On Real-Time Requirements in Specification-Level UML Models

Risto Pitkänen and Tommi Mikkonen
Tampere University of Technology
Finland
`{risto.pitkanen,tommi.mikkonen@tut.fi}`

Motivation for high-level modeling

- Abstraction is a powerful thinking tool
- High-level models
  - Better grasp of system as a whole
  - Early validation and verification
- Real-time systems should be no exception
  - Inherent behavioral complexity
- High-level facilities for expressing time-related issues largely missing from UML

Use cases

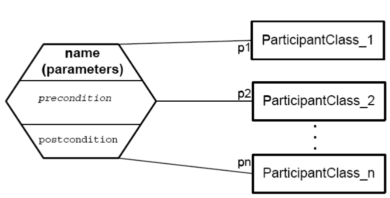
- Requirements/specification level modeling in UML
  - E.g. Unified Process
- Serious handicaps from the point of view of real-time specification:
  - Use case interaction is not supported
    - use case A cannot require that use case B has been executed
  - No explicit time constructs

Formalization of use cases

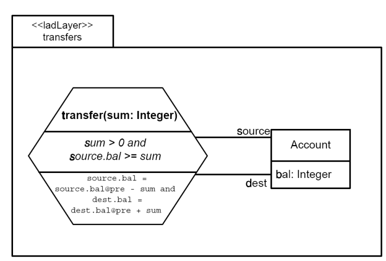
- Catalysis [D'Souza and Wills 1999]
  - Use case = *joint action*
  - Joint actions originate from *DisCo*

```
action assign_mentor(subject: Instructor,
                     watchdog: Instructor)
  post subject.mentor = watchdog and
   let ex_mentee = watchdog.mentee@pre in
     ex_mentee <> null ==> ex_mentee.mentor = null
```

Our approach: joint action



Layered Action Diagrams (LADs)



Properties of LADs

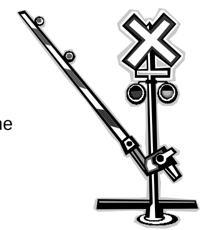
- Simple LADs are superposed onto each other to form a total system
  - Aspect-oriented structuring of a model
- Semi-executable:
  - Generate all joint action instances whose preconditions hold true in present state
  - Pick one, modify participant states such that postcondition becomes true

Extending LADs with real-time issues

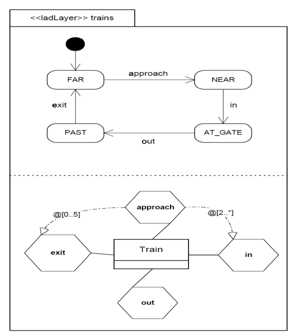
- LADs provide a suitable high-level modeling formalism also for real-time systems
- We want to be able to express real-time dependencies between joint actions
- Aim: easy and natural formalization of requirements such as *"When gas valve is opened, it must be closed within 5 seconds unless the burner is ignited successfully."*

Generalized Railroad Crossing (GRC)

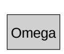
- System controls gate at a railroad crossing
- **Requirements:**
  - **Safety:** Whenever there is at least one train in the crossing, the gate must be down.
  - **Utility:** When there is no train in the crossing, the gate must be raised within a reasonable time.



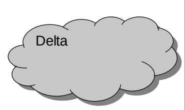
GRC with LADs: Trains



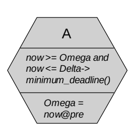
Real-time semantics



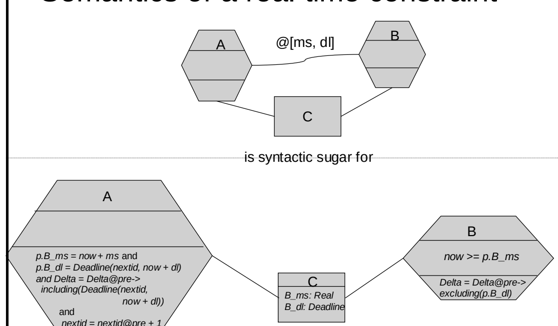
- Global clock *Omega*, typed *Real*, initialized to 0.0.
- Initially empty set *Delta* of deadlines.
  - Holds pairs *(DeadlineId, Real)*



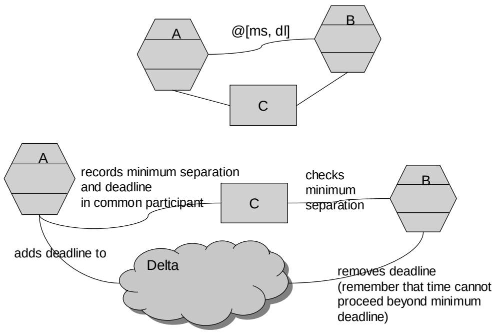
- In each joint action, an implicit parameter *now: Time*.
- Each joint action has an implicit precondition
  - *now >= Omega and now <= Delta->minimum_deadline()*,
- and an implicit postcondition
  - *Omega = now@pre*.
- In effect, this causes time to grow monotonically
  - never exceeding the minimum deadline currently in *Delta*



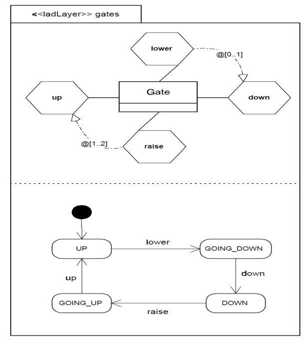
Semantics of a real-time constraint



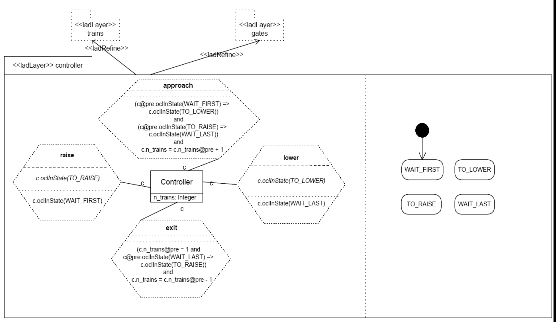
Another way of putting it



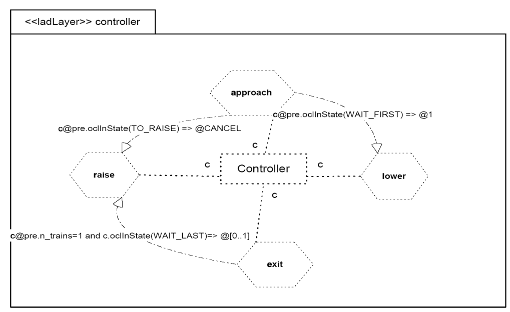
Gate



Controller



Controller real-time constraints



Validation and Verification

- Either theorem proving (using TLA) or model checking (timed automata) can be used for verification and validation
- An equivalent GRC DisCo model has been model checked using the Kronos tool after mapping it to timed automata
- Theorem proving applies to generic models, model checking currently only to specific instances (e.g. two trains)

Conclusions

- Successful application of earlier RT modeling results and techniques in a UML profile based setting
- New approach for expressing real-time requirements for formalized use cases (*joint actions)*
  - Real-time constraint associations
- Methodological and notational support for separating real-time issues from the underlying control logic