Modeling the Railway Control Domain Rigorously with a UML 2.0 Profile

Kirsten Berkenkötter and Ulrich Hannemann

University of Bremen,
P.O. Box 330 440
28334 Bremen, Germany
{kirsten, ulrichh}@informatik.uni-bremen.de

Abstract. We introduce the Railway Control Systems Domain (RCSD) profile of the Unified Modeling Language UML 2.0 as a domain specific modeling language for railway and tramway control systems. The RCSD profile covers the segments of the rail network, sensors, and control elements like signals and switches. Using these terms of the railway domain, it facilitates the communication between domain experts and specialists for embedded control system development. Defined as a profile for UML 2.0, the development of precise RCSD descriptions is supported by standard UML tools, visualizing railway networks in the same way as domain experts are used to. The static description of networks is complemented by the characterization of the dynamics within the network with trains running on predefined routes. This behaviour is provided by the semantics of a state transition system derived from the object diagram of a particular network model. This rigorous semantic approach constitutes a prerequisite for further tool-supported analysis of safety requirements, and generation of the actual control system.

1 Introduction

With the present paper we contribute to the *model driven development* process of railway control systems. With emphasis on a modeling language and its formal semantics, we support the foundation of the widely automated generation of controller components in the railway domain. As we provide a means to capture the requirements of these control components thoroughly and unambiguously, the focus within the development process shifts towards the modeling phase, i.e. the formalization of the application users' view onto the system.

We demonstrate our approach of utilizing a UML 2.0 profile as a *domain specific language* for a problem in the railway control system domain. The *domain of control* – also called *physical model* – consists of a railway network composed of track segments, points, signals, and sensors. Trains enter the domain of control at distinguished entry segments and request to take pre-defined routes through the network. Detection of trains is possible only via sensor observations. A *controller* monitors state changes within the network, derives train locations, and governs signals and points to enable the correct passage of trains through the network.

J. Górski (Ed.): SAFECOMP 2006, LNCS 4166, pp. 398–411, 2006.
© Springer-Verlag Berlin Heidelberg 2006

With all activities, the controller must ensure that no hazardous situation arises, formulated by requiring compliance with a specific set of *safety conditions*.

The railway control domain is a perfect candidate to apply a domain specific language as it contains a rather limited amount of different entities. The specialized objects involved may exhibit only a limited variation of behavior, and the high safety requirements already established in the railway domain have resulted in a decent formalization of component descriptions. Part of the challenge of formulating a domain theory of railways [Rai] lies in the long history of the domain where domain experts gathered a respectable amount of knowledge which is hard to contain in a computing science formalism. Thus, an approach to deal with critical railway control applications has to carefully connect the expertise in railway engineering with the development techniques of safety critical software.

Among the various proposed solutions, we observe a number of characteristics that we deem desirable: (1) The UniSpec language within the EURIS method [FKvV98] provides a domain specific language with *graphical elements* to reflect the topology of a railway network. (2) In order to support the development process with *standard tools* the wide-spectrum Unified Modeling Language UML [RJB04] is used in the SafeUML project [Hun06] which specifically aims at generating code conforming to safety standards. The use of UML is restricted here by guidelines to ensure maintenance of safety requirements which still allow sufficiently expressiveness for the modeling process. (3) In [PBH00, HP02, HP03a] the domain analysis concentrates on the relevant issues for formal treatment of the control problem using a presentation form of tables and lists as foundation for a *formal model*.

Based on these experiences, we propose to use the *profile* mechanism for UML 2.0 [OMG04, OMG05b] to create a *domain-specific* description formalism for requirements modeling in the railway control systems domain (RCSD). This approach allows us to use a graphical representation of the domain elements with domain specific icons in order to facilitate the communication between domain experts and specialists for embedded control systems development. As the profile mechanism is part of the UML standard, the wide-spread variety of existing tools can be adapted within the very spirit of the UML using UML-inherent concepts. Since a profile allows to introduce new semantics for the elements of the profile we can attach a rigorous mathematical model to the descriptions of the domain model. Timed state transition system semantics form the base for formal transformations towards code generation for the controller as well as for the verification task that guarantees conformance to the safety requirements. Consequently, the RCSD profile [BHP] constitutes the first and founding step in a development process for the automatic generation and verification of controllers derived from a domain model as outlined in [HP03a, HPG$^+$04].

The next section gives a brief introduction to the railway control domain terminology as background for the development of a profile. Section 3 explains first the basic concepts and techniques for the construction of a UML 2.0 profile, followed by selected examples of the RCSD profile. An example in Section 4 demonstrates the successful connection between the typical domain notation

and the conceptual view of the profile. In Section 5, we sketch the underlying mathematical model induced by a RCSD description of the physical model and give an overview on the development of the associated controller. We also indicated how this semantic model can be used for tool-supported verification and code generation.

2 Elements of the Railway Domain

Creating a domain specific profile requires identifying the elements of this domain and their properties as e.g. described in [Pac02]. We focus on the modeling of main tracks. All elements that are not allowed on main tracks as e.g. track locks are discarded. The further elements are divided into track elements, sensors, signals, automatic train runnings, and routes. Elements in the domain that come in different but similar shapes like signals are modeled as one element with different characteristics. In this way, we can abstract the railway domain to eight main modeling elements. These are described in the following:

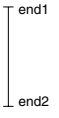

Fig. 1. Segment

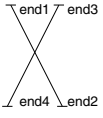

Fig. 2. Crossing

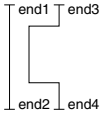

Fig. 3. Interlaced segments

Track Elements. The track network consists of segments, crossings, and points. Segments are rails with two ends (see Fig. 1), while crossings consist of either two crossing segments or two interlaced segments (see Fig. 2 and Fig. 3). In general, the number of trains on a crossing is restricted to one to ensure safety. Points allow a changeover from one segment to another one. We use single points with a stem and a branch (see Fig. 4). There is no explicit model element for double points, as these are seldom used in practice. If needed, they can be modeled by two single points. Single slip points and double slip points are crossings with one, respectively two, changeover possibilities from one of the crossing segments to the other one (see Fig. 5 and Fig. 6). All points have in common that the number of trains at each point in time is restricted to one.

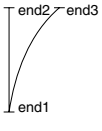

Fig. 4. Single point

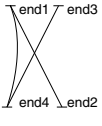

Fig. 5. Single slip point

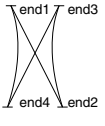

Fig. 6. Double slip point

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. Each sensor is the exit sensor of one track element and the entry sensor of the following one. If the track elements can be used bidirectionally, another sensor is needed that works vice versa.

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 Running. 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 Definition. As sensors are used as connection 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. Some conflicts occur as the required point positions or signal settings are incompatible. Another problem are routes that cross and are potentially safety-critical.

3 The UML 2.0 RCSD Profile

The next step is tailoring the UML 2.0 to the railway domain to provide the previously identified elements of the domain. There are two approaches to achieve this goal. The first one is using the UML 2.0 profile mechanism described in [OMG04] and [OMG05b] that allows for: (1) introducing new terminology, (2) introducing new syntax/notation, (3) introducing new constraints, (4) introducing new semantics, and (5) 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. The second approach is adapting the UML 2.0 metamodel to the needs of the railway domain by using MOF 2.0 (see [OMG06]). This approach offers more possibilities as elements can be added to or removed from the metamodel, syntax can be changed, etc. In fact, a new metamodel is created that is based on UML but is not UML anymore.

We have chosen the first approach - defining a UML 2.0 profile - as this supports exactly the features we need: the elements of the railway domain are new terminology that we want to use as modeling elements. To simplify communication between domain specialists and system developers, the usual notation of the railway domain should be used in a defined way. Therefore, constraints are needed to determine the meaning of the new elements. Track networks described with the new profile are transfered to transition systems. This is done

by transformation rules. Also, we have valid UML models and therefore various tool support.

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. The RCSD profile uses either *Class*, *Association*, or *InstanceSpecification* as basis of stereotypes. In addition, new primitive datatypes and enumerations can be defined as necessary.

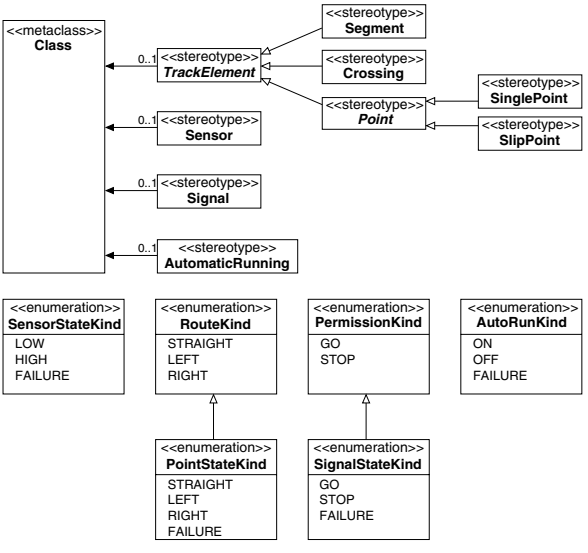

Fig. 7. Network elements part of the RCSD profile

Unfortunately, defining eight stereotypes as suggested by the domain analysis in Sec. 2 is not sufficient. New primitive datatypes and enumerations are needed to model attributes adequately. Special kinds of association are needed to model interrelationships between stereotypes. Furthermore, UML supports two modeling layers, i.e. the model layer itself (class diagrams) and the instances layer (e.g. 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. They consist of the same components but with different characteristics. Second, object diagrams show explicit track layouts for such a model. Here, the symbols of the railway domain have to be used. We need stereotypes on the object level to define these features. For these reasons, the RCSD profile is structured in six parts: the definition of primitive datatypes, network elements on class level, associations between these elements, network elements and associations on object level, routes, and top-level constraints.

Defining new primitive types is the easiest part. New datatypes must be identified and their range of values specified. In our case, these are identifiers for all controllable elements, identifiers for routes (e.g. to specify conflicting ones), time

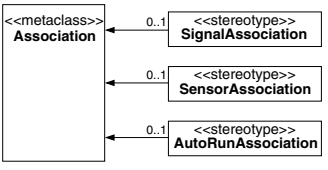

Fig. 8. Associations part of the RCSD profile

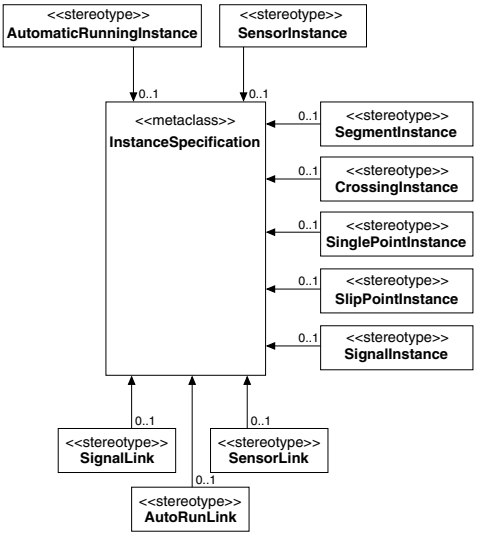

Fig. 9. Instances of network elements and associations part of the RCSD profile

instants and durations. All of them have in common that the value domain is $\mathbb{N}$. Nevertheless, defining different datatypes is important as we have to consider constraints like: all signal identifiers are unique, all point identifiers are unique and so on.

The next part of the profile defines all track network elements, i.e. segments, crossing, points, signals, sensors, and automatic train runnings (see Fig. 7). *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*. All elements are equipped with a set of constraints that define which properties each element must support and how it is related to other elements.

To give an example, each *TrackElement* has at least two ends that are connected to at most two *Sensors*, one entry sensor and one exit sensor. The number of sensors depends on the function of the track element, i.e. if it is used uni- or bidirectionally or if it is a sink or source of the track network. As properties, the maximal number of trains allowed on the element at one point in time and - optional - fixed speed limits are needed. These features are defined by OCL 2.0 (see [OMG05a]) constraints that each *TrackElement* in a model must fulfill.

```
(ownedAttribute->one(a1 |
  a1.name='limit' and
  a1.oclIsTypeOf(Integer) and
  a1.upperBound()=1 and
  a1.lowerBound()≥0 and
  a1.isReadOnly=true)) or
(not ownedAttribute->exists(a2 |
  a2.name='limit'))
```

```
(ownedAttribute->one(a1 |
  a1.name='e1Entry' and
  a1.oclIsTypeOf(Sensor) and
  a1.upperBound()=1 and
  a1.lowerBound()≥0 and
  a1.isReadOnly=true and
  a1.association.oclIsTypeOf
    (SensorAssociation))) or
(not ownedAttribute->exists(a2 |
  a2.name='e1Entry'))
```

We can see above two example constraints. The first one describes that each track element has an optional attribute *limit*. If this attribute exists, its type is *Integer*, its multiplicity is *0..1* or *1*, and its value is constant. Else, no attribute at all with this name exists to avoid confusion. The second example describes an attribute that will be modeled by an association in a class diagram. Each association goes hand in hand with an attribute at each navigable end. In this case, we have an optional association to a sensor where the associated property is *end1Entry*. The type of the attribute is *Sensor* as this is the type at the other end of the association which itself has the type *SensorAssociation* that is also defined in the profile. Again, the attribute has a constant value and either *0..1* or *1* as multiplicity.

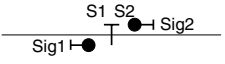

Fig. 10. RCSD notation

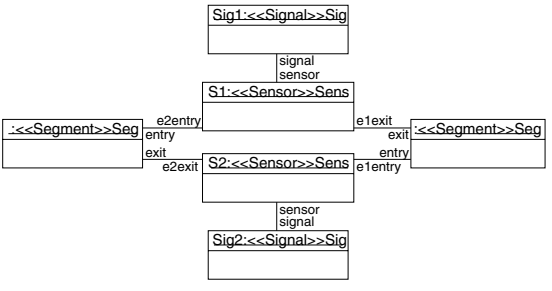

Fig. 11. UML notation

Such constraints describe the appearance of each stereotype. To give another example, *Points* have at least three ends with associated sensors and are not sinks or sources of the track network. They also have more attributes, i.e. *pointId*, *actualState*, *requestedState*, *requestTime* and *delta_p*. These are their identification number as points are controllable, the actual state that is either *STRAIGHT*, *LEFT*, *RIGHT*, or *FAILURE*, the requested state that is either *STRAIGHT*, *LEFT*, or *RIGHT*, the time of the last request, and the duration needed to switch the point after a request has been received. The possible values for the requested and actual state of the point are defined by *RouteKind* and *PointStateKind* as shown in Fig. 7. Other elements have similar required attributes and associations. An example model is shown in Sec. 4.

As associations *SensorAssociation* that connect track elements and sensors, *SignalAssociations* that connect signals and sensors, and *AutoRunAssociations* that connect automatic train runnings and sensors are used as shown in Fig. 8. Here, constraints are needed to determine the kind of stereotype at the ends of each association. Most important, two constraints of *SensorAssociation* describe that each sensor is the exit sensor of one track element and the entry sensor of the following one. In that way, routes can be defined as sequences of sensors.

For each non-abstract modeling element and each association, there exists a corresponding instance stereotype (see Fig. 9). Most important is the definition of domain-specific notation. Of course, usual UML notation can be used but is

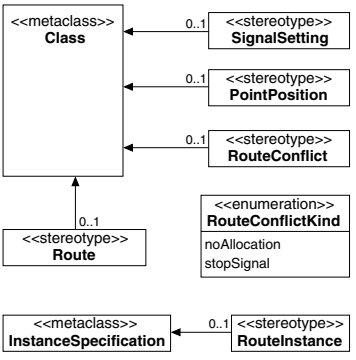

Fig. 12. Route definition part of the RCSD profile

infeasible as we can see in the direct comparison. In Fig. 10, two bidirectional segments connected by two sensors *S1* and *S2* are shown. Signal *Sig1* is associated to *S1*, signal *Sig2* is associated to *S2*. For comparison, the same constellation in object notation is given in Fig. 11.

Furthermore, the profile defines routes and their instances. Each *Route* is defined by an ordered sequence of sensors. Also, the signal setting for entering the route is given. Other properties are ordered sets of required point positions and of conflicts with other routes. The stereotypes to describe this information are given in Fig. 12. Again, constraints are used for unambiguous and strict definitions of attributes and suchlike.

The last part of the profile is a set of top-level constraints that describe interrelationships between stereotypes. The first example states that all point ids have to be unique. The second example describes that all sensors in route definitions really exists:

```
SinglePointInstance::allInstances()->
  collect(slots)->union
    (SlipPointInstance::allInstances()->
      collect(slots))->
select(s1 |
  s1.definingFeature.name='pointId'
  or
  s1.definingFeature.name='pointIdOpp')->
isUnique(s2 | s2.value)
```

```
def: r1:RouteInstance::allInstances()->
  collect(slots)->
    select(s2 |
      s2.definingFeature='routeDefinition')

def: r2:SensorInstance::allInstances()->
  collect(slots)->
    select(s1 |
      s1.definingFeature.name='sensorId')->
        collect(value)

r1.forAll(s3 | r2->including(s3.value))
```

4 Modeling with the RCSD Profile

The stereotypes and data types defined in the profile are used in UML diagrams. A class diagram is used to model a concrete problem in the railway domain, e.g. trams. The concrete track networks are object diagrams related to the class diagram.

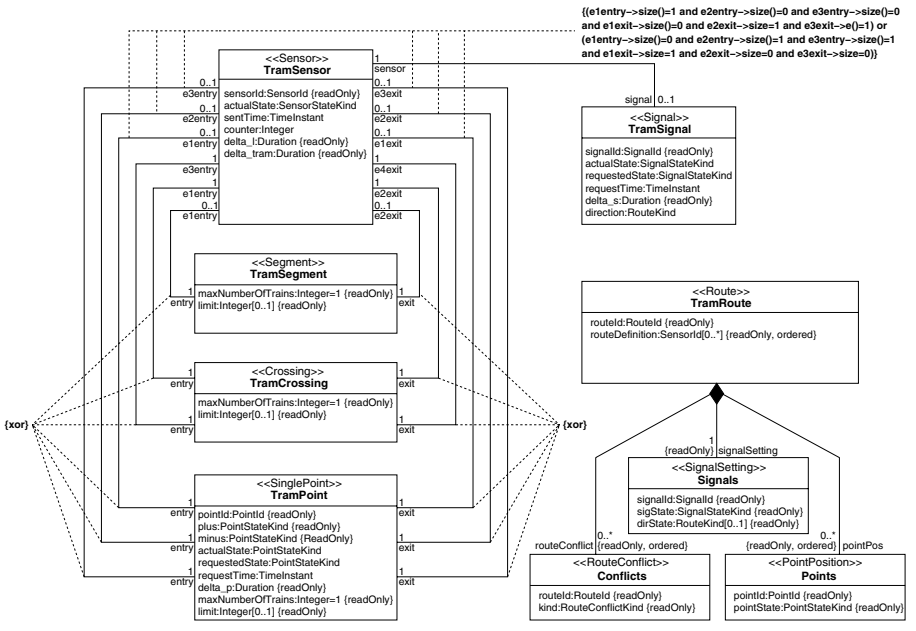

Fig. 13. Generic tram network

In our example, a tram track network is given in a class diagram as shown in Fig. 13. The interrelationships between the different stereotypes from RCSD are concretized for trams: there are no automatic running systems and no slip points, all segments are used unidirectionally, and signals do not use speed limits. The maximal number of trains allowed on each segment is 1. The network description of a concrete tram track network to be controlled is an *object diagram* that is based on the class diagram given above. An object diagram in RCSD profile notation is shown in Fig. 14.

In Fig. 15, a fragment of the same track network is shown in usual UML notation, i.e. an object diagram. Comparing the two figures, it is obvious that the RCSD profile notation is more comprehensible and therefore preferable in the communication process between domain experts and software designers.

5 Semantics

In this section we describe the behavior of the physical model – as captured in a RCSD object diagram – as a *Timed State Transition System* (TSTS). We extend this view by incorporating the behavior of a controller which has to guarantee safety conditions for the running system. The operations of this controller are given as generic patterns independent from the particular physical model. The composition of the controller and the individual domain of control should then allow only sequences of transitions which never violate a safety condition. Since

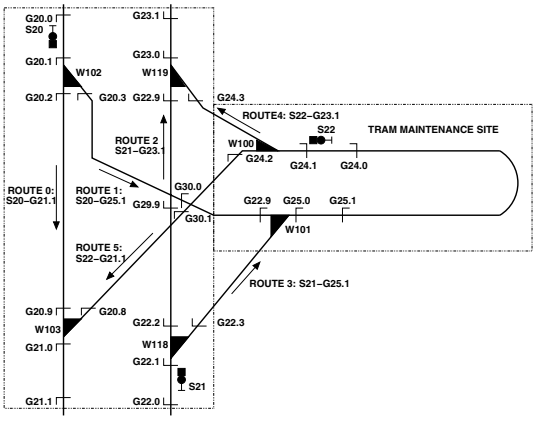

Fig. 14. Concrete tram network in profile notation

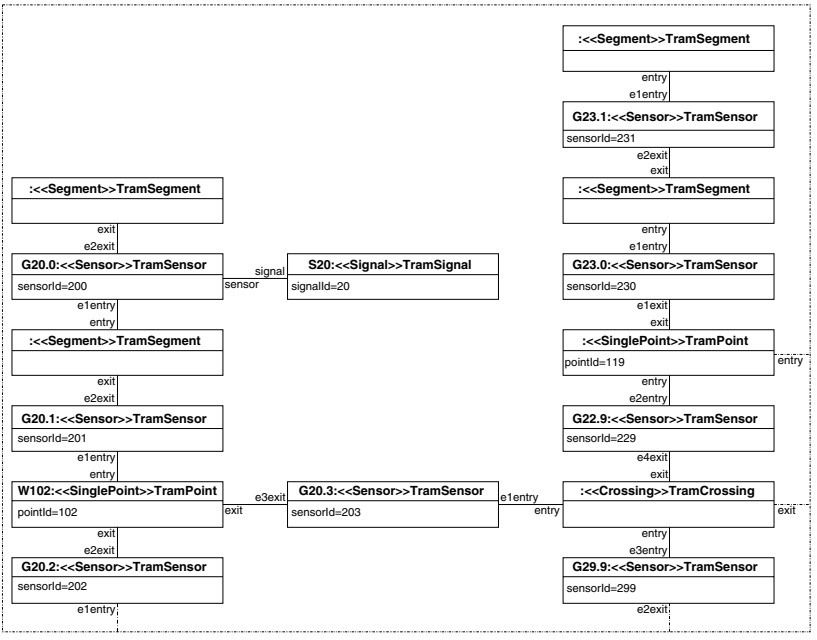

Fig. 15. Fragment of a concrete tram network in usual UML notation

we give a strict mathematical model of this composition, it can be proven by *bounded model checking* techniques.

As listed in Section 2, a physical rail network consists of a set of segments, a set of signals, and a set of sensors. Among the segments, the set of points and the set of crossings are of special interest for the formulation of safety conditions. Each element e in one of these sets owns some attributes according to the

profile definition, which we will denote by $var(e)$. Since we require that components fulfill some real-time restrictions, we use a distinguished variable $t \in \mathbb{N}$ to denote the actual time. The set of variables of a network is thus given by $VAR = \bigcup_e var(e) \cup \{t\}$, the union of the variables of all elements and t. As customary, a *state* $\sigma \in \Sigma : VAR \mapsto VAL$ is a type-consistent mapping from variables to values. A physical network thus induces a timed state transition system (Σ, σ_0, T) with $T \subseteq \Sigma \times \Sigma$ a set of transitions, and $\sigma_0 \in \Sigma$ an initial state, where all variables are given some default value. A transition $t = (\sigma, \sigma')$ will also be denoted by $\sigma \rightarrow \sigma'$. The state change of variables $\bar{x}$ to values $\bar{v}$ from σ to σ' is expressed by $\sigma' = (\sigma : \bar{x} \mapsto \bar{v})$[1].

The network description with its track segments and control elements exhibits some behavior for the following reasons: (1) Change of a sensor status due to a train passing. (2) Change of a component state according to its normal individual behavior, e.g. setting the state of a signal to the requested state (3) Changes due to decisions of the control component. We give some examples for the first two groups of state changes, an extended selection with in-depth explanations is listed in [HPG+04].

As trains themselves do not belong to the model, their presence is only noted via sensors, which change their *actualState* to $HIGH$ and record this in their attributes *sentTime* and *counter*. An example transition $\sigma \rightarrow \sigma'$ for a sensor *sen* would be enabled if $\sigma(sen.actualState) = LOW \wedge \sigma(t) > \sigma(sen.sentTime) + sen.delta\_tram$, i.e., if at least *sen.delta_tram* time units have passed since the last detection, modeling a mandatory minimal distance between trains to ensure individual detection. The resulting state is then $\sigma' = (\sigma : sen.actualState, sen.sentTime, sen.counter \mapsto HIGH, t, \sigma(sen.counter) + 1)$.

Typical transitions associated with some point p which represent its intended behavior are: (1) Change its *actualState* to meet a request. This is formalized by a rule $\sigma \rightarrow \sigma'$ requiring condition $\sigma(p.requestState) \neq \sigma(p.actualState) \wedge \sigma(p.requestTime) + \sigma(p.delta\_p) \leq \sigma(t)$, changing the state to $\sigma' = (\sigma : p.actualState \mapsto \sigma(p.requestState))$.(2) Fail to comply to a request in time. This is formalized by a rule $\sigma \rightarrow \sigma'$ requiring condition $\sigma(p.requestState) \neq \sigma(p.actualState) \wedge \sigma(p.requestTime) + \sigma(p.delta\_p) > \sigma(t)$, changing the state to $\sigma' = (\sigma : p.actualState \mapsto FAILURE)$. Similar transitions exist for signals. Additionally, we have a *clock rule*, which allows time to proceed, expressed by a transition $\sigma \rightarrow \sigma'$ with $\sigma' = (\sigma : t \mapsto \sigma(t) + 1)$.

Note that all transitions of the domain model have their effects restricted to the variables of one object, allowing to model the network in a compositional fashion as parallel composition of its constituents as illustrated in [HP03b]. Yet, all changes of the physical network, i.e. point positions, signal positions, and sensor states are covered by these transitions only.

The *controller* observes sensor state changes, deriving the current train locations within the network from them. Trains may issue requests to pass through the network on pre-defined routes. The controller issues commands to switch

[1] For brevity, we use a *simultaneous assignment* notation here which coincides with a reasonable granularity of operations which should be uninterruptible.

signal and point states and monitors the correct state of these track elements. The architectural model of the controller consists of three components [HP02]: (1) a *route dispatcher* which registers the requests of individual trains for specific routes and administers the allocation of these routes, (2) a *route controller* responsible for setting points and signals for active routes, and (3) a *safety monitor* checking for deviations of the actual from the expected state and subsequently ensuring a safe state.

These separated tasks of the controller are executed by sets of transition *patterns*, i.e. generic transitions which abstract from the concrete physical model. As an example, we have for each route r a transition for the route dispatcher component which checks whether r is requested, r is not yet scheduled for activation, and no conflicting routes are active or scheduled.

The state space Σ' of the controller is an extension of the state space Σ of the domain model as additional variables are needed, e.g. for the administration of routes some data structure is needed to store the current state of a route reservation. The controller model together with the domain model (Σ, σ_0, T) induces a TSTS (Σ', σ_0', T') where σ_0' is the extension of σ_0 by assigning appropriate initial values to the additional variables of Σ'. The set of transitions T' contains the set T and the set of transitions generated from the controller model and the domain data. As demonstrated in [HPG$^+$04], the concrete transitions are derived by instantiating the generic rules according to the physical layout of the network and the route definitions.

In an independent derivation, we formulate a set of safety conditions out of the physical model. These safety requirements are informally listed as: (1) No trains are driving in opposite directions on the same segment. (2) No trains are moving in opposite direction towards the same sensor. (3) The number of trains approaching a point from stem and branches at the same time is at most 1. (4) On each segment the number of trains passing in one direction is less than a predefined maximum. (5) There are no trains residing simultaneously on crossings. Observe that these requirements are formulated generically in terms of track elements only and that each condition involves only a very limited part of the network. Within the physical model, these requirements are instantiated to a set of constraints. As an example from Fig. 14, we take a closer look on the crossing between sensors $G20.3$ and $G30.0$, and respectively, between $G29.9$ and $G22.9$. Clearly, requirement (5) has to be satisfied for this track element. The fact that a train is present in a segment, can be deduced from a comparison of the counters of its entry and exit sensors: Assuming the default settings that no train is present in a segment and both sensors have the same value of their respective *counter* attribute, the predicate $\sigma(G20.3.counter) > \sigma(G30.0.counter)$ models the fact that at least one train is still located between the sensors $G20.3$ and $G30.0$. Consequently, requirement (5) can be formalized for this instance of a crossing as $\neg(\sigma(G20.3.counter) > \sigma(G30.0.counter) \wedge \sigma(G29.9.counter) > \sigma(G22.9.counter))$.

For a given network, we can thus automatically derive a TSTS as model which exhibits the behavior of this network under supervision of the controller

and a set of constraints which have to hold throughout all executions within our model. Analyzing whether a violation of safety requirements is possible amounts to a check whether some state is reachable that does not satisfy all constraints.

While we present a rather abstract mathematical model here, it is an easy task to come up with an equivalent model which encodes all transitions as guarded commands. Representation of timed state transition systems in a special programming language suited for further analysis then boils down to encode states and transitions into this language. This has been worked out in [HPG+04] for SystemC [GLMS02, MRR03], as input language for a proof by bounded model-checking that the safety requirements are satisfied. The SystemC code can be compiled into C/C++ code, leading to the generation of executable code for the controller. In a similar fashion, it is even possible to take the generated machine code and verify its correctness w.r.t. the requirements on the domain model by comparing the TSTS model of the network/controller model to a TSTS model of the machine code. This comparison is feasible, as the abstraction function from the data structures of the machine code to the data model of the domain description is total, and for all transition patterns of the above model a correct refinement in machine code exists.

6 Conclusions

We have presented the RCSD profile for UML 2.0 as suitable formalism to capture the domain specific requirements of the railway control systems domain. In particular, the feature to use a domain specific graphical notation for the domain description helps to bridge the gap between domain experts and developers of controller systems. As any model denoted in the RCSD profile formalism is still a UML model, standard tools which support the profile mechanisms can be used in the development process. Existing UML tools support the analysis of RCSD models as textual representations of a model serve as input for the generation of the TSTS model or the SystemC model. Another example would be the application of graph transformations to the RCSD object instance diagram for simulation purposes [GZ04].

We associate a formal behavioral model with RCSD descriptions as any object diagram induces a timed state transition system, which covers the behavior of the domain of control. Together with the utilization of design patterns for the controller component, we generate a model which serves as foundation for the formal verification of this model w.r.t. a set of safety requirements. In addition, the TSTS model serves as reference for the verification of executable code for the controller. A major advantage of using the transition system model as the semantical model of a RCSD description lies in the fact that various concrete programming languages as well as other formal specification languages can use the transition system model as reference.

References

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

[FKvV98] W. J. Fokkink, G. P. Kolk, and S. F. M. van Vlijmen. EURIS, a specification method for distributed interlockings. In *Proceedings of SAFECOMP '98*, LNCS, volume 1516, pp. 296–305. Springer-Verlag, 1998.

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

[GZ04] M. Gogolla and P. Ziemann. Checking BART Test Scenarios with UML's Object Constraint Language. Formal Methods for Embedded Distributed Systems - How to master the complexity. F. Kordon, M. Lemoine (Eds.), Kluwer, Boston. pp. 133-170, 2004.

[HP02] A.E. Haxthausen and J. Peleska. A Domain Specific Language for Railway Control Systems. In *Proceedings of the Sixth Biennial World Conference on Integrated Design and Process Technology, (IDPT2002), Pasadena, California*, June 23-28 2002.

[HP03a] A. E. Haxthausen and J. Peleska. Automatic Verification, Validation and Test for Railway Control Systems based on Domain-Specific Descriptions. In *Proceedings of the 10th IFAC Symposium on Control in Transportation Systems*. Elsevier Science Ltd, Oxford, 2003.

[HP03b] A. E. Haxthausen and J. Peleska. Generation of Executable Railway Control Components from Domain-Specific Descriptions. In *Proceedings of the Symposium on Formal Methods for Railway Operation and Control Systems (FORMS'2003), Budapest/Hungary*, pp. 83–90. May 15-16 2003.

[HPG+04] A.E. Haxthausen, J. Peleska, D. Große and R. Drechsler. Automated Verification for Train Control Systems. Proceedings of Symposium FORMS/FORMAT 2004, Braunschweig, Germany, 2nd and 3rd Dec. 2004.

[Hun06] H. Hungar. UML-basierte Entwicklung sicherheitskritische Systeme im bahnbereich. In *Dagstuhl Workshop MBEES - Modellbasierte Entwicklung eingebetteter Systeme*, Informatik Bericht, pp. 63–64, TU Braunschweig, Jan 2006.

[MRR03] W. Müller, J. Ruf, and W. Rosenstiel. *SystemC – Methodologies and Applications*, chapter 4, pp. 97–126. Kluwer Academic Publishers, 2003.

[OMG04] Object Management Group. Unified Modeling Language (UML) Specification: Infrastructure, version 2.0. http://www.omg.org/docs/ptc/04-10-14.pdf, October 2004.

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

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

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

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

[PBH00] J. Peleska, A. Baer, and A. E. Haxthausen. Towards Domain-Specific Formal Specification Languages for Railway Control Systems. In *Proceedings of the 9th IFAC Symposium on Control in Transportation Systems 2000, June 13-15, 2000, Braunschweig, Germany*, pp. 147–152, 2000.

[Rai] A grand challenge for computing science: Towards a domain theory of railways. http://www.railwaydomain.org.

[RJB04] J. Rumbaugh, I. Jacobson, and G. Booch. *The Unified Modeling Language – Reference Manual, 2nd edition*. Addison-Wesley, July 2004.