ISS 08-07-2021

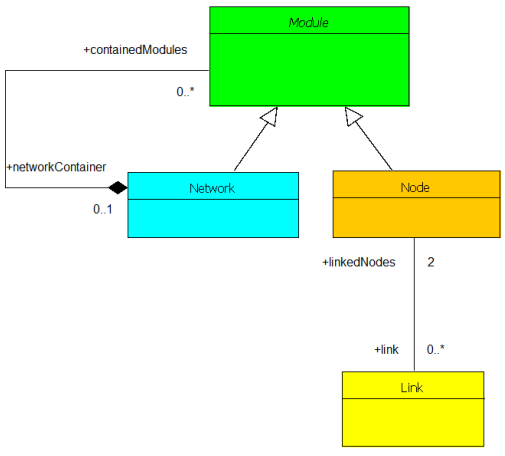
I. Let us consider a network containing nodes and links between two nodes. A network may contain also other networks, nodes, and links which connect two nodes. Networks are organized in a structured manner. There is a unique root network which is not contained in any other network. Excepting the root network, all the other networks are directly contained (owned) by a network. Nodes can/may be connected by links. Links may be included in a network or the may connect two different networks.

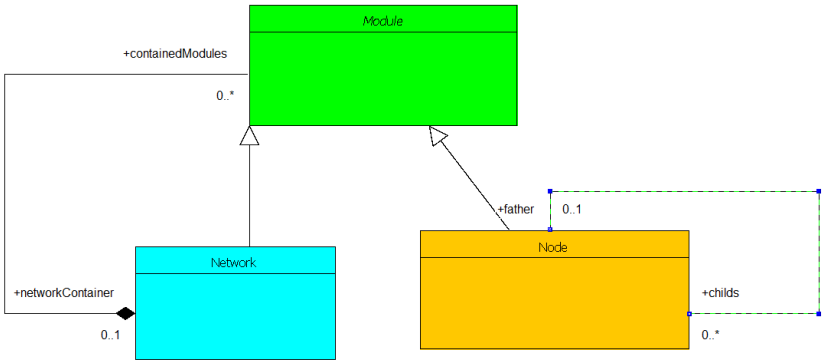
a. Using the UML, please design the structural view of a model complying with the above-mentioned requirements. 2pt

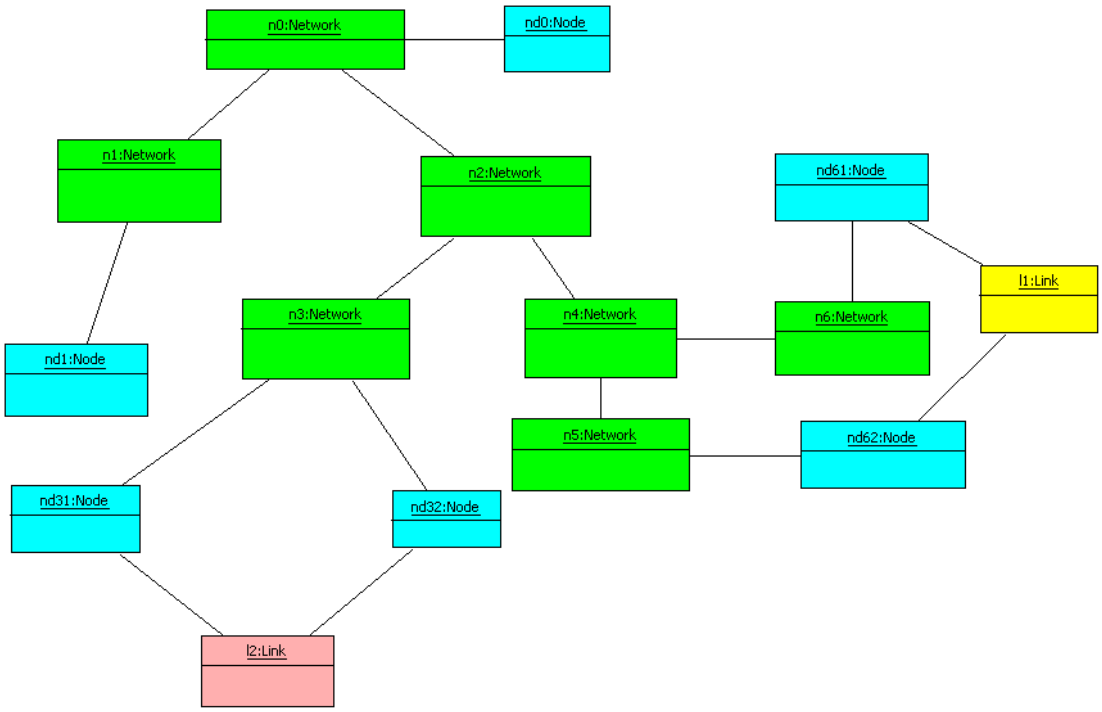
b. Using the OCL, please specify an invariant imposing that in case of links connecting two networks, the nodes of the link are from the two networks connectd. 0.5pt

c. Using the OCL, please specify in the context Network observers returning the root network, and the set of all networks contained in the nested networks in which is included the current network. 1.5pt

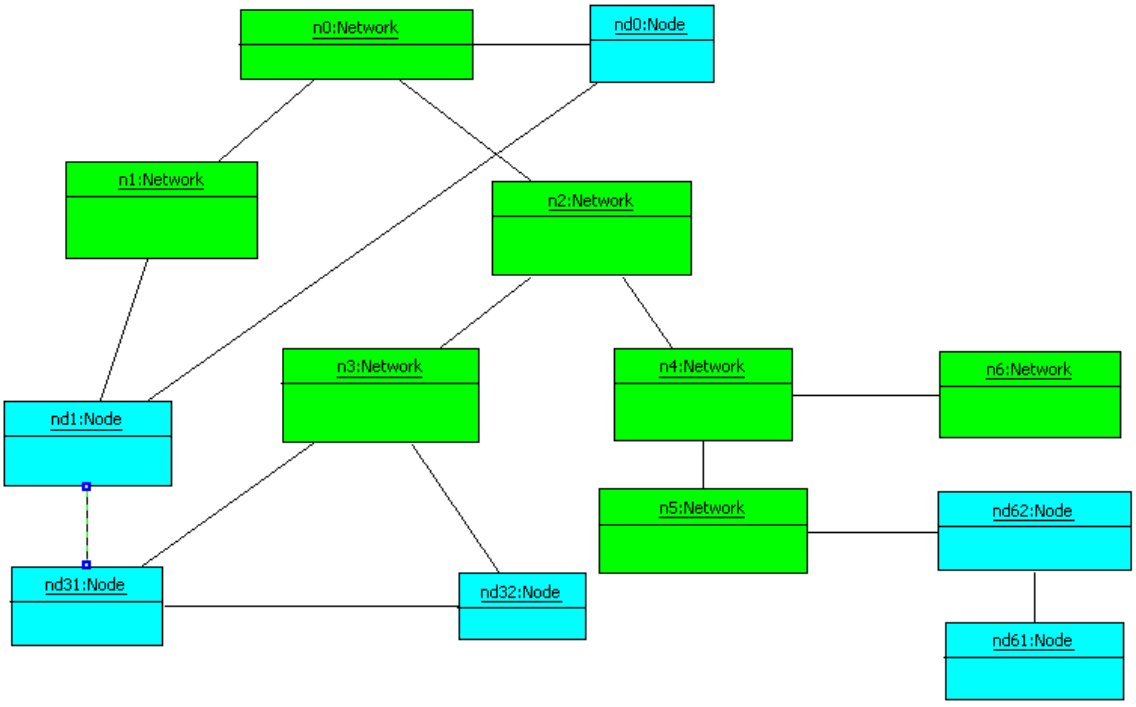
```
context Link

    inv linkBetween2Networks:

        self.nodes.networkContainer->asSet->size = 2

context Network

    def allNetworks:

        let allChildNetworksFL:Set(Network) = (self.containedModules->select(m:Module | m.oclIsTypeOf(Network)).oclAsType(Network))->asSet

        let allChildNetworksDE:Set(Network) = allChildNetworksFL->union(allChildNetworksFL->closure(n:Network | n.allChildNetworksFL))

        let networkRoot: Network =  if self.networkContainer->isEmpty
                                        then self
                                        else self.networkContainer.networkRoot
                                    endif

        let allNetworks:Set(Network) = networkRoot.allChildNetworksDE->including(networkRoot)
```





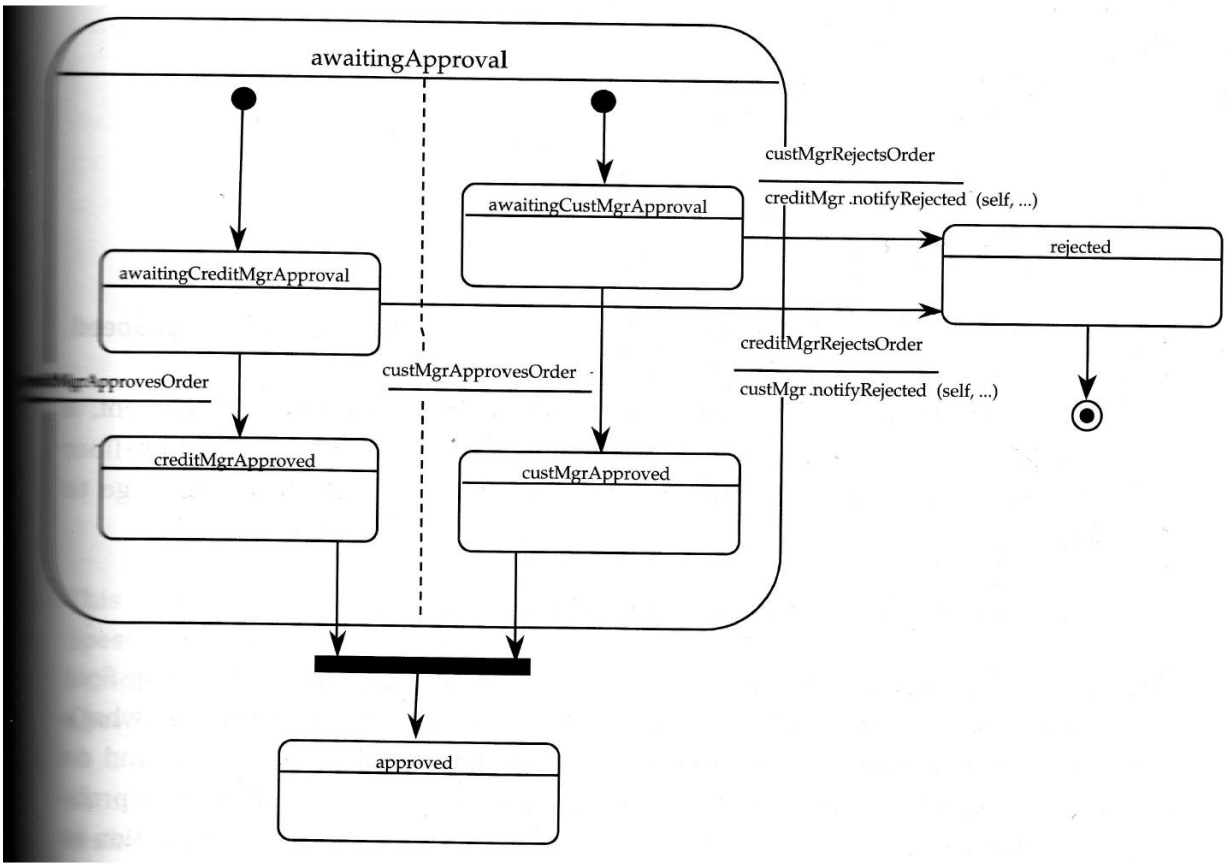


A snapshot obtained by instantiating the above left diagram



A snapshot obtained by instantiating the above right diagram

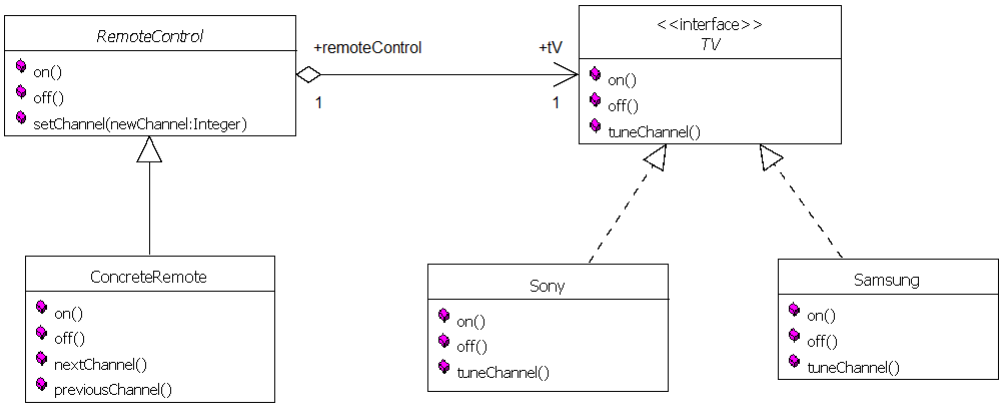
II. Let us consider the diagram represented in the figure above. Please describe the functionality specified by the diagram, mentioning the model elements used to specify this behavior. Please mention the kind of appropriate objects to specify the behavior using this kind of diagram. What is the distinctiveness of the above diagram compared with similar diagrams you encountered often? (the blurred name is creditMgrApprovesOrder) 2pt



State Transition Diagrams/State Machine Diagrams are apropriate to describe/specify the behavior of so named **reactive objects**. By reactive object we mean an object whose behavior depends on the stimulus received and on the current state of the object. The diagram represented in the figure above is an **ortogonal STD** – used to **specify the behavior of concurent reactive objects**. Ortogonal STDs may have 2 or many ortogonal regions. At any moment of its lifetime, the object is in a state which is a tuple of concret states (one for each ortogonal region). In our case, the first concret state is defined by the tuple (**awaitingCreditMgrApproval, awaitingCustMgrApproval**). In this concrete state, if/when one of the messages/stimuli/triggers **creditMgrRejectsOrder** or **custMgrRejectsOrder** is received, the object will transit from the composed state **awaitingApproval** into the concrete state **rejected** and after automaticalli in the **output state** because the object behavior is not important. The transition into the **rejected** state is preceded by sending a message **notifyRejected(self, …)** towards the object **creditMgr** or

custMgr depending on the trigger received **creditMgrRejectsOrder** or **custMgrRejectsOrder.** If a stimulus **creditMgrApprovesOrder** will be received when the concrete state is (**awaitingCreditMgrApproval, awaitingCustMgrApproval**), then the **left ortogonal region** will transit into the concrete state **creditMgrApproved**. The object will remain in this state until a stimulus **custMgrApprovesOrder** will be received, because in this situation, the right ortogonal region will transit into the state **creditMgrApproved**. When the object is in the state (**creditMgrApproval, custMgrApproval**) then, the object will transit automaticalli into the concrete state **approved**. A similar situation happens when the message **custMgrApprovesOrder** will be received before the message **creditMgrApprovesOrder.**

III. Let us suppose that you need to design and implement a software to manage the TV remote's control for different TVs like Samsung, Sony, a.s.o. It is resonable to suppose that the application would not be right from the first time. Moreover, as the number of feedbacks from users will increase, you'l have the oportunity to improve the aplication. Also, it is resonable to suppose that new versions and generations of TVs will require new functionalities. Using UML, please design an apropriate architecture of the application supporting in an easy manner the updating and development of the application. For the beginning let us suppose that the firsts TV functionalities needing to be implemented will be on, off and tuneChannel. 1pt



The BridgeDesignPattern has two hierarchies, one for the remotes and a separate one for platform specific TV implementations. This architecture allows to change either side of the two hierarchies independently enabling an easy update/extenssion of the aplication.

IV. Using your own words, please explain all the information included in an UML model which can be used to generate the code correspondig to the model, in an automated manner. Which information is in your oppinion not yet used in the code generated and why? Which are in your opinion the drawbacks of automated code generation? Why? 2pt

Almost all the tools supporting automatic code generation of UML models uses the architectural information which specifies all the components of the architectural view (classes, interfaces and packages including their structure and behavioral elements - opertions), and different kinds of relationships among these components. Inheritance and associations among classes, implementation between classes and

interfaces, import of different components of a package into other packages (specified in UML by dependency relationships). There are different levels of detail regarding these implementations. The complete management of all kinds of associations is the most importat. Also, there are differences among the implementation of explicit constructors. However, we may conclude that all the architectural information could be used in direct engineering. The usage of the information provided by the behavioral view is done only partial. There are some kind of diagrams like STDs and ADs(Activity Diagrams) whose information is apropriate to be used in direct engineering. The Sequence Diagrams or Collaboration/Communication Diagrams do not offer the whole information needed generating in an automated manner all the code corresponding to the described behavior. Only method composition and the sequence of sending messages is specified in diagrams, the rest of the code must be implemented manually.

However, the code corresponding to observers and constraints can be completely generated in an automated manner, as happened in OCLE for example.

To be really uselful, the automatic code generation needs to be preceded by checking UML model compilability and behavior checking by means of simulations of apropriate model instantiations.