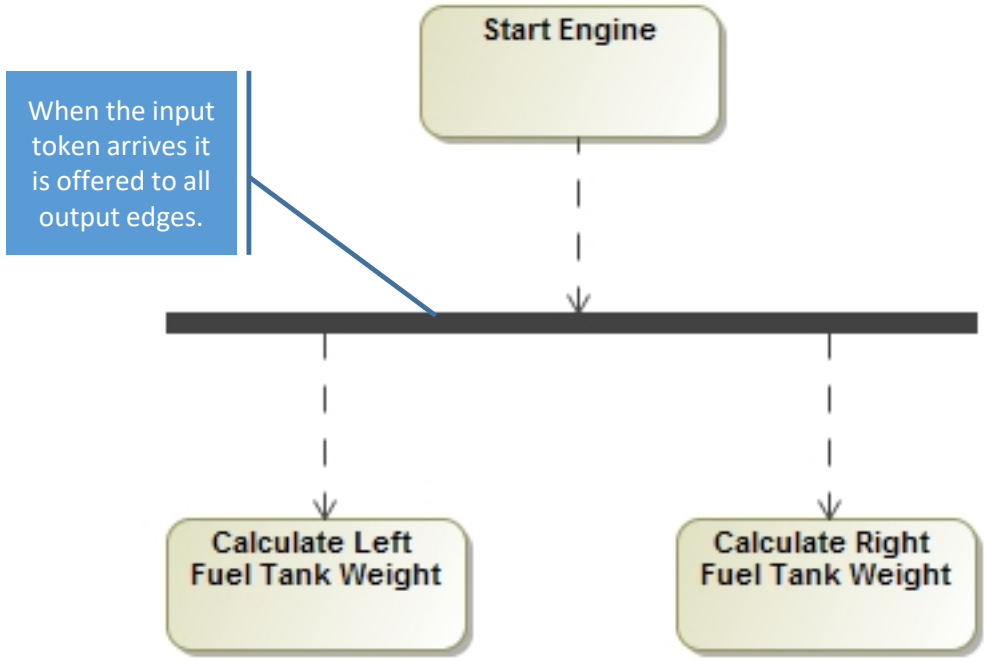
These nodes address **AND** conditions

Join Node



Multiple inputs, one output.

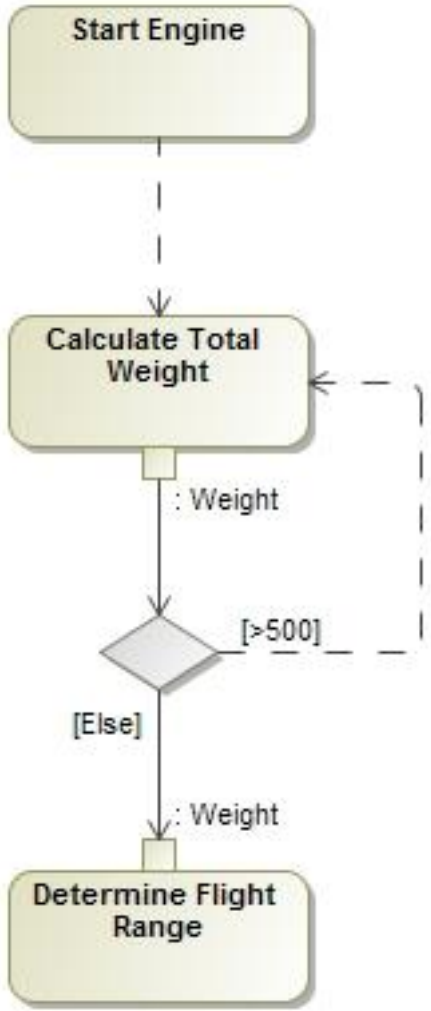
Fork Node



One input, multiple outputs.

Question 1

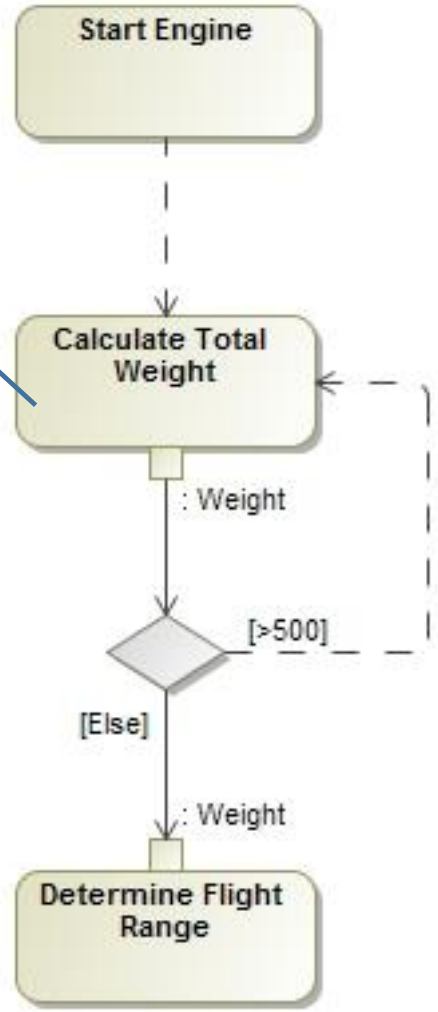
What is the problem with this diagram and what might you do to correct it?



Answer 1

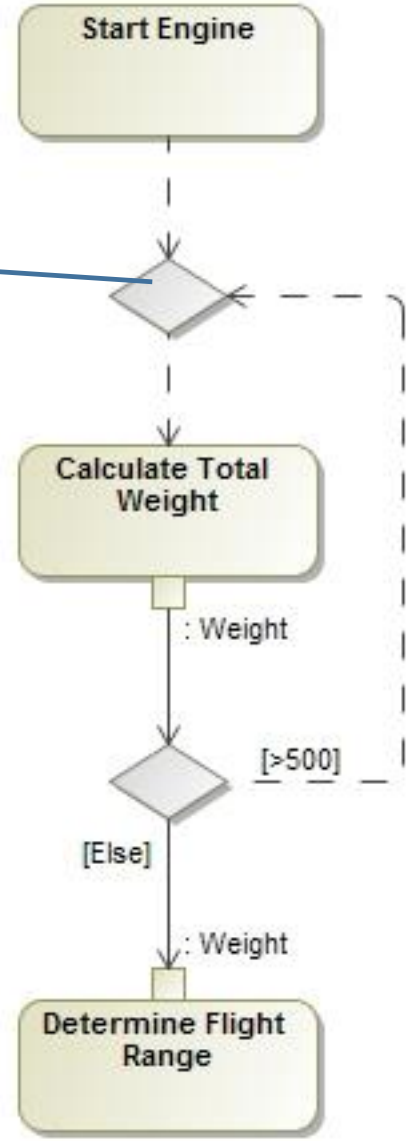
What is the problem with this diagram?

This Action requires two control tokens to execute but the second token can only be generated by the Action's execution. So the Action never starts.



What might you do to correct the problem?

Add a Merge Node. It offers the control token to the output edge whenever it receives any (one) input control token.



Question 2

Discuss the differences between Diagrams **A** and **B**. Is one better than the other? How is *Determine Flight Range* affected?

Discuss the differences between Diagrams **B** and **C**. Is one better than the other? How is *Determine Flight Range* affected?

Diagram A

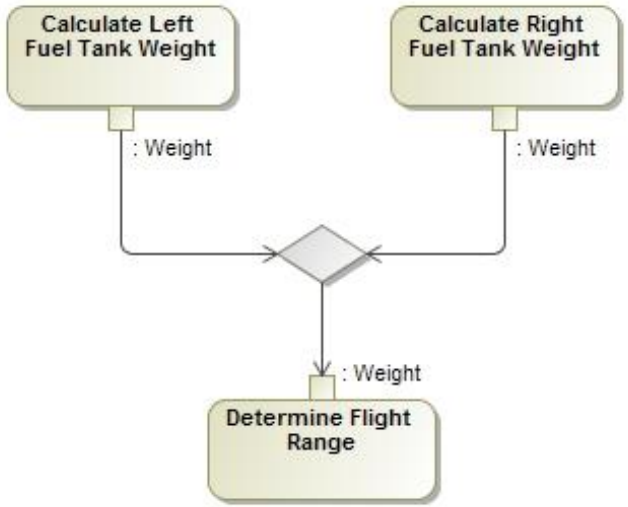


Diagram B

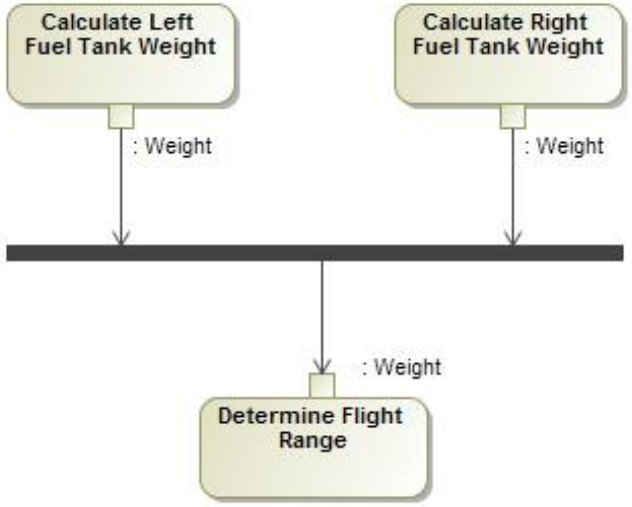
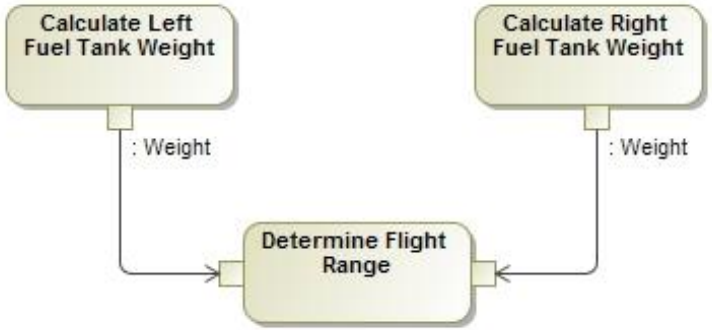


Diagram C



Answer 2

The two diagrams **A** and **B** represent two possible methods for determining flight range. As such, there is no correct diagram. Modelers need to produce diagrams representative of the system as designed. And obviously, the flight range calculation is different for each diagram.

The two diagrams **B** and **C** are semantically identical and the flight range calculations are the same: As such, modelers are free to choose the representation they feel is visually most appealing. Note that the fork has additional semantics allowing modelers to specify a subset of tokens before the output token is offered.

Diagram A

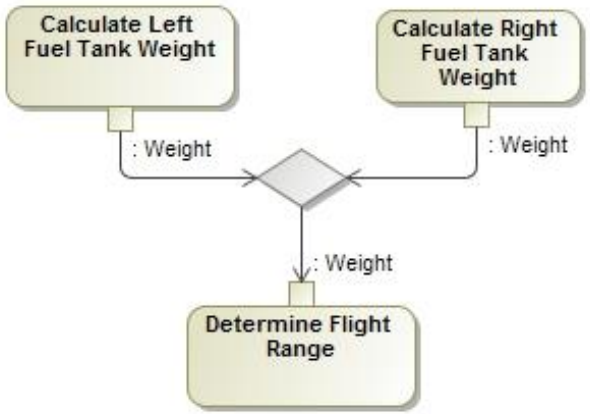


Diagram B

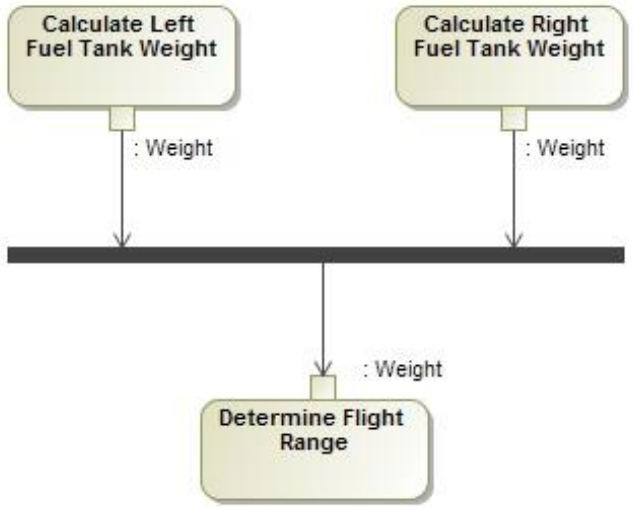


Diagram C