Recent Trends in OO Modelling Languages

JML, rCOS, CREDO

Bernhard K. Aichernig

Institute for Software Technology
Graz University of Technology
Graz, Austria

United Nations University
International Institute for Software Technology
Macao S.A.R. China

Overture 2006

Outline

1. JML
2. rCOS
3. CREDO
4. Questions

Java Modelling Language (JML)

- inspired by VDM + Larch + ...
- Design by Contract: abstraction level lower than VDM++
- but not that low:
  - model fields (e.g. to specify interfaces)
  - refinement and retrieve functions
  - sets, sequences and maps for modelling
- Undefinedness: all function applications denote
  - Original motivation: theorem provers need this semantics
  - ICFEM 2006: G. Leavens reconsiders this (Patrice Chaline).
- VDM's notation is more elegant!

Alarm System

Classes	Methods
AlarmSystem	expertToPage
Qualification	expertIsOnDuty
Alarm	numberOfExperts
Period	
Expert	
(Description)	

Declarations of the Methods

R6:

```
public int numberOfExperts(Period p);
```

R7:

```
public Period[] expertIsOnDuty(Expert e);
```

R8:

```
public Expert expertToPage(Alarm a, Period p);
```

Alarm System: Schedule

R1 implies a kind of schedule:

```
/*@ public model instance JMLObjectToObjectRelation schedule;
  @ public invariant
  @         (\forall JMLType dv, rv; schedule.has(dv,rv);
  @                                  dv instanceof Period &&
  @                                  rv instanceof Expert);
  @*/
```

Alarm System: Alarms

```
/*@ public model instance JMLValueSet alarms;
  @ public invariant
  @        (\forall JMLType e; schedule.has(e);
  @                          e instanceof Alarm);
  @*/
```

Alarm System: Invariant

At any period experts need to be on duty who can cope with any possible alarm!

```
/*@ public invariant
  @   (\forall Alarm a; alarms.has(a);
  @      (\forall JMLType per; schedule.isDefinedAt(per);
  @         qualificationOk(schedule.elementImage(per),
  @                                  a.needed)));
  @*/
```

Auxiliary Function

Auxiliary function to make the invariant more readable:

```
/*@
   public model pure boolean
          qualificationOk(JMLObjectSet exs, Qualification q)
   {
     return
       (\exists Expert ex; exs.has(ex);
          (\exists int i; 0 <= i && i < ex.quali.length;
                   ex.quali[i] == (q)));
   }
  @*/
```

but, JML does not allow this.

Auxiliary Function

Auxiliary function to make the invariant more readable:

```
/*@
   public model pure boolean
          qualificationOk(JMLObjectSet exs, Qualification q)
   {
     return
       (\exists Expert ex; exs.has(ex);
          (\exists int i; 0 <= i && i < ex.quali.length;
                   ex.quali[i] == (q)));
   }
  @*/
```

but, JML does not allow this.

Corrected Auxiliary Function

playing a trick:

```
/*@ ensures (\exists Expert ex; exs.has(ex);
  @                (\exists int i; 0 <= i && i < ex.quali.length;
  @                    ex.quali[i] == q));
   public model pure boolean
          qualificationOk(JMLObjectSet exs, Qualification q)
   {
     return true;
   }
  @*/
```

Method 1

```
/*@ requires schedule.isDefinedAt(p);
  @ ensures \result == schedule.elementImage(p).size();
  @*/
 public int numberOfExperts(Period p);
```

Method 2

Calculating the result using a SetComprehension:

```
/*@
  @ ensures \result ==
  @   (new JMLObjectSet
  @         {Period p | schedule.domain().has(p) &&
  @                     schedule.has(p,e)}).toArray();
  @*/
 public Period[] expertIsOnDuty(Expert e);
```

Method 3

Implicit result specification:

```
/*@ requires schedule.isDefinedAt(p) && alarms.has(a);
  @ ensures
  @   schedule.has(p,\result) &&
  @   (\exists int i; i >= 0 && i < \result.quali.length;
  @       \result.quali[i] == a.needed);
  @*/
 public Expert expertToPage(Alarm a, Period p);
```

rCOS: Refinement of Component and Object Systems

- UNU-IIST: Liu Zhiming, He Jifeng et al.
- based on UTP: Hoare and He's Unifying Theories of Programming (as Woodcock's Circus)
- Methods are modelled as (guarded) UTP designs:
  - execution is relation btw. states of a program.
  - provides refinement calculus in relational or predicate transformer semantics (weakest-precondition)
  - reactive designs add a Boolean variable wait
  - guarded designs set wait true if guard is false
- Interfaces: collection of features (field and method declarations)
- Contract: Interface, Init, Methods, Protocol
- Undefinedness: not explicitly handled

UTP: Theory of Designs

Designs

Let p and Q be predicates not containing ok or ok' and p having only undecorated variables.

$$p \vdash Q \quad =_{df} \quad (ok \wedge p) \; \Rightarrow \; (ok' \wedge Q)$$

A design is a relation whose predicate is (or could be) expressed in this form.

Refinement

Correctness is defined via implication:

$$\forall v, w, \cdots \in A \bullet P \; \Rightarrow \; S, \qquad \text{for all } P \text{ with alphabet } A.$$

we write $[P \; \Rightarrow \; S]$ or $S \sqsubseteq P$

UTP: Theory of Designs

Designs

Let p and Q be predicates not containing ok or ok' and p having only undecorated variables.

$$p \vdash Q \quad =_{df} \quad (ok \wedge p) \;\Rightarrow\; (ok' \wedge Q)$$

A design is a relation whose predicate is (or could be) expressed in this form.

Refinement

Correctness is defined via implication:

$$\forall v, w, \dots \in A \bullet \; P \;\Rightarrow\; S\,, \qquad \text{for all } P \text{ with alphabet } A.$$

we write $[P \;\Rightarrow\; S]$ or $S \;\sqsubseteq\; P$

Refinement of Pre-Postconditions

Theorem

$[(p_1 \vdash Q_1) \Rightarrow (p_2 \vdash Q_2)]$ **iff** $[p_2 \Rightarrow p_1]$ and $[(p_2 \wedge Q_1) \Rightarrow Q_2]$

Like in VDM: preconditions are weakened and postconditions are strengthened.

CREDO = REO + Creol

- FW6 project: CWI, Oslo, UNU-IIST, Uppsala et al.
- modelling and simulation of evolving component networks
- Components
  - Collections of Creol Classes (Oslo)
  - Type-safe runtime class upgrades
  - Interfaces
  - Operational Semantics in Maude
- Glue
  - REO networks: a calculus of mobile channels (CWI)
  - Glue will be a special component
- Undefinedness in Maude: via sorts extended with undefined terms (kinds) and conditional membership (no logic)

CREDO = REO + Creol

- FW6 project: CWI, Oslo, UNU-IIST, Uppsala et al.
- modelling and simulation of evolving component networks
- Components
  - Collections of Creol Classes (Oslo)
  - Type-safe runtime class upgrades
  - Interfaces
  - Operational Semantics in Maude
- Glue
  - REO networks: a calculus of mobile channels (CWI)
  - Glue will be a special component
- Undefinedness in Maude: via sorts extended with undefined terms (kinds) and conditional membership (no logic)

CREDO = REO + Creol

- FW6 project: CWI, Oslo, UNU-IIST, Uppsala et al.
- modelling and simulation of evolving component networks
- Components
  - Collections of Creol Classes (Oslo)
  - Type-safe runtime class upgrades
  - Interfaces
  - Operational Semantics in Maude
- Glue
  - REO networks: a calculus of mobile channels (CWI)
  - Glue will be a special component
- Undefinedness in Maude: via sorts extended with undefined terms (kinds) and conditional membership (no logic)

CREDO = REO + Creol

- FW6 project: CWI, Oslo, UNU-IIST, Uppsala et al.
- modelling and simulation of evolving component networks
- Components
  - Collections of Creol Classes (Oslo)
  - Type-safe runtime class upgrades
  - Interfaces
  - Operational Semantics in Maude
- Glue
  - REO networks: a calculus of mobile channels (CWI)
  - Glue will be a special component
- Undefinedness in Maude: via sorts extended with undefined terms (kinds) and conditional membership (no logic)

CREDO: REO

- by Farhad Arbab et al., CWI
- paradigm for composition of software components based on mobile channels
- motivation: compositional construction of glue code
- dogma: exogenious coordination (coordination from outside) like in dataflow models (e.g. Unix pipes).

CREDO: REO Connectors

- atomic connector: channels
  - two directed ends: source and sink (read and write data)
  - identity, dynamic creation, mobile
  - channel types: synchronous, asynchronous (buffered), lossy, fifo, set, etc.
- connector: set of channels organised in a graph: nodes are channel ends, edges are channels
  - source, sink and mixed nodes.
  - node operations: read from sink nodes, write to source nodes, move node to new location, hide a node, etc.

Questions

- What kind of challenges are JML, rCOS, CREDO to VDM?
- How to react to these challenges?
- Generating VDM to Java + JML?
- VDM++ semantics in rCOS (UTP)
- VDM++ interpreter in Maude?
- Adding REO-connectors to VDM++

Questions

- What kind of challenges are JML, rCOS, CREDO to VDM?
- How to react to these challenges?
- Generating VDM to Java + JML?
- VDM++ semantics in rCOS (UTP)
- VDM++ interpreter in Maude?
- Adding REO-connectors to VDM++

Questions

- What kind of challenges are JML, rCOS, CREDO to VDM?
- How to react to these challenges?
- Generating VDM to Java + JML?
- VDM++ semantics in rCOS (UTP)
- VDM++ interpreter in Maude?
- Adding REO-connectors to VDM++

Questions

- What kind of challenges are JML, rCOS, CREDO to VDM?
- How to react to these challenges?
- Generating VDM to Java + JML?
- VDM++ semantics in rCOS (UTP)
- VDM++ interpreter in Maude?
- Adding REO-connectors to VDM++

Questions

- What kind of challenges are JML, rCOS, CREDO to VDM?
- How to react to these challenges?
- Generating VDM to Java + JML?
- VDM++ semantics in rCOS (UTP)
- VDM++ interpreter in Maude?
- Adding REO-connectors to VDM++

Questions

- What kind of challenges are JML, rCOS, CREDO to VDM?
- How to react to these challenges?
- Generating VDM to Java + JML?
- VDM++ semantics in rCOS (UTP)
- VDM++ interpreter in Maude?
- Adding REO-connectors to VDM++