# Specification of an Ontology for Route Graphs*

Bernd Krieg-Brückner, Udo Frese, Klaus Lüttich, Christian Mandel,
Till Mossakowski, and Robert J. Ross

FB3 Mathematik und Informatik, Universität Bremen,
Postfach 330 440, D-28334 Bremen
bkb@informatik.uni-bremen.de

**Abstract.** This paper describes the general concept of Route Graphs, to be used for navigation by various agents in a variety of scenarios. We introduce the concept of an ontology and describe the modelling of general graphs as an example. This approach is then applied to define a "light-weight" ontology of Route Graphs in an indoors environment, giving at first just a taxonomy of (sub)classes and relations between them, as well as to other (spatial) ontologies. Finally, we show how to formalise ontologies using a First Order Logic approach, and give an outline of how to develop actual data structures and algorithms for Route Graphs.

## 1 Introduction

Route Graphs have been introduced as a general concept ([1]), to be used for navigation by various agents in a variety of scenarios such as humans using railway, underground, road or street networks, as travellers, car drivers or pedestrians, resp. Each application scenario or kind of Route Graph will introduce special attributes on the general structure in a hierarchy of refinements.

While the concept was originally introduced to mediate terminology between artificial intelligence and psychology in spatial cognition, scenarios are not restricted to human users. Route Graphs are also intended for interaction between service robots, such as the Bremen autonomous wheelchair Rolland (cf. Fig. 1), and their users, as well as between robots, for example as a compact data structure for on-line communication in an exploration scenario. Moreover, a bare-bones Route Graph representation is easily constructed from pre-existing map-like representations (at least for floor plans of buildings) or by robot exploration, and we hope that cognitively (more) adequate maps can be constructed from it when the semantic interrelation can be taken into account.

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. We intend to show that an ontological approach is suitable

* This paper has been supported by the Deutsche Forschungsgemeinschaft (DFG) in the SFB/TR 8 Spatial Cognition.

C. Freksa et al. (Eds.): Spatial Cognition IV, LNAI 3343, pp. 390–412, 2005.

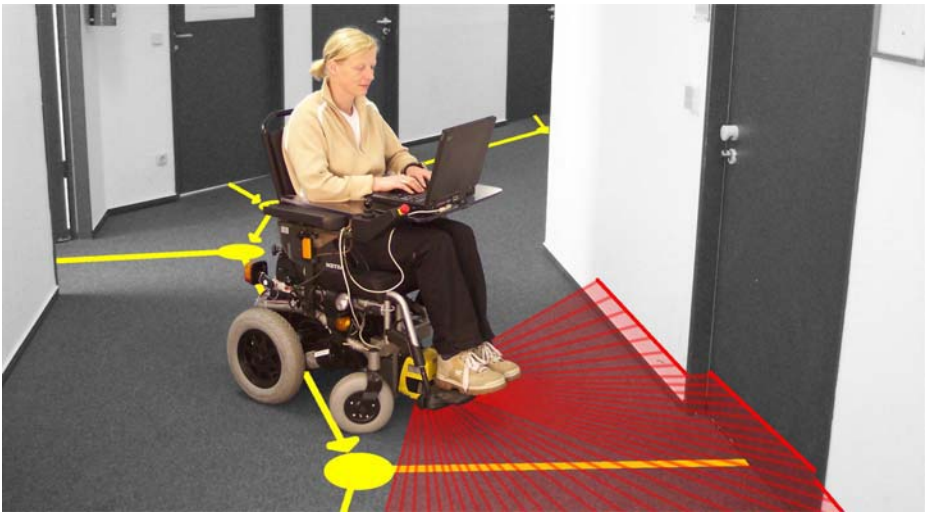

**Fig. 1.** Rolland following a Route Graph

for both: to define the generic concept of Route Graphs and to instantiate it for a particular scenario in detail, leading eventually to a concrete data structure. Moreover, the example of Route Graphs is suitable for demonstrating that adequate formalisation of ontologies can be introduced step by step in this process.

In this paper, we will show the application to an indoors scenario for Rolland. Thus we have to model detailed information for navigation at the "robot level" as well as more abstract concepts of space at the "user level", and the relationship between these and "common sense" concepts in the environment, such as rooms, doors or windows as route marks (cf. [2]). For a dialogue between user and wheelchair, we want to relate the concepts of Route Graphs to linguistic ontologies; this is described e.g. in [3, 4].

The paper is structured as follows: We first sketch the general concept of Route Graphs in Sect. 2. In Sect. 3 we introduce the concept of an ontology and describe the modelling of general graphs as an example. This approach is then applied to define a "light-weight" ontology of Route Graphs in an indoors environment, giving at first just a taxonomy of (sub)classes and relations between them, as well as to other (spatial) ontologies, in Sect.4. Finally, we show in Sect.5 how to formalise ontologies using a First Order Logic approach, and give an outline of how to develop actual data structures for Route Graphs.

## 2 Route Graphs

### 2.1 Sample Scenarios

Figure 2 shows some sample navigation scenarios. We may distinguish between navigation in systems 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);

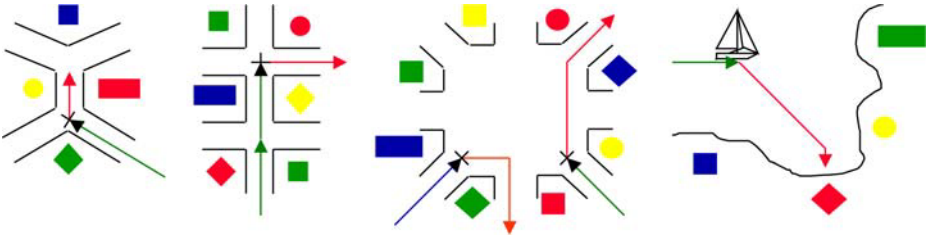

**Fig. 2.** Examples of routes

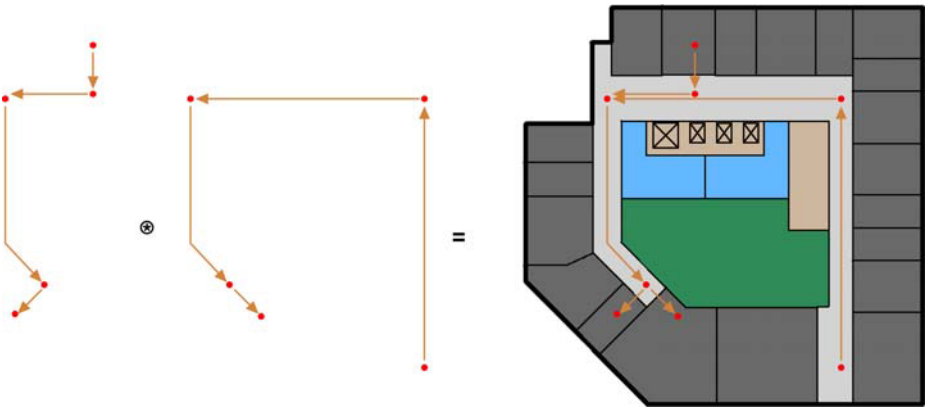

**Fig. 3.** Two routes and their union

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. [5].

While *Route Graph*s should apply to all such scenarios (cf. [1]), we will concentrate on the indoors scenario here, for Rolland and its user as in Fig. 1, as an example for service robot applications. We briefly introduce the basic concepts of Route Graphs here; more detail will follow in Sect. 3.2 and Sect. 4.

**Sample Route: to the Secretary's Office.** Consider Fig. 3: two separate routes are united into a simple Route Graph. Let us take the first route as an example. It can be described by directions in natural language as follows:

- Leave Room MZH 8210 into the corridor
- Turn right, go straight ahead to the window
- Turn left, follow the corridor until the last door on the right-hand-side
- Face it, enter Room MZH 8080

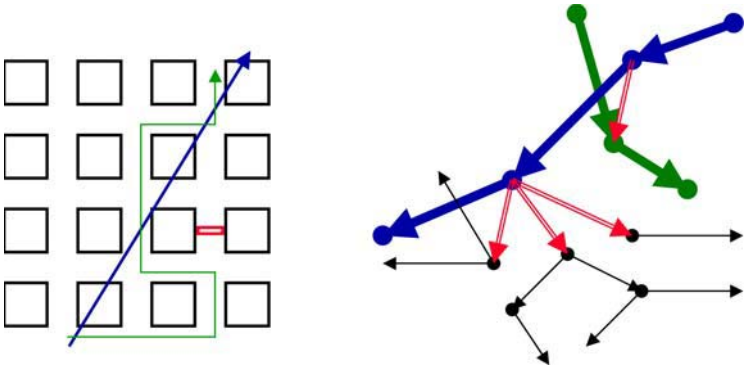

**Fig. 4.** Layers and transfers

**Sample Route Segments.** The first two lines can then be translated into the following two route segments (taking additional information about doors, lifts, windows into account, see Sect. 4.4):

| | |
|---|---|
| *Source*: room MZH 8210 | *Source*: corridor, facing the lift |
| *Entry*: turn towards the door | *Entry*: turn right |
| *Course*: go through the door | *Course*: follow corridor to the window |
| *Exit*: [turn to face the lift] | *Exit*: [turn to face the window] |
| *Target*: corridor, facing the lift | *Target*: T-crossing, facing the window |

**Segments.** An edge of a Route Graph is directed from a source node to a target node. We call an edge a *(route) segment*; it always has three additional attributes: an entry, a course and an exit. Exactly what information is associated with these attributes is specially 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 the exit into a harbour may specify a particular orientation along a quay.

**Places.** A node of a Route Graph, called a *place*, has a particular position and orientation, its *origin*. 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 (see Sect. 4.3).

### 2.2 Homogeneous Route Graphs, Transfers

Fig. 4 shows, on the right hand side, an example where various Route Graphs have been united to one heterogeneous Route Graph. We say a Route Graph of a particular kind *is homogeneous*, if all its segments are of the same kind. In the

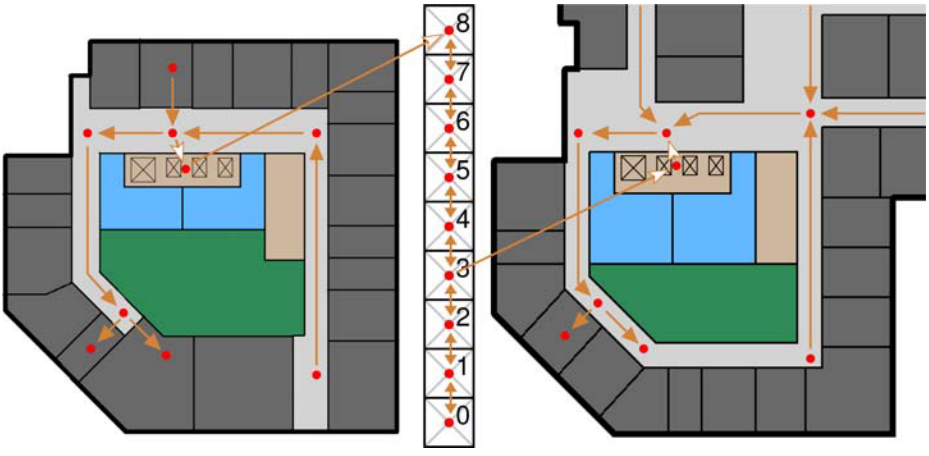


**Fig. 5.** Two levels and lift

**Fig. 6.** Voronoi representation for levels 8 and 3

example, the routes depicted by fat arrows might correspond to underground train lines, those with fine arrows to a network of pedestrian passages; both are homogeneous. There is then a need to introduce *transfer segment*s for connection, or indeed *transfer Route Graph*s that have transfer segments at their fringes. In the example they would correspond to a pedestrian transfer passage between two underground stations, or several exits from an underground station to the pedestrian network above (note that these might be connected to other exit or entry routes underground). Fig. 5 shows a transfer between a Route Graph at level 8 of our office building, the lift system, and level 3.

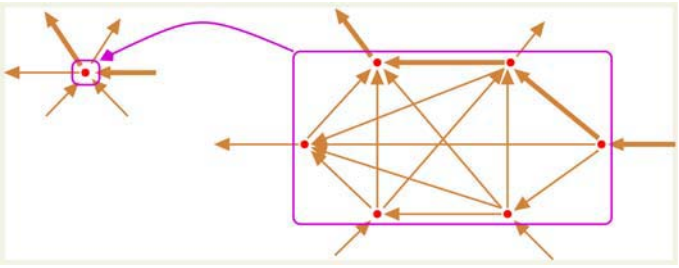

**Fig. 7.** Abstraction to place

### 2.3 Layers and Abstraction

We may wish to separate Route Graphs into *layer*s at different levels for conceptual reasons, possibly unconnected. One reason might be that we want to represent route and overview knowledge (cf. [1]). The left hand side of Fig. 4 sketches a route in an urban block scenario, and the target designated by an arrow. When circumnavigating a road blockage, for example, we are likely as humans to use such an arrow as a "sense of direction" while negotiating our way in the block maze – we are using and combining two layers of knowledge at the same time; it is well known that other animals are much better in this respect.

**Abstraction.** Another reason is abstraction from a more concrete and detailed lower layer to a higher one, as in the abstraction from the robot level to the user level (see also Sect. 4.1). Fig. 5 refers to the user level, while Fig. 6 shows a Voronoi diagram as a representation of space at the robot level (cf. also Sect. 4.2) which corresponds directly to a detailed Route Graph, if we replace the trajectories between places by straight directed edges, in both directions.

What is required here is a relation `abstractsTo` between graphs (in fact a graph morphism), more specifically from a route to a segment, or a graph to a node as in Fig. 7 (cf. also Fig. 16). When a set of nodes SN on the fringe of the graph G at the lower layer is abstracted to a single node N, some obvious conditions must hold, e.g. all incoming/outgoing edges to a node in SN must correspond to incoming/outgoing edges for N; for each pair of incoming and outgoing edges for N there must be a corresponding connecting path in G; etc.

## 3 Modelling Via Ontologies

**Ontologies** provide the means for establishing a semantic structure. An ontology is a formal explicit description of concepts in a domain of discourse [6]. Ontologies are becoming increasingly important because they also provide the critical semantic foundations for many rapidly expanding technologies such as software agents, e-commerce, or the "Semantic Web" [7]. In artificial intelligence, ontologies have been used for knowledge representation ("knowledge engineer-

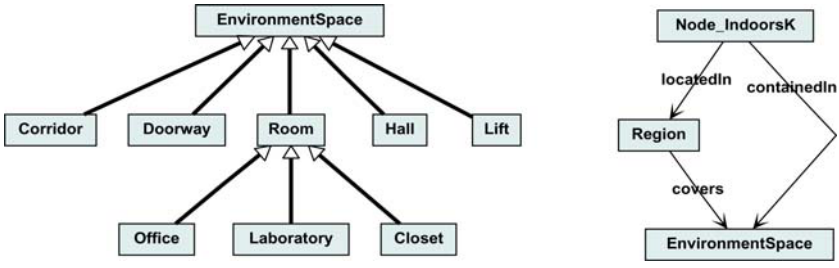


**Fig. 8.** Subclasses of `EnvironmentSpace` and relations

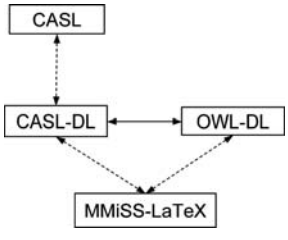


**Fig. 9.** Various ontology specification formats

ing"). The general idea is to make knowledge explicit by expressing concepts and their relationships formally with the help of mathematical logics.

An *ontology* consists of (a hierarchy of) concepts and relations between these concepts, describing their various features and attributes. Classes and relations are used for defining the abstract semantical level; they provide a vocabulary and properties to characterise the concrete entities corresponding to these concepts. Once the abstract notions are declared in terms of classes, objects can be used to denote entities of semantic concepts. As a simple example for an ontology consider Fig.8. On the left, it depicts a hierarchy of subclasses of the *class* `EnvironmentSpace` – an extract of a taxonomy as it might appear in a "Common Sense Ontology" of space[1], cf. [2]. The relation *is a subclass of* (or "*is a*") is depicted by a fat arrow with a hollow tip. For example, an `Office` is a `Room` which is in turn an `EnvironmentSpace`.

A class represents a set of *objects*; for example `Office8210` is an object (depicted with leading and trailing underscores in the diagrams below) of class `Office`, or `Office8210: Office`.

On the right, we see declarations of *relations* depicted by pointed arrows, with classes as domains and co-domains, to be formalised (see Sect.5) and used eventually for the definition of relations between objects. ***`containedIn`***, for example, relates the class `Node_IndoorsK` of our RouteGraph ontology (to be de-

[1] We refer to different ontologies separately here although they are in fact parts of one combined ontology; for the structuring of ontologies see Sect.4.8.

fined below) to `EnvironmentSpace` in the "Common Sense Ontology"; moreover, `Node_IndoorsK`s in the RouteGraph ontology are related by *`locatedIn`* to (a separate ontology for) a calculus of `Region`s. Thus we should eventually be able to deduce, given
`Place8210: Node_IndoorsK`, `Region8210: Region`, and `Office8210: Office`, then
`Place8210 locatedIn Region8210` and `Region8210 covers Office8210` implies `Place8210 containedIn Office8210`

### 3.1 Various Ontology Specification Languages

**Description Logic in** OWL**.** Ontologies may come in various specification formats, cf. Fig.9 (or [8]). We adhere to the OWL standard [9]. Its description logic DL has been primarily defined to be decidable for the Semantic Web.

**Lightweight Ontologies in LaTeX.** For lightweight ontologies, just (sub)classes, objects and relations as in the example above, we use a special LaTeX format that can be translated to OWL and vice-versa (a tool [10], based on the generic graph visualisation tool DaVinci [11], supports incremental presentation of and navigation in such graphs, as in the examples shown in this paper). It allows the specification of ontologies for documents to enable their semantic interrelation, enabling much more advanced document management facilities ([12, 13]).

**First Order Logic in CASL.** In contrast, Casl, the Common Algebraic Specification Language approved by IFIP WG1.3 ([14, 15]), covers full First Order Logic, plus predicates, partial and total functions, and subsorting; more-

MMiSSLaTeX

```
\Class{Graph}{graph}{}
      \RelType{hasNode}{Graph}{Node}
      \RelType{hasEdge}{Graph}{Edge}
\Class{Node}{node}{}
\Class{Edge}{edge}{}
      \RelType{hasSource}{Edge}{Node}
      \RelType{hasTarget}{Edge}{Node}

      \Relation{*-*}{hasNode}{has a node}{}
      \Relation{*-*}{hasEdge}{has an edge}{}
      \Relation{->}{hasSource}{has the source}{}
      \Relation{->}{hasTarget}{has the target}{}

\Class{Sequence_Edge}{\Ref{Edge} list}{}
 \Class{Path}{path}{Sequence_Edge}
  \Class{Route}{route}{Path}
```

**Fig. 10.** Path and route

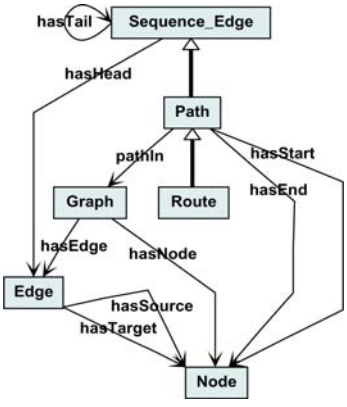


**Fig. 11.** Simple graph ontology (extract)

over, features for structuring, or "specification in the large", have been included (cf. Sect.4.8). Thus full formalisation of ontologies becomes possible, (as in the DOLCE framework [16, 17]), needed eventually here, see Sect.5. A sublanguage, CASL-DL, has been defined to correspond to OWL DL [18] such that the mappings/embeddings of Fig.9 between the various representations can be realized.

### 3.2 Ontology of Graphs

As an example, consider a simple ontology of graphs. Fig.10 shows the core in MMiSSLaTeX: A `Graph` has (one ore more) `Node`s and `Edge`s. The MMiSSLaTeX operation `Class` declares these classes: the first parameter denotes the semantics term, the second a default textual phrase (for use in LaTeX documents), the third the superclass.The operation `Relation` is analogous, but contains as an additional first parameter an indication of the kind of relation, e.g. "`*-*`" to denote "many to many", "`->`" to denote "onto" (a function), or "`<`" to denote "strict partial order". With the operation `RelType` relations may be "typed" by source and target classes to allow (static) checking of conformance when objects are related. Thus each `Edge` has exactly one source and one target `Node`.

Note that, at the level of an ontology specification, we do formalise differently than in mathematics or a classical modeling in a specification or programming language: in the latter case, we would introduce a graph as a pair for a set of nodes and edges; here, a `Graph` contains both sets – the abstraction from the pair is perhaps even more natural and definitely adequate as a first go. We show in Sect.5.1 how a more "data structure" oriented design may come back in.

### 3.3 Paths and Routes

The last lines of Fig.10 show an extension of the basic graph specification. A `Path` is (a subclass of) a `Sequence` of `Edge`s (`Sequence` is a basic concept with additional relations, instantiated here), where the target of the first edge is the

source of the second, and so on. Note that a `Path` corresponds to some graph traversal, possibly with cycles. In contrast, a `Route` contains no repetition of `Edges`. Such properties will have to be formally specified by additional axioms on the subclasses, successively refined from `Sequence_Edge` over `Path` to `Route`, cf. Sect.5.1. The visualisation in Fig.11 shows a few more relations, sometimes with multiple relation arrows, e.g. `hasSource` and `hasTarget` from `Edge` to `Node`.

## 4 A Route Graph Ontology for an Indoors Scenario

### 4.1 Instantiation to Particular Route Graphs

We assume that the general properties of (labelled) directed graphs are specified along the outline above; similarly, abstract route planning algorithms could be specified at this level without knowing (much) more about `Nodes` or `Edges`.

**Indoors Instantiation and Kinds.** We instantiate this general graph ontology here for an indoors application of Route Graphs, see Fig.13 (cf. also Sect.4.8 for the instantiation aspect). Thus `Graph` becomes `Graph_IndoorsK`, `Node` becomes `Node_IndoorsK`, and so on. `KindRG` denotes the kind of Route Graph, where `IndoorsK` may be refined later on as shown in Fig.12; other kinds might denote `RailwayK`, `UndergroundK` or `RoadK` Route Graphs, for example.

**User Level and Robot Level.** Our `IndoorsK` instantiation refers to the (indoors) operating environment of a robot. Note that we do a particular modelling for Route Graphs here which is separate from, but related to, the "Common Sense Ontology" for `EnvironmentSpace` in Fig.8; we have to expect that they are structured differently. As we shall see below, a `Graph_IndoorsK` may refer to a rather detailed description at the level of a robot, e.g. the wheelchair Rolland, or its abstraction at the level of an operator, e.g. a Rolland user. User and robot

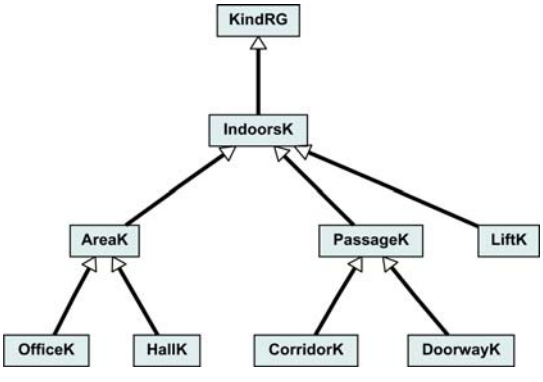


**Fig. 12.** Ontology of `KindRG`

interact at this abstract level, while robots may interact among themselves at the detailed level, for example during a multi-robot exploration phase.

### 4.2 Edges and Route Segments

With the transition from graphs to Route Graphs, an `Edge_IndoorsK` (with source and target) is refined to a `Segment_IndoorsK` (see Fig.14), with an additional entry, course and exit, see Fig.15. Each `KindRG` of Route Graph will introduce its special aspects, in particular (relations to) components.

**Vectors and Voronoi Diagrams.** In the specialisation of `IndoorsK` Route Graphs, a segment is essentially modelled as a vector in polar coordinates: the entry gives the angle from (the origin of) the source place, the course the distance to the target , and the exit the angle to the target, to assume the orientation of (the origin of) the target place (cf. Sect.4.3). In addition, we attach information derived from a Voronoi diagram representation (see also [19]): each point has a maximal circle of available free space around it, represented by its diameter or `Width`. Thus each `Place_IndoorsK` has a `Width`, and the `Width` of a `Course_IndoorsK` denotes the minimal width of the points along its course.

**Abstraction of Segments.** Fig.16 shows a Voronoi diagram at the bottom corresponding to a detailed Route Graph for the route in the corridor which has been abstracted to a single segment at the top, showing the `CorridorWidth`.

Consider also the example in Fig.17, corresponding to that in Fig.3 and given verbally in Sect.2.1: `Segment1` represents a passage through a doorway, of `DoorWidth`; `SegmentCorridor2` represents (part of) the corridor, of `CorridorWidth`. The entry and exit angles are depicted by fat little (angular) arrows. Fig.15 and Fig.18 show the ontology of segments and the objects related to the particular segment `SegmentCorridor2`. Note that the entry angle and the distance of the course denote the *direct* vector to the target place thus defining a spatial relation between the two nodes; the actual Voronoi trajectory (cf. e.g. `Segment1`) and its length (which may be larger than the distance) are not represented here; they could be added to the course as additional information, but are

MMiSSLaTeX

```
\Class{Graph_IndoorsK}{\Ref{IndoorsK} graph}{}
      \RelType{hasNode}{Graph_IndoorsK}{Node_IndoorsK}
      \RelType{hasEdge}{Graph_IndoorsK}{Edge_IndoorsK}
\Class{Node_IndoorsK}{\Ref{IndoorsK} node}{}
\Class{Edge_IndoorsK}{\Ref{IndoorsK} edge}{}
      \RelType{hasSource}{Edge_IndoorsK}{Node_IndoorsK}
      \RelType{hasTarget}{Edge_IndoorsK}{Node_IndoorsK}
\Class{Sequence_Edge_IndoorsK}{\Ref{Edge_IndoorsK} list}{}
 \Class{Path_IndoorsK}{\Ref{IndoorsK} path}{Sequence_Edge_IndoorsK}
  \Class{Route_IndoorsK}{\Ref{IndoorsK} route}{Path_IndoorsK}
```

**Fig. 13.** Indoors Graph ontology

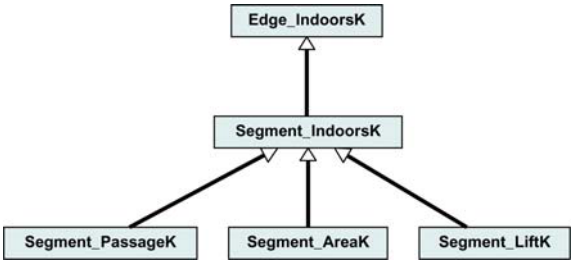


**Fig. 14.** Ontology of indoors edge subclasses

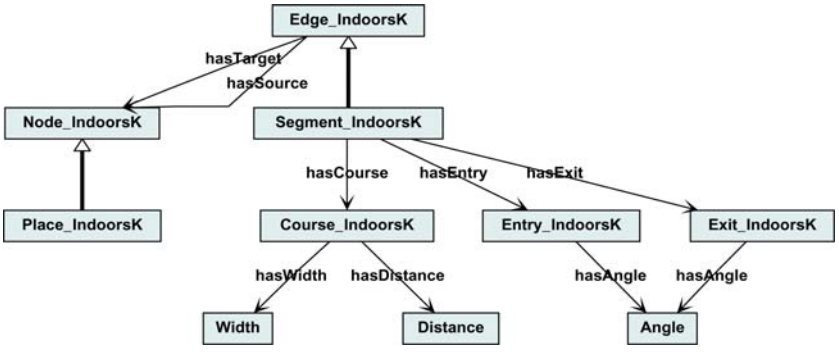


**Fig. 15.** Ontology of indoors segment subclasses

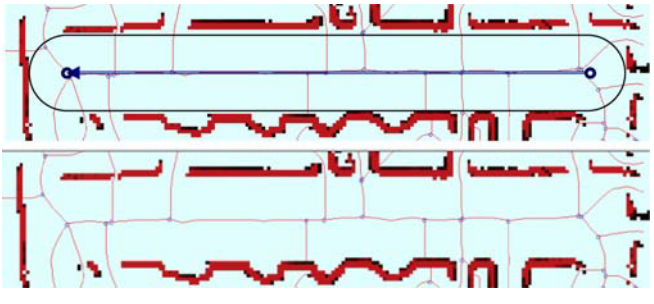

**Fig. 16.** Abstraction of route in corridor to route segment

probably not necessary since the construction of the actual navigation trajectory from the Route Graph has to take e.g. dynamic obstacles into account.

**Modelling Precision.** All measurements above (distances, angles, etc.) are intended to be qualitative; in fact no units have been specified so far. Precision can be introduced by an additional attribute, for example an interval of tolerance.

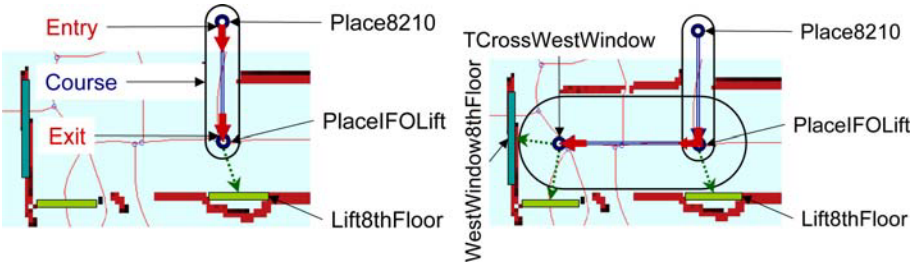


**Fig. 17.** `Segment1` and `SegmentCorridor2`

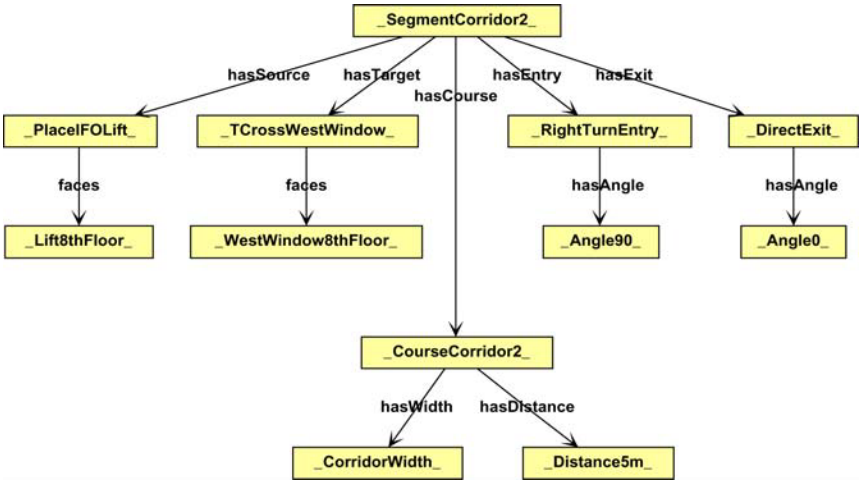


**Fig. 18.** Ontology of object instantiations representing `SegmentCorridor2`

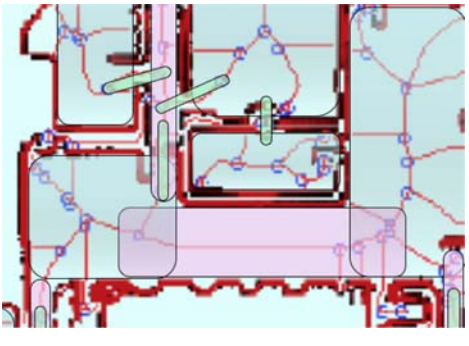

**Fig. 19.** Overlapping regions

### 4.3 Nodes and Places

A `Node_IndoorsK` is refined to a `Place_IndoorsK`; each place has its own *origin*, i.e. a well-defined position and orientation with a zero angle. For example (cf.

Fig.18), `Place8210 faces` the office door, `PlaceIFOLift faces` the lift, and `TCrossWestWindow faces` the west window. We might wish to define an origin to coincide with such a `faces` relation, but in fact it does not matter much as it is only important to have a well-defined origin at all.

**Place Integration.** How is this origin preset? When a node has just one segment attached, its entry angle is zero for an outgoing segment, or its exit angle is zero for an incoming segment.[2] When adding an additional segment, this origin will have to be adhered to. When integrating two routes or in a union of Route Graphs, *place integration* has to be performed for all segments with common places: for any two places to be integrated, a common origin will have to be chosen and the entries/exits of the segments re-computed, if necessary. This is the price to pay for the fact that a place has now become a position of choice for outgoing segments, with different entries in general.

**Regions.** Places are contained in regions. Recall Fig.8: If a `Place` is `locatedIn` a `Region` and this `Region covers` an `EnvironmentSpace`, say an `Office`, then we may conclude that this `Place` is `containedIn` this `Office`.

Fig.19 shows overlapping and nested regions, where the regions can be classified according to the `KindRG`s in Fig.12.

### 4.4 Relation to "Common Sense Ontology"

The relations `locatedIn`, `covers` and `containedIn` are examples of relations between different ontologies (or parts of one big joint ontology); this issue is taken up further in [2]. For example, there are various calculi for regions or other spatial configurations (e.g. [20, 21]); the ontology of such calculi, formally specified as suggested in Sect.5, could be associated here.

**Facing Windows.** is another example. Consider Fig.20 (also the objects in Fig.18): a place can be classified (as a subclass of `Place_IndoorsK`) as a `Place_InFrontOfWindow`, a `Place_InFrontOfLift`, etc.; it may then be related by `faces` to an object in the "Common Sense Ontology", e.g. a `Window` or `Lift`.

**Routemarks.** In such a situation, we may wish to attach an actual pointer to such an object, i.e. a `Vector_Mark_IndoorsK` having a `Node_IndoorsK`, e.g. a `Place_InFrontOfWindow`, as source and a `Routemark` as target; this `Routemark` is then a `Point` that `marks` a `Window` (e.g. in its centre). As examples, consider the dotted arrows in Fig.17. Routemarks help self-localisation during navigation using such vectors for triangulation.

### 4.5 Reference Systems

Analogously, a place may be rooted in a global reference system by a vector from its origin; this could also be a 3D Cartesian vector, of course. In fact, it is quite likely that such reference systems correspond to nested regions (Sect. 4.3).

[2] A origin has to be set before a pointer to a routemark is specified, cf. Sect.4.4.

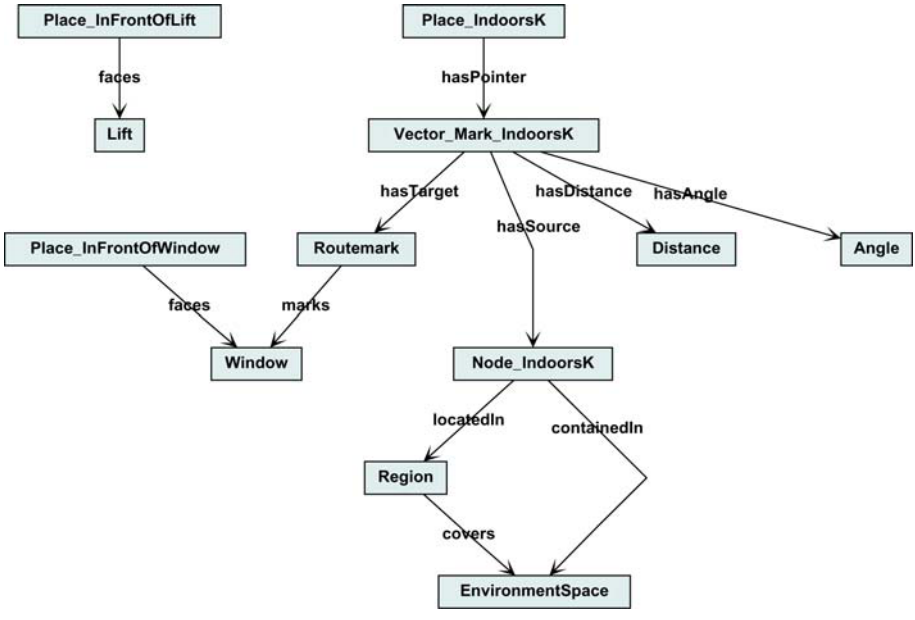


**Fig. 20.** Places facing a window, and routemarks

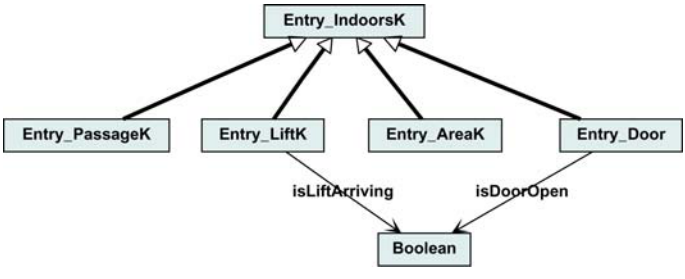


**Fig. 21.** Entries with dynamic predicates

### 4.6 Multi-robot Exploration

Multi-Robot Exploration may require additional information to be attached to places, for example a tag marking it as a fringe node for an explored region, to be extended. When various robots co-operate during exploration, place integration becomes of utmost importance, i.e. a criterion for matching two places, to become the same when "closing the loop" (cf. [22, 23, 21]). In such as case, different co-existing Route Graphs denote different possible worlds, with a certain probability attached; identical sub-graphs may of course be shared. We hope that the Route Graph representation will prove to be sufficiently compact for interaction between robots in such situations.

MMiSSLaTeX

```
\NewDecl{GenGraph}[1]  %[KindRG]
{\Class{Graph_#1}{\Ref{#1} graph}{}
      \RelType{hasNode}{Graph_#1}{Node_#1}
      \RelType{hasEdge}{Graph_#1}{Edge_#1}
\Class{Node_#1}{\Ref{#1} node}{}
\Class{Edge_#1}{\Ref{#1} edge}{}
      \RelType{hasSource}{Edge_#1}{Node_#1}
      \RelType{hasTarget}{Edge_#1}{Node_#1}
\GenSequence{Edge_#1}
 \Class{Path_#1}{\Ref{#1} path}{Sequence_Edge_#1}
  \Class{Route_#1}{\Ref{#1} route}{Path_#1}
```

**Fig. 22.** MMiSSLaTeX source of generic Graph ontology

### 4.7 Modelling Dynamic Information

So far, all the information associated with Route Graphs was static. For use in navigation, it would be very convenient to be able to model dynamic information as well. Consider Fig.21: entries are guarded by dynamic predicates,[3] to check whether a door is open or a lift is arriving. This way, Rolland may for example choose among segments to several lifts, depending on which is about to arrive. Thus status information is modelled; our graph can be regarded as a kind of state transition graph.

From a puristic point of view, this modelling is adequate for a data structure, but violates the idea of an ontology. However, the ontology (and the "data base" of its associated objects) should be the central source of information, the basic knowledge representation. It will yet have to be seen how these two views can be reconciled better, perhaps by representing dynamic information in a completely separate part of the ontology.

### 4.8 Structuring Ontologies

Finally, a word about structuring ontologies. Ontologies, as all descriptions or formal specifications, may become quite large and unwieldy; structuring becomes a necessity. We have learned a lot from structuring formal specifications or theories in the context of CASL, cf. [14, 15]: composite specifications may e.g. be constructed by (conservative) extension or union, items may be re-named, morphisms ("views") may be applied, (remote) libraries may be organised as nested folders.

We suggest to apply similar structuring mechanisms to ontologies. One way to do this is the approach of the MMiSS project, where general documents are structured along these lines, and change management supports sustainable development (cf. [12, 13]).

[3] Boolean valued binary relations to emphasise the dynamic query rather than unary predicates represented e.g. as subclasses.

**Generic Ontologies.** A particularly important structuring mechanism is parameterisation, or abstraction to generic modules, generic (sub)ontologies here. Fig.22 shows a generic graph ontology, as a generalisation of Fig.10: the (only) parameter "#1" may then be instantiated to different specialisations of `KindRG`, e.g. `IndoorsK`, `RailwayK` or `RoadK`. Note that, in its body, `GenSequence` is instantiated, a similarly generalised version of sequences.

## 5 Formalisation in CASL

This section contains the formalisation of the Route Graph ontology and the RouteGraph data type specification. There are several advantages for using CASL for the entire development of a data structure, together with its functions, from an ontology (see [14, 15, 16, 17], cf. Sect. 3.1, Fig. 9):

CASL covers full First Order Logic, predicates, partial and total functions, and subsorts. Tools in HETS (see below) are available for strong (sub-)type checking, overloading resolution, and interfaces to verification tools.

CASL-DL is a sublanguage of CASL, restricted to the Description Logic underlying OWL DL. It offers precise concepts and definitions together with a translation from and to OWL DL, cf. Fig. 9 and [18]. Thus a variety of tools developed for OWL DL become available for CASL-DL. The LaTeX ontology representation of **??**FirstOrderLogic can be translated to CASL-DL (or CASL) to provide a bare-bones specification.

**spec** GENGRAPH [**sort** *Kind*] =
  **sorts** *Graph*[*Kind*], *Node*[*Kind*], *Edge*[*Kind*]
  **preds** *hasNode* : *Graph*[*Kind*] × *Node*[*Kind*];
    *hasEdge* : *Graph*[*Kind*] × *Edge*[*Kind*]
  **ops** *hasSource*, *hasTarget* : *Edge*[*Kind*] → *Node*[*Kind*]
  ∀ *e*: *Edge*[*Kind*]; *s*, *t*: *Node*[*Kind*]; *g*: *Graph*[*Kind*]
  • *hasEdge*(*g*, *e*) ∧ *hasSource*(*e*) = *s* ∧ *hasTarget*(*e*) = *t* ⇒
    *hasNode*(*g*, *s*) ∧ *hasNode*(*g*, *t*)
**then**
  GENSEQUENCE [**sort** *Edge*[*Kind*]] **with** *hasHead*, *hasTail*, *EmptySeq*, *freq*
**then** ...
  **op** *sources* : *Sequence*[*Edge*[*Kind*]] → *Sequence*[*Node*[*Kind*]]
  ...
  **pred** *connected*(*l*: *Sequence*[*Edge*[*Kind*]]) ⇔
    (∀ *e1*, *e2*: *Edge*[*Kind*]
    • *hasHead*(*l*) = *e1* ∧ *hasHead*(*hasTail*(*l*)) = *e2* ⇔
      *hasTarget*(*e1*) = *hasSource*(*e2*)) ∧ *connected*(*hasTail*(*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*)}
**end**

**Fig. 23.** Generic Graph ontology in CASL

```
from Basic/StructuredDatatypes get List
from Basic/Numbers get Nat
from Basic/Graphs get NonUniqueEdgesGraph

spec Graph2 [sort Kind] given Nat =
      sorts  Node, Edge
      ops    source, target : Edge → Node
then
      NonUniqueEdgesGraph [sort Node] [sort Edge] and
      List2 [sort Edge] with [], _::_, atMostOnce
then
      pred  connected : List[Edge]
      ∀ rs1, rs2: Edge; rsl: List[Edge]
      • connected([])                                          %(connected_empty)%
      • connected([rs1])                                       %(connected_singleton)%
      • connected(rs1 :: rs2 :: rsl) ⇔
          target(rs1) = source(rs2) ∧ connected(rs2 :: rsl)     %(connected_rec)%
then ...
      op     sources : List[Edge] → List[Node]
      ...
      sorts  Path = {l: List[Edge] • connected(l)};            %(def_path)%
             Route = {l: Path • atMostOnce(sources(l)) ∧ atMostOnce(l)}
                                                               %(def_route)%
end
```

**Fig. 24.** Design specification in CASL

HETS – the Heterogeneous Tool Set – provides support for heterogeneous specification in various different logics (e.g. ModalCASL) and associated tools, most notably the interactive verification system Isabelle [24] and an increasing number of (semi-)automatic theorem provers, e.g. SPASS [25]. Translators to some target programming languages such as HASKELL or JAVA are under development. Moreover, HETS will support re-use of proofs with the Development Graph Manager MAYA [26].

We illustrate the development process proposed here with the example of generic (Route) Graphs.[4] It starts with an ontology stating only classes, relations and their axioms in a very loose fashion. This gives a top-level overview of the intended concepts and their interrelation.

### 5.1 Generic Graph Ontology in CASL

Fig. 23 shows the ontology of Route Graphs of a specific kind, given as parameter; cf. also Sect. 3.2 and Fig. 22. It starts with the declarations of the basic sorts and relations, and an axiom stating that no dangling edges are allowed.

[4] The formal specification of other parts of the ontology such as parts of the "Common Sense Ontology" or various spatial calculi (cf. Sect. 4.4) would also be quite interesting from the point of view of spatial cognition; we defer this to another paper.

Haskell

```
class Edge a where
    source :: a -> Node
    target :: a -> Node

data Path = forall a . (Eq a,Edge a) => ConPath [a]
data Route = forall a . (Eq a,Edge a) => ConRoute [a]

connected :: Edge a => [a] -> Bool
connected [] = True
connected ([_]) = True
connected (s1:s2:l) = target (s1) == source (s2) && connected (s2:l)

noBranches :: Edge a => [a] -> Bool
noBranches (l) = let sources = map source in atMostOnce (sources l)

isPath :: Edge a => [a] -> Bool
isPath = connected

isRoute :: (Eq a,Edge a) => [a] -> Bool
isRoute l = isPath l && noBranches(l) && atMostOnce(l)
```

**Fig. 25.** Haskell implementation of Edge

**Sequences of Edges.** For the relation of a graph to sequences of edges, the parameterized specification GenSequence is instantiated. It provides some basic relations (omitted here) such as *hasHead* and *hasTail* and some auxiliary functions for specification purposes, such as *sources*, yielding the sources of all edges in a graph as a sequence, or *freq* for counting the number of elements in a sequence. Note that parameterized specifications often use compound names which are to be instantiated, e.g. *Edge*[*Kind*] or *Sequence*[*Edge*[*Kind*]].

**Paths and Routes.** *Path*[*Kind*] is defined as a subsort of *Sequence*[*Edge*[*Kind*]] with the auxiliary predicate *connected*, then *Route*[*Kind*] as a subsort of *Path*[*Kind*] without duplicate branches or edges (this implies no cycles).

### 5.2 Design Specification in Casl

A Casl specification of Graph2 based on predefined basic data types such as List and NonUniqueEdgesGraph is presented in Fig. 24. It povides the same sorts as the ontological definition of GenGraph in Fig. 23 and is the first "design specification" in a software engineering sense. We can prove rather straightforwardly that this specification satisfies the requirements set out in Fig. 23.

### 5.3 Implementation of Inheritance in Haskell

We now turn to an operational implementation in the functional programming language Haskell and demonstrate some of the subtleties in the translation.

Fig. 25 shows the polymorphic implementation of the predicates `connected` and `noBranches` based on the type class `Edge`. The implementation of `connected` uses the same recursive definition as in Fig. 24. The existential data types `Path` and `Route` require type classes `Eq` and `Edge` as minimal context. All this, including constructor functions for `Path` and `Route` (not shown), can be implemented without knowledge about the actual structure of later instantiations.

The Route Graph specialisation of class `Edge` to class `Segment` is shown in Fig. 26. Note that providing special information for `entry`, `course` and `exit` is optional and yields a safe `Nothing` when e.g. an instance does not need an exit.

Haskell

```
class Edge a => Segment a where
    entry   :: a -> Maybe Angle
    entry _ = Nothing::Maybe Angle
    course   :: a -> Maybe Course
    course _ = Nothing::Maybe Course
    exit   :: a -> Maybe Angle
    exit _ = Nothing::Maybe Angle

data SegmentIndoorD = SegInd {segI_source :: Node,
                              segI_entry  :: Angle,
                              segI_course :: Course,
                              segI_exit   :: Angle,
                              segI_target :: Node}

instance Edge SegmentIndoorD where
    source = segI_source
    target = segI_target

instance Segment SegmentIndoorD where
    entry  = Just . segI_entry
    course = Just . segI_course
    exit   = Just . segI_exit
```

**Fig. 26.** Haskell implementation of `SegmentIndoor`

For the instantiation to indoors navigation, the data type `SegmentIndoorsD` is defined with the constructor `SegInd` (with special components/selector functions); it instantiates the classes `Edge` and `Segment`. The subsorting relation of SEGMENT and EDGE in CASL is translated to a class hierarchy in Haskell, where the data type includes all the information needed for all classes of the hierarchy.

Functions in class `Segment` are only accessible for data stored as `Path` or `Route` if we add the context `Segment` to the existential data types. Thus, we have to use the lowest class in the hierarchy that provides access to all information we need when we introduce existential types to hold heterogenous data.

## 6 Conclusion

We have presented the generic concept of Route Graphs, modelled by an ontology, and its step by step formalisation. We hope that the introduction of a "light-weight" ontology first, restricted to (sub)classes and relations only, will appeal to those who are not so interested in formalisation, and provides a general overview to those who are. Such an ontology specification can be done rather quickly in the special LaTeX format, and ontology visualisation tools (e.g. [10]) can already be applied. Moreover, such an ontology is already suitable for the semantic interrelation of documents (cf. [13]), as e.g. in the repository for a joint interdisciplinary research project such as the SFB/TR8 "Spatial Cognition".

The subsequent formalisation of the Route Graph ontology in CASL will appeal to those who emphasise the need for complete formalisation of ontologies, as in the DOLCE approach [16, 17]; here the structuring concepts of CASL and the availability of verification tools are perhaps the primary advantage.

The development of an actual data structure from a top-level ontology in CASL is a new experiment in Software Engineering since this first (loose) requirement specification is tailored to ontologies and not to software. Nevertheless, we believe that the approach works quite well, and the process will hopefully be further automated as much as possible. In the context of cooperation among and integration between a large interdisciplinary diversity of research and application areas (such as AI, Cognitive Science, Linguistics, and Psychology) we see it as a definite advantage to have a common "language", viz. semantic description formalism, to start from, *the* central (joint) ontology. We hope that those who are only interested in the result, e.g. a data structure for interaction in the robotics domain, will be patient with this "interdisciplinary definitional overhead", and happy with the outcome. We expect tools for the (formalised) ontology and interfaces to the data structure, with appropriate operations to simulate access "as an ontology", to coexist peacefully in the running system.

Finally, we hope to have made a contribution to clarifying concepts and providing a data structure for human–robot and robot–robot interaction for the indoors scenario. In particular, the (indoors) Route Graph ontology shall serve as a basis for linguistic augmentation to enable a dialogue between user and wheelchair, as outlined in [3, 4].

The approach presented in this paper will have to be extended, at the "user level", by a relation to spatial calculi, e.g. for treating "right" and "left", and nested overlapping regions. Formalisation of these calculi in CASL, at the ontology level, should be rather straightforward. The challenge will be the choice and combination of suitable calculi, and the transition between them.

We are looking forward to other instantiations of the generic Route Graph ontology (and data structures) in application domains such as geo-information systems or location-based service scenarios.

**Acknowledgement.** We thank the other members of the Route Graph and Ontology Working Groups of the SFB/TR 8, particularly John Bateman, Scott Farrar, Achim Mahnke, Reinhard Moratz, Hui Shi, and Tilman Vierhuff.

## References


1. Werner, S., Krieg-Brückner, B., Herrmann, T.: Modelling navigational knowledge by route graphs. In Freksa, C., Habel, C., Wender, K., eds.: Spatial Cognition II. Number 1849 in LNAI. Springer (2000) 295–317
2. Bateman, J., Farrar, S.: Modelling models of robot navigation using formal spatial ontology. In Freksa, C., Knauff, M., Krieg-Brückner, B., Nebel, B., Barkowsky, T., eds.: Spatial Cognition IV – Reasoning, Action, Interaction. Number 3343 in LNAI. Springer (2004) 366–389
3. Krieg-Brückner, B., Shi, H., Ross, R.J.: A safe and robust approach to shared control via dialogue. Journal of Software **15(12)** (2004) 1744–1755
4. Ross, R., Shi, H., Vierhuff, T., Krieg-Brückner, B., Bateman, J.: Towards dialogue-based shared control of navigating robots. In Freksa, C., Knauff, M., Krieg-Brückner, B., Nebel, B., Barkowsky, T., eds.: Spatial Cognition IV – Reasoning, Action, Interaction. Number 3343 in LNAI. Springer (2004) 478–499
5. Krieg-Brückner, B., Röfer, T., Carmesin, H.O., Müller, R.: A taxonomy of spatial knowledge for navigation and its application to the bremen autonomous wheelchair. In Freksa, C., Habel, C., Wender, K., eds.: Spatial Cognition. Number 1404 in LNAI. Springer (1998) 373–397
6. Uschold, M., Grüninger, M.: Ontologies: Principles, methods and applications. Knowledge Engineering Review **11** (1996) 93–155
7. Gómez-Pérez, A., Benjamins, V.R., eds.: Knowledge Engineering and Knowledge Management. Ontologies and the Semantic Web, 13th International Conference, EKAW 2002, Siguenza, Spain, October 1-4, 2002, Proceedings. In Gómez-Pérez, A., Benjamins, V.R., eds.: EKAW. Volume 2473 of Lecture Notes in Computer Science., Springer (2002)
8. Corcho, O., Gómez-Pérez, A.: A roadmap to ontology specification languages. In: Proceedings of the 12th European Workshop on Knowledge Acquisition, Modeling and Management, Springer (2000) 80–96
9. Bechhofer, S., van Hermelen, F., Hendler, J., Horrocks, I., McGuinness, D.L., Patel-Schneider, P.F., Stein, L.A.: W3C: OWL Web Ontology Language – Reference. W3C Recommendation `http://www.w3.org/TR/owl-ref/` (2004)
10. (`http://www.informatik.uni-bremen.de/agbkb/publikationen/softlibrary_e.htm`)
11. (`http://www.informatik.uni-bremen.de/~davinci/`)
12. Krieg-Brückner, B., Hutter, D., Lindow, A., Lüth, C., Mahnke, A., Melis, E., Meier, P., Poetzsch-Heffter, A., Roggenbach, M., Russell, G., Smaus, J.G., Wirsing, M.: Multimedia instruction in safe and secure systems. In: Recent Trends in Algebraic Development Techniques. Volume 2755 of Lecture Notes in Computer Science., Springer (2003) 82–117
13. Krieg-Brückner, B., Lindow, A., Lüth, C., Mahnke, A., Russell, G.: Semantic interrelation of documents via an ontology. In Engels, G., Seehusen, S., eds.: DeLFI 2004, Tagungsband der 2. e-Learning Fachtagung Informatik. Volume P-52 of Lecture Notes in Informatics., Springer (2004) 271–282
14. Astesiano, E., Bidoit, M., Krieg-Brückner, B., Kirchner, H., Mosses, P.D., Sannella, D., Tarlecki, A.: CASL – the common algebraic specification language. Theoretical Computer Science **286** (2002) 153–196 `www.cofi.info`.
15. Mosses, P.D., ed.: CASL Reference Manual. Volume 2960 of Lecture Notes in Computer Science. Springer (2004)
16. Gangemi, A., Guarino, N., Masolo, C., Oltramari, A., Schneider, L.: Sweetening ontologies with DOLCE. [7] 166–181

17. Lüttich, K., Mossakowski, T.: Specification of ontologies in CASL. In Varzi, A.C., Vieu, L., eds.: Formal Ontology in Information Systems, Proceedings of the Third International Conference (FOIS 2004). Volume 114 of Frontiers in Artificial Intelligence and Applications., IOS-Press (2004) 140–150
18. Lüttich, K., Krieg-Brückner, B., Mossakowski, T.: Ontologies for the semantic web in CASL. 17th Int. Workshop on Algebraic Development Techniques (2004) (to appear).
19. Wallgrün, J.O.: Autonomous construction of hierarchical voronoi-based route graph representations. In Freksa, C., Knauff, M., Krieg-Brückner, B., Nebel, B., Barkowsky, T., eds.: Spatial Cognition IV – Reasoning, Action, Interaction. Number 3343 in LNAI. Springer (2004) 413–433
20. Randell, D., Cui, Z., Cohn, A.: A spatial logic based on regions and connection. In: Proceedings KR-92, San Mateo, Morgan Kaufmann (1992) 165–176
21. Moratz, R., Wallgrün, J.O.: Spatial reasoning about relative orientation and distance for robot exploration. In Kuhn, W., Worboys, M., Timpf, S., eds.: Spatial Information Theory: Foundations of Geographic Information Science. Conference on Spatial Information Theory (COSIT). Lecture Notes in Computer Science, Springer (2003) 61–74
22. Frese, U., Larsson, P., Duckett, T.: A multigrid algorithm for simultaneous localization and mapping. IEEE Transactions on Robotics (2004) (to appear).
23. Frese, U.: A discussion of simultaneous localization and mapping. Autonomous Robots (2004) (submitted).
24. Nipkow, T., Paulson, L.C., Wenzel, M.: Isabelle/HOL — A Proof Assistant for Higher-Order Logic. Springer Verlag (2002)
25. Weidenbach, C., Brahm, U., Hillenbrand, T., Keen, E., Theobalt, C., Topic, D.: SPASS version 2.0. In Voronkov, A., ed.: Automated Deduction – CADE-18. Volume 2392 of Lecture Notes in Computer Science., Springer (2002) 275–279
26. Autexier, S., Hutter, D., Mossakowski, T., Schairer, A.: The development graph manager MAYA (system description). In Kirchner, H., Reingeissen, C., eds.: Algebraic Methodology and Software Technology, 2002. Volume 2422 of Lecture Notes in Computer Science. Springer (2002) 495–502