TRAMWAY NETWORKS AS ROUTE GRAPHS

Klaus Lüttich, Bernd Krieg-Brückner, Till Mossakowski
Universität Bremen
Address: P.O.B 33 04 40, D-28334 Bremen, Germany
Phone: +49-421-218-7129, Fax: +49-421-218-3054, E-Mail: {luettich,bkb,till}@informatik.uni-bremen.de

Abstract: The purpose of this paper is threefold:
(1) to show a methodology for introducing formality step-by-step, starting from a "light-weight" ontology,
(2) to apply it to the generic Route Graph model, instantiated here to Tramway Networks, and
(3) to demonstrate that separation of concerns into layers of abstraction and application of generic formal notions such as graph abstraction lead to a useful decomposition of safety properties.
Keywords: Tramway, Route Graph, Safety, Ontology, Common Algebraic Specification Language (CASL)

1. INTRODUCTION

Route Graphs have been introduced as a general concept (Werner *et al.*, 2000), to be used for navigation by several agents in a variety of scenarios such as humans using railway, underground, road or street networks, as travellers, car drivers or pedestrians. Each application scenario or kind of Route Graph will introduce special attributes on the general structure in a hierarchy of refinements.

Tramway Devices and Networks. In this paper the Route Graph concepts are used to model a tramway network as described in (Haxthausen and Peleska, 2002) and its static safety requirements. Their example of a Tramway Network Description in Fig. 1 corresponds to the extract from a *TramDevices Route Graph* in Fig. 2. In addition to this detailed Devices level we will model an abstraction, the *TramNetwork Route Graph* which only contains tram sensors, signals and other user information, and the relevant tracks to tackle exclusion conflicts. Thus different levels of detail are separated out; safety requirements are expressed at a high level of abstraction.

Abstraction and Refinement. The two levels are related by an abstraction relation – in fact the abstract TramNetwork Route Graph can be derived automatically from the TramDevices Route Graph, which might in turn be derived from a concrete layout in a CAD representation. Alternatively, planning of a Tramway Network may start at the abstract level, where safety requirements may be checked for an abstract layout, and further development may then proceed by automated refinements, introducing detail as required.

Ontologies. The terminology and knowledge represented at the distinct levels of TramDevices and TramNetwork is different. As we are dealing with abstract concepts and interrelations between them, and want to standardise or mediate between different uses of terminology, we are using an *ontology* as *the* central definitional approach and data structure for Route Graphs. An ontology may provide a first "light-weight" view of the concepts involved; in the future, the ontology for the eventual tram driver containing information about maximally allowed speed etc. might even be separated out into yet another ontology level for a specialised purpose; similarly tram users would be interested in derived information about schedules, etc. (not considered here).

We intend to show that an ontological approach is suitable for both: to define the generic concept of Route Graphs, and to instantiate it for a particular scenario in detail, leading to a concrete data structure. Moreover, the example of Route Graphs demonstrates that adequate *formalisation of ontologies* can be introduced step by step in this process, cf. also (Krieg-Brückner *et al.*, 2004*c*).

Structure of the Paper. In Sect. 2. we introduce the basic notions of Route Graphs, followed by a "light-weight" formalisation using ontologies in Sect. 3.. In Sect. 4., these concepts are formally specified in CASL and, in Sect. 5., instantiated to tramways at two levels of abstraction: the *TramDevices level*, introducing concepts such as points, sensors and signals as nodes, and the *TramNetwork level*, which abstracts away from segments between points to track segments between simple sensors and junctions including signals. Finally, we show in Sect. 5.4 that abstraction leads to a useful separation of concerns to tackle various safety properties.

2. ROUTE GRAPHS

2.1 Sample Scenarios

Fig. 3 shows some sample navigation scenarios. We may distinguish between navigation in systems

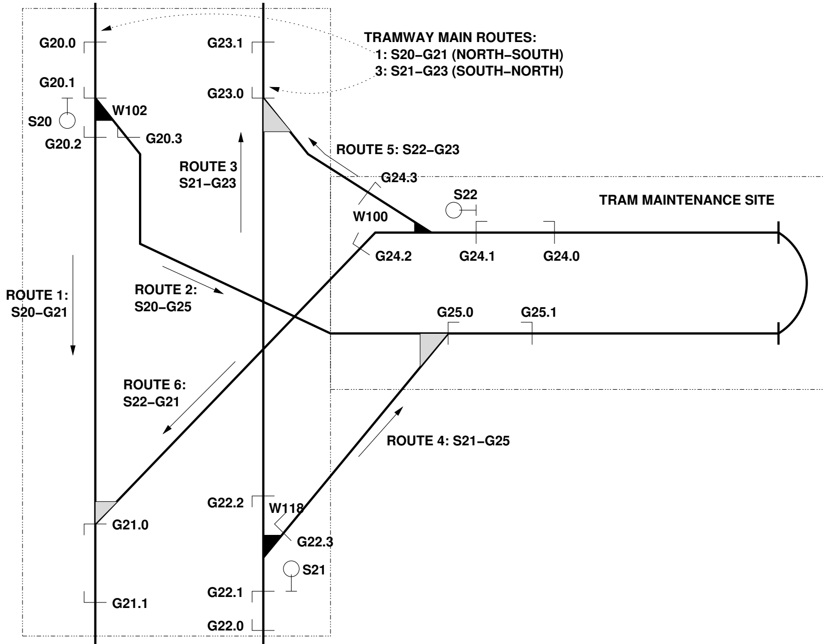

Fig. 1. Tramway Network Description

of passages (e.g. road networks or corridors) or in areas of open space (e.g. on a lake; on a market place, surrounded by buildings; in a hall); In the former, our course is more or less centred within enclosing borders (e.g. curbstones, walls) and guided by routemarks along the way; in the latter the course is given by a vector to the target and we are guided by global landmarks (e.g. a lighthouse, the sun or a church's spire), cf. (Krieg-Brückner *et al.*, 1998).

While the concept of *Route Graph*s was originally introduced to mediate terminology between artificial intelligence and psychology in spatial cognition (Werner *et al.*, 2000), scenarios are not restricted to human users. Route Graphs are also intended for interaction between service robots, such as the Bremen autonomous wheelchair Rolland, and their users, as well as between robots, for example as a compact data structure for online communication in an exploration scenario (Krieg-Brückner *et al.*, 2004*c*; Krieg-Brückner *et al.*, 2004*b*; Ross *et al.*, 2004). Others are applying the Route Graph concepts to public transportation networks (Timpf, submitted).

In this paper, we will show the application to Tramway Networks. The tramway scenario poses similar problems in human-machine interaction as the service robot scenario, such as the shared control problem described e.g. in (Krieg-Brückner *et al.*, 2004*b*; Ross *et al.*, 2004) between driver and automated system. Here we will concentrate on the network itself and its safety, based on the approach by (Haxthausen and Peleska, 2002).

We will now briefly introduce the general concepts of Route Graphs; more detail will follow in Sect. 3..

2.2 Segments

An edge of a Route Graph is directed from a source node to a target node. We call an edge a (route) *segment* when it has three additional attributes: an *entry*, a *course* and an *exit*. Exactly what information is associated with these attributes is specifically defined for each Route Graph instantiation of a particular *kind*. For example, an entry to a highway may denote a particular ramp, an exit another, while the course is just characterised by a certain length. Additional information may be added, e.g. that the course has three lanes. As another example, the entry and course for a boat route segment may be given as a vector in geo-coordinates, while

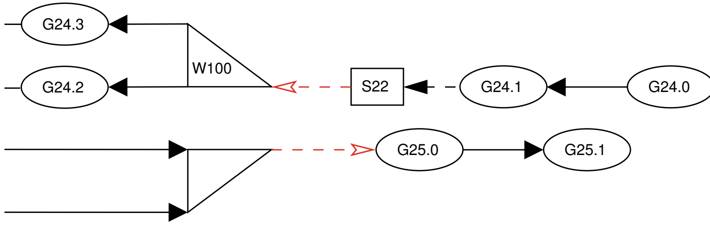

Fig. 2. TramDevices Example Route Graph (Extract)

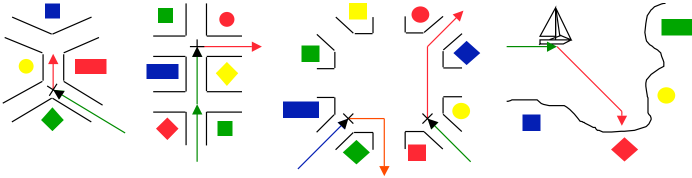

Fig. 3. Examples of Routes

the exit into a harbour may specify a particular orientation along a quay.

For tramways, the course is e.g. associated with the allowed speed; the entry at a "decision point" is guarded by a signal and requires correct setting of the point; an exit requires particular attention on behalf of the tram driver when different incoming track segments (and trams!) join and right-of-way may have to be considered (this is the case in Bremen at least).

2.3 Places

A node of a Route Graph, called a *place*, corresponds to a decision point between branches. It has an *origin* with a particular position and orientation. Thus each node has its own "reference system"; it may, but need not, be rooted in a (more) global reference system, such as a 3D geo-system. The origin becomes particularly important in a union of routes or Route Graphs, when *place integration* is required; cf. different origins (denoted by little crosses, with a direction) for different routes in Fig. 3, c – these would have to be integrated into one.

The example of tramways shows that a global reference system is not really required; more important for the driver are the course, i.e. the egocentric scenario along the track, as well as the signals, the setting of the points, etc.

3. ONTOLOGY OF ROUTE GRAPHS

Ontologies. In an ontology for Tramway Networks and Route Graphs, the conceptual terminology is modelled by classes, their interrelation by (declarations of) relations, and the knowledge contained in actual networks by objects of these classes, related by applications of the relations. We refer to such ontologies as "light-weight" ontologies since they offer a first overview of the concepts, "unburdened" by formalisation; they will be presented as diagrams in this paper.

Fig. 4 shows the taxonomy and some relations of an ontology extract for graphs, cf. (Krieg-Brückner *et al.*, 2004*c*). Boxes denote classes, and arrows with hollow tips denote the *subclassOf* ("is a") relationships between classes. The other arrows denote either binary relations or unary functions. Objects of such classes could be added in a separate diagram.

Fig. 4 introduces the notion of a (generic) *Graph* related to its *Edges* and *Nodes* by *hasEdge* and *hasNode*. Each *Edge* is related by a function to its *source* and *target Node*. *Sequence_Edge* has the partial functions *head* and *tail*. *Path* is a specialisation of *Sequence_Edge*, and *Route* of *Path* (cf. Fig. 5 for the axioms in CASL).

Note that, at the level of an ontology specification, the formalisation is often different from

a conventional approach in mathematics or computer science, i.e. in specification or programming languages: The latter specification formalisms would introduce a graph as a pair of a set of nodes and a set of edges; in the ontology approach only the observation that a graph is related to its components is given: a graph is a set of edges and nodes. This approach seems more natural when only terminology is introduced; the underlying data structure is not relevant yet and will be introduced in a later development step, cf. (Krieg-Brückner *et al.*, 2004*c*).

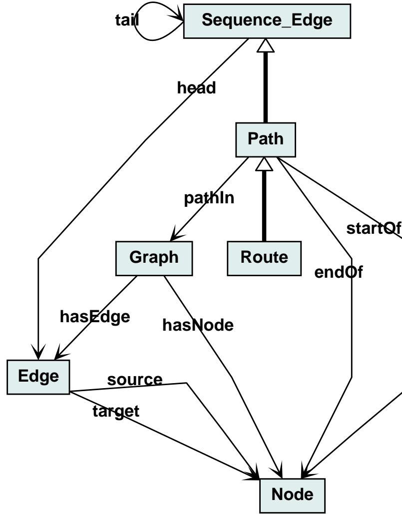

Fig. 4. Simple Graph Ontology (extract)

4. FORMALISATION IN CASL

4.1 CASL

The *Common Algebraic Specification Language*, CASL, is an international development of the "Common Framework Initiative" of IFIP WG1.3 "Foundations of System Specification" (Astesiano *et al.*, 2002; Mosses, 2004). It has been officially approved by IFIP WG1.3 and thus serves as a de-facto standard for algebraic specification.

CASL covers full First-Order Logic (FOL) plus predicates, partial and total functions, and subsorting. *Basic Specifications* consist of declarations of sorts, operations (ops), and predicates (preds), along with definitions of types, abbreviating standard data structures, and FOL axioms. *Structured Specifications* allow the reuse of named specifications and the instantiation of parameterised specifications, combined with various ways for combining and extending specifications ("**then**"), hiding, revealing and renaming of symbols ("**with**"). Specifications are collected in named *libraries* for reuse in other specification contexts, e.g. the extensive library of predefined Basic Specifications.

```
spec GENGRAPH [sort Kind] =
      sorts Graph[Kind], Node[Kind], Edge[Kind]
      preds hasNode : Graph[Kind] × Node[Kind];
            hasEdge : Graph[Kind] × Edge[Kind]
      ops   source, target : Edge[Kind] → Node[Kind]
      ∀ e: Edge[Kind]; s, t: Node[Kind]; g: Graph[Kind]
      • hasEdge(g, e) ∧ source(e) = s ∧ target(e) = t ⇒
          hasNode(g, s) ∧ hasNode(g, t)
then GENSEQUENCE [sort Edge[Kind]
                      fit Elem ↦ Edge[Kind]]
      with head, tail, EmptySeq, freq
then GENSEQUENCE [sort Node[Kind]
                      fit Elem ↦ Node[Kind]]
then ops   sources, targets : Sequence[Edge[Kind]] →
          Sequence[Node[Kind]]
      ...
      pred  connected(l: Sequence[Edge[Kind]]) ⇔
            (∀ e1, e2: Edge[Kind]
             • head(l) = e1 ∧ head(tail(l)) = e2 ⇔
               target(e1) = source(e2)) ∧
               connected(tail(l))
      sorts Path[Kind] =
              {l: Sequence[Edge[Kind]] • connected(l)};
            Route[Kind] =
              {p: Path[Kind]
               • (∀ n: Node[Kind]
                  • freq(sources(p), n) ≤ 1) ∧
                 (∀ e: Edge[Kind] • freq(p, e) ≤ 1)}
      ops   startOf(l: Sequence[Edge[Kind]]):
                      Node[Kind] =
              source(head(l));
            endOf(l: Sequence[Edge[Kind]]):
                      Node[Kind] =
              target(last(l))
      ...
```

Fig. 5. Generic Graph Ontology

For the formalisation of ontologies, (sub)sorts are used to form the taxonomic hierarchy instead of untyped unary predicates for the subclassOf relationship; binary relations are extended to n-ary relations, if required, and represented as predicates or functions as appropriate; FOL axioms formalise the relations (Lüttich and Mossakowski, 2004).

4.2 Route Graphs in CASL

We start the formalisation in CASL at the level of the ontology in Fig. 4 with a loose (requirement) specification of generic graphs, see Fig. 5. The specification includes axioms that determine the properties of subsorts of sequences of edges by predicates and relations. Paths in Route Graphs are defined in a weaker way than paths in ordinary graph theory: an edge may be included more than once in a path. Routes have this restriction, and branching is not allowed. The instantiations below are all based on this generic specification.

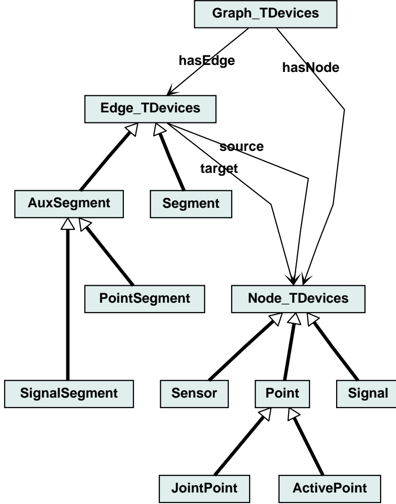

Fig. 6. TDevices Route Graph Ontology

The next step would be the development of a more data type oriented specification. There the Route Graph and its auxiliary data structures like lists of segments and places are based on the basic libraries of CASL approved by CoFI. This implies a proof obligation that the axioms of the loose ontology (requirement) specification still hold in the data type oriented specification. In a third step a Haskell programm may be derived from the inductive definitions found in the second specification, see (Krieg-Brückner *et al.*, 2004*c*).

5. INSTANTIATIONS FOR TRAMWAYS

The different aspects of Tramway Networks are separated into two layers of Route Graphs, related by an abstraction function. The detailed layer is called TRAMDEVICES_ROUTEGRAPH and is discussed in Sect. 5.1. The abstract layer TRAMNETWORK_ROUTEGRAPH only contains sensors (possibly with signals) and segments between them; it is described in Sect. 5.2. The abstraction function between these graphs is presented in Sect. 5.4. Finally, route conflicts, signal settings and point positions are defined as predicates over routes in the TRAMNETWORK_ROUTEGRAPH in Sect. 5.5.

5.1 TramDevices Route Graph

As an introduction to the terms used in this section, Fig. 6 displays the taxonomy and some relations. Fig. 7 shows the formalisation of the TramDevices Route Graph in CASL based on the instantiation of the GENGRAPH specification (cf. Fig. 5). Part of the specified Route Graph is visualized in Fig. 2.

Signals, points and sensors form the nodes of the TramDevices Route Graph. They are declared as subsorts (subconcepts) of *Node*[*TDevices*]. Points are further specialized into *JointPoint* and *ActivePoint*; only the latter have to be controlled for the selection of a route.

```
spec TRAMDEVICES_ROUTEGRAPH =
  GENGRAPH [sort TDevices]
then
  sorts ActivePoint, JointPoint, Sensor, Signal,
        AuxSegment
  generated types Point ::=
                    sorts ActivePoint, JointPoint ;
                Node[TDevices] ::=
                    sorts Sensor, Signal, Point
  free types Edge[TDevices] ::=
                sorts Segment, AuxSegment;
             AuxSegment ::=
                sorts SignalSegment, PointSegment
  sorts SignalSegment =
         {e: Edge[TDevices]
          • source(e) ∈ Sensor ∧ target(e) ∈ Signal};
        PointSegment =
         {e: Edge[TDevices]
          • (source(e) ∈ JointPoint ∧
              target(e) ∈ Sensor) ∨
            (source(e) ∈ Signal ∧
              target(e) ∈ ActivePoint)};
        Segment =
         {e: Edge[TDevices]
          • (source(e) ∈ Sensor ∧
              (target(e) ∈ Sensor ∨
                target(e) ∈ JointPoint)) ∨
            (source(e) ∈ ActivePoint ∧
              target(e) ∈ Sensor)}
  free type ActivePointState ::= straight | left | right
  sorts Entry[TDevices], Course[TDevices]
  ops   entry : Edge[TDevices] → Entry[TDevices];
        pointDirection : Entry[TDevices] →?
                  ActivePointState;
        course : Edge[TDevices] →? Course[TDevices];
        maxSpeed, duration : Course[TDevices] → Nat
  ∀ e: Edge[TDevices]
  • (e ∈ Segment ∧ source(e) ∈ ActivePoint) ⇔
    def pointDirection(entry(e as Edge[TDevices]))
  • def course(e) ⇔ e ∈ Segment
  pred  __crosses__ : Edge[TDevices] × Edge[TDevices]
  ∀ e1, e2: Edge[TDevices]
  • e1 crosses e2 ⇒ e2 crosses e1
  • ¬ e1 crosses e1
end
```

Fig. 7. TramDevices Route Graph in CASL

Edge[*TDevices*] is specialised into subsorts: *SignalSegment* and *PointSegment* are auxiliary edges to connect the devices needed along a track; *Segment* may carry additional information, e.g. an allowed maximal speed and a duration associated with the course along the segment (not shown here). All subsorts of *Edge*[*TDevices*] are defined in terms of their allowed sources and targets. Only *Segment*s originating from an *ActivePoint* have an *Entry* that carries the information in which direction the point has to be switched before the segment may be entered. An exit before a *JointPoint* requires particular attention on behalf of the tram driver (at least in Bremen) for the potential right-of-way of incoming trams on other incoming track segments; this is not considered here.

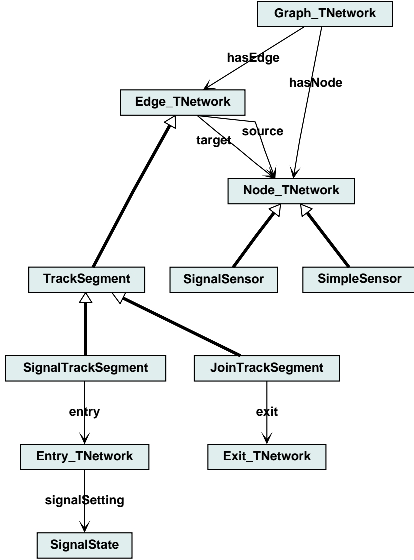

Fig. 8. TNetwork Route Graph Ontology

5.2 TramNetwork Route Graph

An abstract Route Graph layer on top of the TramDevices Route Graph models only *TrackSegment*s and sensors (corresponding to *Node*[*TNetwork*]), divided in two groups (cf. Fig. 8 and Fig. 9):

- *SimpleSensor*s to recognise passing trams;
- *SignalSensor*s with additional information about signals.

TrackSegment has two specialisations:

- a *SignalTrackSegment* (denoted by a dashed/dotted purple arrow) originating from *SignalSensor* places with an *Entry* that denotes the *SignalState* setting needed to enter this *SignalTrackSegment*;
- a *JoinTrackSegment* (denoted by a dashed green arrow) with an *Exit* that draws attention to the potential right-of-way of incoming trams on other incoming *JoinTrackSegment*s, see the TramDevices Route Graph description above.

Information attached to a course is as above.

A *TramRoute* is a specialised *Route* that starts at a *SimpleSensor* followed by a *SignalSensor* (cf. (Haxthausen and Peleska, 2002) for this definition).

```
spec TRAMNETWORK_ROUTEGRAPH =
     GENGRAPH [sort TNetwork]
     with Edge[TNetwork] ↦ TrackSegment
then
     sorts SignalSensor, SimpleSensor
     free type Node[TNetwork] ::=
             sorts SignalSensor, SimpleSensor
     sorts TramRoute =
             {r: Route[TNetwork]
              • startOf(r) ∈ SimpleSensor ∧
                target(head(r)) ∈ SignalSensor ∧
                (∀ e: TrackSegment
                 • hasElem(tail(r), e) ⇒
                   source(e) ∈ SimpleSensor ∧
                   target(e) ∈ SimpleSensor)};
           SignalTrackSegment, JoinTrackSegment <
              TrackSegment
     free type SignalState ::= straight | left | right | stop
     sort  Entry[TNetwork], Exit[TNetwork]
     ops   entry : SignalTrackSegment →
                     Entry[TNetwork];
           exit : JoinTrackSegment → Exit[TNetwork];
           signalSetting : Entry[TNetwork] →
                     SignalState
     preds __crosses__ : TrackSegment × TrackSegment;
           __crossingConflictWith__,
           __exclusionConflictWith__:
              TramRoute × TramRoute
     ∀ e, e′: TrackSegment; r1, r2: TramRoute
     • e crosses e′ ⇒ e′ crosses e
     • ¬ e crosses e
     • r1 crossingConflictWith r2 ⇔
       ∃ n_e1, n_e2: TrackSegment
       • hasElem(r1, n_e1) ∧ hasElem(r2, n_e2) ∧
         n_e1 crosses n_e2
     • r1 exclusionConflictWith r2 ⇔
       (target(head(r1)) = target(head(r2)) ∨
        last(r1) = last(r2)) ∧
       ¬ r1 = r2;
end
```

Fig. 9. TramNetwork Route Graph in CASL

5.3 Abstraction

For the abstraction of more detailed Route Graphs to simplified ones, two generic abstraction functions are specified in CASL, cf. Fig. 13. Sequences of edges without branches may be simplified to a single edge by *simplyifyEdges*. A weak surjective graph homomorphism (*abstractPlaces*) is provided to abstract nodes with a common feature to a single node; such a common feature could be “nodes in the same area”. It is weak in the sense that all nodes of the source graph are mapped into the abstracted node, but only those edges are mapped that have source and target nodes that are mapped to different abstracted nodes. Note that attributes attached to edges of the graph to be abstracted are only considered in the mapping if they are attached to edges that are mapped.

As a refinement, one might want to insist that for every incoming edge into the graph to be abstracted there is a path to every outgoing edge; these edges will correspond to incoming and outgoing edges of the abstract node.

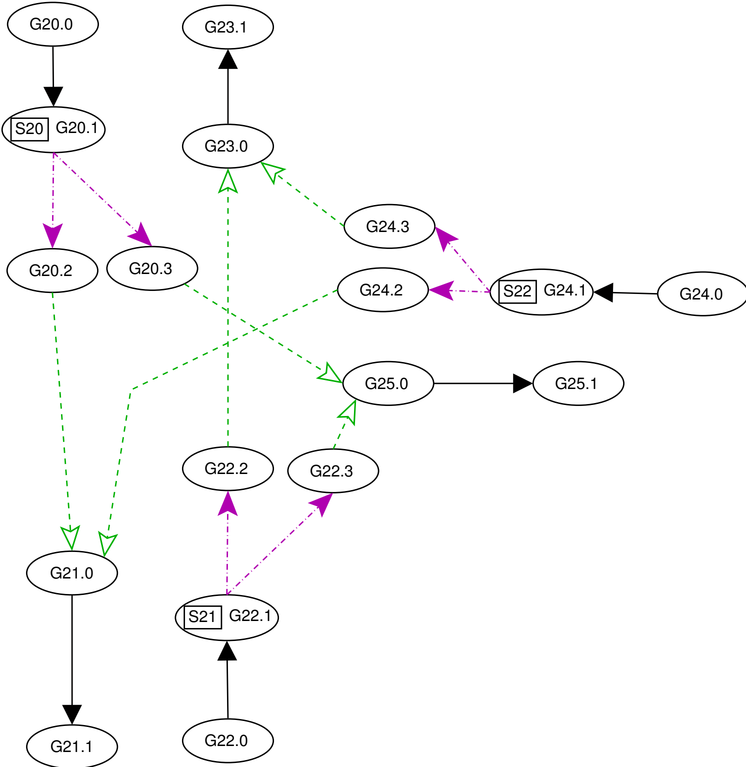

Fig. 10. Route Graph for the TramNetwork Example

5.4 Abstraction of TramDevices Route Graph

The TramNetwork Route Graph of *TrackSegment*s can be derived from the TramDevices Route Graph using the abstraction functions provided above. This is shown for an example in Fig. 11 in an informal way with dashed/dotted lines. Fig. 12 shows the ontology with abstraction relations concerning nodes. In an instantiation of MAPNODE (cf. the CASL specification in Fig. 14), *mapNode* and *mapEdge* are refined in a domain-specific way. This generates a proof obligation that the axioms of MAPNODE still hold.

5.5 Safety Predicates on Routes

This section describes the safety predicates in the Route Graph specifications of the previous sections, cf. Fig. 7 and Fig. 9.

In Fig. 7, an infix predicate *crosses* is very loosely specified on a pair of *Edge*[*TDevices*]s. A proper definition would have to be based on some at least qualitative if not quantitative spatial relations between their source and target nodes – for example, using the dipole calculus (Dylla and Moratz, 2004). The information about actual crossings in a TramDevices Route Graph might be derived from a detailed CAD description of the actual spatial layout. It is carried over to a corresponding overloaded predicate *crosses* on *TrackSegment*s in the abstracted TramNetwork Route Graph by the abstraction functions, cf. Fig. 14.

Alternatively, the *crosses* predicate may be defined directly in the abstract TramNetwork Route Graph when working in the opposite direction by refinement; in this case, the abstract spatial layout must be sufficiently well-defined to establish the *crosses* relation.

On this basis, the predicate *crossingConflictWith* may be defined as the first static safety predicate, see Fig. 9. It is defined on two *TramRoute*s and holds if two *TrackSegment*s exist, one in each route, which are in the *crosses* relation.

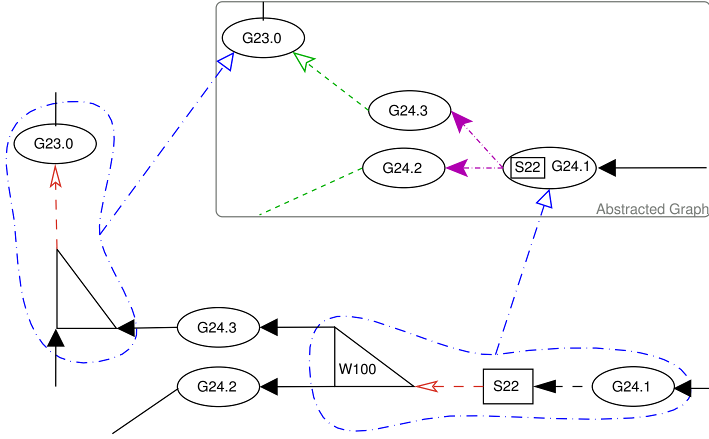

Fig. 11. Abstraction from TramDevices Route Graph (Example)

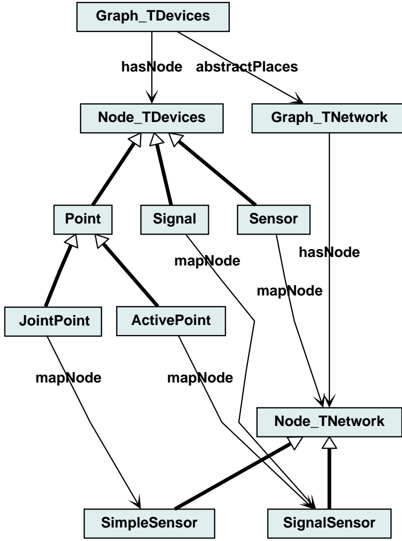

Fig. 12. Abstraction Relations Ontology (extract)

```
spec PLACEABSTRACTION [GENGRAPH [sort Kind1]]
                      [GENGRAPH [sort Kind2]]
   given NAT =
  ops   abstractPlaces : Graph[Kind1] ⟶ Graph[Kind2];
        mapNode : Node[Kind1] ⟶ Node[Kind2];
        mapEdge : Edge[Kind1] ⟶? Edge[Kind2]
  ∀ e_K1: Edge[Kind1]; e_K2: Edge[Kind2];
    n_K1: Node[Kind1]; n_K2: Node[Kind2];
    g_K1: Graph[Kind1]; g_K2: Graph[Kind2]
  • def mapEdge(e_K1) ⇔
    ¬ mapNode(source(e_K1)) = mapNode(target(e_K1))
  • g_K2 = abstractPlaces(g_K1) ⇒
    (hasEdge(g_K1, e_K1) ∧ def mapEdge(e_K1) ⇒
     source(mapEdge(e_K1)) = mapNode(source(e_K1)) ∧
     target(mapEdge(e_K1)) = mapNode(target(e_K1)) ∧
     hasEdge(g_K2, mapEdge(e_K1))) ∧
    (∃ n1, n2: Node[Kind1]; e1, e2: Edge[Kind1]
     • mapNode(n1) = mapNode(n2) ∧
       hasNode(g_K1, n1) ∧ hasNode(g_K1, n2) ∧
       hasEdge(g_K1, e1) ∧ hasEdge(g_K1, e2) ∧
       target(e1) = n1 ∧ source(e2) = n2 ⇒
       (∃ r: Route[Kind1]
        • pathIn(r, g_K1) ∧ startOf(r) = n1 ∧
          endOf(r) = n2))
  • g_K2 = abstractPlaces(g_K1) ⇔
    (hasNode(g_K2, n_K2) ⇔
     ∃ n1: Node[Kind1]
     • hasNode(g_K1, n1) ∧ mapNode(n1) = n_K2) ∧
    (hasEdge(g_K2, e_K2) ⇔
     ∃ e1: Edge[Kind1]
     • hasEdge(g_K1, e1) ∧ mapEdge(e1) = e_K2)
end
```

Fig. 13. Generic Place Abstraction

```
spec DEVICES_TO_TRACKSEGMENTS =
    PLACEABSTRACTION
      [TRAMDEVICES_ROUTEGRAPH
       fit Kind1 ↦ TDevices,
          Route[Kind1] ↦ Route[TDevices],
          Path[Kind1] ↦ Path[TDevices]]
      [TRAMNETWORK_ROUTEGRAPH
       fit Kind2 ↦ TNetwork,
          Edge[Kind2] ↦ TrackSegment,
          Route[Kind2] ↦ Route[TNetwork],
          Path[Kind2] ↦ Path[TNetwork]]
then ∀ g1: Graph[TDevices]; g2: Graph[TNetwork];
       sigSeg: SignalSegment; poiSeg: PointSegment;
       sig_n: Signal; seg: Segment
     • mapNode(source(sigSeg)) =
        mapNode(target(sigSeg))
     • mapNode(source(poiSeg)) =
        mapNode(target(poiSeg))
     • mapNode(sig_n) ∈ SignalSensor
     • ¬ def mapEdge(sigSeg)
     • ¬ def mapEdge(poiSeg)
     • def mapEdge(seg) ∧
        mapEdge(seg) ∈ TrackSegment
     • abstractPlaces(g1) = g2 ⇔
        (∀ n1: Node[TDevices]
         • hasNode(g1, n1) ⇔
          hasNode(g2, mapNode(n1))) ∧
        (∀ e1: Edge[TDevices]
         • def mapEdge(e1) ⇒
          (hasEdge(g1, e1) ⇔
           hasEdge(g2, mapEdge(e1)) ∧
           hasNode(g2, source(mapEdge(e1))) ∧
           hasNode(g2, target(mapEdge(e1)))))
     • source(seg) ∈ ActivePoint ⇒
        mapEdge(seg) ∈ SignalTrackSegment
     • target(seg) ∈ JointPoint ⇒
        mapEdge(seg) ∈ JoinTrackSegment
     pred __crosses__(d_e1, d_e2: TrackSegment) ⇔
          ∃ e1, e2: Edge[TDevices]
          • d_e1 = mapEdge(e1) ∧
             d_e2 = mapEdge(e2) ∧ e1 crosses e2;
end
```

Fig. 14. Abstraction from TramDevices in CASL

Let us consider the TramNetwork Route Graph example in Fig. 10. We assume that a tram intends to use the route "24/21" from G24.1 to G21.1 (right → lower left corner). Then this route "24/21" is in *crossingConflictWith* route "22/23" from G22.1 to G23.1 (bottom → top); this conflict can be avoided by setting signal S21 to "right" (or "stop") – setting the corresponding point to "right" increases safety. The corresponding *crossingConflictWith* route "20/25" from G20.1 to G25.0 (upper left corner → right) could similarly be avoided by setting signal (and point) S20 to "straight".

However, then a new conflict arises: route "24/21" is in *exclusionConflictWith* route "20/21" from G20.1 to G21.1 (top → bottom) since they join at G21.0 and share the last segment. To avoid this, signal S20 must be set to "stop" (alternatively, tram drivers must watch out for other trams and obey the local traffic rules, as in Bremen).

The last kind of conflict, e.g. that route "20/25" is in *exclusionConflictWith* route "20/21", occurs when two routes share a starting segment; the two routes can obviously only be used exclusively in either of the forking directions, with appropriate setting of signal and point S20.

6. CONCLUSION

"Light-Weight" Ontologies. We have described the instantiation of Route Graphs to Tramway Networks and their safety requirements. A "light-weight" ontology was presented to introduce the concepts of generic Route Graphs and Tramway Networks for a first understanding of the terminology used in this paper and of the conceptual interrelation. Such as presentation, particularly in graphical form, is easier to grasp than a complete formalisation. The similarity in appearance to a particular kind of UML class diagram is intended (for a translation see below); however, we are interested in linking to ontology standards such as OWL (Bechhofer *et al.*, 2004; Lüttich and Mossakowski, 2004; Lüttich *et al.*, 2004). Such "light-weight" ontologies may then also be used to help in documentation and to interrelate documents, cf. (Krieg-Brückner *et al.*, 2003; Krieg-Brückner *et al.*, 2004*a*).

Formalisation of Ontologies in CASL. A more precise definition of the ontologies has then been given in CASL. The specifications are all well-formed and statically checked by tools for type correctness; the LaTeX-representation for pretty formatting is automatically generated by a tool from the (already readable) ASCII syntax.

For a separation of concerns, the various aspects of a Tramway Network have been modeled in two layers of Route Graphs, related by an abstraction function. The TramDevices layer, possibly automatically generated from a CAD based network layout description, aims at engineers concerned with the particular placement of the various devices. The abstracted version, the TramNetwork layer, where only signals, sensors and track segments are present, aims at engineers who have to define tram routes, check for their consistency, define scheduling etc. (this might be a future extension). This abstraction preserves the relevant static features that are needed to check the safety requirements for tramway routes.

On the basis of the specifications in this paper, a route maintanance system could be designed in CASL; with the tools available for CASL, the axioms of the loose specifications given could be proved to hold in the design specification. The design specification carries over the static safety requirements; Signal Setting Tables and Point Position Tables may be derived from the tram routes. From the design specification, an executable Haskell programm could then be derived automatically, cf. (Krieg-Brückner *et al.*, 2004*c*).

Extension by Modal Logic. Another future step would be to extend the formalisation presented in this paper with modal logic; the dynamic aspects of points and signals could then be modeled as well. This is also possible within the CASL framework, as there is an extension of CASL, called ModalCASL, which extends CASL's first-order logic with modal logic. States correspond to possible worlds in the Kripke semantics

of modal logic. State transitions caused by an action *pass_train_G20_0* would be specified by using pre- and post-conditions, e.g. $G20_0 = n \Rightarrow$ [*pass_train_G20_0*]$G20_0 = n + 1$, where *G20_0* is a flexible constant having different values in different worlds.

In this modal logic framework, hazards could also be specified, based on the predicate *crossingConflictWith* defined on *TramRoutes*, as follows:

$$
\begin{array}{l}
r1\ hazard\ r2 \Leftrightarrow \\
\quad r1\ crossingConflictWith\ r2\ \wedge \\
\quad counterState(target(head(r1))) > 0\ \wedge \\
\quad counterState(target(head(r2))) > 0\ \wedge \\
\quad pointSate(entry(head(r1))) = \\
\qquad signalState(target(head(r1)))\ \wedge \\
\quad pointState(entry(head(r2))) = \\
\qquad signalState(target(head(r2)))
\end{array}
$$

where $hazard$ is a flexible predicate, *counter-*$State$ is a flexible operation representing the state of the sensor given as argument (greater zero means at least one train has passed since last reset) and $signalState$ and $pointState$ represent the direction given by the signal and the locked point direction. The first three conjuncts express that two trams have reached the sensors of the entries of conflicting track segments; the last two conjuncts express that the track switches are set in such positions that the trams will actually use the conflicting segments.

Another hazard might arise from a combination of exclusion and crossing conflicts.

When a hazard (due to faulty equipment or human error of signal neglect) is detected by sensors for two passing trams, remotely triggered full stop is the only cure.

Heterogeneous Tool Set. CASL and ModalCASL have been integrated with other logics in *Heterogeneous* CASL, HetCASL, for which the *Heterogeneous Tool Set*, HETS, (Mossakowski, 2004) provides, among other things, logic translations and connections to automatic and interactive verification systems.

UML. It is also planned to extend HETS with a translation link between CASL and UML, such that e.g. the route graph ontology is mapped to a UML class diagram, which can then be used in connection with software engineering tools. Note, however, that we do not consider it advisable to start with such a UML class diagram: firstly, the Object Constraint Language (OCL), the logic of UML, limits quantifications to finite domains, which means e.g. that one cannot quantify over routes (generally, there are infinitely many routes for a route graph) and therefore cannot express all of our axiomatization. Secondly, the two-valued logic of CASL is simpler than three-valued OCL, and hence can resort to the whole body of methods and tools that have been developed for classical first-order logic.

RAISE. Future work should integrate the present route graph specifications with existing work on specification of railway nets in the RAISE language. The RAISE language is similar to ModalCASL in that it can deal with states; we prefer to use ModalCASL because it is based on modal first-order logic that has been studied for a long time. We consider a translation from existing RAISE specifications to CASL to be a fruitful task for the future.

(Bjørner, 2000) (to cite one of a number of papers) is tailored to railway systems, a sister domain, and uses various sorts/sets rather than graphs, concentrating more on aspects of scheduling, train plans and time tables, rather than route graph abstractions at different levels.

A connection to the approach of (Lindegaard *et al.*, 2000; Haxthausen *et al.*, 2004) (based on RAISE, an including an implementation) also looks promising; one difference is that they do not use graphs, but specify observer functions stating geographic relations between the track elements and the allowed network topologies. Their state transition system corresponds to Kripke models of modal specifications in our approach.

Acknowledgements

We thank Jan Peleska and Stefan Bisanz for their contribution to the discussions. Jan Peleska contributed the Tramway Network figure (Fig. 1); Achim Mahnke implemented the ontology visualisation tool (Mahnke, 2004) and helped in the preparation of some of the figures.

REFERENCES

Astesiano, E., M. Bidoit, B. Krieg-Brückner, H. Kirchner, P. D. Mosses, D. Sannella and A. Tarlecki (2002). CASL – the common algebraic specification language. *Theoretical Computer Science* **286**, 153–196.

Bechhofer, S., F. van Hermelen, J. Hendler, I. Horrocks, D. L. McGuinness, P. F. Patel-Schneider and L. A. Stein (2004). W3C: OWL Web Ontology Language – Reference. W3C Recommendation `http://www.w3.org/TR/owl-ref/`.

Bjørner, D. (2000). Formal software techniques in railway systems. In: *9th IFAC Symposium on Control in Transportation Systems, Technical University, Braunschweig, Germany, 13-15 June 2000* (E. Schnieder, Ed.). pp. 1–12. VDI/VDE-Gesellschaft Mess- und Automatisieringstechnik, VDI-Gesellschaft für Fahrzeug- und Verkehrstechnik. Invited talk.

Dylla, F. and R. Moratz (2004). Exploiting qualitative spatial neighborhoods in the situation calculus. In: *Spatial Cognition IV* (C. Freksa, M. Knauff, B. Krieg-Brückner, B. Nebel and T. Barkowsky, Eds.). Lecture Notes in Artificial Intelligence. Springer. (to appear).

Haxthausen, A. E. and J. Peleska (2002). A domain specific language for railway control systems. In: *Proceedings of the Sixth Biennial World Conference on Integrated Design and Process Technology*. IDTP2002. Pasadena, Califorania.

Haxthausen, A. E., N. Christensen and R. Dyhrberg (2004). From Domain Model to Domain-specific Language for Railway Control Systems. In: *Proceedings of Formal Methods for Automation and Safety in Railway and Automotive Systems (FORMS/FORMAT 2004)), Braunschweig, Germany*.

Krieg-Brückner, B., A. Lindow, C. Lüth, A. Mahnke and G. Russell (2004*a*). Semantic interrelation of documents via an ontology. In: *DeLFI 2004, Tagungsband der 2. e-Learning Fachtagung Informatik, 6.-8. September 2004, Paderborn, Germany* (G. Engels and S. Seehusen, Eds.). Vol. P-52 of *Lecture Notes in Informatics*. Springer. pp. 271–282.

Krieg-Brückner, B., D. Hutter, A. Lindow, C. Lüth, A. Mahnke, E. Melis, P. Meier, A. Poetzsch-Heffter, M. Roggenbach, G. Russell, J.-G. Smaus and M. Wirsing (2003). Multimedia instruction in safe and secure systems. In: *Recent Trends in Algebraic Development Techniques*. Vol. 2755 of *Lecture Notes in Computer Science*. Springer. pp. 82–117.

Krieg-Brückner, B., R. Ross and H. Shi (2004*b*). A safe and robust approach to shared control via dialogue. *Chinese Journal of Software*. (to appear).

Krieg-Brückner, B., T. Röfer, H.-O. Carmesin and R. Müller (1998). A taxonomy of spatial knowledge for navigation and its application to the bremen autonomous wheelchair. In: *Spatial Cognition* (C. Freksa, C. Habel and K.F. Wender, Eds.). pp. 373–397. Number 1404 In: *Lecture Notes in Artificial Intelligence*. Springer.

Krieg-Brückner, B., U. Frese, K. Lüttich, C. Mandel, T. Mossakowski and R. Ross (2004*c*). Specification of an ontology for route graphs. In: *Spatial Cognition IV* (C. Freksa, M. Knauff, B. Krieg-Brückner, B. Nebel and T. Barkowsky, Eds.). Lecture Notes in Artificial Intelligence. Springer. (to appear).

Lindegaard, M. P., P. Viuf and A. E. Haxthausen (2000). Modelling Railway Interlocking Systems. In: *Proceedings of the 9th IFAC Symposium on Control in Transportation Systems 2000, June 13-15, 2000, Braunschweig, Germany*. pp. 211–217.

Lüttich, K. and T. Mossakowski (2004). Specification of ontologies in CASL. In: *International Conference on Formal Ontology in Information Systems*. IOS-Press. (to appear).

Lüttich, K., T. Mossakowski and B. Krieg-Brückner (2004). Ontologies for the semantic web in CASL. *17th Int. Workshop on Algebraic Development Techniques*. (to appear).

Mahnke, A. (2004). Ontology visualisation tool. `http://www.informatik.uni-bremen.de/agbkb/publikationen/softlibrary_e.htm`.

Mossakowski, T. (2004). Heterogeneous tool set (Hets) web site. `http://www.informatik.uni-bremen.de/cofi/hets`.

Mosses, P. D., Ed.) (2004). CASL *Reference Manual*. Vol. 2960 of *Lecture Notes in Computer Science*. Springer.

Ross, R., H. Shi, T. Vierhuff, B. Krieg-Brückner and J. Bateman (2004). Towards dialogue-based shared control of navigating robots. In: *Spatial Cognition IV* (C. Freksa, M. Knauff, B. Krieg-Brückner, B. Nebel and T. Barkowsky, Eds.). Lecture Notes in Artificial Intelligence. Springer. (this volume).

Timpf, S. (submitted). Modeling the physical complexity of routes in public transportation networks. *Environment and Behaviour*.

Werner, S., B. Krieg-Brückner and T. Herrmann (2000). Modelling navigational knowledge by route graphs. In: *Spatial Cognition II* (C. Freksa, C. Habel and K.F. Wender, Eds.). pp. 295–317. Number 1849 In: *Lecture Notes in Artificial Intelligence*. Springer.