Electronic Communications of the EASST
Volume 5 (2006)

Proceedings of the Sixth OCL Workshop OCL for (Meta-)Models in Multiple Application Domains (OCLApps 2006)

Design of a Railway Domain Profile and its OCL-based Validation

Kirsten Berkenkötter

18 pages

Guest Editors: Dan Chiorean, Birgit Demuth, Martin Gogolla, Jos Warmer
Managing Editors: Tiziana Margaria, Julia Padberg, Gabriele Taentzer
ECEASST Home Page: http://www.easst.org/eceasst/ ISSN 1863-2122

Design of a Railway Domain Profile and its OCL-based Validation

Kirsten Berkenkötter

kirsten@informatik.uni-bremen.de, http://www.informatik.uni-bremen.de/~kirsten
Arbeitsgruppe Betriebssysteme, Verteilte Systeme
Universität Bremen, Germany

Abstract: Domain-specific languages become more and more important these days as they facilitate the close collaboration of domain experts and software developers. One effect of this general tendency is the increasing number of UML profiles. UML itself as the most popular modeling language is capable of modeling all kinds of systems but it is often inefficient due to its wide-spectrum approach. Profiles tailor the UML to a specific domain and can hence be seen as domain-specific dialects of UML. At the moment, profiles mainly introduce new terminology, often in combination with OCL constraints which describe the new constructs more precisely. As most tools do not support validation of OCL expressions let alone supplementing profiles with OCL constraints, it is difficult to check if models based on a profile comply to this profile. A related problem is checking whether constraints in the profile contradict constraints in the UML specification. In this paper, it is shown how to complete these tasks with the tool USE. As an example, a profile from the railway control systems domain is taken which describes the use of its modeling elements quite strictly. Models based on this profile serve as a foundation for automated code generation. Therefore, they require a rigorous and unambiguous meaning. OCL is heavily used to reach this goal.

Keywords: Domain-specific Languages, Railway Domain, UML Profiles, OCL, Validation

1 Introduction

The current interest in model driven architecture (MDA) [OMG03] and its surrounding techniques like metamodeling and model driven development (MDD) has also increased the interest in domain-specific languages (DSL) and their development. MDA enforces the idea of platform independent models (PIM) as main artifact in the design of software systems, while the concrete implementation will be based on a platform specific model (PSM). The step from PIM to PSM is performed by transformations while the generation of code is based on the PSM and a description of the concrete target platform called platform model (PM).

In the context of MDA, several standards have been developed like the Meta Object Facility (MOF) [OMG06] for designing metamodels and the Unified Modeling Language (UML) [OMG05c, OMG05b] as a modeling language. UML has become the de-facto standard for modeling languages and is supported by various tools. Due to its wide-spectrum approach, it can be used for modeling all kinds of systems. This is an advantage as one tool can be used to develop different kinds of systems. In contrast, it may also lead to inefficiency and inaccuracy as each

domain has its own need, e.g. domain-specific terminology that differs from the one of UML may lead to misunderstandings. Another problem are semantic variation points in UML. These are necessary to enable the wide-spectrum approach but not useful if the model is to be used in the MDA context as transformations and code generation cannot be utilized with an ambiguous model as foundation.

A good example are railway control systems that are described in specific terminology and notation. The domain of control are track networks that consist of elements like segments, points, or signals. Routes are defined to describe how trains travel on the network. In addition, there are rules that specify in which way a network is constructed and how it is operated. Some rules apply to all kinds of railway systems and some are specific for each kind of railway system, e.g. tramway or railroads. In principle, UML is capable of modeling such systems: class diagrams can be used to describe segments, points, and other track elements and their dependencies while object diagrams model concrete track layouts and routes. Rules can be specified by means of OCL. The problem is that we have to model each kind of railway system with all rules explicitly. The domain knowledge that covers the common parts of all railway control systems is not captured in such models. Neither is specific notation that is used in the domain like symbols for signals and sensors.

Domain-specific languages are a means to overcome these disadvantages [Eva06]. Designing a new modeling language from scratch is obviously time-consuming and costly, therefore UML profiles have become a popular mechanism to tailor the UML to specific domains. In this way, different UML dialects have been developed with considerably low effort. New terminology based on existing UML constructs is introduced and further supplied with OCL [OMG05a, WK04] constraints to specify its usage precisely. Semantics are often described in natural language just as for UML itself.

With respect to railways, the Railway Control Systems Domain (RCSD) profile has been developed [BHP, BH06] as domain-specific UML derivative with formal semantics.The main reason for developing this profile was to simplify the collaboration of domain experts of the railway domain and software developers that design controllers for this domain. With the help of the profile, the system expert develops track networks for different kind of railway systems consisting of track segment, signal, points, etc. The software specialist works on the same information to develop controllers. In the end, controller code which satisfies safety-critical requirements shall be generated automatically. Railway control systems are especially interesting as the domain knowledge gathered in the long history of the domain has to be preserved while combining it with development techniques for safety-critical systems. Structural aspects are specified by class and object diagrams (see Figure 9 and Figure 10) whose compliance to the domain is ensured by OCL constraints. Semantics are based on a timed state transition system that serves as foundation for formal transformations towards code generation for controllers as well as for verification tasks. In this paper, the focus is on the validation of the structural aspects to ensure the correct and successful application of transformations and verification. Details about semantics can be found in [PBD+05, BH06, BHP].

A problem that has not been tackled until now is to validate that the constraints of a profile comply to the ones of UML and that models using a profile comply to this profile. One reason for this is that CASE tools often support profiles as far as new terminology can be introduced but lack support of OCL [BCC+05]. One of the few tools that support OCL is USE (UML

Specification Environment) [Ric02, GZ04]. It allows the definition of a metamodel supplied with OCL constraints and checks whether models based on this metamodel fulfill all constraints. Using (a part of) the UML metamodel in combination with a profile as the USE metamodel allows for fulfilling three goals: (a) Validating that this profile complies to the UML metamodel as each model has to fulfill the invariants of the UML metamodel and the profile. (b) Validating that class diagrams comply to the profile. (c) Validating that object diagrams comply to the profile if the profile describes instances as well as instantiable elements. This approach has been used to validate the RCSD profile and models based on this profile.

The paper is organized in the following way: the next section gives an introduction to UML profiles and the usage of OCL in this context. After that, the railway domain is briefly introduced in Section 3, followed by a description of the RCSD profile and typical constraints in Section 4. After that, Section 5 describes the validation with USE on the different levels. Future work, especially with respect to automated test case generation, is sketched in Section 6. At last, the results of this validation approach and future work are discussed in Section 7.

2 UML Profiles and OCL

UML profiles as described in in [OMG05b] and [OMG05c] offer the possibility to tailor the UML to a specific domain in several ways: (a) introducing new terminology, (b) introducing new syntax/notation, (c) introducing new constraints, (d) introducing new semantics, and (e) introducing further information like transformation rules.

Changing the existing metamodel itself e.g. by introducing semantics contrary to the existing ones or removing elements is not allowed. Consequently, each model that uses profiles is a valid UML model. Profiles are therefore not a means to develop domain-specific languages that contradict UML constraints or semantics. Due to the wide-spectrum approach of UML, semantics are loosely enough to allow all kinds of profiles. A UML 2.0 profile mainly consists of stereotypes, i.e. extensions of already existing UML modeling elements. You have to choose which element should be extended and define the add-ons. In addition, new primitive datatypes and enumerations can be defined as necessary. In Figure 5, a part of the UML metamodel is shown that it afterwards used as a basis for stereotypes, e.g. in Figure 4.

OCL can be used in various ways to specify the stereotypes more precisely:

(a) Constraining property values: A stereotype has all properties of its base class and can add only attributes. Defining new associations to classes in the reference metamodel or other stereotypes is not allowed. Therefore, constraining values of existing attributes and associations is a useful means to give a stereotype the desired functionality.

(b) Specifying dependencies between values of different properties of one element: Often, it is necessary to describe dependencies between the properties of a modeling element precisely.

(c) Specifying dependencies between property values of different instances of one element: Some properties like identification numbers need specific values for different instances of one element.

(d) Specifying dependencies between property values of different instances of different elements: In the same way, several elements may have properties whose values have some kind of relationship. Here, it is important to chose the context of the constraint carefully such that the constraint is not unnecessarily complicated because another modeling element would have been the better choice as basis for the constraint.

3 Short Introduction to the Railway Domain

Creating a domain specific profile requires identifying the elements of this domain and their properties as e.g. described in [Pac02]. In the railway domain, track elements, sensors, signals, automatic train runnings, and routes have been proven essential modeling elements. They are described shortly in the following, more details can be found in [BH06]:

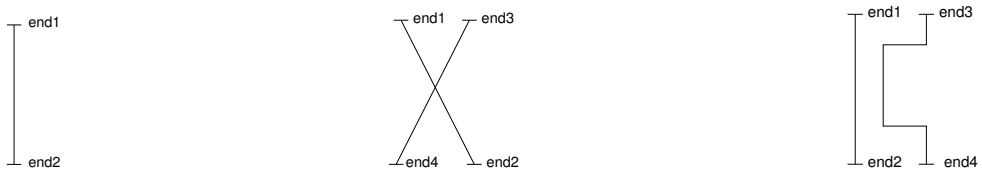

Figure 1: Segment (a), crossing (b), and interlaced segment (c)

Track Elements The track network consists of segments, crossings, and points. Segments are rails with two ends (see Figure 1(a)), while crossings consist of either two crossing segments (see Figure 1(b)) or two interlaced segments (see Figure 1(c)). Points allow a changeover from one segment to another one. Single points have a stem and a branch (see Figure 2(a)). Single slip points and double slip points are crossings with one, respectively two, changeover possibilities (see Figure 2(b) and Figure 2(c)).

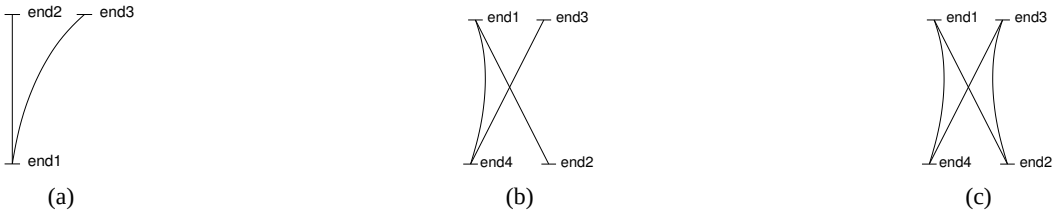

Figure 2: Single point (a), single slip point (b), and double slip point (c)

Sensors Sensors are used to identify the position of trains on the track network, i.e. the current track element. To achieve this goal, track elements have entry and exit sensors located at each end. The number of sensors depends on the allowed driving directions, i.e. the uni- or bidirectional usage of the track element.

Signals Signals come in various ways. In general, they indicate if a train may go or if it has to stop. The permission to go may be constrained, e.g. by speed limits or by obligatory directions

in case of points. As it is significant to know if a train moves according to signaling, signals are always located at sensors.

Automatic Train Runnings Automatic train running systems are used to enforce braking of trains, usually in safety-critical situations. The brake enforcement may be permanent or controlled, i.e. it can be switched on and off. Automatic train running systems are also located at sensors.

Route Definitions As sensors are used as connections between track elements, routes of a track network are defined by sequences of sensors. They can be entered if the required signal setting of the first signal of the route is set. This can only be done if all points are in the correct position needed for this route. Conflicting routes cannot be released at the same time.

4 RCSD Profile

Unfortunately, defining eight stereotypes as suggested by the domain analysis in Section 3 is not sufficient. New primitive datatypes, enumerations, and special kinds of association to model interrelationships between stereotypes are needed. Furthermore, UML supports two modeling layers, i.e. the model layer itself (class diagrams) and the instances layer (object diagrams). In the RCSD profile, both layers are needed: class diagrams are used to model specific parts of the railway domain, e.g. tramways or railroad models, while object diagrams show explicit track layouts for such models. Hence, stereotypes on the object level have to be defined. For these reasons, the RCSD profile is structured in five parts: the definition of primitive datatypes and literals, network elements on class level, associations between these elements, instances of network elements and associations, and route definitions.

4.1 Types and Literals

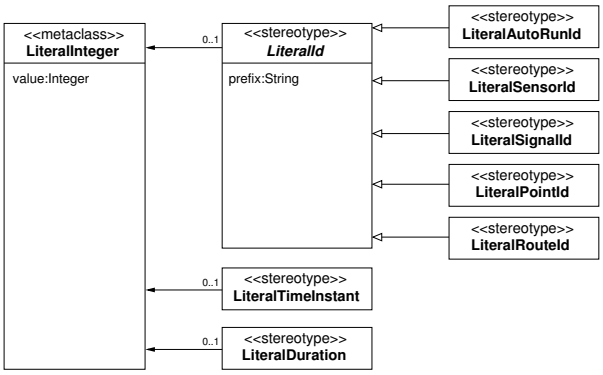

Figure 3: Literals part of the RCSD profile

Several new datatypes are needed: identifiers for all controllable elements, identifiers for routes (e.g. to specify conflicting ones), time instants, and durations. All of them have in com-

mon that the value domain is $\mathbb{N}$. Defining different datatypes facilitates constraints like: all signal identifiers are unique, all point identifiers are unique, and so on. In addition, each new datatype has a dedicated stereotype to model literals of this type (see Figure 3). For the identification types, the corresponding literal consists of an integer value and a prefix character. Literals for time instants and durations are integer values.

```
inv LiteralPointId1:
  value >= 0
inv LiteralPointId2:
  prefix = 'P'
```

OCL constraints for these stereotypes are simple as only values of properties are restricted. Integers values have to be from $\mathbb{N}$; prefixes for different identification types have specific values: *'S'* for sensors, *'Sig'* for signals, *'P'* for points, *'A'* for automatic runnings, and *'R'* for routes. As an example, the two constraints needed for *LiteralPointId* are given above. For the sake of brevity, the name of invariants and the invariants context, where it is unmistakable, are omitted in the following.

4.2 Network Elements

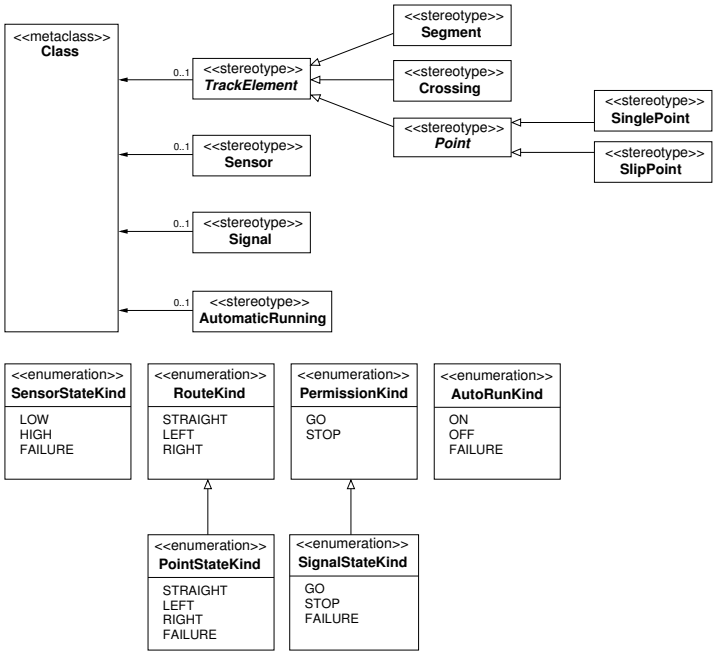

Figure 4: Network elements part of the RCSD profile

The next part of the profile defines track network elements, i.e. segments, crossings, points, signals, sensors, and automatic train runnings (see Figure 4). *Segment*, *Crossing*, and *Point* have in common that they form the track network itself, therefore they are all subclasses of the abstract

TrackElement. Similarly, *SinglePoint* and *SlipPoint* are specializations of *Point*. Enumerations are defined to specify values of properties. All elements are equipped with a set of constraints that define which properties must be supported by each element and how it is related to other elements.

An instance of *TrackElement* on the model layer must provide several properties: *maximalNumberofTrains* to restrict the number of trains on a track element at one point in time (mandatory) and *limit* to give a speed limit (optional). Both properties have to be integers. The first one has a fixed multiplicity 1, the second one may have multiplicities 0..1 or 1. Such requirements for *TrackElement* are defined in the following way:

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

To understand the structure of this constraint, a look at the UML metamodel is helpful. In Figure 5, a part of it is shown, namely the *Classes* diagram of the UML 2.0 *Kernel* package. As all network elements are stereotypes of *Class*, we can refer to all properties of *Class* in our constraints. Properties on the model level are instances of class *Property* on the metamodel level, which are associated to *Class* by *ownedAttribute*. As a *StructuralFeature*, *Property* is also a *NamedElement*, a *TypedElement*, and a *MultiplicityElement*, which allows to restrain name, type, and multiplicity as shown in the constraints above.

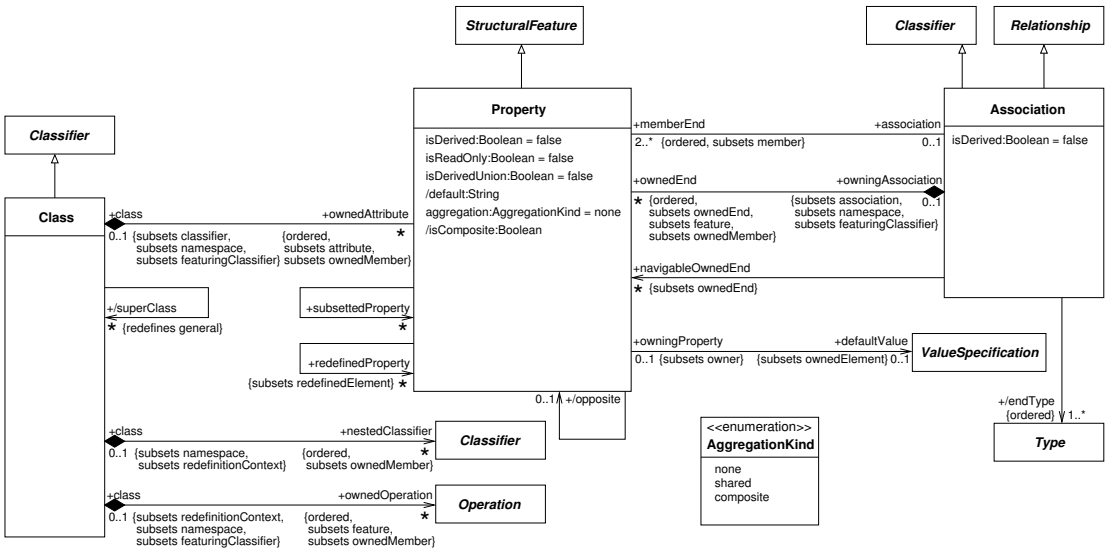

Figure 5: Classes diagram of the UML 2.0 Kernel package

Specifying that some class on model level is the end of an association works in the same way on the metamodel level. As we can see in Figure 5, the ends of each association are properties of classes, i.e. we have to define another property that has to be an association end. *TrackElement* is again used as an example: At each end of a *TrackElement*, entry or exit sensors can be associated.

e1Entry, *e1Exit*, *e2Entry*, and *e2Exit* are used to model these ends of associations to sensors (optional). All outgoing associations must be *SensorAssociations*:

```
ownedAttribute->one(a | a.name->includes('e1Entry')
                    and a.upperBound() = 1 and a.lowerBound() >= 0
                    and a.isReadOnly = true
                    and a.outgoingAssociation.
                          oclIsTypeOf(SensorAssociation)) or
(not ownedAttribute->exists(a2 | a2.name->includes('e1Entry')))
...
ownedAttribute->collect(outgoingAssociation)->
    forAll(a | a.oclIsTypeOf(SensorAssociation) or a.isUndefined)
```

Similar constraints are defined for all network elements. They belong obviously to the category (a) as described in Section 2. They restrict properties on the metamodel level for the usage on the model level.

4.3 Associations

Three types of associations are defined: *SensorAssociations* that connect track elements and sensors, *SignalAssociations* that connect signals and sensors, and *AutoRunAssociations* that connect automatic train runnings and sensors (see Figure 7(a)). Constraints are needed e.g. to determine the kind of stereotype at the ends of each association and their number. As an example, each *SignalAssociation* is connected to one sensor and one signal:

```
inv SignalAssociation1: memberEnd->size() = 2
inv SignalAssociation2: endType->size() = 2
inv SignalAssociation3: endType->one(t | t.oclIsKindOf(Sensor))
inv SignalAssociation4: endType->one(t | t.oclIsKindOf(Signal))
```

Similar constraints are defined for the other kinds of association.

4.4 Instances of Network Elements and Associations

For each non-abstract modeling element and each association, there exists a corresponding instance stereotype (see Figure 7(b)). Here, the domain-specific notation is defined. In Figure 6(a), two unidirectional segments connected by a sensor *S1* are shown. For comparison, the same constellation in object notation is given in Figure 6(b).

(a) (b)

Figure 6: Sensors in RCSD notation (a) and classical UML notation (b)

The instances are heavily restricted by OCL constraints as the instance level serves as the basis for automated code generation. Again, we find several constraints of category (a), where the values of properties are specified explicitly. To give an example, the maximal number of trains on a crossing or point is always defined and the value is 1:

```
slot->one(s1 | s1.definingFeature.name->includes('maxNumberOfTrains')
          and s1.value->size()= 1
          and s1.value->first().oclIsTypeOf(LiteralInteger)
          and s1.value->first()->oclAsType(LiteralInteger).value = 1)
```

Similar constraints appear for all kinds of track elements, e.g. the limit on track elements must have a value from $\mathbb{N}$ if present. More interesting are the constraints from category (b) that describe the dependencies between properties of one stereotype. As an example, each *Point* has a *plus* and *minus* position. One of these has to be *STRAIGHT* and the other one *LEFT* or *RIGHT*:

```
slot->select(s1 | s1.definingFeature.name->includes('minus') or
                  s1.definingFeature.name->includes('plus'))->
  one(s2 | s2.value->size()= 1
       and s2.value->first().oclIsTypeOf(InstanceValue)
       and s2.value->first().oclAsType(InstanceValue).instance.name->
         includes('STRAIGHT')) and
slot->select(s1 | s1.definingFeature.name->includes('minus') or
                  s1.definingFeature.name->includes('plus'))->
  one(s2 | s2.value->size()= 1
       and s2.value->first().oclIsTypeOf(InstanceValue)
       and (s2.value->first().oclAsType(InstanceValue).instance.
              name->includes('LEFT') or
            s2.value->first()->oclAsType(InstanceValue).instance.
              name->includes('RIGHT')))
```

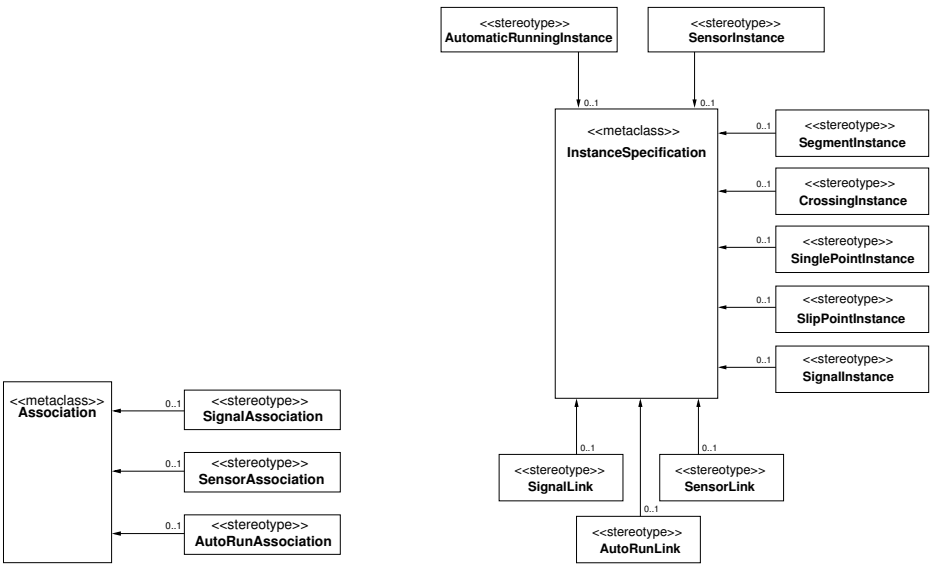

Figure 7: Associations (a) and Instances of network elements and associations (b) parts of the RCSD profile

An example from category (c) are identification numbers of sensors that have to be unique. Each *Sensor* must have a property *sensorId* that is unique with respect to all instances of *Sensor*:

```
SensorInstance.allInstances->collect(slot)->asSet->flatten->
  select(s | s.definingFeature.name->includes('sensorId'))->
    iterate(
      s:Slot;
      result:Set(LiteralSensorId) = oclEmpty(Set(LiteralSensorId)) |
      result->including(s.value->first.oclAsType(LiteralSensorId)))->
    isUnique(value)
```

4.5 Route definitions

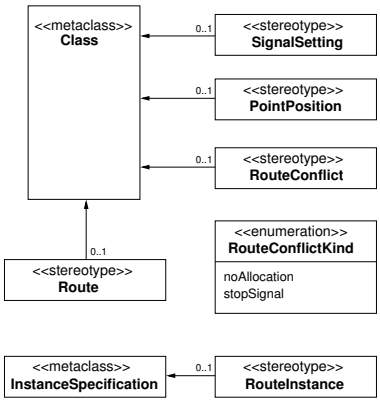

Figure 8: Route definition part of the RCSD profile

Moreover, the profile defines routes and their instances as shown in Figure 8. Each *Route* is defined by an ordered sequence of sensors. The signal setting for entering the route and sets of required point positions and of conflicts with other routes are further necessary information. Again, constraints are used for unambiguous and strict definitions of properties. Constraints from category (d) are typical as sensors, signals, and points are referenced by their id in route definitions. This implies that these ids belong to some existing instances, e.g. the sensor ids given in the definition of a route. Hence, the following constraint must hold for each *RouteInstance*:

```
let i:Set(Integer) =
  slot->select(s | s.definingFeature.name->
    includes('routeDefinition'))->asSequence->first().value->
      iterate(v:ValueSpecification;
              result:Set(Integer)=oclEmpty(Set(Integer)) |
              result->including(v.oclAsType(LiteralSensorId).value))
in
  i->forAll(id | SensorInstance.allInstances->exists(sens |
    sens.slot->select(s | s.definingFeature.name->
      includes('sensorId'))->asSequence->first().value->first().
        oclAsType(LiteralSensorId).value = id))
```

5 Validation of Wellformedness Rules with USE

The next step is adapting the profile and its various invariants to USE for the validation process. USE expects a model in textual notation as input. For syntax details, we refer to [GZ04]. In our case, this is the metamodel consisting of (a part of) the UML metamodel and the profile. On this basis, instance models can be checked with respect to the invariants in the metamodel. In our case, the instance model consists of both class layer and object layer, i.e. models using the RCSD profile. A similar application of USE with respect to the four metamodeling layers of UML is shown in [GFB05].

This metamodel file includes both the necessary part of the UML 2.0 metamodel and the RCSD profile for two reasons: first, the profile cannot exist without its reference metamodel and second, one goal is to check the compliance of the profile to the metamodel. This task must be performed implicitly as USE does not check if the given constraints contradict. Instead, we assume the profile compliant to the metamodel as long as both the constraints in the metamodel and the constraints in the profile are all valid. Contradicting constraints can be identified if all constraints in the profile evaluate to true but some constraint(s) in the metamodel evaluate(s) to false.

5.1 Modeling the UML Metamodel and the RCSD Profile for USE

In the metamodel file, a description of classes with attributes and operations, associations, and OCL constraints is expected. OCL constraints are either invariants as shown in Section 4, definitions of operations, or pre-and postconditions of operations. Only operations whose return value is directly specified in OCL and not dependent on preconditions are considered side-effect free and may be used in invariants. For the validation of the profile, all invariants must be fulfilled by the instance model(s).

From the UML metamodel, the *Kernel* package has been modeled with some modifications: (a) Packages are not needed by the RCSD profile and therefore skipped in all diagrams, diagram *Packages* has been omitted completely. (b) Lower and upper bounds of multiplicities have been changed to *LiteralInteger* instead of *ValueSpecification* for easier handling. One reason is that the invariants in the context of *MultiplicityElement* are not specific enough to guarantee that the *ValueSpecification* really evaluates to *LiteralInteger* as necessary. Therefore, expressions cannot be used to specify multiplicities. The invariants of *MultiplicityElement* have been adapted to this. (c) Several invariants and operations had to be rewritten or omitted completely as they are erroneous in the UML specification. More information about this problem can be found in [BGG04]. (d) Some names in the UML specification had to be changed due to conflicts with USE keywords or multiple usage in the specification which also leads to conflicts. This problem is also described in [BGG04]. (e) USE does not support *UnlimitedNatural* as type. This problem has been overcome by using *Integer* and additional constraints that restrict corresponding values to $\mathbb{N}$. All in all, 34 invariants have been modeled here. Further packages from the UML metamodel are not needed.

Profiles are not directly supported by USE. This problem has been overcome by modeling each stereotype as a subclass from its metaclass, i.e. a metamodel extension. Modeling profiles as restricted extensions to metamodels is feasible with respect to [JSZ+04]. Here, modifications

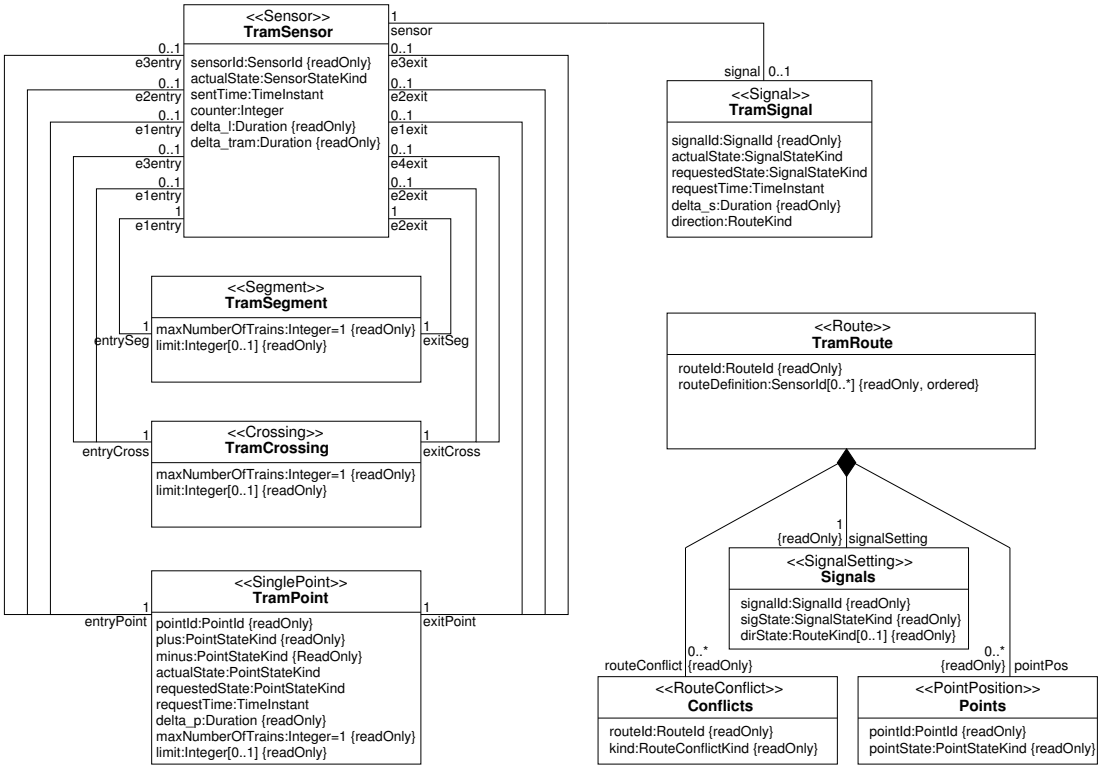

Figure 9: Tram network definitions - class level

to metamodels are classified in level one (all extensions to the reference metamodel allowed), level two (new constructs can be added to the referenced metamodel, but existing ones cannot be changed), level three (each new construct must have a parent in the reference metamodel), and level four (new relationships are only allowed as far as existing ones are specialized. The lower levels include all restrictions of the levels above. Therefore, profiles can be considered a level four metamodel extension and modeled as such in USE. [1] All in all, the following invariants of types (a) - (d) have been specified:

Profile part	(a)	(b)	(c)	(d)
Types and Literals	12	0	0	0
Network Elements	95	0	0	0
Associations	27	0	0	0
Instances	104	37	4	7
Route Definitions	36	1	3	22
Total	274	38	7	29

[1] [JSZ+04] considers profiles as level three which is incorrect as the relationship restriction has to be respected by profiles.

5.2 Compliance of RCSD Model to Profile on Class Level

Evaluating constraints is possible for instances of the given (meta)model. As an example, a tram network description is used on class level. Tram networks consist of segments, crossings, and single points that are all used unidirectionally. Furthermore, there are signals, sensors, and routes, but no automatic runnings. This constellation is shown in Figure 9.

In USE, an instance model can be constructed step by step by adding instances of classes and associations of the metamodel to an instance diagram. More convenient is the usage of a **.cmd* command file where instance creation and setting of property values are specified in textual notation. Again, we refer to [GZ04] for syntax details.

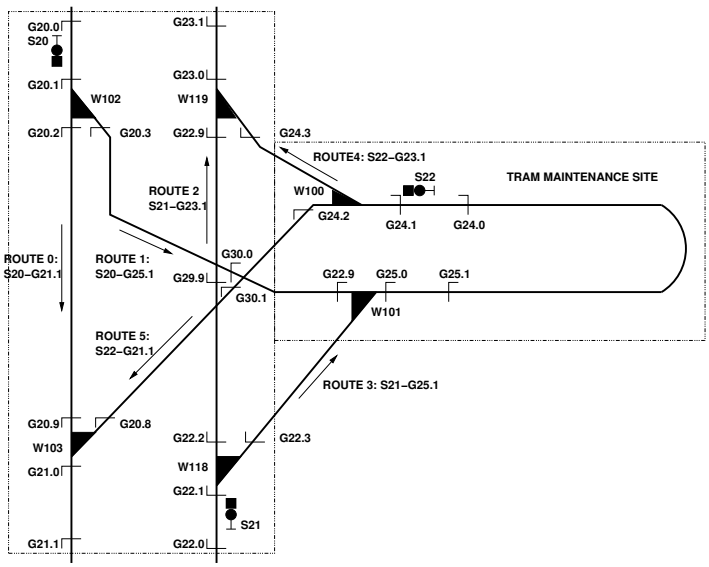

Figure 10: Concrete track network - instance level

5.3 Compliance of RCSD Model to Profile on Instance Level

A concrete network of a tram maintenance site with six routes is shown in Figure 10. Note that this diagram is given in RCSD notation and can also be shown in UML object notation as discussed in Section 4. The explicit route definitions have been omitted for the sake of brevity, but can be easily extracted from Figure 10. This diagram has been used for the validation on the instance level. It consists of 12 segments, 3 crossings, 6 points, 25 sensors, 3 signals, and 6 routes, specified in a second **.cmd* file. The two **.cmd* files form a complete instance model of the metamodel consisting of classes and their instances.

5.4 Results

In this example, all invariants have been fulfilled. The correctness of the OCL constraints could be easily checked by adding intentional errors like incorrect association ends or signals with the

same id. USE facilitates tracing of such errors by (a) showing which instance of the metamodel has violated an invariant and by (b) decomposing the invariant in all sub-clauses and giving the respective evaluation. In Figure 11, we can see that *sensor2* and *sensor3* have duplicate identification numbers.

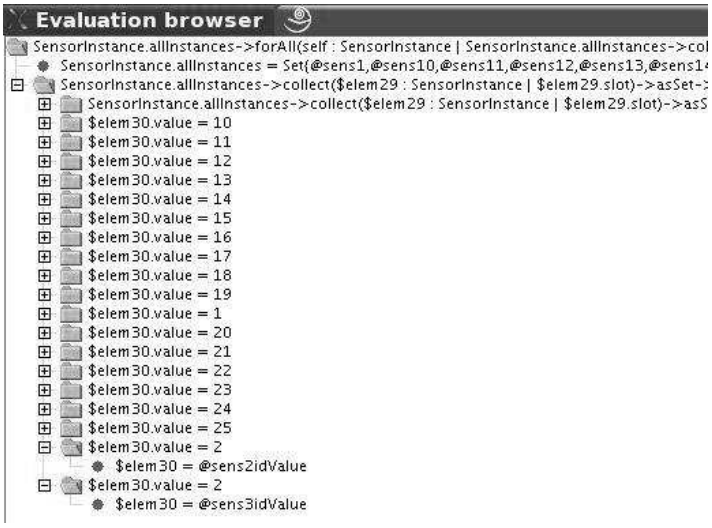

Figure 11: Evaluation example - two identical sensor ids

For the validation process, some effort has to be made for the modeling part. Fortunately, the metamodel and profile have to be modeled only once for each profile. The part of the UML metamodel that has to be included varies from profile to profile depending on the metaclasses references by stereotypes. The current version of the USE model file consists of approximately 4000 lines. As this task is performed once per profile, the effort seems reasonable. With respect of the RCSD profile, the instance model on class level has to be modeled once per specific railway system, e.g. once for trams. With this part of the instance model, all kinds of concrete track layouts can be checked. The tram example consists of approximately 1500 lines of input data to USE. These can be generated from class diagrams by parsing the output of CASE tools and adapting them to USE. Concrete track layout can also be generated, this time from object diagrams. In this way, all kinds of track layouts for one system can be checked. The example track layout needs about 5000 lines. As writing them for each layout would be an obnoxious task, automation is highly required.

6 Related and Future Work

At the moment, the RCSD profiles defines large parts of a domain-specific language for the railway control systems domain. On the one hand, there is the abstract and concrete syntax defined by UML diagrams as shown in the figures in Section 4. On the other hand, there are static semantics defined by OCL constraints (see Section 4) that allow us to validate RCSD models and use them as foundation for further tasks.

Obviously, also behavioral semantics have to be defined. These are captured by a timed state transition system (TSTS) that is based on a RCSD model and also incorporates the behavior of a controller which has to guarantee safety conditions for the running system. The behavior of the controller can be deduced from generic patterns that are derived from domain knowledge. The composition of the controller and the individual RCSD model should then allow only sequences of transitions which never violate a safety condition. A good example here is that a railway controller may never release two conflicting routes at the same time. More details about the TSTS and the generic controller patterns can be found in [BH06, PGHD04, PBD+05].

A TSTS can be encoded in SystemC [GLMS02] and used for verification purposes [GD03]. As described in [PGHD04] and [PBD+05], the verification of the railway domain model can be performed by bounded-model checking as this technique overcomes the problem of state explosion usually occurring with other model checking techniques for railway control systems of realistic sizes. SystemC models also serve as foundation for automated code generation of railway controller code. Automated code generation and the verification of the generated code are ongoing work that is further described in [PBD+05].

An interesting point to investigate is automated test case generation. Even if the correctness of model and generated code can be verified, tests have to be performed at least on hardware-software integration level. This is unavoidable as the hardware integration may expose new kinds of errors that are caused by hardware configurations, memory handling, interface latencies, and similar problems that not present on model level. We expect that the selection of meaningful test cases will be improved significantly due to the domain-specific knowledge provided by RCSD models.

Currently, there are several approaches to model-based automated test case generation that could be adapted to the TSTS. One example is presented in [DGG04]. Test cases are chosen by traversing a graphical representation of the software under test. Paths in the graph are chosen by statistical methods. Another approach – based on timed automata – is presented in [CO02]. Here, heuristics have been developed to chose test cases from the infinitely many ones deduced from paths through the timed automata. Both approaches – and also similar ones – share the same problem: it is not possible to ensure that all relevant test cases for the system under test have been found.

For the railway domain, the domain knowledge can be taken into account with respect to test case generation algorithms. The object model in combination with safety conditions gives important information about the expected behavior of the controller under test. To give an example, we expect that all points assigned to a route have been switched to the requested position before the entry signal of the route is set to *GO*. Other examples are that conflicting routes may not be released at the same time or that only one train is allowed at most on a point at each point in time.

The test case generation algorithm has to be aware of this information. At the moment two possibilities seem feasible. (a) Test case generation is performed at object level based on route information and safety conditions specified in OCL. The needed test cases are then transfered to the SystemC level where we can check if the model is sufficiently covered – dependent on some coverage criterion – and eventually more test cases have to be generated. (b) Another possibility is to incorporate the domain-specific knowledge about relevant test cases into the SystemC model as additional information. In this case, test case generation can be completely

performed on SystemC level. At the moment, we examine the power of these two approaches.

Another advantage of using domain-specific knowledge in automated test case generation is that not only test cases but also meaningful documentation can be generated. This facilitates backtracking of occurring errors as we are able to follow the inputs to the controller under test and its outputs more easily. It is obviously more convenient to be aware that contradicting routes *Route 2* and *Route 4* have been requested and both released due to some error than reconstructing the meaning of the generated test case manually.

Moreover, the generation of USE snapshots of RCSD object diagrams by each test, e.g. in a different log file, seems promising to facilitate error backtracking. As the behavior of the controller is derived from static semantics and safety conditions defined by OCL expressions, it is likely that errors occurring in tests are reflected in invariant violations. Hence, we need a snapshot of an object diagram coinciding with the current system state. In this case, the violated invariant will give information about the cause of the error. The possibilities for using OCL constraints and USE in error backtracking are also currently under investigation.

7 Conclusion

The validation of models of the RCSD profile and the profile itself based on OCL constraints with USE has been proven useful in several ways. It has been shown that the profile complies to UML as it is required and that an example model for tramways is valid in the RCSD context. This makes object diagrams for such tramways applicable for transformation and verification purposes. Another effect of the validation with USE was the improvement of the OCL constraints themselves. As most case tools have no OCL support, it is hard to detect if constraints exhibit syntax errors or if complicated constraints really have the intended meaning.

An adaption of the validation process to other profiles can be performed straightforward as the same kinds of constraints should appear. It is possible that the UML metamodel part has to be enhanced for other profiles as this depends on the metaclasses referenced by stereotypes. Validation is reasonable for each profile whose application relies on a solid and unambiguous model.

With respect to the RCSD profile, future work has to investigate the behavioral aspects of track layouts as described in Section 6. At the moment, only statical aspects have been examined, but USE can also be applied to the validation and test of controllers that have been generated for a concrete track network.

At any rate, verification, automated code generation, and automated test case generation based on RCSD models seem to be promising approaches to improve the development process of railway control systems and their verification. First results also show the impact of domain-specific languages as the domain-specific knowledge covered in such models influences further usage of models as e.g. in automated test case generation significantly.

Acknowledgements: Special thanks go to Fabian Büttner and Arne Lindow for their help with USE and to Ulrich Hannemann and Jan Peleska for their valuable feedback to the first versions of this paper and the related work.

Bibliography

[BCC+05] T. Baar, D. Chiorean, A. Correa, M. Gogolla, H. Hußmann, O. Patrascoiu, P. H. Schmitt, J. Warmer. Tool Support for OCL and Related Formalisms - Needs and Trends. In Bruel (ed.), *Satellite Events at the ModELS'2005 Conference*. LNCS 3844, pp. 1–9. Springer-Verlag, 2005.
doi:10.1007/11663430_1

[BGG04] H. Bauerdick, M. Gogolla, F. Gutsche. Detecting OCL Traps in the UML 2.0 Superstructure. In Baar et al. (eds.), *Proceedings 7th International Conference Unified Modeling Language (UML'2004)*. LNCS 3273, pp. 188–197. Springer, 2004.
doi:10.1007/b101232

[BH06] K. Berkenkötter, U. Hannemann. Modeling the Railway Control Domain Rigorously with a UML 2.0 Profile. In Górski (ed.), *Computer Safety, Reliability, and Security, SAFECOMP 2006*. LNCS 4166, pp. 398–411. Springer, 2006.
doi:10.1007/11875567

[BHP] K. Berkenkötter, U. Hannemann, J. Peleska. The Railway Control System Domain. Draft.
http://www.informatik.uni-bremen.de/agbs/research/RCSD/

[CO02] R. Cardell-Oliver. Conformance Test Experiments for Distributed Real-Time Systems. In Olderog and Steffen (eds.), *International Symposium on Software Testing and Analysis (ISSTA'02)*. ACM Press 1710, pp. 159–163. July 2002.
doi:10.1145/566172.566196

[DGG04] A. Denise, M.-C. Gaudel, S.-D. Gouraud. A Generic Method for Statistical Testing. *ISSRE - 15th International Symposium on Software Reliability Engineering*, pp. 25–34, 2004.
doi:http://doi.ieeecomputersociety.org/10.1109/ISSRE.2004.2

[Eva06] A. Evans. Domain Specific Languages and MDA. 2006.
http://albini.xactium.com/web/downloads/b1a35960appliedMetamodelling.pdf

[GD03] D. Große, R. Drechsler. Formal Verification of LTL Formulas for SystemC Designs. In *IEEE International Symposium on Circuits and Systems*. Pp. V:245–V:248. 2003.
http://ieeexplore.ieee.org/search/wrapper.jsp?arnumber=1206243

[GFB05] M. Gogolla, J.-M. Favre, F. Büttner. On Squeezing M0, M1, M2, and M3 into a Single Object Diagram. Technical report LGL-REPORT-2005-001, Ecole Polytechnique Fédérale de Lausanne, 2005.
http://www.db.informatik.uni-bremen.de/publications/Gogolla_2005_OCLWS.ps

[GLMS02] T. Grötker, S. Liao, G. Martin, S. Swan. *System Design with SystemC*. Kluwer Academic Publishers, 2002.

[OMG03] Object Management Group. MDA Guide Version 1.0.1. June 2003.
http://www.omg.org/docs/omg/03-06-01.pdf

[OMG05a] Object Management Group. OCL 2.0 Specification, version 2.0. June 2005.
http://www.omg.org/docs/ptc/05-06-06.pdf

[OMG05b] Object Management Group. Unified Modeling Language: Superstructure, version 2.0. July 2005.
http://www.omg.org/docs/formal/05-07-04.pdf

[OMG05c] Object Management Group. Unified Modeling Language (UML) Specification: Infrastructure, version 2.0. July 2005.
http://www.omg.org/docs/ptc/04-10-14.pdf

[OMG06] Object Management Group. Meta Object Facility (MOF) 2.0 Core Specification. Jan. 2006.
http://www.omg.org/docs/formal/06-01-01.pdf

[GZ04] M. Gogolla, P. Ziemann. *Checking BART Test Scenarios with UML's Object Constraint Language*. Pp. 133–170. Kluwer, Boston, 2004.
http://www.db.informatik.uni-bremen.de/publications/Gogolla_2004_KLUWER.ps

[JSZ+04] Y. Jiang, W. Shao, L. Zhang, Z. Ma, X. Meng, H. Ma. On the Classification of UML's Meta Model Extension Mechanism. In Baar et al. (eds.), *The Unified Modelling Language: Modelling Languages and Applications*. LNCS 3273, pp. 54–68. Springer, 2004.
doi:10.1007/b101232

[Pac02] J. Pachl. *Railway Operation and Control*. VTD Rail Publishing, Mountlake Terrace (USA), 2002. ISBN 0-9719915-1-0.

[PBD+05] J. Peleska, K. Berkenkötter, R. Drechsler, D. Große, U. Hannemann, A. E. Haxthausen, S. Kinder. Domain-Specific Formalisms and Model-Driven Development for Railway Control Systems. In *TRain workshop at SEFM2005*. September 2005.
http://www.informatik.uni-bremen.de/agbs/jp/papers/peleska_et_al_train2005_slides.pdf

[PGHD04] J. Peleska, D. Große, A. E. Haxthausen, J. R. Drechsler. Automated Verification for Train Control Systems. In Schnieder and Tarnai (eds.), *FORMS/FORMAT 2004 - Formal Methods for Automation and Safety in Railway and Automotive Systems*. Pp. 252–265. Technical University of Braunschweig, 2004.
http://www.informatik.uni-bremen.de/agbs/jp/papers/peleska_et_al_forms2004.ps

[Ric02] M. Richters. *A Precise Approach to Validating UML Models and OCL Constraints*. BISS Monographs 14. Logos Verlag, Berlin, 2002. Ph.D. Thesis, Universität Bremen.

[WK04] J. Warmer, A. Kleppe. *Object Constraint Language 2.0*. MITP-Verlag, Bonn, 2004.