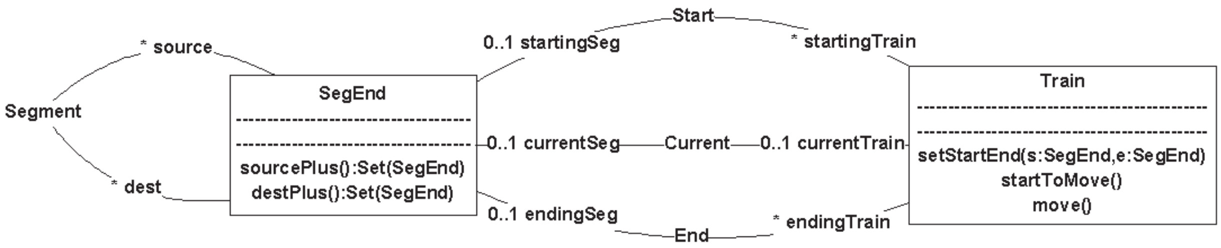
Formal Methods versus UML's OCL

Martin Gogolla
University of Bremen

The talk explains the connection between the Unified Modeling Language UML and its Object Constraint Language OCL. It shows OCL features by considering the BART case study and giving a small abstract specification for it. This specification is animated by means of the UML Specification Environment USE developed at University of Bremen. Finally the talk discusses to what extent OCL is different from existing formal specification languages.




```
context SegEnd inv noDirectLoopsForSource:
  source->excludes(self)

context SegEnd inv noDirectLoopsForDest:
  dest->excludes(self)

context Train inv existsPathFromStartToEnd:
  (startingSeg.isDefined() and endingSeg.isDefined()) implies
    startingSeg.destPlus()->includes(endingSeg)

context Train inv existsPathFromStartToCurrent:
  currentSeg.isDefined() implies
    (startingSeg=currentSeg or
     startingSeg.destPlus()->includes(currentSeg))

context Train inv existsPathFromCurrentToEnd:
  currentSeg.isDefined() implies
    (currentSeg=endingSeg or
     currentSeg.destPlus()->includes(endingSeg))

context Train inv sourceOfCurrentSegEndNotOccupied:
  currentSeg.isDefined() implies
    (currentSeg.source->collect(se|se.currentTrain)->
       select(t|t.isDefined())->isEmpty)

context Train inv destOfCurrentSegEndNotOccupied:
  currentSeg.isDefined() implies
    (currentSeg.dest->collect(se|se.currentTrain)->
       select(t|t.isDefined())->isEmpty)

context Train::setStartEnd(s:SegEnd,e:SegEnd)
pre: startUnequalEnd: s<>e
post assignedStartingSeg: startingSeg=s
post assignedEndingSeg: endingSeg=e

context Train::startToMove()
post startedWithStartingSeg: currentSeg=startingSeg

context Train::move(c:SegEnd)
pre destIncludesSegEndToMoveTo: currentSeg.dest->includes(c)
post assignedCurrentSeg: currentSeg=c
```

Figure 1: UML Class Diagram for Part of BART Case Study

This approach to the BART case study starts with the class diagram given in Figure 1. In the class diagram, tracks are described by objects of class `SegEnd` and an association `Segment` between `SegEnd` objects. Links in the association `Segment` represent directed edges with a source `SegEnd` and a destination `SegEnd` object. Trains are specified by a class `Train`. A journey for a train is determined by starting and ending `SegEnd` objects and a current `SegEnd` object giving the current position of the train. The class diagram

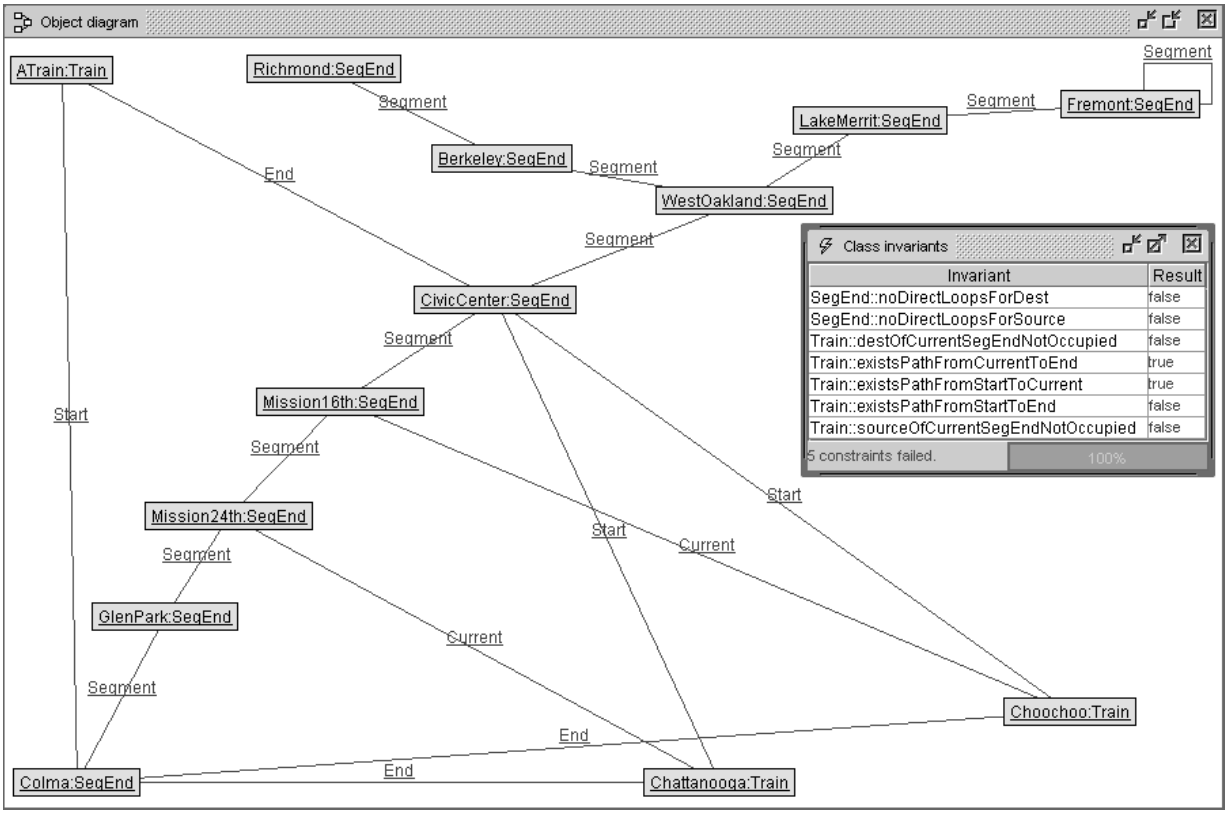


Figure 2: USE Object Diagram

is further restricted by (1) invariants expressing operation independent properties which hold in all system states and (2) pre- and postconditions describing the operations in more detail. A snapshot of the USE animation with stations represented as `SegEnd` objects and `Train` objects is pictured in Figure 2. The window entitled "Class invariants" analyses the specified invariants and reveals that some of them do not hold in the shown system state. Double clicking a "false" invariant would open a further window explaining in detail the reason for invariant failure.