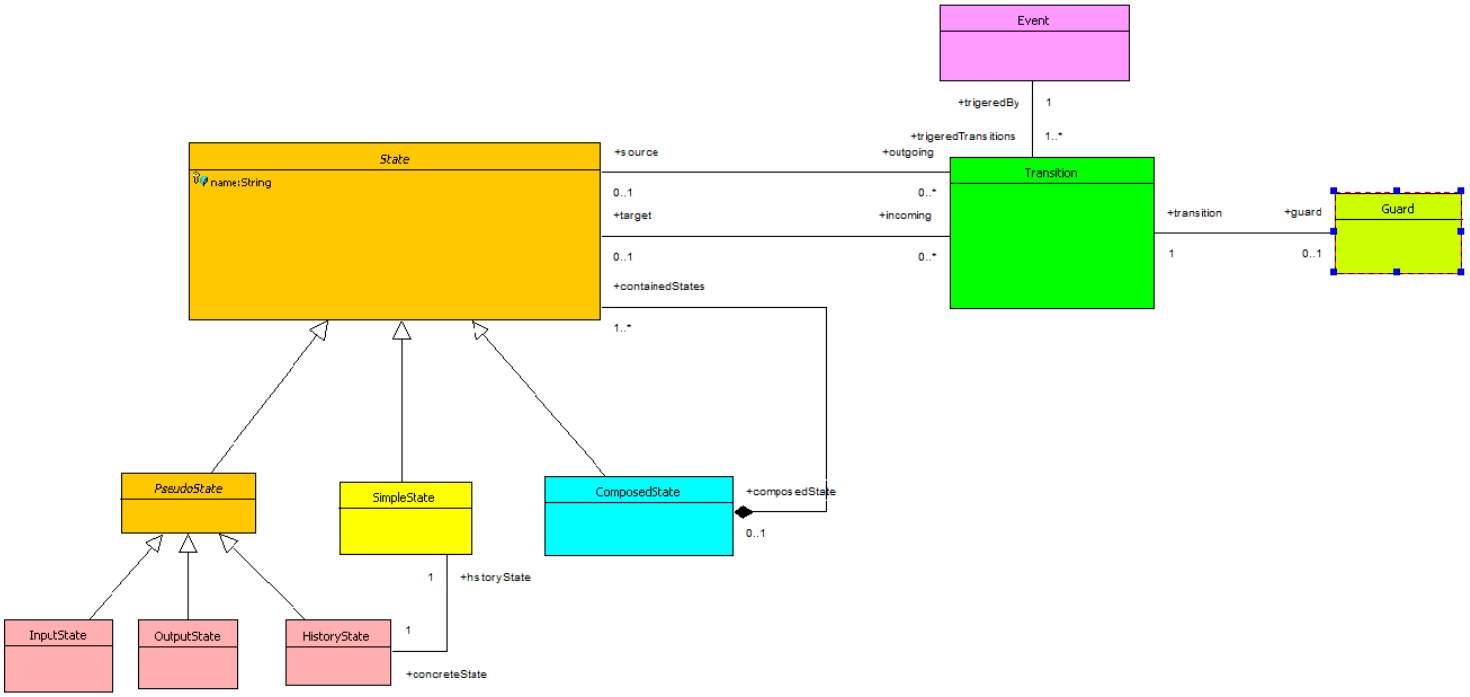
ISS colloquium the 3rd of June 2020

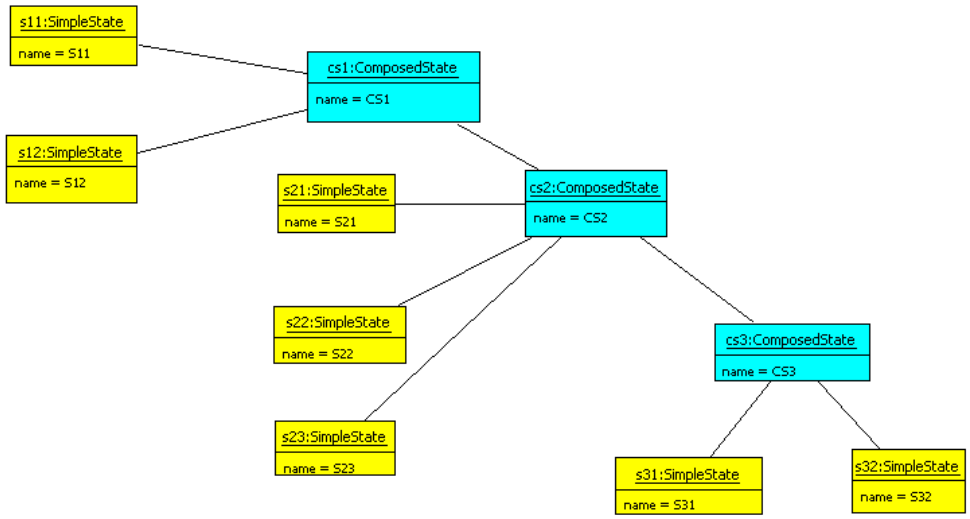
Working time 2h, 1pt by default

I. A state Machine contains states, transitions between states, events triggering transitions and guards that can be associated to transitions. States are named elements. Guards are assertions that enable transitions if and only if the mentioned event appears and the guard is evaluated at **true**. Like guards, events are associated to transitions. There are different kind of states: SimpleStates, CompositeStates (that may contain other states) and PseudoStates. There are three different kinds of PseudoStates. InputStates, OutputStates and HistoryStates. Each HistoryState has a reference towards the last SimpleState from which a transition was triggered when an event appeared at the level of the CompositeState in which the HistoryState is nested.

   a. Using the UML, please draw a class diagram representing the concepts above mentioned and their relationships. 3pt
   b. Using the OCL, please specify appropriate invariants in classes: OutputState and InputState mentioning that: there are not transitions starting from outputStates, only transitions entering in outputStates; and only a transition starting from each inputState. There are not transitions entering in inputStates. 0.5pt
   c. In the context of the CompositeState please specify an observer computing the number of simpleStates contained in that compositeState. 0.5pt

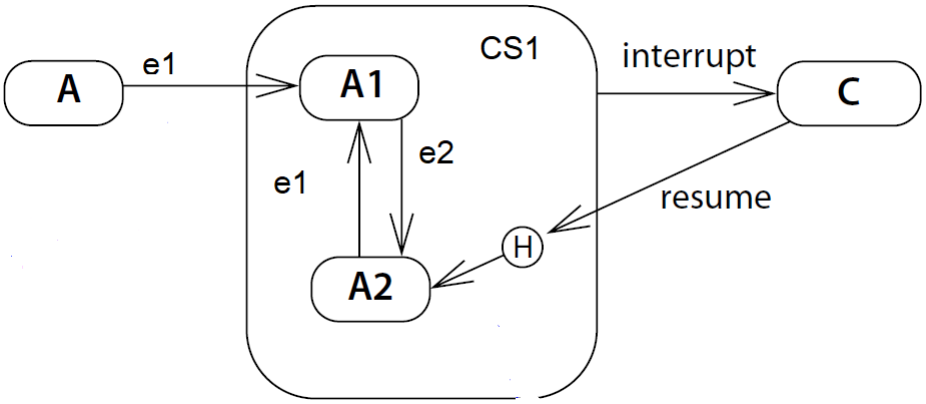


```
model StateMachineModel


context OutputState
    inv: not self.incoming->isEmpty and self.outgoing->isEmpty


context InputState
    inv: self.incoming->isEmpty and self.outgoing->size = 1

context ComposedState

    def numOfSimpleStates:

    let allNumOfSimpleStates:Integer = Set{self.containedStates}->flatten->closure(s | if s.oclIsTypeOf(ComposedState)
                                                                                        then s.oclAsType(ComposedState).containedStates
                                                                                        else s.composedState.containedStates
                                                                                      endif)->select(s | s.oclIsKindOf(SimpleState))->size

    let numOfSimpleStates:Integer = self.containedStates->select(s | s.oclIsKindOf(SimpleState))->size

endmodel
```



II. Using your own words, please explain how the stateMachine represented bellow works.

1.5pt



In each moment, the stateMachine will be in a simpleState. Irrespective of the simpleState in which the stateMachine is, if another event, not associated with the transition will appear, nothing happens. For example, if the statemachine is in the state A and one of the events ***e2***, ***interrupt*** or ***resume*** will appear, nothing happens.

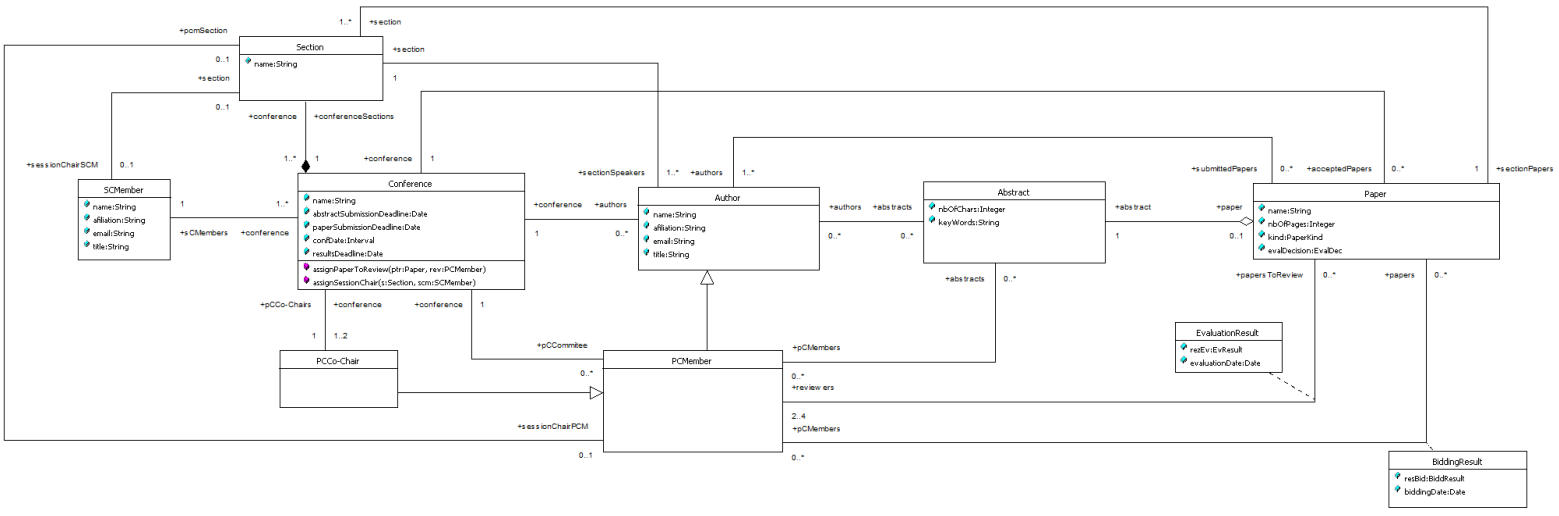
When the stateMachine is in the state **A**, waits until the event ***e1*** will appear. Then a transition will be triggered towards the **A1** state.

In the **A1** state, if the **e2** event will appear, then a transition in the **A2** will be triggered. If the ***interrupt*** event will appear, then a transition towards the state **C** will be triggered.

In the **A2** state, if the ***e1*** event will appear, then a transition in the **A1** will be triggered. If the ***interrupt*** event will appear, then a transition towards the state **C** will be triggered.

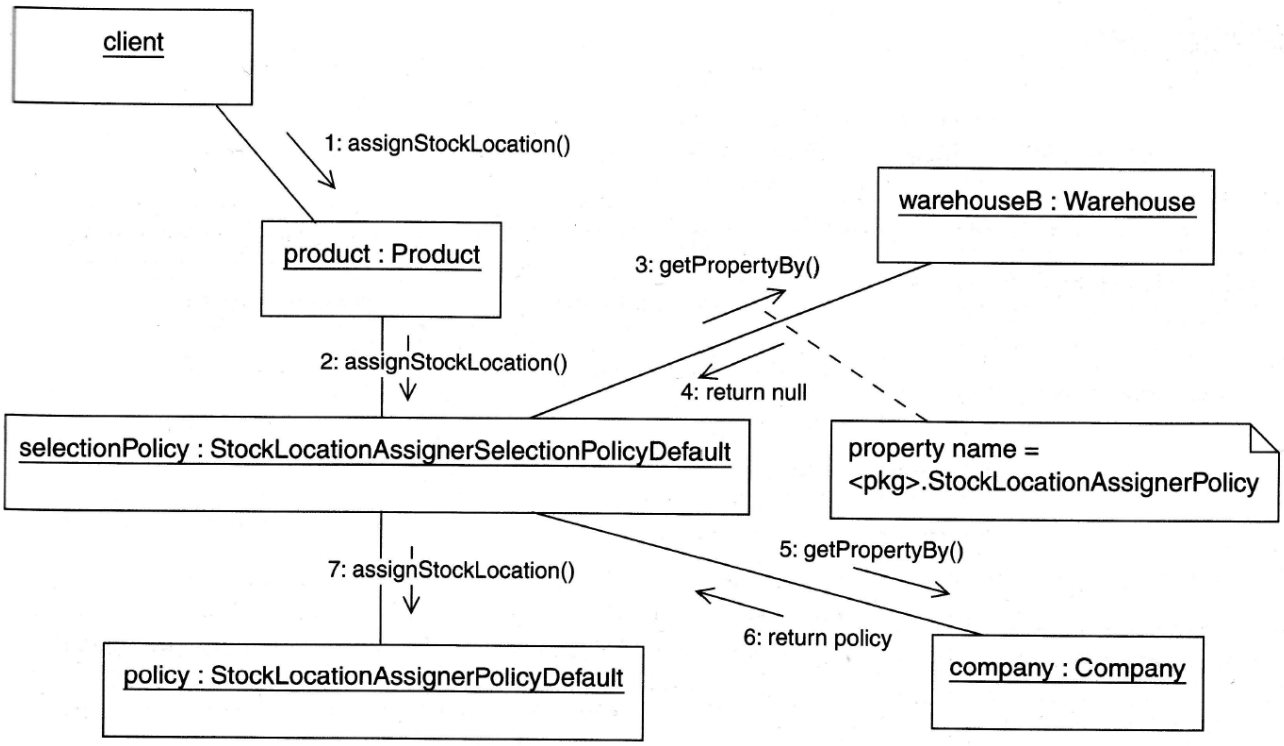
In the state **C**, when a ***resume*** event will appear, a transition towards the last simpleState of **CS1** will be triggered. In other words, the transition triggered by the ***resume*** event will be towards the same simpleState from which the **C** state was entered when the **interrupt** event has appeared.

III. The unicity of elements names contained in a namespace is one of the most important constraints in programming and modeling languages. In case of automate code generation, it is forbidden to name elements with a reserved word in the target programming language. In our case, this language is Java. In this context, please check if the model represented in the next page complies with this rule. If not, please clearly explain all the problems you noticed. 1.5 pt



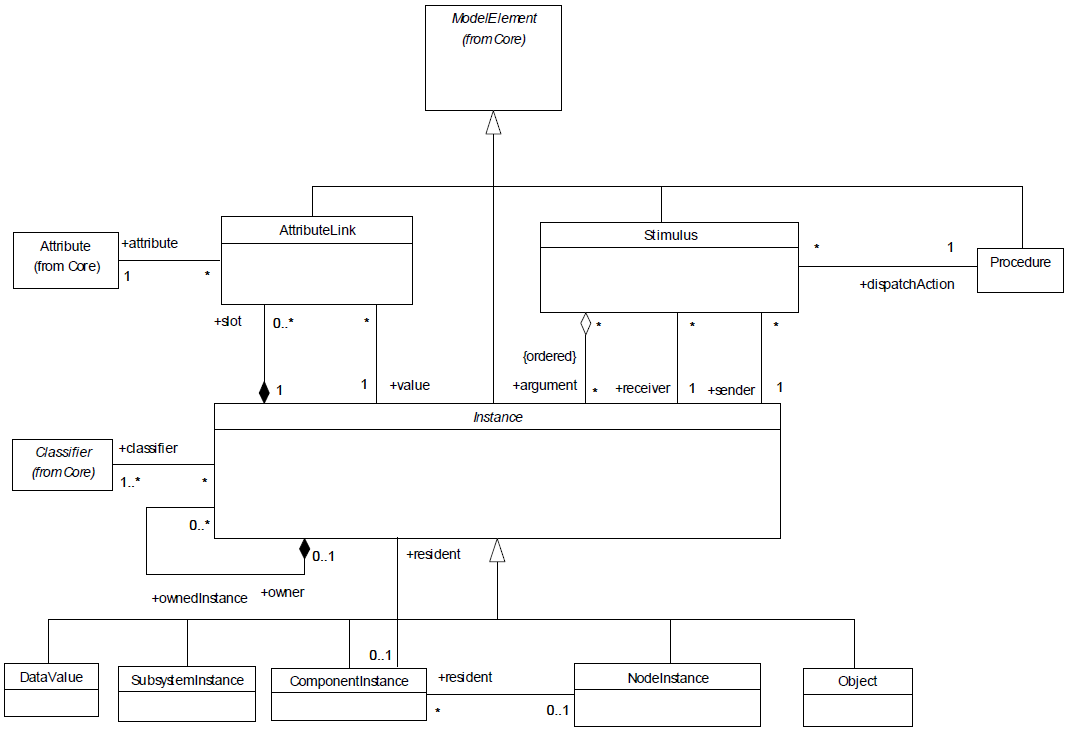
In the PCCo-Chair namespace, there are two oppositeAssociatedEnds named *conference*. One is related to the bidirectional association between PCCo-Chair and Conference classes, the other is inherited from the PCMember class, due to the association between PCMember and Conference. A similar conflict exists in the PCMember class due to opposite associationEnd *conference*, and of the associationEnd having the same name, inherited from the classAuthor. In the class Paper, an opposite associationEnd is named *abstract* which is a reserved word in Java.

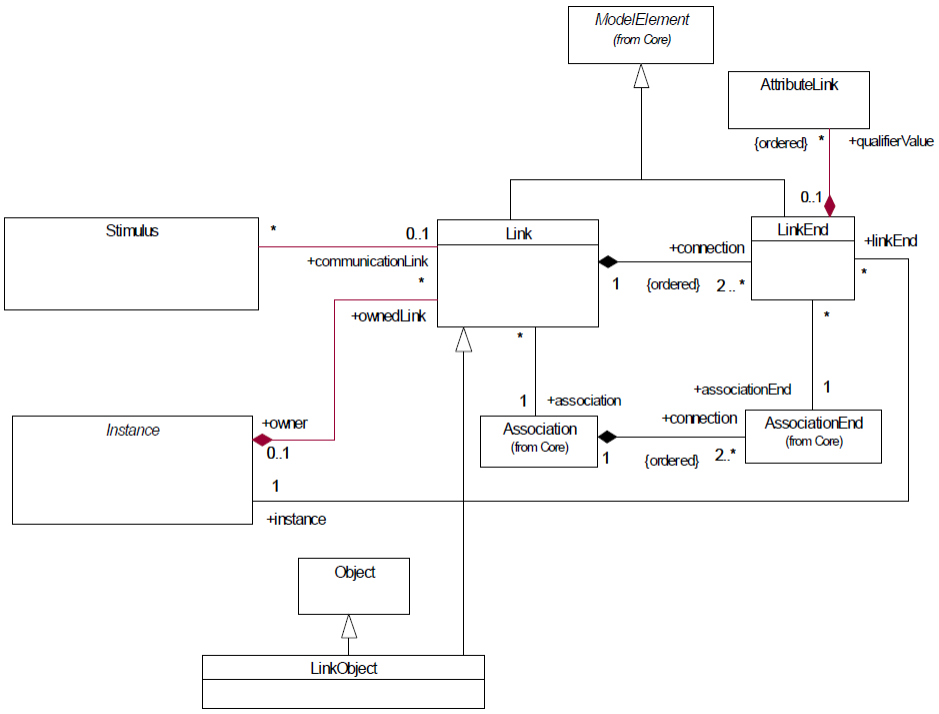
IV. Please name the diagram represented bellow, name the elements represented in this diagram and identify the metaclasses of these elements from the two diagrams represented in the following. 2pt



It is a collaboration diagram;

- *client, product, warehouse, selectionPolicy, company* and *policy* are **Object *instances*,**
- between *client* and *product*, *product selectionPolicy, selectionPolicy* and *warehouse*, *selectionPolicy* and *policy* and between *selectionPolicy* and *company* we have ***instances* of Link,**
- *assignStockLocation(), getPropertyBy()*, *return null* and *return policy* are ***instances* of Stimulus.**


ModelElement
(from Core)
AttributeLink
Attribute
(from Core)
+attribute
1
*
+slot
0..*
*
1
1
+value
Stimulus
*
1
Procedure
+dispatchAction
{ordered}
+argument
*
+receiver
1
+sender
1
*
*
*
Instance
Classifier
(from Core)
+classifier
1..*
*
0..*
0..1
+resident
+ownedInstance
+owner
DataValue
SubsystemInstance
ComponentInstance
0..1
+resident
*
0..1
NodeInstance
Object


ModelElement
(from Core)
AttributeLink
{ordered}
*
+qualifierValue
0..1
Stimulus
*
0..1
+communicationLink
*
Link
+connection
1
{ordered}
2..*
LinkEnd
+linkEnd
*
+ownedLink
*
*
1
+association
+associationEnd
1
Instance
+owner
0..1
1
Association
(from Core)
+connection
1
{ordered}
2..*
AssociationEnd
(from Core)
+instance
Object
LinkObject