Formally Modeling Robot Cooperation in a Small Example Factory
Antonio Vallecillo, Martin Gogolla
University of Malaga, University of Bremen

1. Introduction
2. Structural Elements
3. Behavioral Elements
4. Potential for Expressing and Proving Properties
5. Lessons Learnt
6. Conclusion

References (DBLP: formal robot)
model Hammers

abstract class Part
operations
  comparableTo(p:Part):Boolean=null
end

class Head < Part
operations
  comparableTo(p:Part):Boolean=p.oclIsTypeOf(Head)
end

class Handle < Part
operations
  comparableTo(p:Part):Boolean=p.oclIsTypeOf(Handle)
end

class Hammer < Part
attributes
  isPolished: Boolean
operations
  comparableTo(p:Part):Boolean=p.oclIsTypeOf(Hammer)
end

abstract class Robot
end

abstract class Machine < Robot
operations
  start()
end

abstract class PartGenerator < Machine
end

class HandleGenerator < PartGenerator
attributes
  counter: Integer init: 0
operations
  start()
  begin -- test implementation, it only generates one Part
  declare hl: Handle;
  hl:=self.generate();
  self.output.put(hl);
  end
  generate(): Handle
  begin
  result:=new Handle;
  self.counter:=self.counter+1;
  end
end
class HeadGenerator < PartGenerator
attributes
  counter:Integer init: 0
operations
  start()
    begin -- test implementation, it only generates one Part
    declare hd:Head;
    hd:=self.generate();
    self.output.put(hd);
  end
  generate():Head
    begin
      result:=new Head;
      self.counter:=self.counter+1;
    end
end

class Assembler < Machine
operations
  start()
    begin
      declare hd:Part, hl:Part, hm:Hammer;
      hd:=self.input->select(t|t.parts->size>0 and 
        t.parts->forAll(oclIsTypeOf(Head)))->single().get();
      hl:=self.input->select(t|t.parts->size>0 and 
        t.parts->forAll(oclIsTypeOf(Handle)))->single().get();
      hm:=self.assemble(hd.oclAsType(Head),hl.oclAsType(Handle));
      self.output.put(hm);
    end
  assemble(hd:Head,hl:Handle):Hammer
    begin
      destroy hd,hl;
      result:=new Hammer;
      result.isPolished:=false;
    end
end

class Polisher < Machine
operations
  start()
    begin
      declare hm:Part;
      hm:=self.input->select(t|t.parts->size>0 and 
        t.parts->forAll(oclIsTypeOf(Hammer)))->single().get();
      self.polish(hm.oclAsType(Hammer));
      self.output.put(hm);
    end
  polish(hm:Hammer)
    begin
      hm.isPolished:=true;
    end
end
class Tray < Robot
attributes
cap: Integer -- capacity
operations
put(p: Part)
begin
insert(self, p) into Contains;
end
pre notFull: self.parts->size() < cap
post ElementAdded: self.parts = self.parts@pre->append(p)
get(): Part
begin
result := self.parts->at(1);
delete(self, result) from Contains;
end
pre notEmpty: self.parts->size() > 0
post FirstElementRemoved:
    result = self.parts@pre->at(1) and
    self.parts@pre = self.parts->prepend(result)
size(): Integer = self.parts->size()

statemachines
psm PutGet
states
    init: initial
    Empty [self.parts->size() = 0]
    Normal [0 < self.parts->size() and self.parts->size() < self.cap]
    Full [self.parts->size() = self.cap]
transitions
    init -> Empty { create }
    Empty -> Normal { [self.cap > 1] put() }
    Normal -> Normal { [self.parts->size() < cap - 1] put() }
    Normal -> Full { [self.cap > 1 and self.parts->size() = cap - 1] put() }
    Empty -> Full { [self.cap = 1] put() }
    Full -> Empty { [self.cap = 1] get() }
    Full -> Normal { [self.cap > 1] get() }
    Normal -> Normal { [self.cap > 1 and self.parts->size() > 1] get() }
    Normal -> Empty { [self.parts->size() = 1] get() }
end
end
-- associations
association MachineProduction between
    Machine [1..*] role input
    Tray [1] role output
end

association MachineConsumption between
    Tray [*] role input
    Machine [*] role output
end

composition Contains between
    Tray [1] role tray
    Part [*] role parts ordered
end
constraints

context Tray inv AtLeastOneElem:
    self.cap>0

context Tray inv SamePartsInTrays:
    self.parts->forall(p1,p2|p1.comparableTo(p2))

context Tray inv CapacityRespected:
    self.parts->size<=self.cap

context PartGenerator inv NoInputTray:
    self.input->size()=0

context HeadGenerator inv HeadsOut: self.output->size=1 and
    self.output.parts->forall(p|p.oclIsTypeOf(Head))

context HandleGenerator inv HandlesOut: self.output->size=1 and
    self.output.parts->forall(p|p.oclIsTypeOf(Handle))

context Assembler inv HandlesIn: self.input->size=2 and
    self.input->exists(b|b.parts->forall(p|p.oclIsTypeOf(Handle)))

context Assembler inv HeadsIn:
    self.input->exists(b|b.parts->forall(p|p.oclIsTypeOf(Head)))

context Assembler inv HammersOut: self.output->size=1 and
    self.output.parts->forall(p|p.oclIsTypeOf(Hammer))

context Polisher inv HammersIn: self.input->size=1 and
    self.input->one(b|b.parts->forall(p|p.oclIsTypeOf(Hammer)))

context Polisher inv HammersOut: self.output->size=1 and
    self.input.size=1 and
    self.output.parts->forall(p|p.oclIsTypeOf(Hammer))

-- context HeadGenerator inv MachineConnectionTyping:
--    self.input.input->isEmpty and
--    self.output.output->forall(ooclIsTypeOf(Assembler))
--
-- context HandleGenerator inv MachineConnectionTyping:
--    self.input.input->isEmpty and
--    self.output.output->forall(ooclIsTypeOf(Assembler))
--
-- context Assembler inv MachineConnectionTyping:
--    self.input.input->exists(ooclIsTypeOf(HeadGenerator)) and
--    self.input.input->exists(ooclIsTypeOf(HandleGenerator)) and
--    self.input.input->forall(
--        ooclIsTypeOf(HeadGenerator) or ooclIsTypeOf(HandleGenerator)) and
--    self.output.output->forall(ooclIsTypeOf(Polisher))
--
-- context Polisher inv MachineConnectionTyping:
--    self.input.input->forall(ooclIsTypeOf(Assembler)) and
--    self.output.output->isEmpty
Machines

```
!new HandleGenerator('hag')
!new HeadGenerator('heg')
!new Assembler('asm')
!new Polisher('pol')
```

Trays

```
!new Tray('Handle2Assem')
!Handle2Assem.cap:=4;
!new Tray('Head2Assem')
!Head2Assem.cap:=4;
!new Tray('Assem2Polish')
!Assem2Polish.cap:=4;
!new Tray('Polish2Out')
!Polish2Out.cap:=4;
```

Production Line Connections

```
!insert (hag,Handle2Assem) into MachineProduction
!insert (heg,Head2Assem) into MachineProduction
!insert (Handle2Assem,asm) into MachineConsumption
!insert (Head2Assem,asm) into MachineConsumption
!insert (asm,Assem2Polish) into MachineProduction
!insert (Assem2Polish,pol) into MachineConsumption
!insert (pol,Polish2Out) into MachineProduction
```

Process

```
!heg.start()
!hag.start()
!asm.start()
!pol.start()
```
Testing the model with the USE model validator

<table>
<thead>
<tr>
<th>Class</th>
<th>Min</th>
<th>Max</th>
</tr>
</thead>
<tbody>
<tr>
<td>HandleGenerator</td>
<td>1..1</td>
<td></td>
</tr>
<tr>
<td>HeadGenerator</td>
<td>1..1</td>
<td></td>
</tr>
<tr>
<td>Assembler</td>
<td>1..1</td>
<td></td>
</tr>
<tr>
<td>Polisher</td>
<td>1..1</td>
<td></td>
</tr>
<tr>
<td>Tray</td>
<td>4..4</td>
<td></td>
</tr>
<tr>
<td>Head</td>
<td>2..2</td>
<td></td>
</tr>
<tr>
<td>Handle</td>
<td>3..3</td>
<td></td>
</tr>
<tr>
<td>Hammer</td>
<td>4..4</td>
<td></td>
</tr>
<tr>
<td>MachineProduction</td>
<td>0..*</td>
<td></td>
</tr>
<tr>
<td>MachineConsumption</td>
<td>0..*</td>
<td></td>
</tr>
<tr>
<td>Contains</td>
<td>0..*</td>
<td></td>
</tr>
</tbody>
</table>
potential of developing model-driven robot descriptions with UML and OCL

- visualization of complex structures and processes for better understanding and documentation

- execution and simulation of scenarios, i.e. operation call sequences
  - different scenarios with different structural properties
    e.g. trays with different capacity
  - variations of a single scenario with equivalence checking by checking different operation call orders
    e.g. headG.start();handleG.start() == handleG.start();headG.start()

- checking of structural local and global properties within states by OCL queries
  - e.g. calculating the missing number of heads or handles to be produced to compensate the other over production
  - e.g. dependencies between tray capacities; for example, between number of unpolished and polished hammers
  - e.g. capturing in a global System object
    numOfProducedHeads:Integer
    numOfProducedHandles:Integer
    numOfProducedUnpolishedHammers:Integer
    numOfProducedPolishedHammers:Integer
    aiming towards saturated (or stable) states with (0,0,0,N)

- checking of behavioral properties
  - e.g. testing the executability of an operation by testing the precondition or the guard of an operation

- checking for weakening or strengthening model properties (invariants, contracts, guards) by executing a scenario with modified constraints

- proving of general properties within a finite search space with the USE model validator
  - structural consistency, i.e. all classes are instantiable
  - behavioral consistency, i.e. all operations can be executed
  - checking for deadlocks, e.g. construction of deadlock scenarios due to inadequate buffer capacities

general model architecture
- integrated description of structure and behavior including properties
- construction of a system infrastructure (machines, trays) being later populated dynamically with temporal items (heads, handles, hammers)

example extension: consider further possible system properties
- coordinates of robots, i.e. machines and trays
- processing times of machines