Behavior Modeling with Interaction Diagrams in a UML and OCL Tool

Martin Gogolla, Lars Hamann, Frank Hilken, Matthias Sedlmeier, Quang Dung Nguyen

University of Bremen, Germany
**Aim of USE:** support development by reasoning about model through

(a) **validation**, i.e., checking informal expectations against formally given properties, for example, by stating OCL queries against a reached system state

(b) **verification**, i.e., checking formal properties of the model, for example by considering model consistency or independence of invariants; USE supports making deductions from stated model on the basis of finite search space of possible system states

- USE supports the development of **tests** (scenarios)

- OCL operations **contracts**, i.e., pre- and postconditions, are general OCL formulas

  - in **postconditions**, one can refer with @pre to attribute and association end values at precondition time; postconditions formulate general requirements; not restricted to changes to attribute and association end values; postconditions need not to determine a unique post-state

- **actual changes** by the SOIL operations that are checked against the contract
Tasks for OCL within USE

- in class diagrams for
  (a) class invariants
  (b) operation contracts (pre- and postconditions)
  (c) attribute and association derivation rules
  (d) attribute initializations

- in protocol state machines for
  (e) state invariants
  (f) transition pre- and postconditions

- furthermore for
  (g) ad-hoc OCL queries in object diagrams
  (h) expressions within SOIL

Class diagrams and protocol machines enriched by invariants, operation contracts, statechart constraints and SOIL operation implementations determine system structure and behavior

Sequence and communication diagrams in USE for visualizing and analyzing specified test cases in form of scenarios; interaction diagrams do not restrict system behavior, but document, analyze, and help to understand the interactions
Enter OCL expression:

Point.allInstances->select(p |
  Truck.allInstances->forall(t |
    t.trips->excludes(p))))

Result:

Set{m : Set(Point)}
Example of operation implementation and pre- and postconditions

Truck::move(target:Point)
    begin
    self.trips:=self.trips->including(target);
    self.debt:=self.debt+1;
    delete (self,self.current) from Current;
    insert (self,target) into Current;
    end

pre  currentExists:
    self.current->notEmpty
pre  targetReachable:
    self.current.north->union(self.current.south)
        ->includes(target)

post  debtIncreased:
    self.debt@pre+1=self.debt
post  tripsUpdated:
    self.trips@pre->including(target)=self.trips
post  currentAssigned:
    target=self.current
post  allTruckInvs:
    numIsKey()
Example for motorway with west/east connections
9. new Truck("freds_scania")
10. hh.southConnect(b)
11. b.southConnect(m)
12. !new Point("m")
13. !b.init("B")
14. !hh.init("HH")
15. !new Point("hh")
9. new Point("hh")
10. hh.init("HH")
11. new Point("b")
12. b.init("B")
13. new Point("m")
14. m.init("M")
15. hh.southConnect(b)
16. b.southConnect(m)
17. new Truck("freds_scania")
18. freds_scania.init("BRB-MS 1936")
Sequence diagram

hh:Point  b:Point  m:Point  freds_scania:Truck

init("Hi")

init("B")

init("M")
southConnect(b)
southConnect(m)

init("BRB-MS 1936")

enter(hh)

move(b)

move(m)

pay(3)

bye()
create hh:Point
hh.init('HH')
create b:Point
b.init('B')
create m:Point
m.init('M')

hh.southConnect(b)
b.southConnect(m)"
Messages 1..21 - only messages 9..15 displayed
Messages 1..21 - OCL select for truck object identity
Realized UML/OCL concepts in USE [concepts in brackets not in example]

- class, attribute, datatype, operation, operation signature
- class invariant
- query operation definition with OCL;
  non-query operation with state changes through definition in SOIL
- attribute initialization, derived attribute, [derived association]
- operation contract, i.e., pre- and postconditions
- association, role names, multiplicities,
  [aggregation, composition, qualified association]
- [generalization]
  [subsets constraint, redefines constraint, union constraint]
- protocol state machines with states and transitions
  state invariants, [transition pre- and postcondition]
--------------------------- model TollCollect
model TollCollect
--------------------------- class Truck
class Truck
attributes
  num:String          init: ''
  trips:Sequence(Point) init: Sequence{}
  debt:Integer       init: 0
operations
  init(aNum:String)
    begin self.num:=aNum end
  enter(entry:Point)
    begin insert (self,entry) into Current; self.debt:=1;
       self.trips:=self.trips->including(self.current) end
  move(target:Point)
    begin self.trips:=self.trips->including(target);
       self.debt:=self.debt+1; delete (self,self.current) from Current;
       insert (self,target) into Current end
  pay(amount:Integer)
    begin self.debt:=self.debt-amount end
  bye():Integer
    begin delete (self,self.current) from Current;
       result:=self.debt.abs(); self.debt:=0 end
numIsKey():Boolean=
    Truck.allInstances->forAll(self,self2|
        self<>self2 implies self.num<>self2.num)

statemachines
psm TruckLife
states
    prenatal:initial
    born     [num='']
    noDebt   [num<>' ' and current->isEmpty]
    debt     [num<>' ' and current->notEmpty]
transitions
    prenatal -> born   { create }
    born      -> noDebt { init() }
    noDebt    -> debt   { enter() }
    debt      -> debt   { move() }
    debt      -> debt   { pay() }
    debt      -> noDebt { bye() }
end
end
class Point

attributes
  name: String
  init: ''
  isJunction: Boolean
    derived: north->union(south)->size() >= 2
    -- northSize: Integer
      derived: self.northPlus()->size()
    -- southSize: Integer
      derived: self.southPlus()->size()

operations
  init(aName: String)
    begin self.name:=aName end
  northConnect(aNorth: Point)
    begin insert (aNorth,self) into Connection end
  southConnect(aSouth: Point)
    begin insert (self,aSouth) into Connection end

northPlus(): Set(Point) = self.north->closure(p|p.north)
southPlus(): Set(Point) = self.south->closure(p|p.south)

nameIsKey(): Boolean =
  Point.allInstances->forAll(self,self2|
    self<>self2 implies self.name<>self2.name)
noCycles(): Boolean =
  Point.allInstances->forAll(self|
    not(self.northPlus()->includes(self)))
statemachines
  psm PointLife
  states
    prenatal:initial
    born [name='']
    growing [name<>'''
  transitions
    prenatal -> born   { create }
    born   -> growing { init() }
    growing -> growing { northConnect() }
    growing -> growing { southConnect() }
  end
end

----------------------------------------------- association Current
association Current between
  Truck[0..*] role truck
  Point[0..1] role current
end

----------------------------------------------- association Connection
association Connection between
  Point[0..*] role north
  Point[0..*] role south
end
Point::PointLife {protocol}

create /
prenatal

born
[(self.name = "")

init(aName : String) /

northConnect(aNorth : Point) /
southConnect(aSouth : Point) /

growing
[(self.name <> ")

constraints

context Truck inv numIsKeyInv:
  numIsKey()
context Point inv nameIsKeyInv:
  nameIsKey()
context Point inv noCyclesInv:
  noCycles()

Point::init

context Point::init(aName:String)
pre freshPoint:
  self.name='' and self.north->isEmpty and self.south->isEmpty
pre aNameOk:
  aName<>'' and aName<>null
post nameAssigned:
  aName=self.name
post allPointInvs:
  nameIsKey() and noCycles()
Point::northConnect
context Point::northConnect(aNorth:Point)
pre aNorthDefined:
   aNorth.isDefined
pre freshConnection:
   self.north->excludes(aNorth) and self.south->excludes(aNorth)
pre notSelfLink:
   self<>aNorth
pre noCycleIntroduced:
   aNorth.northPlus()->excludes(self)
post connectionAssigned:
   self.north->includes(aNorth)
post allPointInvs:
   nameIsKey() and noCycles()

Point::southConnect
context Point::southConnect(aSouth:Point)
pre aSouthDefined:
   aSouth.isDefined
pre freshConnection:
   self.south->excludes(aSouth) and self.south->excludes(aSouth)
pre notSelfLink:
   self<>aSouth
pre noCycleIntroduced:
   aSouth.southPlus()->excludes(self)
post connectionAssigned:
   self.south->includes(aSouth)
post allPointInvs:
   nameIsKey() and noCycles()
context Truck::init(aNum:String)
pre freshTruck:
    self.num='' and self.trips=Sequence{} and self.debt=0 and
    self.current->isEmpty
pre aNumOk:
    aNum<>'' and aNum<>null
post numAssigned:
    aNum=self.num
post allTruckInvs:
    numIsKey()

context Truck::enter(entry:Point)
pre noDebt:
    0=self.debt
pre currentEmpty:
    self.current->isEmpty
pre entryOk:
    entry<>null
post debtAssigned:
    1=self.debt
post tripsUpdated:
    self.trips@pre->including(entry)=self.trips
post currentAssigned:
    entry=self.current
post allTruckInvs:
    numIsKey()}
--- Truck::move ---

context Truck::move(target:Point)
pre currentExists:
  self.current->notEmpty
pre targetReachable:
  self.current.north->union(self.current.south)->includes(target)
post debtIncreased:
  self.debt@pre+1=self.debt
post tripsUpdated:
  self.trips@pre->including(target)=self.trips
post currentAssigned:
  target=self.current
post allTruckInvs:
  numIsKey()

--- Truck::pay ---

context Truck::pay(amount:Integer)
pre amountPositive:
  amount>0
pre currentExists:
  self.current->notEmpty
post debtReduced:
  (self.debt@pre-amount)=(self.debt)
post allTruckInvs:
  numIsKey()
--- Truck::move ---

context Truck::move(target:Point)
pre currentExists: self.current->notEmpty
pre targetReachable:
  self.current.north->union(self.current.south)->includes(target)
post debtIncreased: self.debt@pre+1=self.debt
post tripsUpdated: self.trips@pre->including(target)=self.trips
post currentAssigned: target=self.current
post allTruckInvs: numIsKey()

--- Truck::pay ---

context Truck::pay(amount:Integer)
pre amountPositive: amount>0
pre currentExists: self.current->notEmpty
post debtReduced: (self.debt@pre-amount)=(self.debt)
post allTruckInvs: numIsKey()

--- Truck::bye ---

context Truck::bye():Integer
pre currentExists: self.current->notEmpty
pre noDebt: self.debt<=0
post resultEqualsOverPayment: self.debt@pre.abs()=result
post zeroDebt: self.debt=0
post currentEmpty: self.current->isEmpty
post allTruckInvs: numIsKey()
Thanks for your attention!
Motivation and context

- system modeling with UML behavior diagrams

- statecharts and both kinds of interaction diagrams: sequence and communication diagrams

- new implementation features in a UML and OCL modeling tool: USE (Uml-based Specification Environment)

- sequence diagram lifelines are extended with states from statecharts

- communication diagrams introduced as an alternative to sequence diagrams

- assess introduced features and propose systematic set of features that should be available in both interaction diagrams

- emphasize the role that OCL plays for such a feature set
Common selection criteria for sequence and communications diagrams

Selection focusing on objects: Objects could be selected through

- Interactive hide or show for objects
- Objects satisfying resp. violating an OCL invariant during interaction
- Objects satisfying resp. violating an ad-hoc OCL formula during interaction
Common selection criteria for sequence and communications diagrams

Selection focusing on messages: Messages could be selected through

- Interactive hide or show for messages
- Selection through an OCL object query identifying the sending object
- Selection through a satisfied resp. violated OCL pre- or postcondition
- Selection through a satisfied resp. violated ad-hoc OCL formula at pre- or postcondition time during an operation call
- Selection by message kind: object creation, object destruction, link insertion, link deletion, attribute assignment, operation call
- Selection by message number depth
- Determination of a message interval defined by
  - interactively fixed start message and end message
  - start OCL formula and end OCL formula
  - a statechart start state and a statechart end state for a fixed object
Contribution discussed

- how to handle UML interaction diagrams in a model validation tool
- pointed to the link between protocol machine and interaction diagrams
- developed a desirable feature set for both kinds of UML interaction diagrams, namely sequence and communication diagrams

Future work

- complete our current implementation with the missing features in both interaction diagrams
- message selection and message interval selection could offer promising analysis options
- extend the options for interaction analysis with temporal OCL query features
- larger examples and case studies need to validate already existing and planned features for better support of interaction diagrams
Messages 1..21 - OCL select for trucks with coinciding last&first point on trip
Messages 1..21 - OCL select for point connection used twice