Cooperative Computing &
Communication Laboratory

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 *j*
  (by a time-annotated temporal logic formula)

Does *M* satisfy *j* ?

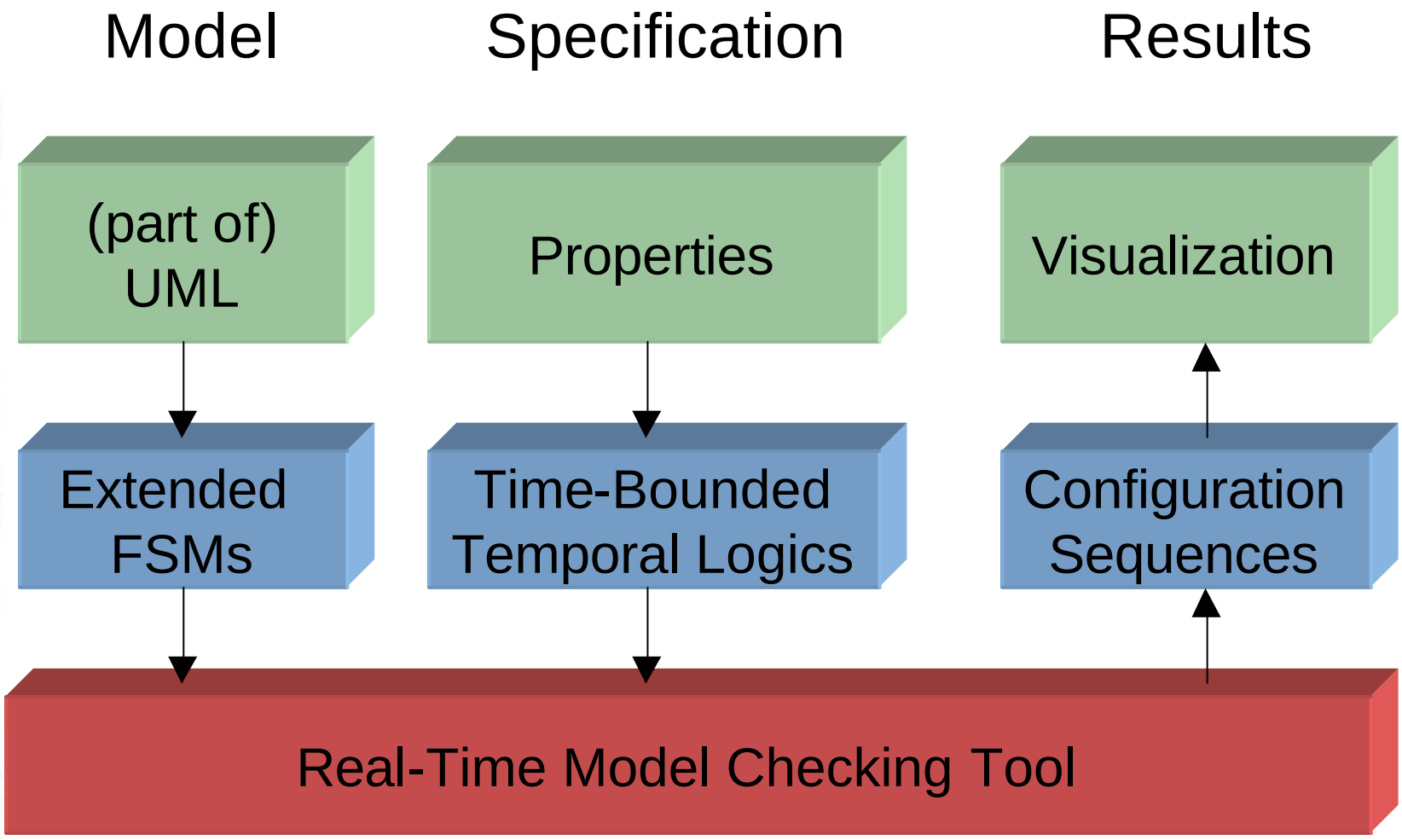
Specification Environment

Motivation

Temporal OCL Extensions

State-based OCL Extension

Conclusion



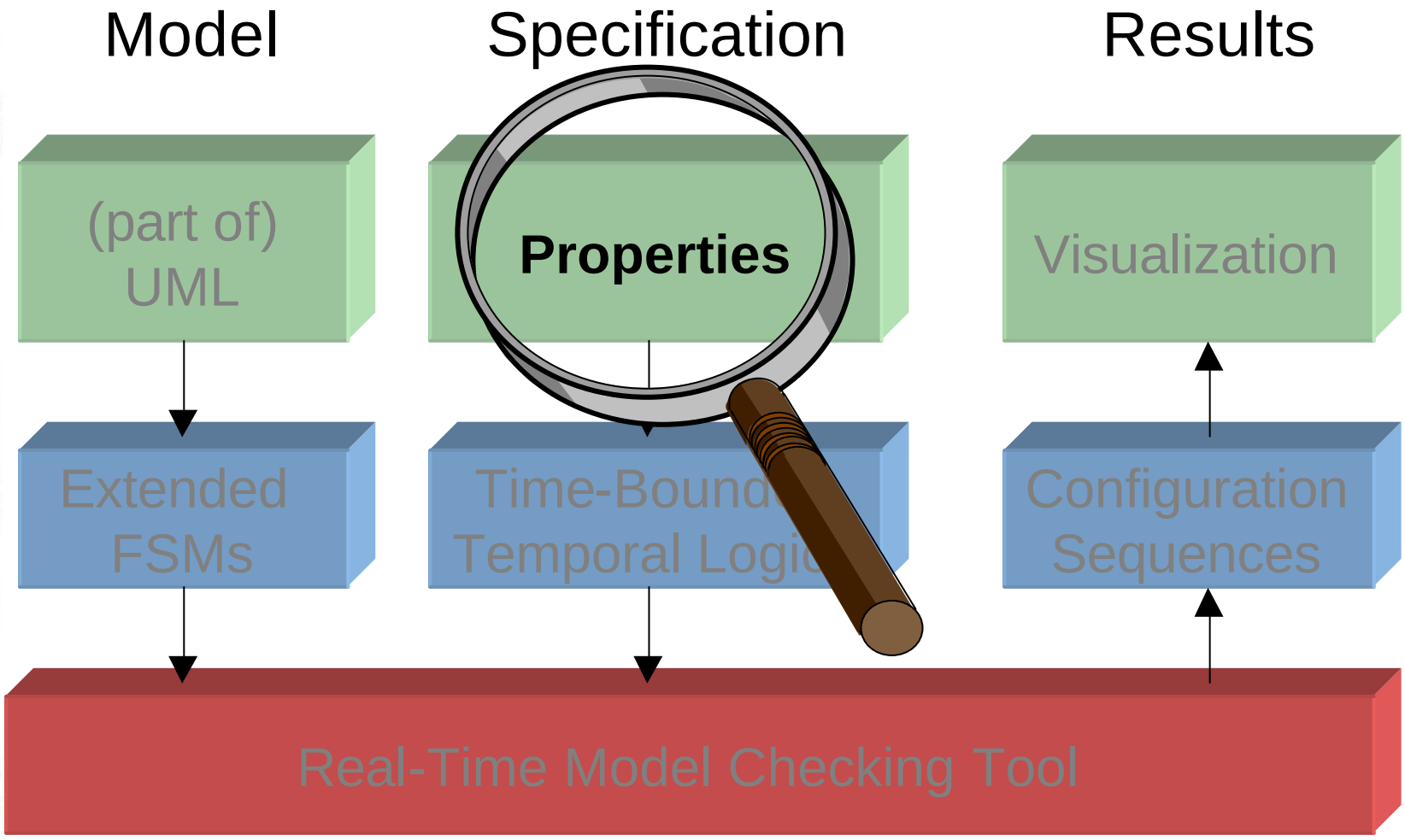
Specification Environment

Motivation

Temporal OCL Extensions

State-based OCL Extension

Conclusion



Motivation

Problem Domain

Formulation of system properties

w.r.t. dynamic, time-bounded behavior

Approaches

- Temporal logics form
- Natural languag
- Diagrams
- Progra g 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

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

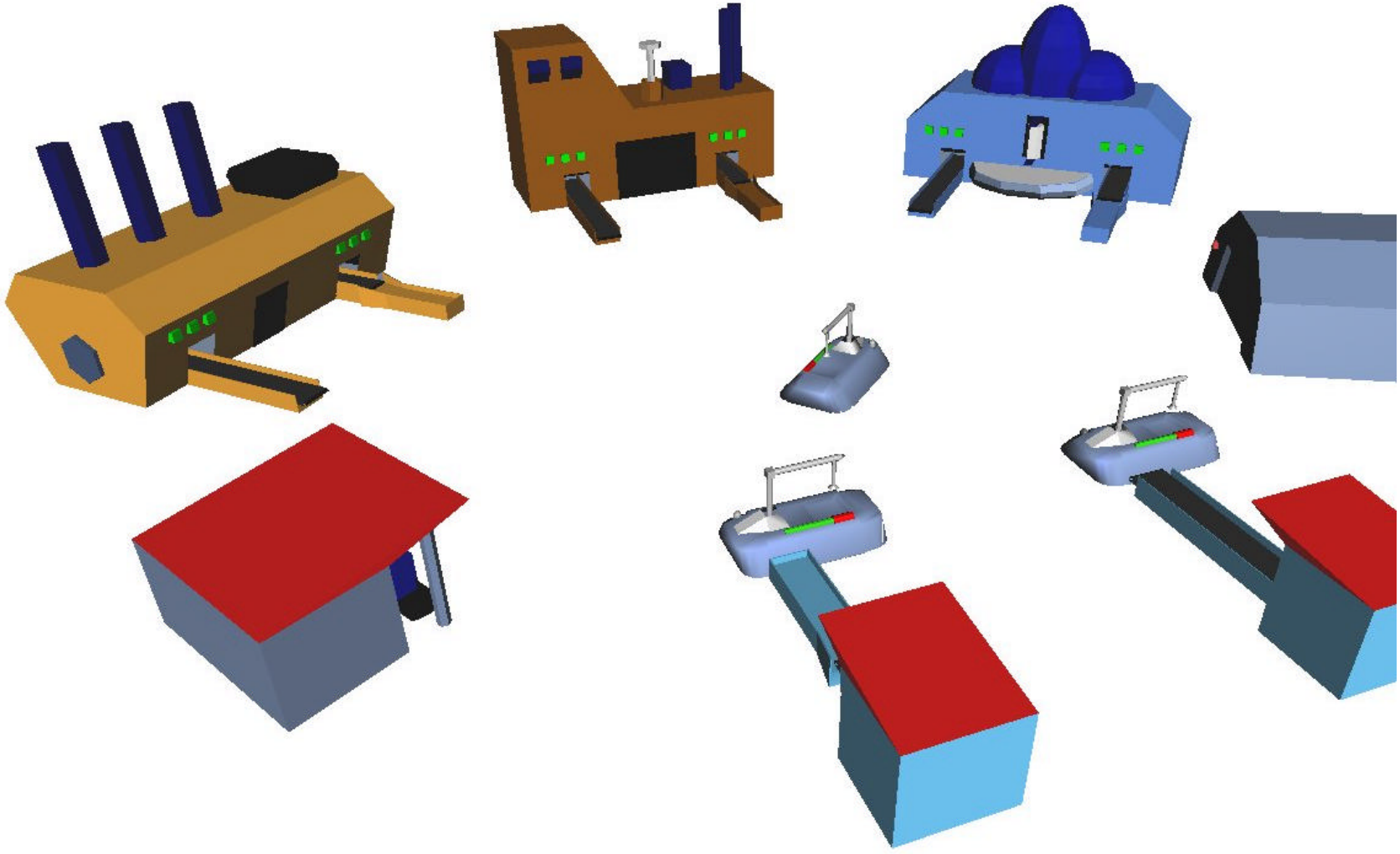
Manufacturing Case Study

Motivation

Temporal OCL Extensions

State-based OCL Extension

Conclusion



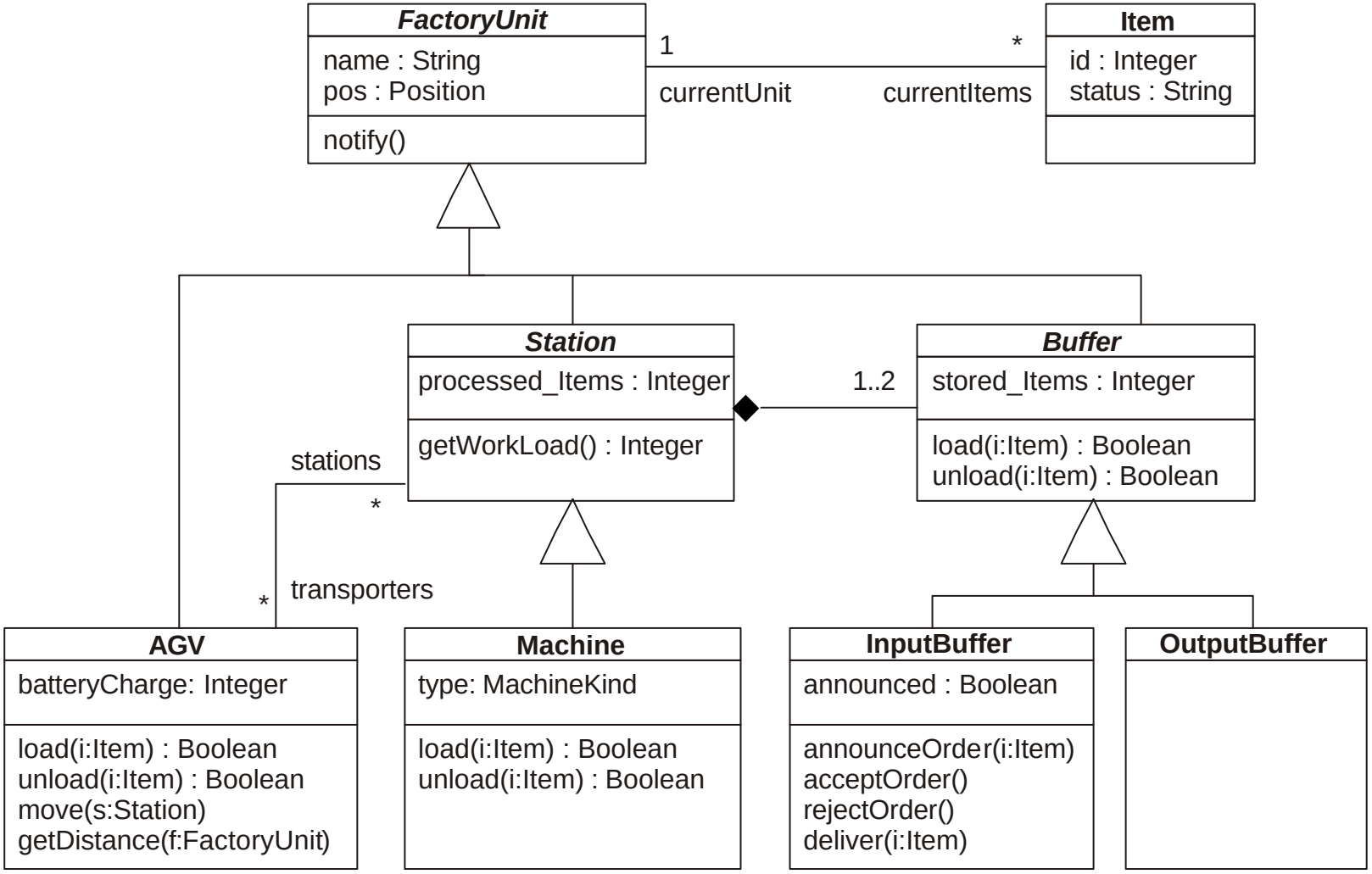
UML Class Diagram

Motivation

Temporal OCL Extensions

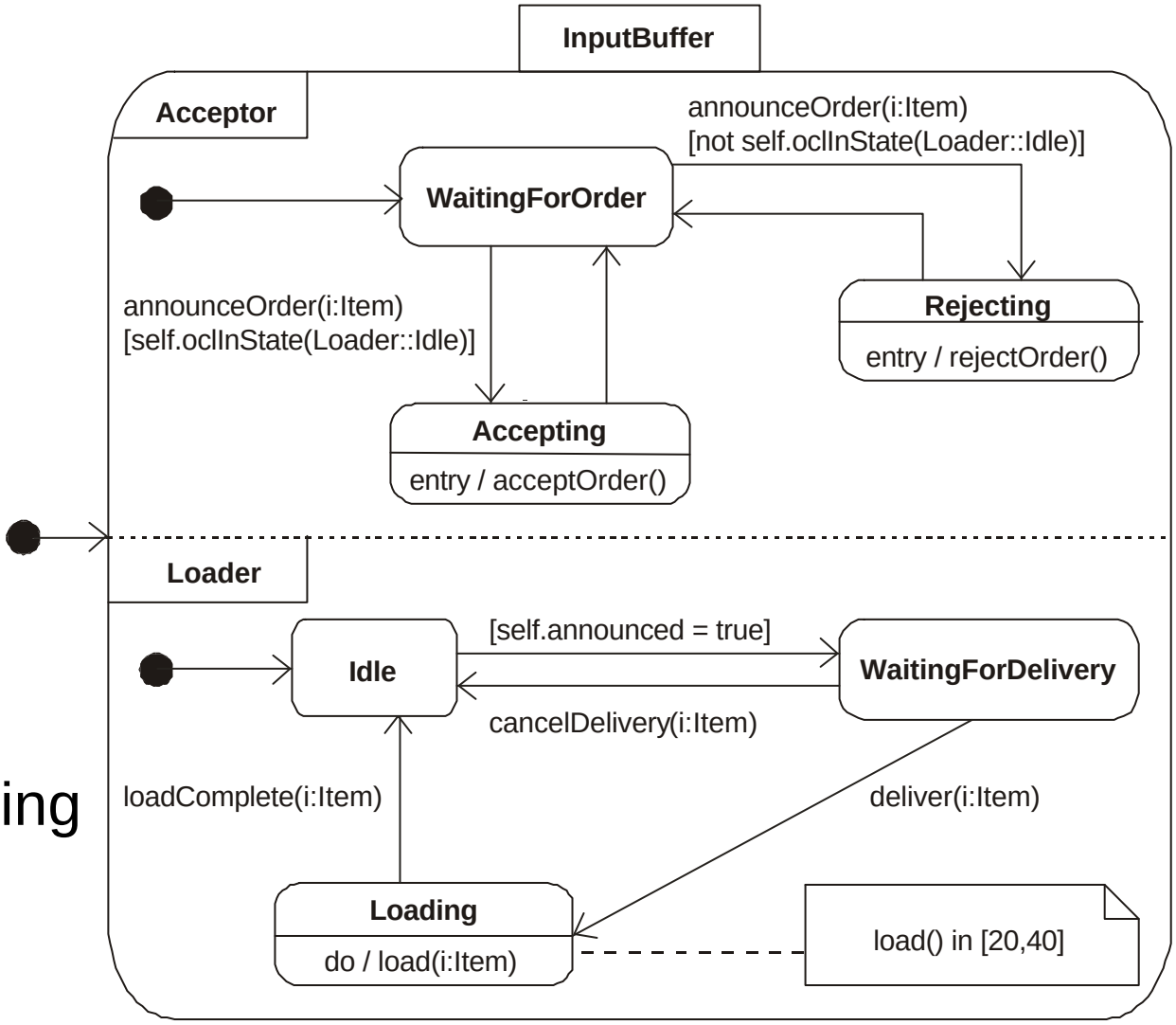
State-based OCL Extension

Conclusion



UML Statechart

- **Acceptor** negotiates orders
- **Loader** observes actual loading



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

```
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.)

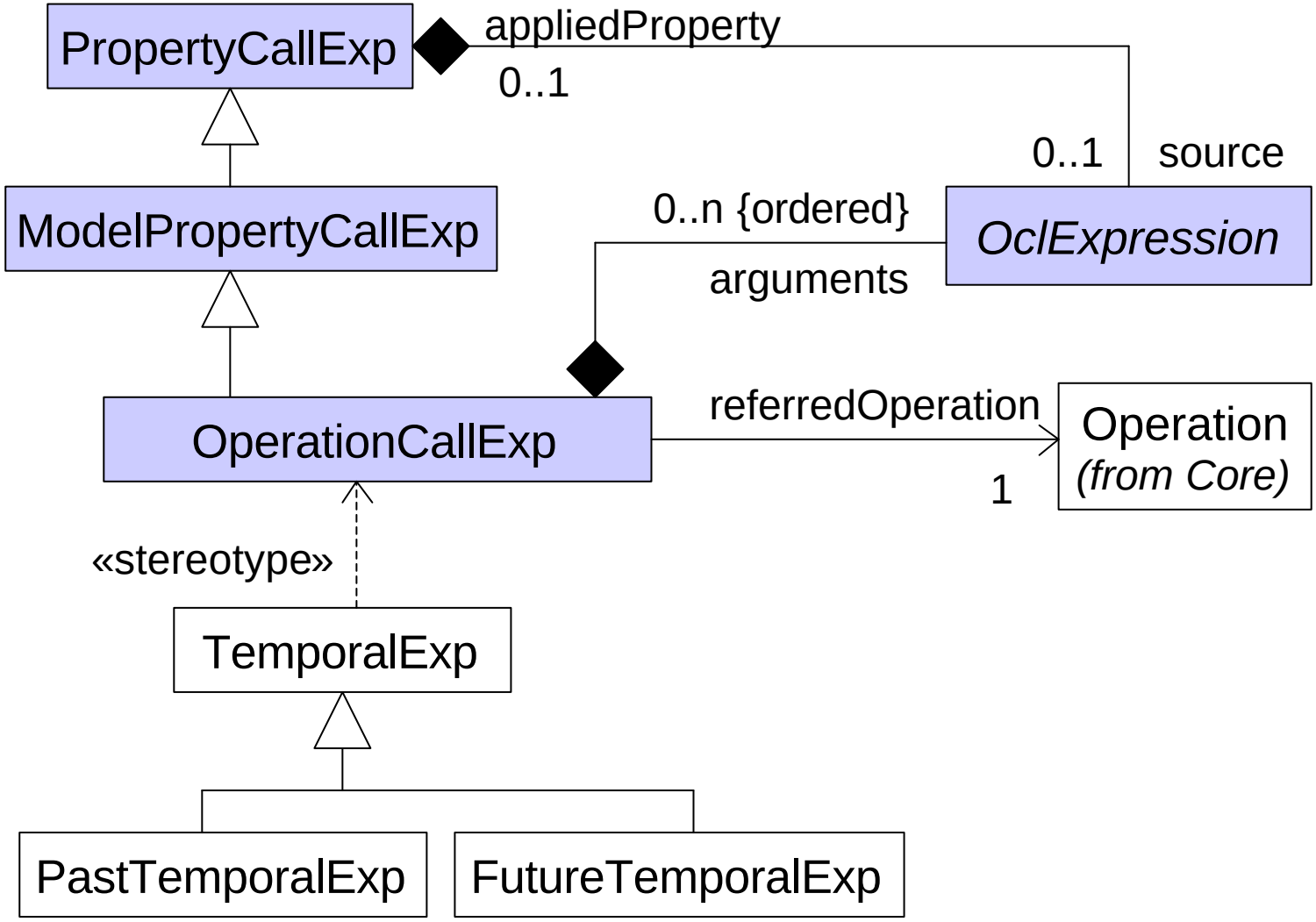
OclPath

- 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



Concrete Syntax

Motivation

Temporal OCL Extensions

State-based OCL Extension

Conclusion

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)
[2] not FutureTemporalExpCS.ast.referredOperation.oclIsUndefined()

Formal Semantics: Mapping to Temporal Logics

CCTL (**C**locked **C**omputation **T**ree **L**ogic)

- 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

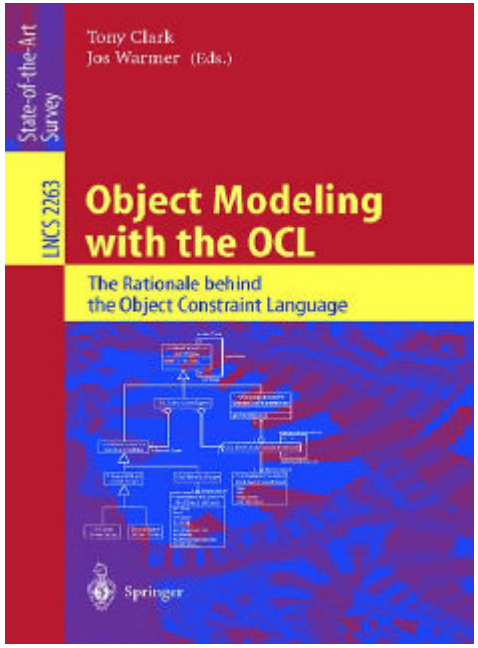
Potential Future Usage:

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

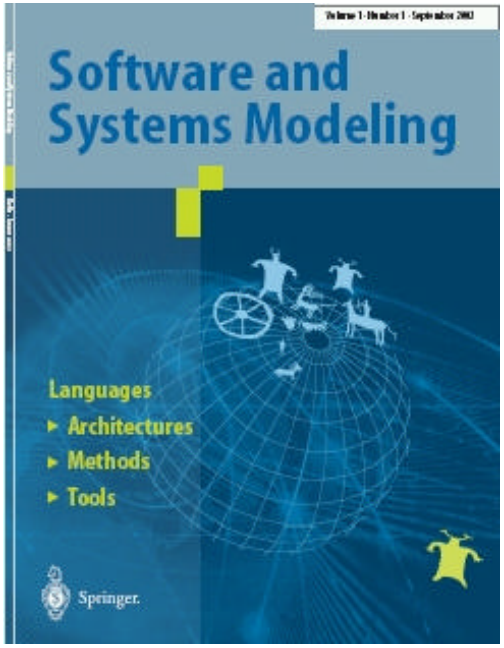
Conclusion

- 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:



LNCS 2263



SoSyM 2(3)

Thank you very much for your attention!