

Modelling the Railway Control Domain rigorously with a UML 2.0 Profile

Kirsten Berkenkötter

Ulrich Hannemann



`kirsten,ulrichh@informatik.uni-bremen.de`

Outline

1. Context
2. Railway Control Systems Domain
3. Domain-specific Languages and UML
4. RCSD Profile
5. Behaviour
6. Validation and further development steps
7. Conclusion

Model based development of railway control systems

- Develop **Platform - Independent Model (PIM)** of the Railway Control Domain as **UML 2.0 Profile**
- Transform UML model into a behavioral model: **Timed State Transition Systems** and **SystemC**
- **Generation of code** out of SystemC model
- Validation, verification and test



Domain-specific Languages and UML

- UML:
 - + Wide-spectrum approach with semantic variation points
 - + Various tool support
 - + Well-known by software developers

- Domain-specific languages:
 - + Domain-specific
 - + Own tool support necessary
 - + Well-known by domain experts

Domain-specific Languages and UML

- Profiles combine both approaches:
 - + Railway domain notation can be used by domain experts
 - + UML notation can be used by software experts
 - + Syntax strictly defined by OCL constraints
 - + Semantics defined by transformation to Timed State Transition System

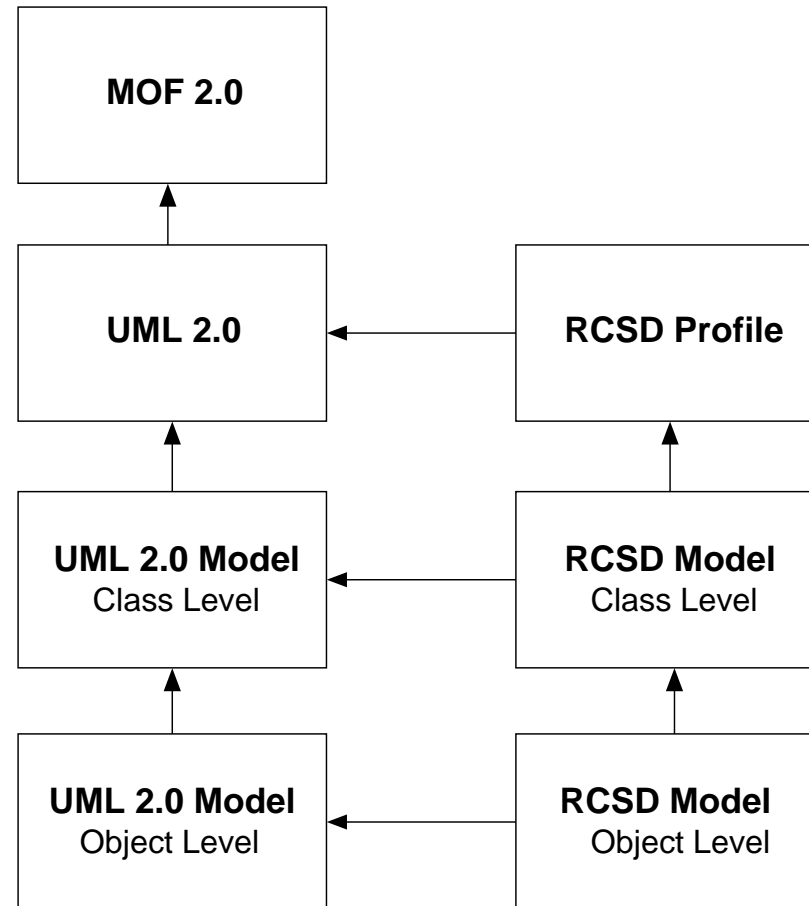


Capabilities

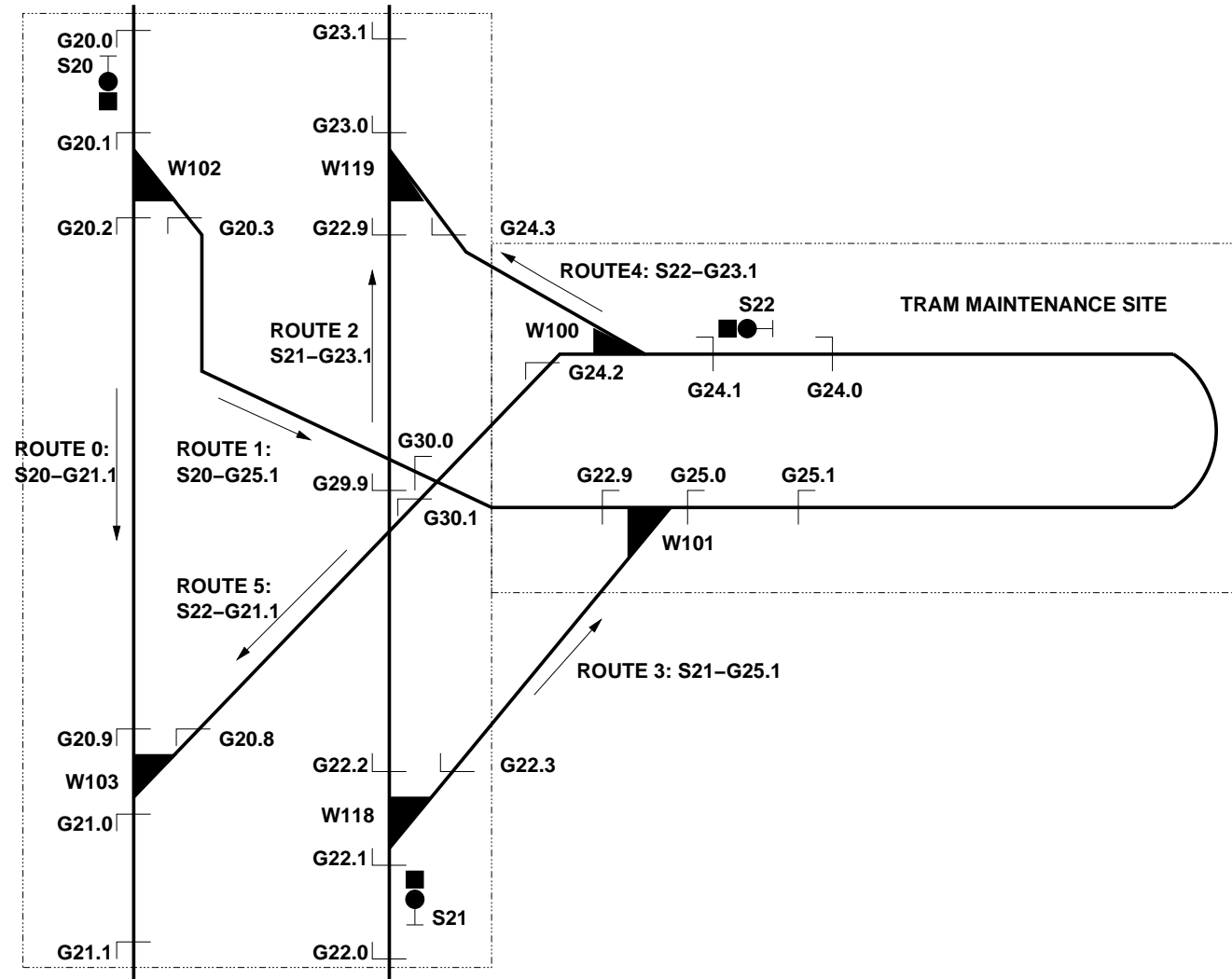
- Capabilities of profiles:
 - + Introduce new syntax
 - + Introduce new constraints
 - + Introduce new semantics
 - + Introduce further information, e.g. transformation rules
 - + Define new primitives and/or enumerations



UML Metalevels



Domain Example



Elements

- Track Elements
- Sensors
- Signals
- Automatic Train Runnings
- Route Definitions



Railway Control Systems Domain

- Track Elements

- + Form a track network
- + Maximal number of trains
- + Optional fixed speed limit

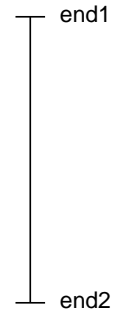
- + Segments
- + Crossings
- + Points



Railway Control Systems Domain

- Segments

+ Two ends



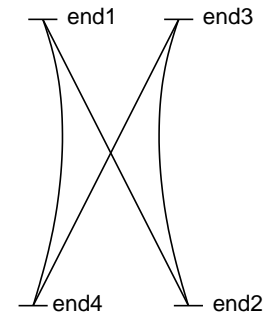
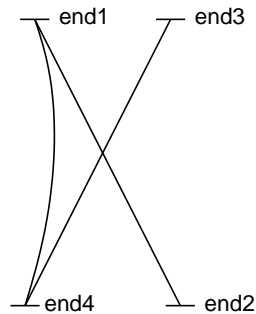
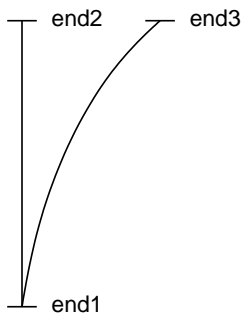
+ Can be sinks or sources of track networks



Railway Control Systems Domain

- Points

- + Single points
- + Single slip points and double slip points
- + One train maximal at each point in time



Railway Control Systems Domain

- Sensors

- + Determine locations of trains
- + Connect track elements to a network
- + Each sensor is exit sensor of one track element and exit sensor of the next one
- + Sensors are locations of all signals and automatic train runnings



Railway Control Systems Domain

- Signals

- + Signal *GO* and *STOP*

- + Signal speed limits (optionally)

- + Signal directions *LEFT*, *RIGHT*, *STRAIGHT* (optionally)



Railway Control Systems Domain

- Routes
 - + Routes are series of sensors
 - + Specific point positions are required
 - + Each route has a start signal
 - + Conflicting routes are known



The RCSD Profile

- How to cook a profile
 - + Each stereotype extends a class in the UML metamodel
 - + Associations can be subsetted but not added
 - + Attributes can be added
 - + Constraints can be added
 - + Semantics can be added
 - + Notation can be added

The RCSD Profile

- Primitives and Literals
- Track Network Elements
- Associations
- Instances
- Route Definitions



The RCSD Profile

- Primitives and Literals

- + Identification numbers for sensors, signals, points ...
- + Points in time
- + Intervals
- + Literals for each primitive type

<<primitive>>
TimeInstant

<<primitive>>
Duration

<<primitive>>
RoutelId

<<primitive>>
AutoRunId

<<primitive>>
SignalId

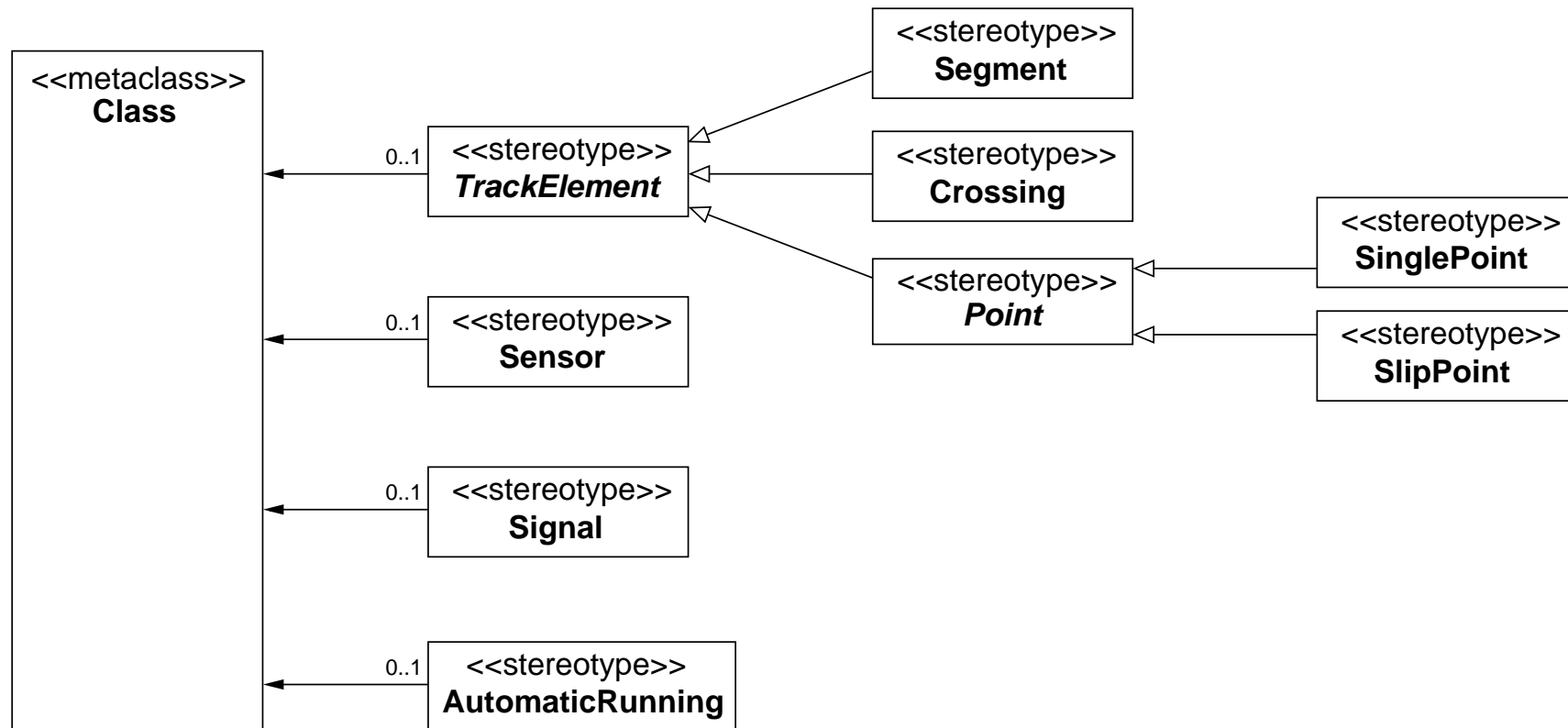
<<primitive>>
PointId

<<primitive>>
SensorId

The RCSD Profile

- Track Network Elements
 - + Segments
 - + Crossing
 - + Single and slip points
 - + Sensors
 - + Signals
 - + Automatic train runnings

The RCSD Profile



The RCSD Profile

- *Signal* stereotype:
 - + *Signal* is an extension of *Class*
 - + Associations are not restricted
 - + No new attribute
 - + Constraints are used to specify properties and their types, e.g. id with type *SignalID*
 - + Semantics are not added
 - + No new notation

The RCSD Profile

- Track Network Elements

Typical constraints:

```
+ ownedAttribute->one(a | a.name->includes('signalId') and  
a.type.name->includes('SignalId') and  
a.upperBound() = 1 and  
a.lowerBound() = 1 and  
a.isReadOnly = true)
```

```
+ ownedAttribute->one(a | a.name->includes('sensor') and  
a.upperBound() = 1 and  
a.lowerBound() = 1 and  
a.outgoingAssociation.  
oclIsTypeOf(SignalAssociation))
```



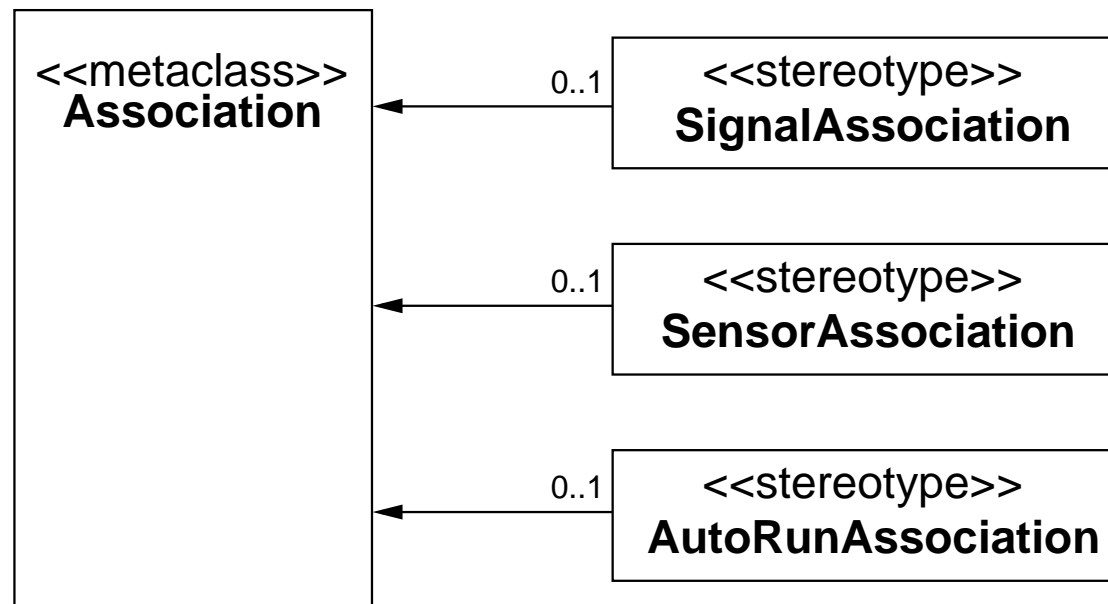
The RCSD Profile

- Associations

- + SensorAssociations between sensors and track elements
- + SignalAssociations between sensors and signals
- + AutoRunAssociations between automatic train runnings and sensors

The RCSD Profile

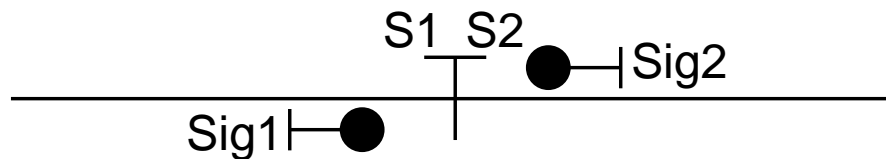
- Associations



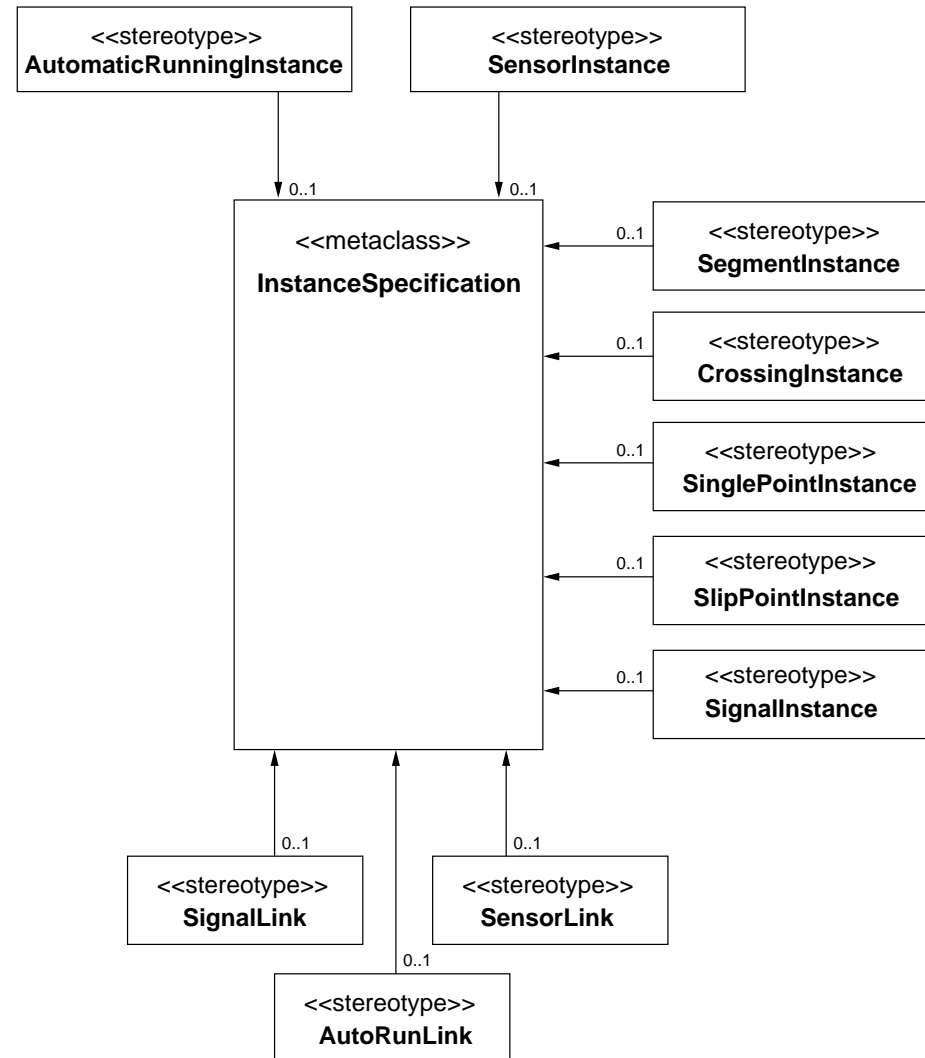
The RCSD Profile

- Instances

- + Segment, crossing, single point, and slip point instances
- + Sensor, signal, and automatic train running instances
- + Domain-specific notation for each instance, e.g. signals



The RCSD Profile



The RCSD Profile

- *SignalInstance* stereotype:
 - + *SignalInstance* is an extension of *InstanceSpecification*
 - + Associations are not restricted
 - + No new attribute
 - + Constraints are used to specify that *SignalInstance* is an instance of *Signal*
 - + Semantics are given by transformation rules
 - + Notation is added with respect to track layout diagrams



The RCSD Profile

- Instances

Typical constraints:

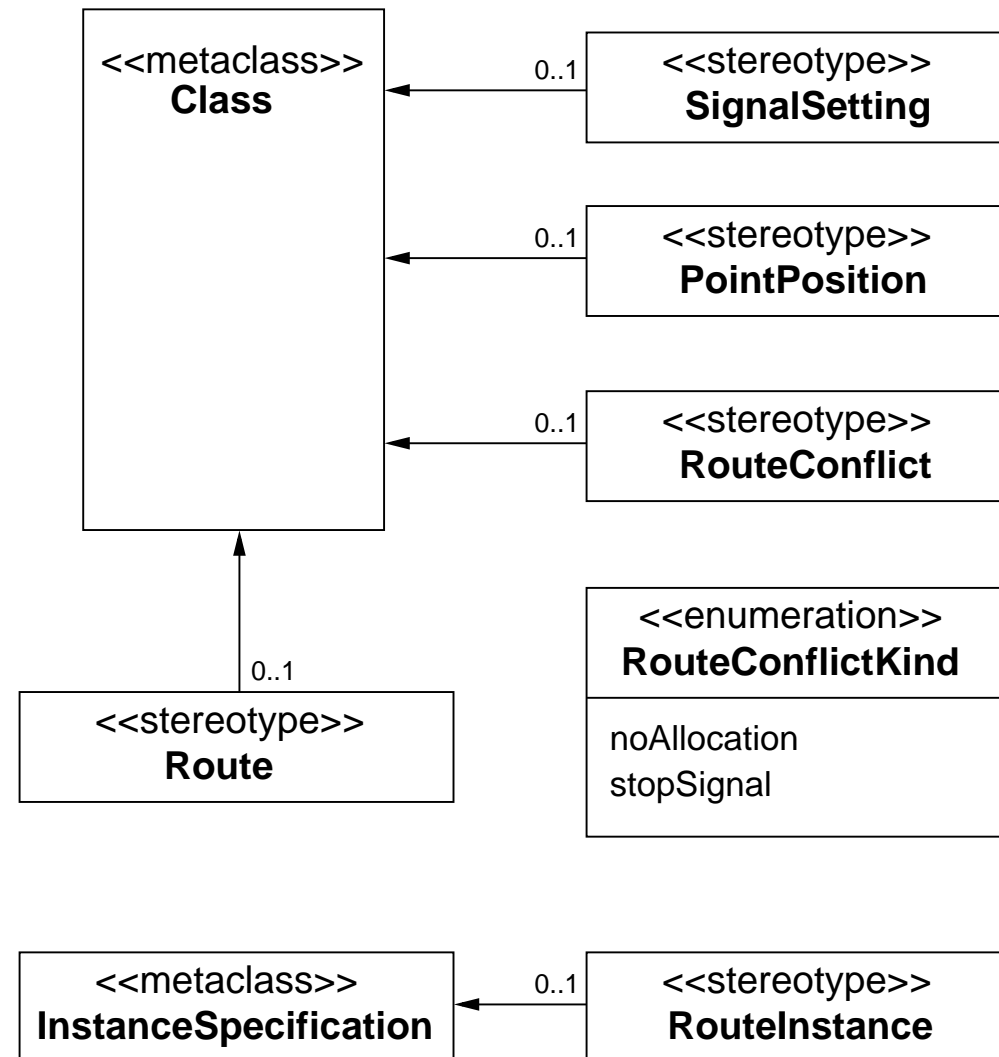
```
+ slot->one(s1 | s1.definingFeature.name->includes('signalId') and  
s1.value->size()= 1 and  
s1.value->first().oclIsTypeOf(LiteralSignalId))
```

The RCSD Profile

- Route Definitions

- + Sequence of sensor ids
- + Sequence of point positions (point id + position)
- + Sequence of signal settings (signal id + setting)
- + Sequence of route conflicts (route id + conflict kind)

The RCSD Profile



Timed State Transition Systems

- + Variables: Properties of all elements of rail network
- + Additional time variable: t
- + State: mapping variables to values
- + Transitions: instances of patterns for types of elements



Transition Examples

- $(w101.reqState \neq w101.actState \wedge w101.reqTime + w101.deltaT \leq t) \Rightarrow w101.actState' = w101.reqState$
- $(w101.reqState \neq w101.actState \wedge w101.reqTime + w101.deltaT > t) \Rightarrow w101.actState' = FAILURE$
- $\Rightarrow t' = t + 1$



Controller Model

- Additional local variables
- **Route dispatcher** registers route requests
- **Route controller** control points and signals of resp. route
- **Safety monitor** checks actual state
- **Safety conditions** are instances of patterns for track elements
- Transitions of controller are also instances of patterns for routes and track elements



UML Specification Environment (USE)

- Validation results for the example:
 - + Compliance of RCSD profile to UML implicitly shown (for this example)
 - + Class diagram is compliant to RCSD profile
 - + Object diagram is compliant to RCSD profile

Validation with USE

- Further results:
 - + USE supports efficient error tracing for constraints
 - + Quality of OCL constraints has been improved
 - + Profile must be modeled once
 - + Class and object diagrams can be translated to USE automatically

Verification of behavioural model

- Timed State Transition Systems represented in SystemC
- Safety conditions generated automatically
- Controller model generated in SystemC
- Bounded model checking of model against safety conditions
- SystemC model is already executable code

Conclusion

- + Efficient modeling of RCSD models
- + Models can be described explicit enough for transformation and verification
- + Validation of static properties with USE
- + Compliance of UML and profile models can be shown
- + Automated generation of safety conditions
- + Automated generation of controller model