Temporal OCL Extensions for Specification of Real-Time Constraints

Stephan Flake

SVERTS Workshop at UML 2003, San Francisco, USA, October 20, 2003
Overview

- Motivation
- Overview of Temporal OCL Extensions
- State-oriented Real-time OCL Extension
- Conclusion
Motivation

Formal Verification with Real-Time Model Checking

Given

• a model $M$
  (by a time-annotated Kripke structure)

• a desired property $\varphi$
  (by a time-annotated temporal logic formula)

Does $M$ satisfy $\varphi$?
Specification Environment

Model
(part of) UML
Extended FSMs

Specification
Properties
Time-Bounded Temporal Logics

Results
Visualization
Configuration Sequences

Real-Time Model Checking Tool

Motivation
Temporal OCL Extensions
State-based OCL Extension
Conclusion

© Siemens and Paderborn University 2003
Specification Environment

Model
(part of)
UML
Extended FSMs

Specification
Properties
Time-Bounded Temporal Logics

Results
Visualization
Configuration Sequences

Real-Time Model Checking Tool

Motivation
Temporal OCL Extensions
State-based OCL Extension
Conclusion

© Siemens and Paderborn University 2003
Motivation

Problem Domain

Formulation of system properties w.r.t. dynamic, time-bounded behavior

Approaches

- Temporal logics formulae
- Natural language
- Diagrams
- Programming language oriented

Support in the UML?
Support in the UML?

UML Extension Mechanisms, esp. RT Profile with Timing Mechanisms like:

- Timers
- Clocks
- TimedEvent
- Timeout ...

Still no possibility to specify state-related properties (e.g. liveness, safety properties) w.r.t. dynamic, time-bounded behavior!
## Temporal OCL Extensions - Overview

<table>
<thead>
<tr>
<th>Approach</th>
<th>Syntax</th>
<th>Formal Semantics</th>
<th>Real-Time</th>
</tr>
</thead>
<tbody>
<tr>
<td>Ramakrishnan et al.</td>
<td>OCL + future oriented temporal operators</td>
<td></td>
<td>-</td>
</tr>
<tr>
<td>Conrad/Turowski</td>
<td>OCL + temporal operators</td>
<td></td>
<td>-</td>
</tr>
<tr>
<td>Kleppe/Warmer</td>
<td>OCL + action clause</td>
<td></td>
<td>-</td>
</tr>
<tr>
<td>Bradfield et al.</td>
<td>OCL + temporal templates</td>
<td>observational mu-calculus</td>
<td>-</td>
</tr>
<tr>
<td>Distefano et al.</td>
<td>similar CTL</td>
<td>BOTL</td>
<td>-</td>
</tr>
<tr>
<td>Roubtsova et al.</td>
<td>parameterized classes (stereotypes)</td>
<td>TCTL</td>
<td>✓</td>
</tr>
<tr>
<td>Sendall/Strohmeier</td>
<td>OCL-consistent</td>
<td></td>
<td>✓</td>
</tr>
<tr>
<td>Cengarle/Knapp</td>
<td>OCL + temporal operators</td>
<td>Trace semantics</td>
<td>✓</td>
</tr>
</tbody>
</table>

**Motivation**

- Temporal OCL Extensions
  - State-based OCL Extension
  - Conclusion
Manufacturing Case Study

Motivation

Temporal OCL Extensions

State-based OCL Extension

Conclusion
UML Class Diagram

Motivation

Temporal OCL Extensions

State-based OCL Extension

Conclusion

**FactoryUnit**
- name : String
- pos : Position
- notify()

**Item**
- id : Integer
- status : String

**Station**
- processed_Items : Integer
- getWorkLoad() : Integer

**Buffer**
- stored_Items : Integer
- load(i:Item) : Boolean
- unload(i:Item) : Boolean

**AGV**
- batteryCharge : Integer
- load(i:Item) : Boolean
- unload(i:Item) : Boolean
- move(s:Station)
- getDistance(f:FactoryUnit)

**Machine**
- type : MachineKind
- load(i:Item) : Boolean
- unload(i:Item) : Boolean

**InputBuffer**
- announced : Boolean
- announceOrder(i:Item)
- acceptOrder()
- rejectOrder()
- deliver(i:Item)

**OutputBuffer**
UML Statechart

- **Acceptor** negotiates orders

- **Loader** observes actual loading

```uml
actor InputBuffer

state Acceptor
    * WaitingForOrder
        * announceOrder(i:Item) [not self.oclInState(Loader::Idle)]

state Loader
    * Idle
        * loadComplete(i:Item)
            - [self.announced = true]

    * Loading
        * do / load(i:Item)

    * WaitingForDelivery
        * deliver(i:Item)
            - load() in [20, 40]

    * Rejecting
        * entry / rejectOrder()

    * Accepting
        * entry / acceptOrder()
```

**Motivation**

- Temporal OCL Extensions
- State-based OCL Extension

**Conclusion**

© Siemens and Paderborn University 2003
OCL Extension + Statecharts

Informal Natural Language

Items must always arrive at the input buffer within timing intervals of at most 400 time units.

State-oriented real-time OCL

context InputBuffer

inv: InputBuffer::Loader@post(1,400) ->forall( p:OclPath | p->includes(Loading) )
OCL Extension + Statecharts (cont.)

Informal Natural Language

The input buffer must not accept a new order while waiting for loading a previously accepted order.

Extended OCL

```ocl
context InputBuffer
inv: let errorCfg = Set {Acceptor::Accepting, Loader::WaitingForDelivery}
in
  self@post() ->forall (p:OclPath | p->excludes(errorCfg))
```
New OCL Types and Operations

OclState

• currently, only one operation on states:
  self.oclInState(stateName)

• introduction of new operations
  compliant with UML StateMachine metamodel,
  e.g., self.oclInCfg(cfg:Set(State))

OclConfiguration

• similar to Set(OclState)

• necessary because of concurrent substates

• usual OCL operations for sets
New OCL Types and Operations (cont.)

OclPath

- similar to `Sequence(OclConfiguration)`
- possible future execution paths
- usual OCL operations for sequences

OclAny

- `@pre` OCL postcondition operator becomes an operation
- `@post(a,b)` returns the set of possible future execution paths
UML Profile for OCL Extension

- **PropertyCallExp**
  - **appliedProperty**: 0..1

- **ModelPropertyCallExp**
  - **source**: 0..1
  - **arguments**: 0..n {ordered}

- **OperationCallExp**
  - **referredOperation**: 1

- **TemporalExp**
  - **FutureTemporalExp**
  - **PastTemporalExp**

**OclExpression**

- **Operation** *(from Core)*

**Motivation**
**Temporal OCL Extensions**
**State-based OCL Extension**
**Conclusion**
Concrete Syntax

FutureTemporalExpCS ::= OclExpressionCS ’@’ simpleNameCS ’(’ argumentsCS?’ )’

Abstract Syntax Mapping:
FutureTemporalExpCS.ast : FutureTemporalExp

Synthesized Attributes:
FutureTemporalExpCS.ast.source = OclExpressionCS.ast
FutureTemporalExpCS.ast.arguments = argumentsCS.ast
FutureTemporalExpCS.ast.referredOperation = …

Inherited Attributes:
OclExpressionCS.env = FutureTemporalExpCS.env
argumentsCS.env = FutureTemporalExpCS.env

Disambiguating Rules:
[1] Set{’post’}->includes(simpleNameCS.ast)
Formal Semantics: Mapping to Temporal Logics

**CCTL (Clocked Computation Tree Logic)**
- future-oriented, branching-time
- time-annotated temporal operators
- real-time Model Checking (RAVEN)

OCL constraints with @post operations can directly be mapped to CCTL formulae.
Temporal OCL Extensions

Current Use:

• "Pure" Modeling (e.g., Protocol Specification)
• (Real-Time) Model Checking

Potential Future Usage:

• Other diagrams, e.g. Activity Diagrams
• UML Profiles based on the OCL 2.0 Metamodel
• UML Profile for Scheduling, Performance, and Time
Conclusion

• Already a number of temporal OCL extensions with different focus, e.g.,
  • future vs. past time
  • event vs. state-oriented

• Our OCL extension
  • provides means for real-time constraints w.r.t. state-oriented future execution paths,
  • completely retains current OCL syntax,
  • has a formal semantics by a mapping to temporal logics.
More Details:

Thank you very much for your attention!