Specification and Validation of Safety-critical Interlocking Functionality using

EXECUTABLE UML

UIC

EURO-INTERLOCKING

Christian Bühler, KnowGravity Inc,
Badenerstrasse 808, 8048 Zürich, Switzerland
Tel. +41-44'43'42'000, Fax. +41-44'43'42'009

Contents

- The Context
  - The Customer
  - The Project
- CASSANDRA/xUML
- Some of CASSANDRA's advanced UML Modeling Features
  - Time Events
  - Model Instantiation
  - Behavior Inheritance
  - Concurrent State Regions
  - Change Transitions
  - Model Validation
- Demonstration

KnowGravity Inc.

Consulting

specific application of know-how
within our core competencies in guiding
customer projects

Training

training of know-how within our
core competencies

Doing

application of know-how within our core
competencies in realization projects

Know-how

Running

tools for automated
know how application

Brokering

mediation of know-how outside our
core competencies

Processes & Methods

Software engineering based on OO, SSADM or SA/SD,
Hermes, V-Modell, Software Process Improvement (SPI);
Software Ergonomics

Technology

Business Rules, Artificial Intelligence, COM, CORBA, EJB,
XML, Java, Prolog, C++, VB, Web-Technologies, CASE and
other development tools

The GENERIS Project

EURO-INTERLOCKING

The Euro-Interlocking Consortium

The **Euro-Interlocking Project** is an organisation of **17 railways** from across Europe, working together under the auspices of the **UIC (International Union of Railways)** and in close collaboration with the industry's **supplier organisations**, the **European Union** and the **ERTMS Users Group** in Brussels. The project is based in Swiss Federal Railway offices in **Zurich**.

The Project "Euro-Interlocking"

The project's primary aim is to reduce the life-cycle costs of future interlocking systems by promoting **harmonisation in the description of railway requirements for interlocking systems and by the standardization of interlocking interfaces**. The project also aims at improving the reliability and availability of future interlocking systems as well as promoting the international cross-acceptance of products.

Having completed its work on **qualitative requirements** for interlocking systems, the project has now turned its attention to **functional requirements**…

The Challenges

- Each railway has evolved different policies and practices for handling its functional requirements
  ⇨ Develop a common method for describing functional requirements
- Each railway tends to believe that its approach is the only possible solution, as its functionality is unique.
  ⇨ Careful introduction of new approaches
- Safety-critical systems require careful validation and formal verification
  ⇨ Leading edge techniques and technologies
- Project team distributed all over Europe
  ⇨ Reliance on web-based IT infrastructure for sharing information
- Completely different cultures of participating railways
  ⇨ Huge amount of social as well as political skills

The Approach

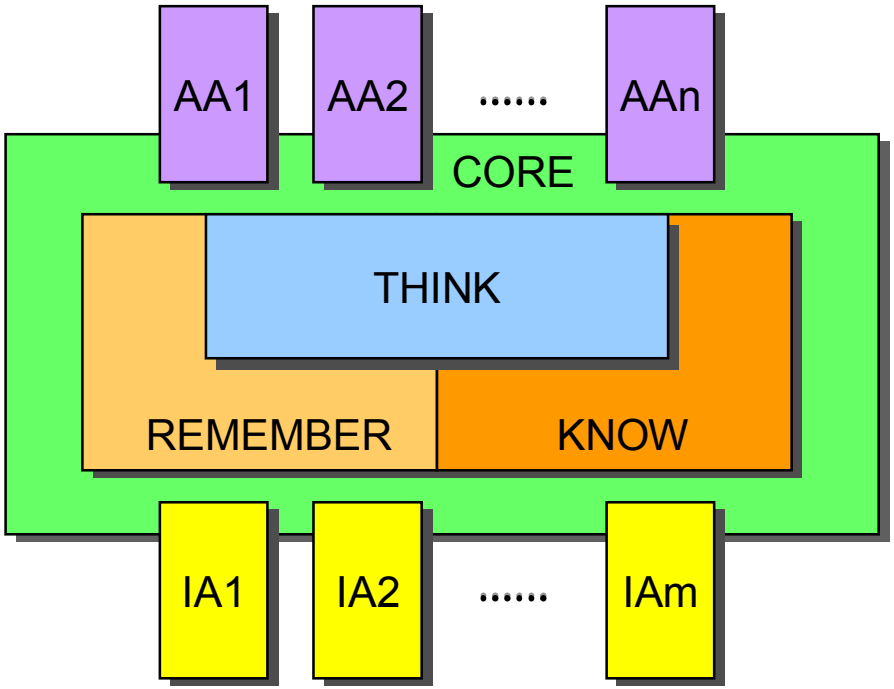
- To create an atomised database of written requirements for a given railway.
- To model these requirements in UML, taking care to show how and where each individual requirement is represented in the model.
- To simulate the model in order that the railway's domain experts can verify its functionality.
- Once this has been done, the requirements of a second railway can be added to the textual database by 'tagging' common requirements and adding new functionality as appropriate.
- Any differences can then be introduced to the model using separate classes, in order to retain a clear distinction in the model between the requirements of each railway.

Interlocking Models

- Micro Interlocking
  - Extremely simplified model to illustrate some key concepts
  - Specific model
  - For this presentation only
- Mini Interlocking
  - Initial feasibility model with typical track elements and GUI
  - Generic model with two national instantiations
  - Euro-Interlocking UML reference model
- GENERIS
  - The “real” Euro-Interlocking requirements model
  - Generic model currently with one national instantiation
  - Work in progress

CASSANDRA/xUML

CASSANDRA: A Platform for Advanced SE



- **CORE:** basic infrastructure (GUI, XML, persistency, licensing, etc.)
- **REMEMBER:** a UML-based declarative database
- **KNOW:** common sense in form of a „class model of the world“
- **THINK:** an inference engine to process know how
- **IAx:** A set of Interface Agents for various CASE tools
- **AAx:** A set of (hopefully useful) Application Agents

CASSANDRA-based Executable UML

- Based on KnowGravity's CASSANDRA platform
- Provides model simulation based on a UML Virtual Machine
  - Actors and use cases
  - Sequence diagrams and events
  - Classes, associations, and instances
  - State diagrams as multiple instances of communicating state machines
  - Simulation time
- Versatile functionality
  - Model extraction from CASE tool (ARTiSAN RtS)
  - Generic and user-specific simulation GUI
  - Black box and glass box observation
  - Regression testing

UML 2.0 Executable Language

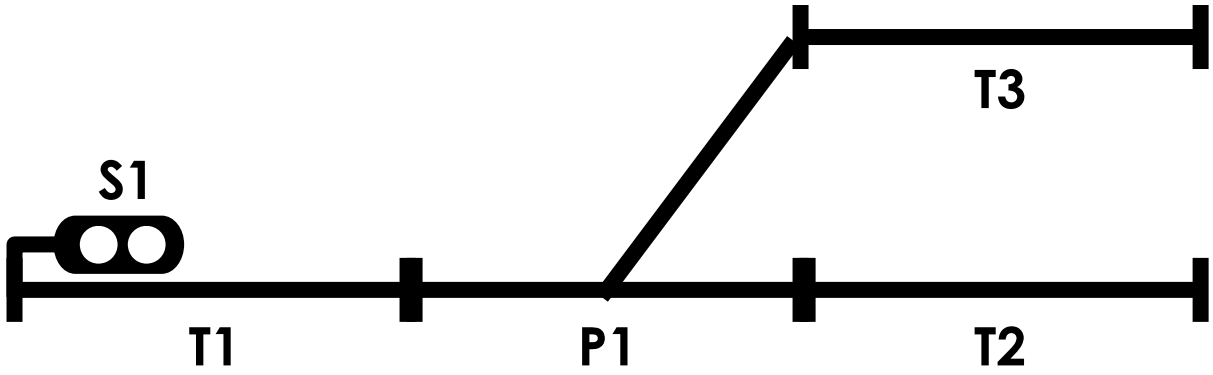
UML 2.0 based action language for transition actions

- Instance construction and destruction
- Attribute manipulation
- Link construction and destruction
- Association navigation
- Events for synchronous and asynchronous communication
- (Filtered) event broadcasts over associations
- Control structures

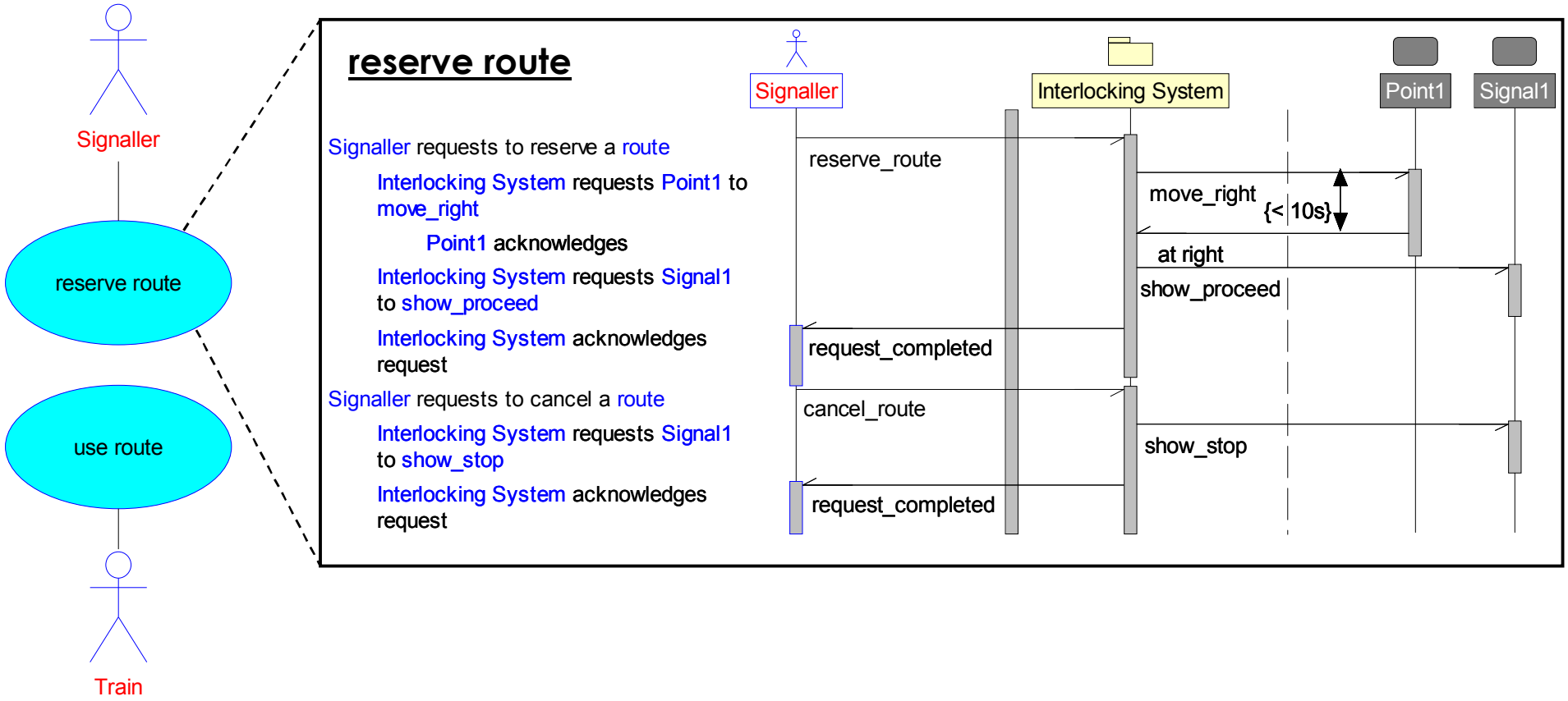
UML 2.0/OCL-based language for transition guards and actions

- Complex boolean, arithmetic, set and term expressions
- All and exist quantifiers
- Reflective and meta evaluation

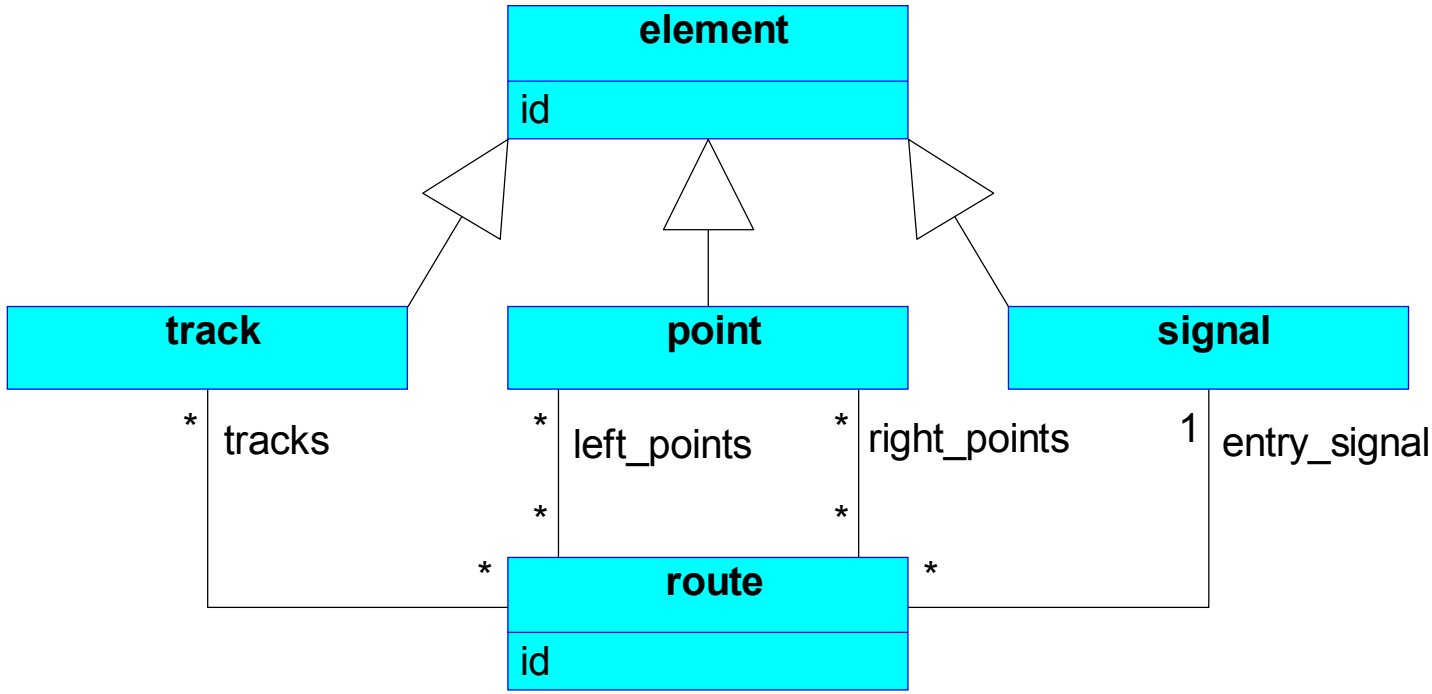
Some of CASSANDRA's advanced UML Modeling Features illustrated on Micro Interlocking



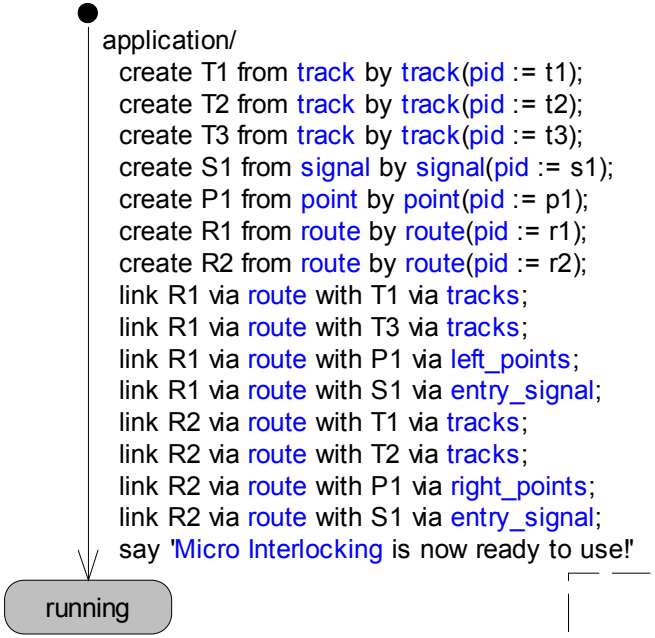
Micro Interlocking Use Cases

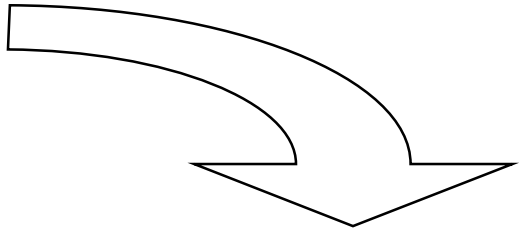


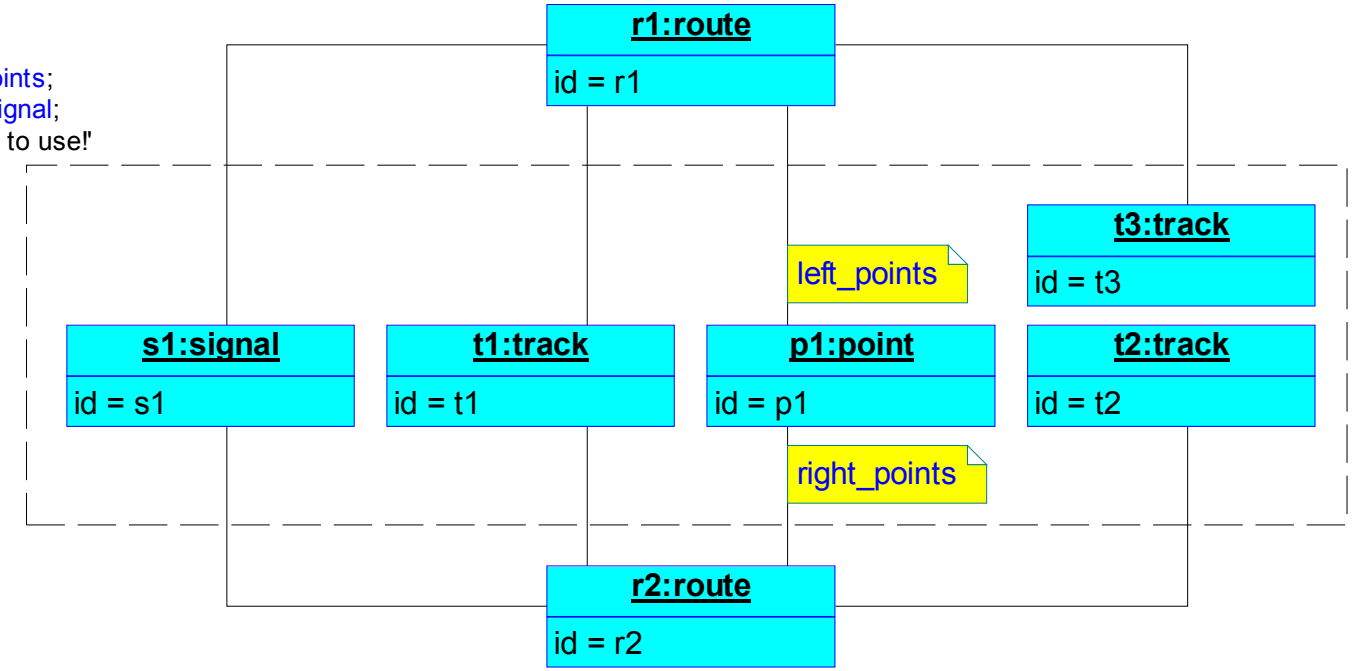
Micro Interlocking Domain Objects



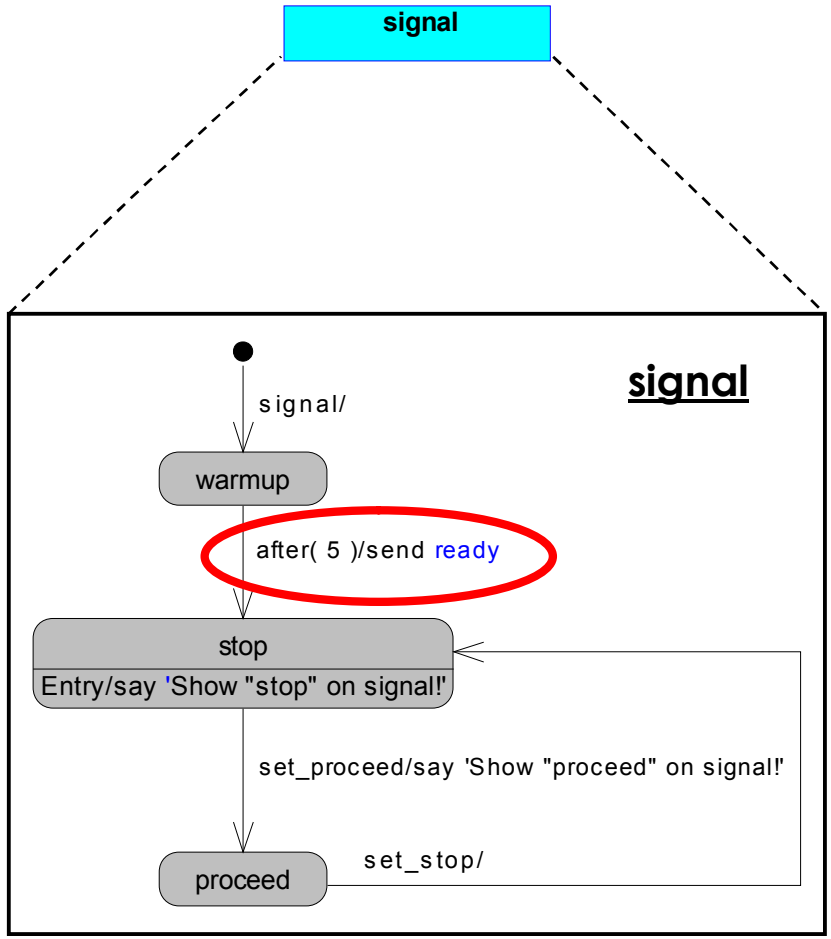
Model Instantiation







Time Events



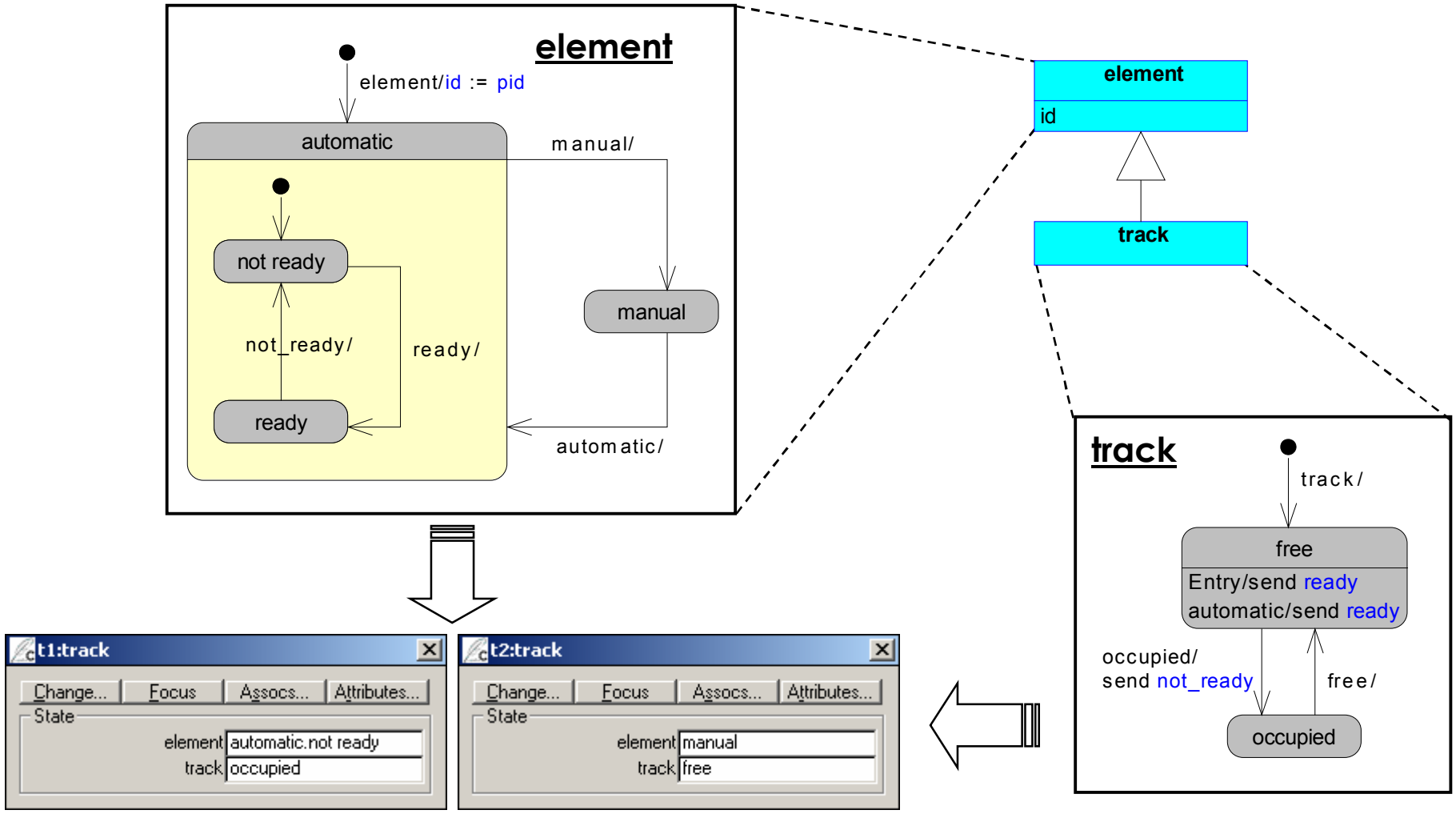
Time units are in simulation time:

- either user controllable units
- or real-time (seconds)

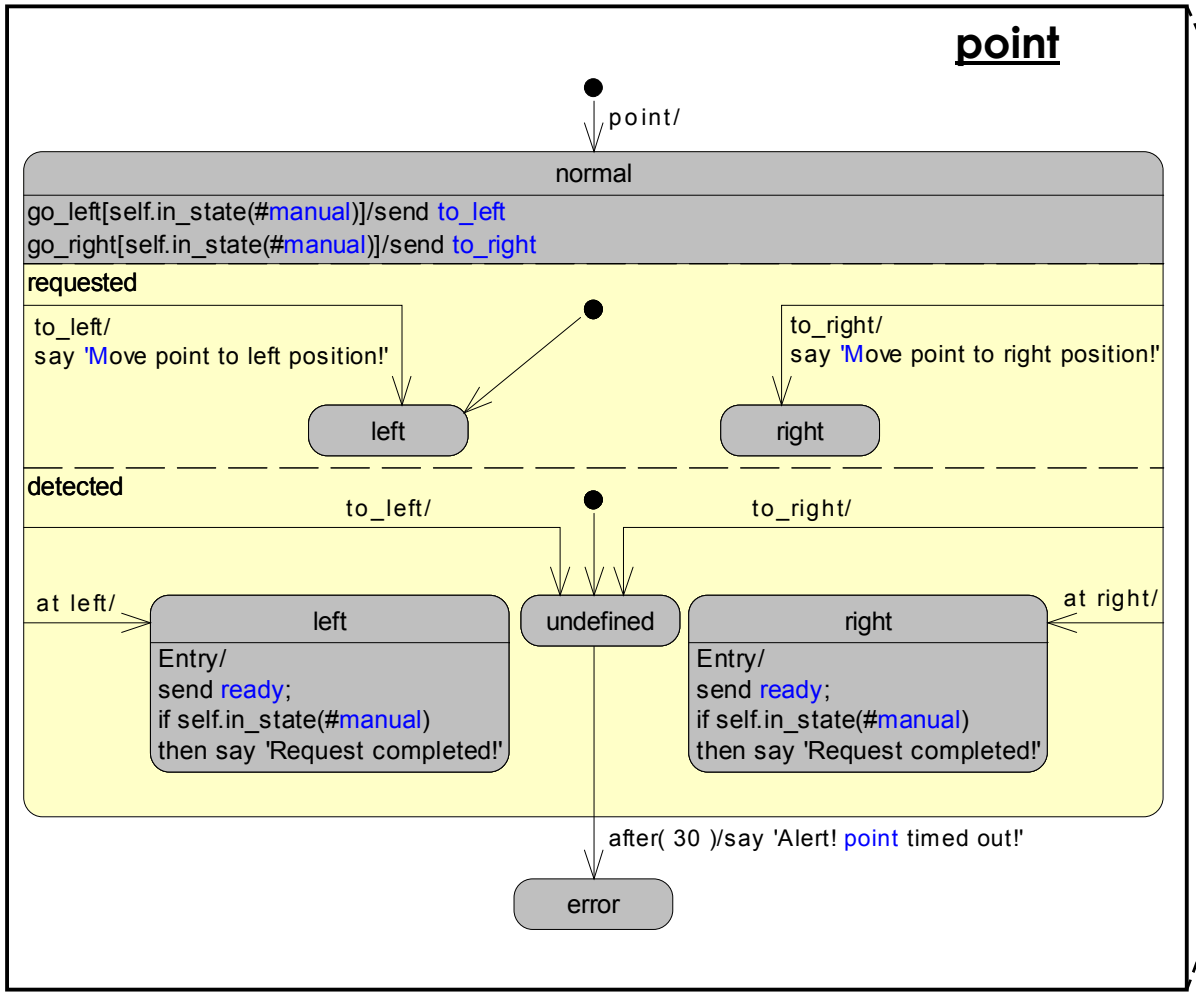
Alternative (non-UML) form:

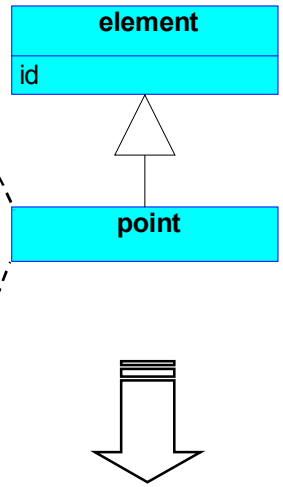
- send <event> after <time>

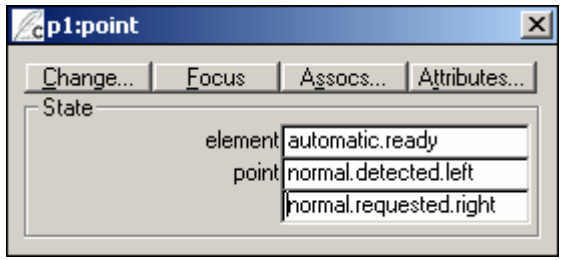
Behavior Inheritance



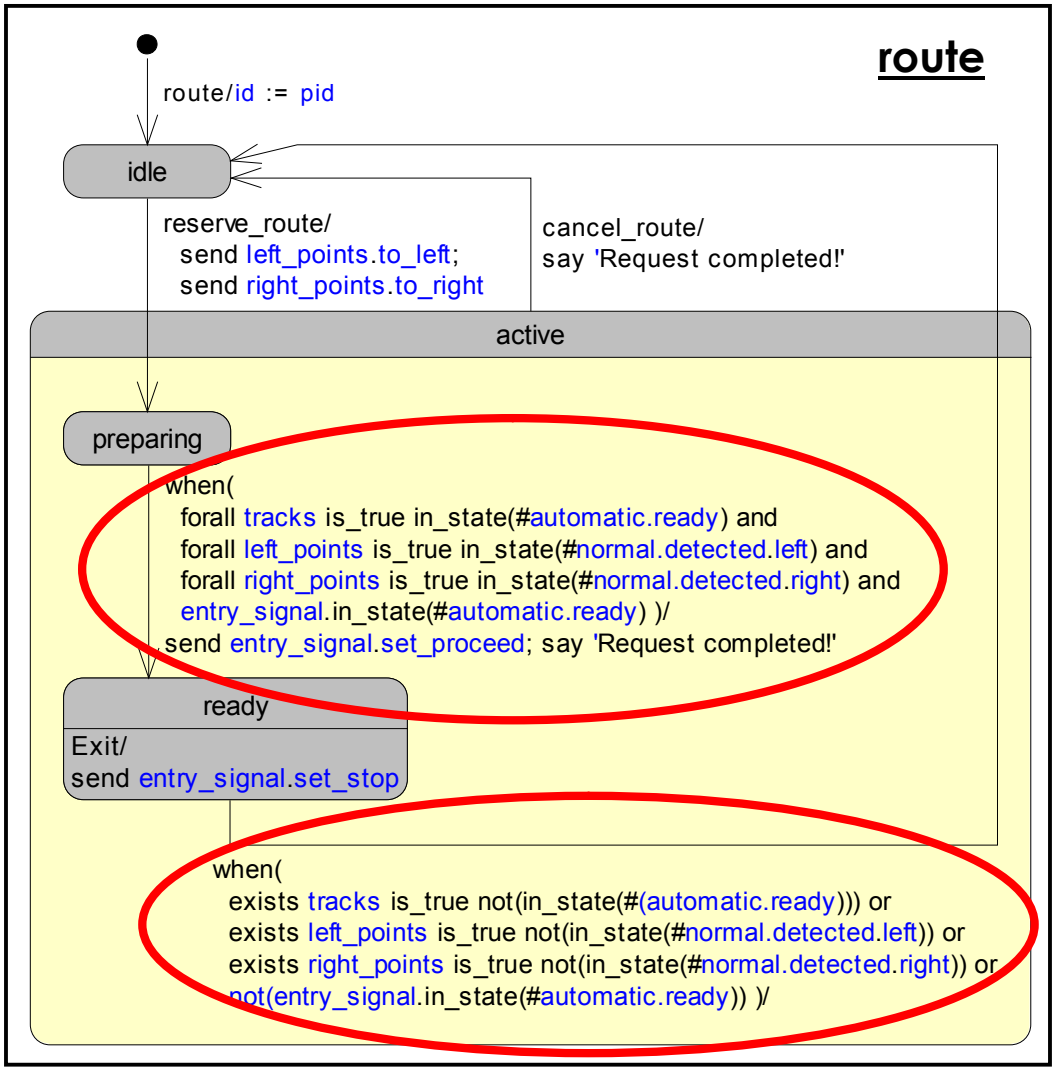
Concurrent State Regions







Change Transitions



Change Transitions

- are triggered by a condition
- have no explicit triggering event

Such a Change Condition

- may be arbitrarily complex and stated over the whole model
- is automatically checked whenever something changes in the model

Model Validation

Model-level Debugging:

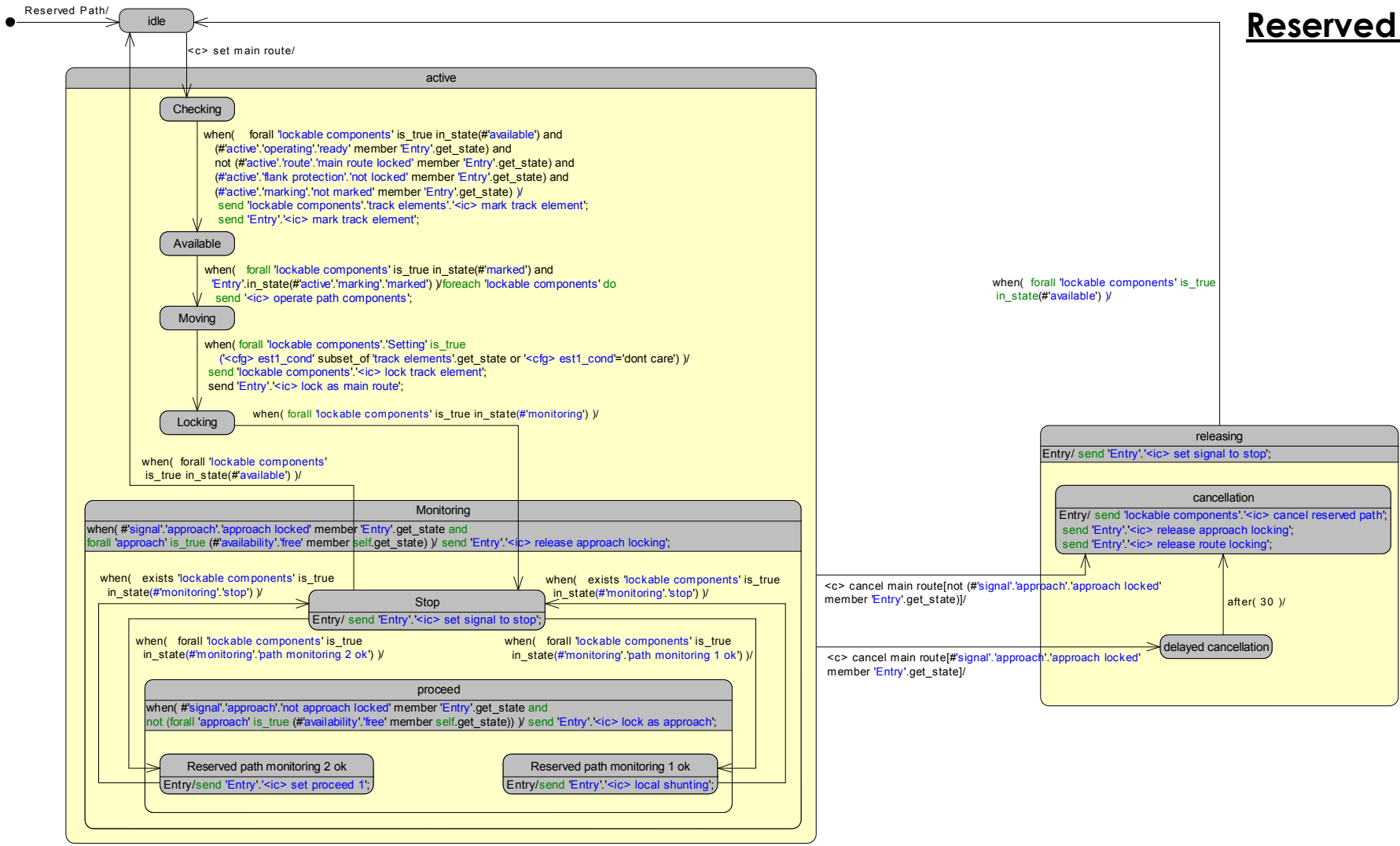
- Object Manipulator to directly create and manipulate object instances
- Object Inspectors to observe any number of objects at runtime
- Save and restore of model state (snapshot)
- Selective logging of interesting events
- Speech output for interesting situations

Regression Testing:

- Simple recording of named test sequences
- Automatic verification of test sequences
- Grouping of test sequences

Experiences and Summary

GENERIS: The "Real Thing"



Summary & Experiences

- ☺ Despite having 'complete' textual requirements, the process of modelling still raised many questions about functionalities
- ☺ UML provides highly compact solutions for the modelling of large and complex requirements
- ☺ Simulation is helpful for validating the given functionality
- ☺ UML is becoming accepted by railway signallers

- ☹ Complex models are still difficult to understand
- ☹ Model-level debugging could be improved
- ☹ Performance of ARTiSAN RtS ⇔ CASSANDRA/xUML coupling could be improved
- ☹ Layout problems with state diagrams having complex conditions and actions