INFORMATIK
BERICHTE

360 – 08/2011

A Generic Data Model for Moving Objects

Jianqiu Xu, Ralf Hartmut Güting

Fakultät für Mathematik und Informatik
Postfach 940
D-58084 Hagen

A Generic Data Model for Moving Objects

Jianqiu Xu, Ralf Hartmut Güting

Database Systems for New Applications, Mathematics and Computer Science
FernUniversität Hagen, Germany
{jianqiu.xu,rhg}@fernuni-hagen.de

July 29, 2011

Abstract

The current data models for managing moving objects can be classified by environments such as *road network* or *indoor*. *Road network* is for moving objects like cars, taxis, and *indoor* is for humans moving inside a building. However, each model addresses only one specific case. To represent the complete movement for a moving object, especially a human who can move inside a building, walk on the pavement outside, drive the car on the road, and use the public vehicles (bus or train), more than one model are needed. In this paper, we propose a generic data model for representing moving objects that encapsulates multiple environments so that the database system only needs one model to manage all the data. Abstractly, the method is to let the *space* where an object moves be covered by a set of so-called *infrastructures* and to represent the location data by *referencing* to these *infrastructures*. Each *infrastructure* provides an environment for moving as well as its available places. It consists of a set of *infrastructure objects*. For example, road network is a kind of *infrastructure* and *indoor* environment is also a kind of *infrastructure* where the roads, streets and rooms are *infrastructure objects*. Besides, transportation modes (e.g., *Car, Bus, Walk*) are seamlessly integrated into the model to enrich moving objects with semantic data, making the representation more expressive than only focusing on location data. Because for humans' movement, it is necessary to know both *where* and *how*. In addition, our model represents the location in a multiresolution way where both imprecise and precise data is managed. With the generic model, a complete trajectory covering multiple environments can be represented.

The new model is carefully integrated with the type system and the framework of operators provided in earlier work for free space and for road networks. In this way, a powerful set of generic query operations is inherited and available for querying, together with extensions dealing with the new infrastructures and transportation modes. We demonstrate these capabilities by formulating a set of sophisticated queries across all infrastructures.

1 Introduction

Moving objects databases have been extensively studied in the last decade due to their wide applications, e.g., location-based services [21, 57, 23], transportation and road networks [34, 44, 6, 33], nearest neighbor queries [47, 22, 32, 18], trajectory searching [9, 27, 10]. Basically, a moving objects database manages spatial objects continuously changing locations over time. Although there has been a lot of work [42, 19, 13, 21, 45, 17] on modeling moving objects, the methods only address the issue in one specific environment. According to the environment, the current state-of-the-art can be classified into three categories:

1. *free space* [42, 53, 13, 19, 14];

2. *road (spatial) network* [49, 21, 45, 17];

3. *indoor* [25, 24].

Free space is an environment where the movement has no constraint and the locations are represented by coordinates. In the real world, objects usually move only on a pre-defined set of trajectories as specified by the underlying network (road, highway, etc) where the movement is limited by the environment. This is the *road network* environment. These first two environments are for *outdoor* movement. Meanwhile, people spend large parts of their lives in *indoor* spaces such as office buildings, shopping centers, etc. According to a Nokia report [1], people spend up to 90% of their time indoors (GPS only works for *outdoor*). Thus, the models for *indoor* moving objects are proposed. These models are diverse for different environments, e.g., location representation and data manipulation. Each data model manages the data individually and there is no interaction or connection between movement in different environments. Thus, for different applications the database system has to manage several models where each fits into one case. Besides, the queries are limited to one specific environment. The system cannot answer queries covering different environments, e.g., *indoor* and *outdoor*. Whereas for humans, the movement can cover all the above three cases, the complete trajectory cannot be managed by existing techniques. In this paper, we propose a generic data model to manage moving objects in all these contexts. It provides a global view of space where moving objects are located instead of focusing on an individual environment.

We consider the movement for humans that can cover several environments. To motivate the scope of this paper, consider the following two example movement scenarios of *Bobby*:

M_1: *walks from his house to the parking lot, and then drives the car along the road and highway to his office building, finally walks from the underground garage to his office room.*

M_2: *walks from the house to a bus stop, and then takes a bus to the train station, moves from one city to another by train, finally walks from the train station to his office room.*

The current techniques cannot manage the complete movement as the trajectories are split into several parts each of which fits into an environment (e.g., $road\ network$, $indoor$). Thus, queries like

Q_1:"*where is Bobby at 8:00 am, e.g., at home, on the street or in the office room?*", and

Q_2: "*how long does Bobby walk during his trip?*"

cannot be supported in one system. Because for Q_1, one first has to identify which moving environment $Bobby$ belongs to (e.g., $road\ network$, $indoor$) as the trajectory data is managed separately, and for Q_2 the walk movement is dispersed in different environments ($outdoor$ and $indoor$). Most existing methods [42, 53, 13, 19] use a pair (x, y) to identify the location; obviously it cannot apply for $indoor$ as it is a 3D environment. Of course, it is possible to extend the pair to a triple (x, y, z) to represent the location. But the method only focuses on geometric properties (e.g., coordinates). The raw location data (x, y, z) means nothing else than three real numbers so that one cannot recognize which part is for walking and which is for driving. To completely and richly identify the movement, one needs two factors: *where* and *how*. As for humans' movement, it can have several kinds of transportation modes, e.g., Car, Bus, $Walk$. To answer Q_2, one initially has to recognize the $Walk$ part in the whole movement, but raw location data cannot express such a kind of information.

Recently, in Microsoft's *GeoLife* project [2] researchers work on exploring and learning transportation modes of people's movement [60, 59, 58] from raw GPS data to enrich the mobility with informative and context knowledge. By mining multiple users' location histories, one can discover the top most interesting locations, classical travel sequences and travel experts in a given geospatial region, hence enable a generic travel recommendation. The difficulty is how to partition a user's trajectory into several segments and identify the transportation mode

for each piece. This is because the user's trajectory can contain multiple kinds of modes and the velocity of a mode suffers from the variable traffic condition.

Besides communication, mobile phones are also used to determine the transportation modes of an individual when outside [39]. A convenient classification system is proposed that used a mobile phone with a built-in GPS receiver and an accelerator where an accuracy level of 93.6% is achieved.

In a transportation system, it is more meaningful to support trip planning with different kinds of motion modes (e.g., $Walk \rightarrow Bus \rightarrow Train$) and the constraint on a specific mode, for example, less than two bus transfers. Whereas paper [6] presents a data model for trip planning in a multimodal transportation system, they do not address modeling moving objects. We propose a method for modeling moving objects in various environments and it seamlessly integrates transportation modes. The data model is more expressive as it can represent the semantic data as well as locations. Some researchers focus on mining semantic locations from GPS data [30, 61, 8].

Basically, there are two contrasting approaches for modeling geometric location data which can be classified into two categories: the *field-based* model and the *object-based* model [41]. The first takes into account the space as being continuous and empty, and there are discrete entities (objects) moving inside the space having their own properties, i.e., coordinates. References [42, 53, 13, 19] belong to this category. Another considers the world as a surface littered with recognizable and geographic objects that are associated with spatial attributes, and every location in space is represented by mapping it to those objects. Papers [40, 51, 7, 36, 17] fall into this category. Besides, trajectory pattern [3, 43] also takes into consideration the background geographic space where trajectories are located. A preprocessing step is involved there to construct the trajectory with semantic units, e.g., interesting places (hotels and airports).

In this paper, we follow the method of the *object-based* approach and propose a generic and expressive data model for moving objects. We consider the geographic *space* to be covered by a set of so-called *infrastructures*, where each corresponds to an environment. For example, *road network* is a kind of *infrastructure*, and $indoor$ is also an *infrastructure*. For each infrastructure consisting of a set of *infrastructure objects*, we model its available places as well as the relative position within a place. Afterwards, we represent the location by *mapping* it to these georeferenced *infrastructures*, more specifically, *mapping* it to *infrastructure objects*. Papers [19, 7, 36, 17] only consider one case where the model is only available for one environment, e.g., free space, road network, but here we model all environments and the representation is consistent with the previous ones for each specific case. We define a general framework for moving objects location representation that subsumes all specific environments. Besides, transportation modes are well embedded into the model. We know both where the object is and how it moves so that example queries Q_1 and Q_2 can be answered via one model.

In addition, our model represents the location data in a multiresolution mechanism where both imprecise and precise data is managed. This is because for some applications [36, 37], it may be not necessary to represent the location in its full complexity. For example, consider the query

Q_3:"*find all travelers passing two states during their trip*".

It is only necessary to manage the trajectory in such a resolution that one can determine whether the trajectory of a traveler intersects the state region or not, while the precise movement inside can be ignored. Because the higher the resolution is, the more space is needed to store the data and the more time is required for processing. Therefore, it is inefficient to work at full resolution if it is not required. But for most applications, a high resolution representation (precise location) is preferable which is adopted by most existing models [42, 19, 17]. Currently, there does not exist a method which can combine both of them, but our model can.

The database system is able to tune the level of scale to the appropriate value, making therefore the system much more powerful, while which level is preferable depends on the application requirement.

The main contribution of this paper is the design of a generic data model for managing moving objects in various environments. It includes the following specific contributions:

- We define a set of *infrastructures* covering the $space$ where an object moves and a general definition for all infrastructure objects. A generic model is proposed to represent the location data in various environments. And a framework for generic moving objects is proposed which can represent moving objects in all defined *infrastructures*. The multiresolution location representation is also covered by the model, i.e., precise and imprecise data.

- We model available places for each *infrastructure* and give the data type representing the geo-space it covers as well as the relative position inside. Applying the proposed framework of generic moving objects, we show how the location data is represented for each infrastructure. For one specific environment *indoor*, we also define a graph model for navigation supporting optimal routes searching with respect to different costs (e.g., distance, time).

- We define the type system for the generic model and design a set of operators on generic moving objects as well as those in a specific environment. The syntax and semantics of proposed operators are also defined. Type system and operations are carefully integrated and consistent with the well-established models in [19, 17] to obtain a powerful query language.

- An interface is provided to exchange information with a relational environment and a set of example queries are formulated which consider moving objects in all environments, demonstrating the expressiveness of the resulting querying framework.

The rest of the paper is organized as follows: The generic data model is presented in Section 2. It includes generic location representation, the framework for generic moving objects and multiresolution location representation. Section 3 describes how the location is represented in each *infrastructure*. Section 4 defines the type system and the operators on the proposed data types. A relational interface for exchanging data is defined in Section 5 and a comprehensive set of example queries are formulated in Section 6. In Section 7 we describe briefly the current state of the implementation. Section 8 reviews related work. Finally, Section 9 concludes the paper.

2 Generic Model

In this section, we present the generic location representation for moving objects in various environments. To define the data types in our model, we employ some basic data types from [13] which are introduced in Section 2.1. Section 2.2 introduces the transportation modes that we consider, defines a finite set of *infrastructures* and gives the generic location representation. The framework for generic moving objects is presented in Section 2.3 and the multiresolution location representation is discussed in Section 2.4.

2.1 Preliminaries

We give the carrier set of basic types that we use for the definitions in the following sections [1].

Definition 2.1 *Base Types*
$D_{\underline{int}} = \mathbb{Z} \cup \{\perp\}$
$D_{\underline{real}} = \mathbb{R} \cup \{\perp\}$
$D_{\underline{bool}} = \{FALSE, TRUE\} \cup \{\perp\}$

Each domain is based on the usual interpretation with an extension by the undefined value $\{\perp\}$. We give three data types representing time: *time instant*, *time interval* and *time range*, defined as follows:

Definition 2.2 *Time Types*
time instant: $D_{\underline{instant}} = \mathbb{R} \cup \{\perp\}$
time interval: $D_{\underline{interval}} = \{(s, e, lc, rc) | s, e \in D_{\underline{instant}}, lc, rc \in D_{\underline{bool}}, s \leq e,$
$(s = e) \Rightarrow (lc = rc = true)\}$
time range: $D_{\underline{periods}} = \{V \subseteq D_{\underline{interval}} |\ (u, v \in D_{\underline{interval}} \wedge u \neq v)$
$\Rightarrow$ *disjoint(u,v)* $\wedge \neg adjacent(u, v)\}$

We use $\underline{instant}$ to denote the data type for time instant, which is based on the $\underline{real}$ type. The value of a time interval, which is to define a set of time instants, is represented by two end instants and two flags lc and rc indicating whether at the end instant it is closed or open where the start instant s must be no later than the end instant e. Type $\underline{periods}$ represents a set of disjoint and non-connected time intervals. We also give the $\underline{intime}$ type constructor which yields types whose values consist of a time instant and a value. Let α denote an abstract type (excluding time type) which can be $\underline{int}$, $\underline{real}$, etc.

Definition 2.3 $D_{\underline{intime}} = D_{\underline{instant}} \times D_{\alpha}$

In addition, we also employ four spatial data types, $\underline{point}$, $\underline{points}$, $\underline{line}$ and $\underline{region}$, and two temporal types $\underline{mbool}$ (moving bool) and $\underline{mpoint}$ (moving point). Their corresponding domains are denoted by $D_{\underline{point}}$, $D_{\underline{points}}$, $D_{\underline{line}}$, $D_{\underline{region}}$, $D_{\underline{mbool}}$ and $D_{\underline{mpoint}}$ (for the technical definition we refer readers to [13, 19]).

2.2 Data Model

Using the *object-based* method, we let the $space$ where an object moves be covered by a set of *infrastructures* (introduced in detail in Section 2.2.1). Then, the location of a moving object is represented by mapping it to these *infrastructures*. Each *infrastructure* covers some places and is also related to a kind of transportation mode. For example, if the *infrastructure* is road network, the mode can be Car [2], $Taxi$, etc. If the *infrastructure* is the area such as the pavement or footpath, the mode is $Walk$. Table 1 lists the classification of movements on available places as well as the corresponding transportation modes. For a special case that the movement has no constraint in the environment (i.e., free space), we let the mode be $Free$.

Let $\underline{tm}$ denote the data type for transportation modes, whose carrier set is defined in the following.

Definition 2.4 *Transportation Mode*
$D_{\underline{tm}} = \{Car, Bus, Train, Walk, Indoor, Metro, Taxi, Bicycle, Free\}$

[1] Using the algebraic terminology that for a data type α, its domain or carrier set is denoted as D_{α}.
[2] we use Italics font to denote a kind of transportation mode.

Places	Transportation Modes
bus routes	Bus
railways	$Train$, $Metro$
buildings	$Indoor$
pavements, footpaths	$Walk$
roads, highways, streets	Car, $Bicycle$, $Taxi$
free space	$Free$

Table 1: Classification of Movements

It can be considered as an enumeration type. As people also walk in the $indoor$ space, in the following when we say mode $Walk$ it means outdoor environment by default, which is to distinguish the modes betweeen $Walk$ and $Indoor$. Here, we only take into consideration transportation modes for objects moving on the ground. Of course, there are still two kinds of transportation modes: $Ship$ and $Airplane$. Compared with the modes listed in Table 1, normally people do not frequently change to these two modes in daily life. So, we do not consider them in this paper but they can be easily integrated because both of them can be considered as public transportation vehicles.

2.2.1 Infrastructures and Infrastructure Objects

Each *infrastructure* describes a kind of environment for objects moving and it is composed of a set of elements called *infrastructure objects*. Each object is represented by a value where the type of the value is according to the *infrastructure* characteristic. For instance, in the road network (an *infrastructure*), a $line$ value is used to describe the geometrical property of a road or a highway, and in free space (also considered as an *infrastructure*) a $polygon$ or a $region$ (in the following, we use terms $polygon$ and $region$ interchangeably) is used to identify the pavement area for people walking. Besides, the *infrastructure* can also be an environment where the elements are moving objects, such as buses and trains in the public transportation network. The location of moving objects (specifically, humans as travelers) can be represented by *referencing* to these *infrastructure objects*. For example, streets and highways are treated as *infrastructure objects* which constitute the *infrastructure* road network. Then, the location of a driver in a car moving on a highway can be represented by a two-tuple (rid, pos) where rid corresponds to the highway and pos denotes the relative position along that highway [17]. In mobility pattern queries [36], they take into account objects moving in a free space and partition the space into a set of disjoint zones (*infrastructure objects*) each of which is represented by a $region$ and for each zone a label is assigned. Thus, the location is represented by mapping it to these zones. According to the movement classification in Table 1, a taxonomy of *infrastructures* considered in this paper is listed as follows:

(1) I_{ptn} : public transportation network (ptn for short);
(2) I_{indoor} : Indoor;
(3) I_{rbo} : Region-based Outdoor (rbo for short);
(4) I_{rn} : Road Network (rn for short);
(5) I_{fs} : Free Space (fs for short).

We believe they cover all cases of ground movement. For I_{ptn}, *infrastructure objects* are buses, trains and metros. For $indoor$, rooms, staircases, hallways, etc. are the elements, and for I_{rbo} the partitioned zones compose the *infrastructure*. The road network I_{rn} consists of streets, highways, etc. Free space is an empty *infrastructure*, i.e., no *infrastructure objects*.

The detailed representation for each *infrastructure* is presented in Section 3. As stated above, for each *infrastructure*, its elements are associated with a value of a data type representing the background geographic information. To be more general, let α be a general data type where the specific value we consider for each infrastructure is listed as below:

(1) I_{ptn}: $\alpha \rightarrow \underline{mpptn}$ (introduced in Section 3.1)
(2) I_{indoor}: $\alpha \rightarrow \underline{groom}$ (introduced in Section 3.2)
(3) I_{rbo}: $\alpha \rightarrow \underline{region}$
(4) I_{rn}: $\alpha \rightarrow \underline{line}$

First, we use the type $\underline{mpptn}$ representing moving points in public transportation network, such as buses, trains and underground trains. In this case, the movement of people can be represented by *referencing* to the bus/train they take. Second, for $indoor$ environment, basically it consists of 3D objects, such as rooms, corridors and staircases where the movement can be both horizontal and vertical. We use the type $\underline{groom}$ to describe 3D *infrastructure objects*. Next, a polygon is used to describe the area covered by the pavement and footpath, which is for the mode $Walk$. Last, a $line$ is the abstract description for the street, road, or highway. It is for Car, $Bicycle$, and $Taxi$. As I_{fs} is an empty set, no data type is needed.

To be clear with data type notations, we define a symbol for each data type. Let $IOType$ be the set of all data types for infrastructure objects and $IOSymbol$ be a set where its elements are the symbols for infrastructure objects data types, defined as below.

Definition 2.5 *Infrastructure Objects Data Types and Symbol Data Types*
IOType = $\{\underline{mpptn}, \underline{groom}, \underline{region}, \underline{line}, \perp\}$
IOSymbol = {*MPPTN, GROOM, REGION, LINE, FREESPACE*}

Let $\mu : IOSymbol \rightarrow IOType$ be a mapping from symbols to the full data types, e.g., $\mu(MPPTN) \rightarrow \underline{mpptn}$. In *free space*, there is no infrastructure object so that we let the data type be $\perp$. Then, we give the definition for general *infrastructure object* which consists of four attributes.

Definition 2.6 *General Infrastructure Object*
Let $IO(oid, s, \beta, name)$ denote an infrastructure object where $oid(\in D_{\underline{int}})$ is a unique object identifier, $s(\in IOSymbol)$ is a symbol for a data type, $\beta(\in D_{\mu(s)})$ is the value of a certain infrastructure object data type, and a string value $name$ is to describe the name of the object.

Let $I = \{I_{ptn}, I_{indoor}, I_{rbo}, I_{rn}, I_{fs}\}$ be the set of all *infrastructures*, and $Dom(I_i)(I_i \in I)$ be the set of values (*infrastructure objects*) for I_i. Then, we define the $space$ where an object moves as follows:

Definition 2.7 *A $space$ is defined as a set of infrastructure objects from all infrastructures, denoted by*

$$Space = \bigcup_{I_i \in I} Dom(I_i)$$

Let $\underline{space}$ be the data type representing the space, its domain is:
$D_{\underline{space}} = Space$
We let the space be comprised of a set of *infrastructure objects* which can be *static* or *dynamic*. The *static* components correspond to spatial objects, e.g., $regions$ or $lines$ and the *dynamic* part is for public transportation vehicles, such as buses and trains. Based on the *full* space (covered by *infrastructure objects*), in the next subsection we propose the general method to represent locations for moving objects.

2.2.2 Location

Using the *reference* method, there is a simple way for representing the location. That is, the location is represented by a function from time to an *infrastructure object*. But there is a significant drawback that the location is imprecisely described, i.e., it is only at the level of an object, e.g., a $region$. It is also necessary to know the exact and precise location in that object. Assuming the *referenced* objects now are the partitioned zones, queries like "*find all trips moving from zone A to zone B passing a location p in A*" can not be answered because it is unknown how the object moves inside A. To solve this problem, we propose that the *infrastructure object* should be combined with a second value representing the precise location within that *referenced object*. For the second value, we simply use a pair $i_{loc} = (x, y)$ to denote the position. The value i_{loc} is according to the *referenced (infrastructure) object* instead of the coordinate in the whole space.

Take into consideration a simple example in Figure 1 where there is a polygon denoted by $pl = \{(x, y)|5 \leq x \leq 35 \wedge 10 \leq y \leq 25\}$ covering a certain area (e.g., a plaza) in space and a point p located inside pl. Using the above method, we represent p by $(pl, (15, 10))$ where $(15, 10)$ denotes the local position in pl (we let the left lower point of the bounding box built on pl be the origin point). Alternatively, if it uses the global coordinate value in space, the point is denoted by $p' = (20, 20)$. In Section 3, we present in detail how the location i_{loc} is represented for each *infrastructure*. Because for different infrastructures, the values for describing *infrastructure objects* have different data types, e.g., $\underline{line}$, $\underline{region}$.

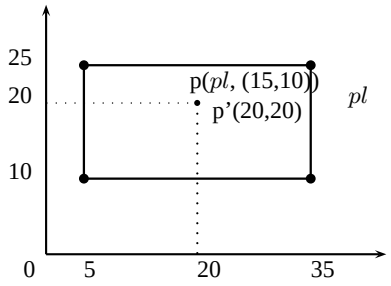

Figure 1: location description in an *infrastructure object*

To sum up, we represent the location by two parts where the first is an object identifier corresponding to an *infrastructure object* and the second stands for the relative position in that object. More formally, it is defined as follows.

Definition 2.8 *Generic Location*
$Loc = \{(\delta_1, \delta_2)|\delta_1, \delta_2 \in D_{\underline{real}}\}$
$D_{\underline{genloc}} = \{(oid, i_{loc})|oid \in D_{\underline{int}}, i_{loc} \in Loc\}$

We call it *Generic Location* as it can represent locations of moving objects in all environments listed in Table 1. Each element in Loc has two attributes denoted by δ_1, δ_2 for the position description. We do not use symbol x and y because the semantic has some other meaning (introduced in Section 3) other than only coordinate value.

For the elements in $D_{\underline{genloc}}$, each consists of two attributes where oid is the infrastructure object identifier and i_{loc} describes the position in the object. The representation supports different coordinate reference systems where *global* and *local* representation can be translated to each other (seeing the example in Figure 1). Based on Definition 2.8, now we give the representation for moving objects location.

Definition 2.9 *Let f be the location function projecting from time to a location. The definition is as follows:*

$$f : D_{\underline{instant}} \rightarrow D_{\underline{genloc}}$$

The continuous changing location data is represented by a function f where $D_{\underline{instant}}$ (domain) denotes the time instant and $D_{\underline{genloc}}$ (range) represents the set of elements for locations. Different from the method in [13, 19] which maps the time to a space domain which is for free space, here the elements in $D_{\underline{genloc}}$ can represent locations in multiple environments. Each element includes the space data that is achieved by a two level representation (imprecise and precise). It is richer than [19] in two respects:

- It is a generic data model that can be applied for various moving environments, e.g., *spatial network, indoor*. The location representation is not limited to a specific surrounding but can cover multiple cases (introduced in detail in Section 3). Both *global* and *local* coordinate reference systems are supported. At the same time, topological relations like "*contains*" and "*inside*" can be derived by oid. The model covers this aspect explicitly while involving geometric operations results in costly computation. Besides, the existing queries are preserved because the precise location is represented.

- The location is represented in a multiresolution way. The method of using the object identifier is an approximate location description, while the precise data is also managed by i_{loc}. This provides a flexible way to describe objects' movement, while the resolution depends on the specific application requirement.

Next, we define a data type called $\underline{genrange}$ representing sets of generic locations. Is is used on the one hand to represent the trajectory[3] of a generic moving object, i.e., its projection into generic space. On the other hand, it can represent any kind of curve or region in the generic space, for example, a road in the road network, a park in the region based outdoor space, a collection of rooms in a building, or an arbitrary region in free space. It is defined as follows.

Definition 2.10 *Generic Range*

$$Subrange = \{(oid, l, m) | oid \in D_{\underline{int}}, l \in D_{\underline{points}} \cup D_{\underline{line}} \cup D_{\underline{region}} \cup \{\perp\}, m \in D_{\underline{tm}} \cup \{\perp\}\}$$
$$D_{\underline{genrange}} = 2^{Subrange}$$

Each $\underline{genrange}$ value is a set of elements each of which is denoted by sub_traj_i ($\in Subrange$). sub_traj_i is composed of three attributes: oid is the infrastructure object identifier, l denotes a set of generic locations and m describes the transportation mode.

When a $\underline{genrange}$ value is used to represent the trajectory of a moving object, then l contains the locations passed by the object (usually as a $\underline{line}$ value with coordinates relative to the given infrastructure object). Also the transportation mode is defined.

When it is used to represent an arbitrary region in the generic space, l may be a $\underline{points}$ or $\underline{region}$ value or undefined (an undefined value meaning that all locations in the infrastructure object are valid). In this case the transportation mode may be undefined.

[3] In [19] the term *trajectory* is used for a moving object's projection into space, and there is an operator **trajectory** to perform this projection (also used later in this paper). Meanwhile, in the literature the term trajectory is often used to refer to a moving point object as a whole, including the time dimension.

2.3 Moving Objects Representation

In this paper, we focus on the complete movement history of moving objects (also called *trajectory* in the literature), instead of current and future position. To represent the continuous changing location data, a standard way to model it is by the *regression* method that first segments the data into pieces or regular intervals within which it exhibits a well-defined trend, and then chooses the basis and mathematical functions most appropriate to fit the data in each piece [13, 14, 48]. Using the method of *sliced representation* [13, 29], we represent a moving object as a set of so-called *temporal units (slices)*. Each $unit$ defines a time interval as well as the movement during it. First, we give a general definition for temporal units that applies to all units discussed in this paper. Let R be a set. The definition of *abstract temporal units* is given as follows:

Definition 2.11 *Abstract Temporal Units*
$Atu = D_{\underline{interval}} \times R$

The values of abstract temporal units have two components where the first defines a time interval and the second identifies an element from a set. Here a pair or unit $(i, r) \in Atu$ ($i \in D_{\underline{interval}}, r \in R$) says that during the time interval i, the movement of the unit is identified by r. The specific representation for r is introduced progressively in the following. Applying the above framework, we define the so-called *generic temporal units* for moving objects as follows:

Definition 2.12 *Generic Temporal Units*
$Gentu = \{(i, oid, i_{loc_1}, i_{loc_2}, m) | i \in D_{\underline{interval}}, oid \in D_{\underline{int}}, i_{loc_1}, i_{loc_2} \in Loc, m \in D_{\underline{tm}}\}$

Each element has five components where i defines a time interval, oid maps to an infrastructure object, i_{loc_1}, i_{loc_2} identify two positions in the object and m describes the transportation mode. Considering Definition 2.11, for an abstract unit (i, r), here r applies to the part $(oid, i_{loc_1}, i_{loc_2}, m)$. Each unit references to an infrastructure object with a certain transportation mode during the time interval. It says that at time instant $i.s$ the object is located at i_{loc_1} in the *infrastructure object* oid and at time instant $i.e$ it has moved to i_{loc_2} in that infrastructure object with the transportation mode m. The position during time interval $[i.s, i.e]$ is achieved by a discrete function associated with the object oid. Notice that the mathematical functions manipulating the data are diverse for different infrastructure objects (introduced in Section 3). Given two units $u_1, u_2 \in Gentu$ that cover the same time interval ($u_1.i = u_2.i$), some relationships can be exploited between them:

1. $u_1.m \neq u_2.m$

   It denotes two units with different transportation modes. For example, $u_1.m$ = Car and $u_2.m$ = $Walk$. Queries on transportation modes extract data from this attribute.

2. $u_1.m = u_2.m \wedge u_1.oid \neq u_2.oid$

   The transportation modes are the same, but the *referenced* infrastructure objects are different. Let $u_1.m$ = Bus, $u_2.m$ = Bus, then $u_1.oid$ and $u_2.oid$ denote different buses.

3. $u_1.m = u_2.m \wedge u_1.oid = u_2.oid$

   In this case, both transportation modes and infrastructure objects are the same. But the precise location in space can be different. For example, two pedestrians walk on the same pavement or two clerks walk inside the same office room. They can have different locations in space which are distinguished by i_{loc_1} and i_{loc_2}.

Given two generic temporal units $u_1, u_2 \in Gentu$ (with $u_1.i.e \leq u_2.i.s$), let sub_traj_1, $sub\_traj_2 (\in Subrange)$ denote their projection movements in space. Since each unit represents a linear movement of the object, we can use a function for the evaluation at time t along the line. Let $p_1 (\in D_{\underline{point}})$ be the start location of u_1 and $p_2 (\in D_{\underline{point}})$ be the start location of u_2 which correspond to the start endpoints of sub_traj_1 and sub_traj_2, respectively. Let T_1, T_2 (e.g., $[0, u_1.i.e - u_1.i.s]$) be the relative time intervals of u_1 and u_2, and we give the evaluation functions for u_1 and u_2.

$f_{u_1}(t) = \{(x, y) | x = p_1.x + a_1 \cdot t, y = p_1.y + b_1 \cdot t, t \in T_1, a_1, b_1 \in \mathbb{R}\}$
$f_{u_2}(t) = \{(x, y) | x = p_2.x + a_2 \cdot t, y = p_2.y + b_2 \cdot t, t \in T_2, a_2, b_2 \in \mathbb{R}\}$
Then we define two units u_1 and u_2 are ***mergeable*** $\Leftrightarrow$
(i) $u_1.oid = u_2.oid \wedge u_1.m = u_2.m$
(ii) $u_1.i$ is ***adjacent*** to $u_2.i$
(iii) $f_{u_1}(T_1.e) = f_{u_2}(T_2.s) \wedge a_1 = a_2 \wedge b_1 = b_2$
Based on the generic temporal units, we give the definition of generic moving objects.

Definition 2.13 *Generic Moving Objects*
$D_{\underline{genmo}} = \{< u_1, u_2, ..., u_n > | n \geq 0, n \in D_{\underline{int}}$, ***and***
(i) $\forall i \in [1, n], u_i \in Gentu$
(ii) $\forall i, j \in [1, n], i \neq j \Rightarrow u_i.i \cap u_j.i = \oslash \wedge u_i, u_j$ ***are not mergeable*** $\}$

Each moving object consists of a sequence of temporal units where each unit corresponds to an infrastructure object, the movement in that infrastructure object and a kind of transportation mode. We only consider the case that the geometry property of a moving object is abstracted as a *moving point* (i.e., focus on the position but ignore the extent variation of the object), while *moving line* and *moving region* are out scope of this paper. Within this framework, we can describe objects' movement in various environments.

2.4 Multi-Scale Representation

In our model, we represent the location data in a multi-scale way: the imprecise level by only referencing to an *infrastructure object* and the precise level by both the *infrastructure object* and the relative position inside it. Each *infrastructure object* is associated with a value representing the space it is covering, e.g., a $line$. By *referencing* to it, we know that the object's position is bounded or inside that area, which is an approximate value. If both levels are used, the accurate location of moving objects is known. But for some applications [37, 36], the full resolution for location data may not be necessary so that it is much simpler and more efficient to work with the approximate location. Thus, we can represent the location by only *referencing* to an infrastructure object, while the precise movement inside is ignored. Applying the framework of *generic temporal units* in Definition 2.12, we give the definition of so-called *low resolution temporal units* as follows.

Definition 2.14 *Low Resolution Temporal Units*
$Lowgentu = \{(i, oid, i_{loc_1}, i_{loc_2}, m) |$
(i) $i \in D_{\underline{interval}}$***;***
(ii) $oid \in D_{\underline{int}}$***;***
(iii) $i_{loc_1}, i_{loc_2} \in Loc, i_{loc_1} = (\perp, \perp) \wedge i_{loc_2} = (\perp, \perp)$***;***
(iv) $m \in D_{\underline{tm}}\}$

It is a subset of the domain of *generic temporal units* where the value in i_{loc_1}, i_{loc_2} is set as undefined, while the framework of *generic moving objects* still applies. In Section 3.3, we will

show an example of it. Here, for the low resolution of location data, we only represent it by mapping the location to an infrastructure object.

In some cases, this rough description may not provide enough information for the query user if the infrastructure object is very large in space. For example, if it is on a highway the length of which can be hundreds of kilometers or on a relatively long city street, e.g., two or three kilometers, then the rough location representation with only an object identifier seems to be not enough. To provide the location data in a more precise and meaningful way, we can do a second partition on each infrastructure object. The area covered by an infrastructure object can be partitioned into a set of pieces and $i_{loc_1}.\delta_1$ ($i_{loc_2}.\delta_1$) can be used to store the partition identifier. The partition methods are different for each kind of infrastructure depending on the data type used to represent the infrastructure objects. For the infrastructures and infrastructure objects defined in Section 2.2.1, we propose the partition methods in the following.

- $I_{ptn}(\underline{mpptn})$

  We believe it is enough to know which bus or train the traveler takes, while the relative location inside can be ignored. Thus, for this kind of object no partition is needed.

- $I_{indoor}(\underline{groom})$

  We first briefly describe the data type $\underline{groom}$. Basically, it represents a room by a 2D area plus a value denoting the height above some *ground level* of the building. We use the type $\underline{region}$ to denote a 2D area and do a cell partition on it where each cell can be represented by a rectangle (region). So, the low resolution location description is composed of two parts: a room identifier oid and a cell *id*. The cell size depends on the application requirement.

- $I_{rbo}(\underline{region})$

  We use the same method as for I_{indoor} that a region is partitioned into a set of disjoint cells.

- $I_{rn}(\underline{line})$

  As a line consists of a sequence of segments, each partition can correspond to a segment. The rough location description first references to a line and second references to a segment in that line.

- I_{fs} No partition.

Although the second partition is better than only referencing to an object identifier, the location description is still at an imprecise level. Here, we just show that our model can represent the low resolution location in two levels, while whether the second partition is required is determined by the application. In the rest of the paper, we assume only the object identifier is used for rough location description.

3 Representation for Infrastructures

In this section, we give the location representation for each infrastructure. Section 3.1 introduces the public transportation network and Section 3.2 addresses the *indoor* environment. Region-based outdoor is discussed in Section 3.3. Road network and free space are described in Sections 3.4 and 3.5, respectively.

3.1 Public Transportation Network

The elements (*infrastructure objects*) in this kind of infrastructure include buses, trains, and underground trains. As they have similar characteristics, without loss of generality we take the public bus as an example to present how the infrastructure and location is represented. The bus network is located in the environment *road network* but with additional features. It has two types of components: *static* and *dynamic*, where the first includes *bus stops* and *bus routes*, and the second are moving buses. Each bus has a regular mobility pattern as it normally moves along a pre-defined bus route and has a sequence of bus stops where the movement suspends.

3.1.1 Static

A bus stop identifies a location in space where a bus first arrives, lets passengers get on and off, and then departs (if it is not the last stop). Each bus stop belongs to one bus route, but several bus stops from different routes can have the same spatial location where the transfer occurs. A bus route contains a sequence of bus stops which partition the whole route into a sequence of sub lines, each part is called a *route segment* (*segment* for short). Each segment defines the connection between two consecutive stops. The bus can only stop at these places for passengers getting on and off. A route has a start location and an end location which can be equal to each other (a cyclic route). The route may be not the shortest path connecting two locations in a road network, but it defines a path along which the bus should move.

Bus Stop: Intuitively, a bus stop corresponds to a point in space, but it also has some other information, e.g., which bus route it belongs to. Instead of simply using a point to identify a bus stop, we define a data type named $\underline{busstop}$ with its carrier set given in the following.

Definition 3.1 *Bus Stop*

$D_{\underline{busstop}} = \{(rid, pos) | rid, pos \in D_{\underline{int}}, rid \geq 0 \wedge pos \geq 0\}$

rid and pos denote which bus route the stop belongs to and the relative order in that route, respectively. We define a relationship called *adjacent* between two bus stops. Given bs_1, $bs_2 \in D_{\underline{busstop}}$,

bs_1 is *adjacent* to $bs_2 \Leftrightarrow bs_1.rid = bs_2.rid \wedge bs_1.pos + 1 = bs_2.pos$

Let p_bs_i $(\in D_{\underline{point}})$ be the spatial point that bs_i $(\in D_{\underline{busstop}})$ corresponds to (later, we introduce how to retrieve it). Then, the equality of two bus stops is defined as:

$bs_1 = bs_2 \Leftrightarrow p\_bs_1 = p\_bs_2$.

There are two possibilities if $bs_1 = bs_2$:

(1) $bs_1.rid = bs_2.rid \wedge bs_1.pos = bs_2.pos$;

(2) $bs_1.rid \neq bs_2.rid$.

The first case should be straightforward, while the second illustrates that two different bus routes can have an intersection bus stop where the *bus transfer* can happen. The geo-location of a bus stop is described by a point, but a point can correspond to several bus stops belonging to different bus routes.

Bus Route: A bus route consists of a sequence of *segments*, each of which represents the geographic connection between two *adjacent* bus stops. Consequently, we represent a bus route by a sequence of segments. Let $Busseg$ be the set of bus segments defined as below:

Definition 3.2 *Bus Segment*

$Busseg = \{(bs_1, bs_2, geo) | bs_1, bs_2 \in D_{\underline{busstop}}, bs_1, bs_2$ *are adjacent*, $geo \in D_{\underline{line}}\}$

Each segment is composed of two bus stops recording two endpoints and a line geo denoting the geographical connection in space. It represents the connection between two *adjacent* bus stops. Given two segments seg_1, $seg_2 \in Busseg$, we define

seg_1 is $linkable$ to $seg_2 \Leftrightarrow seg_1.bs_2 = seg_2.bs_1$,
that is $seg_1.bs_2$ and $seg_2.bs_1$ have the same spatial point in space.
Let $\underline{busroute}$ be the data type for *bus routes* and the domain is:

Definition 3.3 *Bus Route*
$D_{\underline{busroute}}$ = $\{< seg_1, seg_2, ..., seg_n > | n \geq 1, n \in D_{\underline{int}}$, *and*
(1)$\forall i \in [1, n], seg_i \in Busseg$;
(2) $\forall i \in [1, n\text{-}1], seg_i.bs_1.rid = seg_{i+1}.bs_1.rid \wedge seg_i$ *is* $linkable$ *to* seg_{i+1} $\}$

A *bus route* is defined as a sequence of bus segments all of which have the same rid and each two consecutive segments are *linkable*. Let operator **geo_data** return the geometry line of a bus segment. Then the whole line of a bus route can be constructed via $\bigcup_{i=1}^{n}$**geo_data**(seg_i). Two consecutive segments seg_i and seg_{i+1} have an intersection point in space where seg_i ends and seg_{i+1} starts, and the point identifies the position of a bus stop. Given a bus route r, let m be the number of its segments and n be the number of bus stops. Then, m and n satisfy $r.n = r.m + 1$. Given a $\underline{busstop}$ $bs = (rid, pos)$, the indices for the bus segments ending and starting at it are $bs.pos - 1$, $bs.pos$. Thus, the geometry data (a $\underline{point}$) for the bus stop can be obtained which is the endpoint of a bus segment.

To illustrate how the above data types work together, Figure 2 gives an example with three bus routes $\{r_1, r_2, r_3\}$ and five points $\{s_1, s_2, s_3, s_4, s_5\}$ to identify bus stop locations. The points are connected by segments from different bus routes (labeled by symbol seg with a subscript) where seg_1, seg_2 are for r_1, seg_3, seg_4 are for r_2, and seg_5, seg_6 are for r_3. Figure 3 lists the representation for bus stops and bus routes. For brevity, we omit the detailed geometry description for points and lines here. The points from $\{s_1, s_2, s_3, s_4\}$ represent the intersection locations of two bus routes where each denotes the position of two bus stops. This implies that a *bus transfer* can happen here.

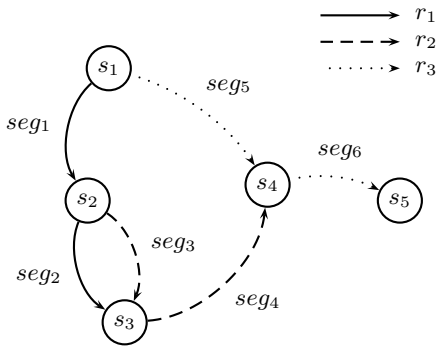

Figure 2: A Simple Bus Network

One issue that needs to be discussed is that in the real world a bus stop is combined with a name and normally it has two different positions in the road where each corresponds to one direction along the route (*Up* and *Down*). In fact, the two stops belong to one bus route but with different directions. This invokes the question how to distinguish between them. To solve the issue, we define each bus route with two directions as two different routes where each is uniquely identified and represented by different geometrical lines. In the real world, the *Up* and *Down* routes are normally located on two different lanes of a road and the geometry description for them should be different.

Point	Bus Stops
s_1	$bs_1 =< 1,1 >$, $bs_7 =< 3,1 >$
s_2	$bs_2 =< 1,2 >$, $bs_4 =< 2,1 >$
s_3	$bs_3 =< 1,3 >$, $bs_5 =< 2,2 >$
s_4	$bs_6 =< 2,3 >$, $bs_8 =< 3,2 >$
s_5	$bs_9 =< 3,3 >$

(a)

RouteId	Bus Routes
1	$br_1 = \{seg_1(bs_1, bs_2), seg_2(bs_2, bs_3)\}$
2	$br_2 = \{seg_3(bs_4, bs_5), seg_4(bs_5, bs_6)\}$
3	$br_3 = \{seg_5(bs_7, bs_8), seg_6(bs_8, bs_9)\}$

(b)

Figure 3: Static Component

3.1.2 Dynamic

A moving bus contains two types of information: *bus route* and *schedule*. The route is fixed and limits the bus movement. The buses belonging to the same route have different time schedules, i.e., *departure* and *arrival* time at each stop. Each schedule for a bus is called a *bus trip*. We model each *bus trip* as a moving point and represent its movement as follows. Let $D_{\underline{busloc}}$ be the domain of bus locations.

Definition 3.4 *Bus Location*

$D_{\underline{busloc}} = \{(br\_id, bs\_id, pos) | br\_id, bs\_id \in D_{\underline{int}}, pos \in D_{\underline{real}}\}$

The location of a bus is represented by a bus route id, a stop number of that route and the relative distance from the stop position along the route. Note that the definition is consistent with the generic location in Def. 2.8 ($oid \rightarrow br\_id, i_{loc} \rightarrow (bs\_id, pos)$), where here it is a specific case for the location on a bus route. To get the spatial point of a bus location, one can call the function **geodata** with the input bus location and corresponding bus route. Given $bloc_1, bloc_2 \in D_{\underline{busloc}}$, we define $bloc_1$ and $bloc_2$ are *successive stops* $\Leftrightarrow$

(1) $bloc_1.br\_id$ = $bloc_2.br\_id$;

(2) $bloc_1.bs\_id + 1$ = $bloc_2.bs\_id$;

(3) $bloc_1.pos$ = $bloc_2.pos$ = 0.

Let $Bustu$ be the domain of temporal units for a *bus trip*, which is defined in the following.

Definition 3.5 *Temporal Units for a Bus Trip*

$Bustu = \{(i, bloc_1, bloc_2) | i \in D_{\underline{interval}}, bloc_1, bloc_2 \in D_{\underline{busloc}},$ *and*

(i) $i.s = i.e \Rightarrow bloc_1 = bloc_2$*;*

(ii) $bloc_1, bloc_2$ *are successive stops or* $bloc_1, bloc_2$ *are equal* $\}$

The values have three components where the first defines a time interval and the second and third stand for two locations on a bus route. The above definition is derived from Definition 2.11 where R is instantiated to $D_{\underline{genloc}} \times D_{\underline{genloc}}$. Each element in $Bustu$ describes that at $i.s$ the location is identified by $bloc_1$, and at $i.e$ the object has moved to $bloc_2$. According to the value of $bloc_1$ and $bloc_2$ during i, a unit $tub \in Bustu$ represents two cases:

case 1: $bloc_1, bloc_2$ are *successive stops*

case 2: $bloc_1, bloc_2$ are *equal*

The first illustrates a normal movement from one bus stop to the successive one where both of them belong to the same bus route. The second denotes a small deviation time at the bus

stop illustrating the behavior that a bus stays at the stop waiting for passengers getting on and off. With the above two cases, the temporal unit can represent different movement behaviors of a bus. **Case 1** can be considered as the movement in both *temporal* and *spatial* dimension, and **Case 2** is the movement only in *temporal* dimension. Note that Definition 3.5 is consistent with the definition of temporal units in Definition 2.12. Let $v \in Gentu, u \in Bustu$, and we have

(1) $v.i \rightarrow u.i$;
(2) $v.oid \rightarrow u.bloc_1.br\_id(u.bloc_2.br\_id)$;
(3) $v.i_{loc_1} \rightarrow (u.bloc_1.bs\_id, u.bloc_1.pos)$;
(4) $v.i_{loc_2} \rightarrow (u.bloc_2.bs\_id, u.bloc_1.pos)$;
(5) as u denotes a specified infrastructure unit, $v.m(Bus)$ is not needed in u.

Let $\underline{mpptn}$ be the data type representing *bus trips*. Based on Definition 3.5, we give the definition as follows.

Definition 3.6 *Bus Trips*
$D_{\underline{mpptn}} = \{< tub_1, tub_2, ..., tub_n > | n \geq 1, n \in int$, *and*
(i)$\forall i \in [1, n], tub_i \in Bustu$;
(ii) $\forall i, j \in [1, n], i \neq j \Rightarrow tub_i.i \cap tub_j.i = \emptyset$;
(iii) $\forall i, j \in [1, n], tub_i.bloc_1.br\_id = tub_j.bloc_1.br\_id$; $\}$

Condition (iii) is to guarantee that each *bus trip* only belongs to one bus route so that all *temporal units* have the same route identifier. For a bus, the regular mobility pattern is that it starts from a bus stop, goes forward to another, normally staying there for a short time period (e.g., 30 seconds), and then proceeds to the next. We define the behavior that the bus stays at a bus stop for a short time as a special kind of movement where the spatial location remains constant over time, called t_{move}. In contrast to t_{move}, the behavior that the bus moves from one bus stop to another is called st_{move} where the spatial content changes over time. These two movements occur alternating. Instead of explicitly identifying the location by geographical data, the trajectory of a moving bus is represented by a sequence of units describing all bus stops it reaches in conjunction with the temporal property. Each *unit* is described by a time interval and two locations on the bus route. The two locations can be equal or different, which correspond to t_{move} and st_{move}, respectively. As each bus moves along a bus route represented by a curve in space, the position between two successive stops can be determined by *linear interpolation* along that curve. This representation is more compact because compared with the times of frequently updating raw location data (e.g., coordinates), the number of bus stops is very small. Both update and storage cost are reduced. The units in each bus trip fit into the framework of generic temporal units and bus trips also fit into the framework of generic moving objects in Definition 2.13.

3.1.3 Infrastructure Objects

With the above data types, we give the specific representation for *infrastructure objects* in I_{ptn}. For static infrastructure objects, we extend Definition 2.5 to include the symbols and types for bus stops and bus routes.

Definition 3.7 *Extended Infrastructure Objects Data Types and Symbol Data Types*
IOType := IOType $\cup \{\underline{busstop}, \underline{busroute}\}$
IOSymbol := IOSymbol $\cup$ {*BUSSTOP, BUSROUTE*}

Applying Definition 2.6, there are two kinds of *infrastructure objects* in bus network:

- static: $IO_{ptn}(oid, \textbf{BUSSTOP}, \beta, name)(\beta \in D_{\underline{busstop}})$
  $IO_{ptn}(oid, \textbf{BUSROUTE}, \beta, name)(\beta \in D_{\underline{busroute}})$
- dynamic: $IO_{ptn}(oid, \textbf{MPPTN}, \beta, name)(\beta \in D_{\underline{mpptn}})$

The *static* objects are *referenced* by *dynamic* objects, while the movement of humans is represented by *referencing* to *dynamic infrastructure objects*.

However, here one has to be a bit careful. One may be tempted to assume that to define the movement of a traveler on a bus, specifying the time interval and a reference to the bus object in the infrastructure is sufficient. However, the infrastructure object describes the *scheduled bus movement* from which the real bus movement may deviate (e.g. due to delays). For example, one would like to determine in a query at which bus stop a passenger entered the bus. Deriving this from the scheduled location of the bus at the time the passenger enters may lead to errors and inconsistencies with the remaining trip of the passenger (e.g. in the region based outdoor infrastructure). We therefore include in the unit describing a passenger's trip also the start and end locations on the respective bus route.

Applying the framework of *generic temporal units* in Definition 2.12, let $u_{ptn} = (i, oid, i_{loc_1}, i_{loc_2}, m)$ be a temporal unit representing the movement of a human in ptn where the meanings are:

(1) $oid \rightarrow IO_{ptn}(oid, \textbf{MPPTN}, \beta, name)(\beta \in D_{\underline{mpptn}})$;
(2) $i_{loc_1} \rightarrow (bs\_id, pos)$;
(3) $i_{loc_2} \rightarrow (bs\_id, pos)$;
(4) $m \rightarrow Bus$.

The unit represents that during time interval i the object's movement is determined by the *infrastructure object* identified by oid, moving from i_{loc_1} to i_{loc_2}. The values of i_{loc_1} and i_{loc_2} represent the start and end locations (bus stops) of the traveler where bs_id records the stop number and pos records the distance from the stop position. Note that the bus route id is already handled by the referenced infrastructure object (the moving bus). The representation can efficiently reduce the storage size for the traveler's trajectory. Assuming a traveler takes a bus, instead of recording the units at all possible locations that the bus speed or direction changes, now only one unit is required. And if several passengers take the same bus, all of them can *reference* to the same *infrastructure object*. Otherwise, the location data for each of them has to be stored. The travelers may get on and off the same bus at different stops, which are distinguished by i_{loc_1} and i_{loc_2}. In addition, one does not have to update the data for travelers until they get off the bus or switch to another one.

Bus Trajectory and Traveler's Trajectory

According to Definition 2.10, the bus or a traveler's trajectory is a set of $sub\_traj_i (\in Subrange)$ where the attribute values are:

(1) $sub\_traj_i.oid \rightarrow IO_{ptn}(oid, \textbf{BUSROUTE}, \beta, name)(\beta \in D_{\underline{busroute}})$;
(2) $sub\_traj_i.l$ stores the movement along the route;
(3) $sub\_traj_i.m \rightarrow Bus$.

3.2 Indoor

In the *indoor* environment, there are two kinds of elements where one is for the place that people can stay and move inside such as rooms, chambers, corridors, and the other is for transitions between two places, such as office doors, entrances/exits for staircases. We call the first $groom$ (general room) and the second $door$. Normally, a $groom$ contains at least one door. For example, an office room may have only one door, but a corridor can have several doors via

which people can leave one room, and enter another. The space of a door is covered by the $groom$ it belongs to. And a door builds the connection between two $grooms$. We represent the position of an *indoor* moving object by mapping it to a $groom$. In Section 3.2.1, we give the data type representing $groom$ and $door$ as well as the location representation for *indoor* moving objects. In Section 3.2.2 we create a graph for *indoor* navigation.

3.2.1 Modeling Indoor Space

groom: The whole indoor environment comprises of a set of 3D geometry objects where each corresponds to an office room, a corridor, a chamber, etc. The object can be modeled as a 3D polygon. Let $Region3d$ be the set of 3D polygons defined as follows:

Definition 3.8 *Region3d*

$Region3d = \{(poly, h) | poly \in D_{\underline{region}}, h \in D_{\underline{real}}\}$

The attribute $poly$ describes the 2D area in space and h denotes the height of the room above the ground level of the building. We assume the unit for the height is $meter$. Given $r_1, r_2 \in Region3d$, we define

$r_1 = r_2 \Leftrightarrow r_1.poly = r_2.poly \wedge r_1.h = r_2.h$

Based on Figure 1, consider an example shown in Figure 4 where a large rectangle bounded by bold lines denotes the vertical view of an office building. There is an office room inside denoted by r_{312} with the bounding box plotted by dotted lines. We assume the room height is 9 (meter). Then, the room is represented by $r_{312} = (poly_{312}, 9.0)(poly_{312} \in D_{\underline{region}}$, we omit the detailed description here) and the bounding box of r_{312} is denoted by $bb\_r_{312} = (bb_{312}, 9.0)$ ($bb_{312} = \{(x, y) | 10 \leq x \leq 21 \wedge 13 \leq y \leq 20\}$). A point p is located in r_{312} which is denoted by $(r_{312}, (5, 3))$ and $(5, 3)$ is the relative value according to bb_r_{312} (the left lower point of bb_r_{312} is the origin point). But its global coordinate in space can still be transformed by p and bb_r_{312}, that is $p' = (15, 16)$.

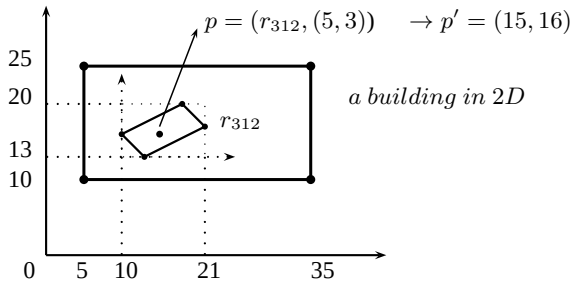

Figure 4: Relative Position in $Indoor$

In the $indoor$ environment, the case exists that the height above ground level of a moving object changes during its movement. Usually this occurs when the object moves on a staircase, but it can also happen in an amphitheater or a chamber where one room has several floors with different altitudes. This results in 2D areas having different height values for one room. To be more general, based on Definition 3.8 we define a data type named $\underline{groom}$, given in the following:

Definition 3.9 *Data Type for General Room*

$D_{\underline{groom}} = \{GR \subseteq Region3d | (gr_1, gr_2 \in GR \wedge gr_1 \neq gr_2) \Rightarrow$
$disjoint(gr_1.poly, gr_2.poly)\}$

Each element in $D_{\underline{groom}}$ is a set of 3D regions where the 2D areas are disjoint. Using the staircase as an example, a staircase (an *infrastructure object*) between two floors is modeled as a set of 3D regions where each represents one footstep of the staircase. For an elevator where the 2D area is the same on each floor, we can represent it by several groom objects each of which has only one element recording the 2D area for one floor (a rectangle) accompanied by the height value.

With the data type above, the location in *indoor* environment is represented as follows. Let IO_{indoor} denote an *infrastructure object* and $u_{indoor} = (i, oid, i_{loc_1}, i_{loc_2}, m)$ be a temporal unit for *indoor* moving objects. Applying the definition of *generic temporal units* in Definition 2.12, the corresponding attributes in u_{indoor} map to:

(1) $oid \rightarrow IO_{indoor}(oid, \textit{GROOM}, \beta, name)(\beta \in D_{\underline{groom}})$;

(2) $i_{loc_1} \rightarrow (x, y)$;

(3) $i_{loc_2} \rightarrow (x, y)$;

(4) $m \rightarrow Indoor$.

The coordinate (x, y) denotes the relative position in the 2D area IO_{indoor} projects to. If the groom object has several elements, the origin point is set as the left lower point of the overall bounding box. The height of a moving object is determined by which *groom* the object is located in. To clarify terms, when we speak of a *groom*, it is a short description for a general room which can be an office room, a chamber, etc., while $\underline{groom}$ denotes the data type representing a *groom*. According to Definition 2.10, the elements of a trajectory projecting to this infrastructure are specified as:

(1) $sub\_traj_i.oid \rightarrow IO_{indoor}(oid, \textit{GROOM}, \beta, name)(\beta \in D_{\underline{groom}})$;

(2) $sub\_traj_i.l$ records the movement inside $IO_{indoor}.\beta$;

(3) $sub\_traj_i.m \rightarrow Indoor$.

door: Each door occupies a position in space and builds the connectivity between two $grooms$. As a door is shared by two $grooms$, the location of the door is represented by two objects where each denotes the relative position in the $groom$ it belongs to. Note that the absolute position in space for them is the same. For each object, let $\underline{doorpos}$ be the data type representing it, defined as follows.

Definition 3.10 *Geo-location for Door*

$Doorpos = \{(gr\_id, pos) | gr\_id \in D_{\underline{int}}, pos \in D_{\underline{line}}\}$.

gr_id is a *groom* identifier and pos records the position of the door in that groom. In the real case, besides the location information the door has another attribute describing whether the connectivity is available, i.e., *open* or *closed*. Sometimes the door is *open* so that people can go through it, but sometimes it is *closed* (Of course you can enter if you have the key. But in this paper we assume people have the general authority, in a visitor's context). To model the time-dependent state, we let $\underline{mbool}$ (employed from [13]) describe the temporal attribute, the value of which is represented as a sequence of units where each has two values: a temporal interval and a $\underline{bool}$ value. Each unit describes whether during the time interval the state is *open* (*true*) or *closed* (*false*). We give the data type representing doors in the following.

Definition 3.11 *Data Type for Door*

$D_{\underline{door}} = \{(dp_1, dp_2, tp, genus) | dp_1, dp_2 \in Doorpos, tp \in D_{\underline{mbool}},$
$genus \in \{nonlift, lift\}\}$.

Each door has dp_1, dp_2 representing the relative position in the two $grooms$ it is connecting, a temporal type tp denoting the time-dependent state (*open* or *closed*) and $genus$ describing whether the door is for an elevator or a normal office room door. We distinguish between

lift doors and non-lift doors because the time-dependent state for non-lift doors can be known, while the state for lift doors is unknown. Usually, the state for an office room door or a chamber door has a regular pattern, e.g., it is open from 8:00 am to 6:00 pm and closed at the other time. But for an elevator, the door (entrance/exit) of which at each level can be considered as the connection between different floors, it is unknown at which floor the lift currently stays so the state for an lift door is uncertain. Hence for a lift door, tp is set to undefined.

3.2.2 Navigation

We define a graph on *grooms* and *doors* for indoor trip plannings as follows:

Definition 3.12 *Indoor Graph*
An indoor graph is defined as $G_{Indoor}(N, E, \sum_{groom}, \sum_{door}, l_v, l_e, W)$ *where:*

1. N *is a set of nodes;*
2. $E \subseteq V \times V$ *is a set of edges;*
3. $l_v : N \rightarrow \sum_{door}$ *is a function assigning labels to nodes;*
4. $l_e : E \rightarrow \sum_{groom}$ *is a function assigning labels to edges;*
5. *each edge is associated with a weight value from W which denotes the shortest distance for connecting two endpoints.*

We also introduce a data type $\underline{indoorgraph}$ whose values are indoor graphs as defined in Def. 3.12.

Each node represents a door which records the position in two *grooms* it belongs to. Each edge corresponds to a general room and builds the connection between two nodes (doors) without passing through a third door. It says that one door is reachable from another via a *groom*. One *groom* can have several edges in the graph if it has more than two doors, because each pair of doors indicates a connection. For example, the elevator is a *groom* consisting of several 3D regions, each of which is the cube between two floors. The entrance/exit on each floor is represented as a node and the connection between two floors is represented as an edge. Each edge is associated with the shortest path between two doors inside the *groom* and the weight is the length of the path. The weight may be not the Euclidean distance as there can be obstacles inside rooms. So, it includes both Euclidean distance and obstructed distance [56, 55]. To build the graph, one needs to precompute the distance between two doors inside a groom. Usually, one room does not have too many doors so the cost is not high.

Take into consideration a simple example in Figure 5(a) which has four rooms $\{gr_1, gr_2, gr_3, gr_4\}$ and four doors $\{d_1, d_2, d_3, d_4\}$ where gr_1, gr_2, gr_3 can be considered as office rooms and gr_4 is a hallway. Objects d_1, d_2, d_3 are the corresponding doors for gr_1, gr_2, gr_3, and d_4 is the entrance/exit for the hallway. The connectivity graph is depicted in Figure 5(b). Each node corresponds to a door which connects two grooms (e.g., $d_1 \rightarrow gr_1, gr_4$) except d_4. And each edge denotes a room connecting two doors (for simplicity, we omit the shortest path stored in the edge). Our model can support searching for three types of optimal routes: (1) shortest distance; (2) smallest number of rooms; (3) minimum traveling time.

shortest distance: we can directly apply Dijkstra's algorithm on G_{Indoor} to find a route with the shortest distance. Note that a preprocessing step is required before applying the normal shortest path algorithm as the input data is an indoor location which could be located in an office room, corridor or even on the staircase. So, the algorithm first has to connect the start and end locations to all doors that their corresponding rooms have, where the doors correspond to the

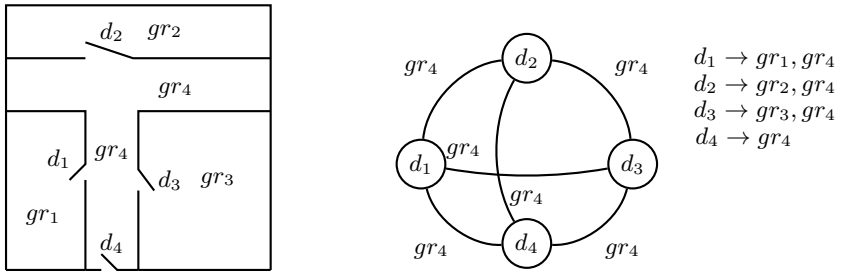

Figure 5: Floor Plan and Connectivity Graph

nodes in the graph. For the termination condition, as each node (door) has two positions where each maps to a room, the process stops when one of the rooms equals to the destination. To improve the searching procedure, the A^* algorithm can be applied if the start and end locations are at the same level where the Euclidean distance between them is set as the heuristic value. As the position of a door is represented as a 2D line in a groom, we take the middle point to compute the distance. Given two points p, q in 3D space, let $p(i)$, $q(i)$ denote the value in dimension i (i=$\{1, 2, 3\}$). The distance between p and q is calculated by

$$dist(p, q) = (\Sigma_{i=1}^{3}(p(i) - q(i))^2)^{1/2} \quad (1)$$

If the start and end locations are at different levels, the above heuristic value might not be efficient (still correct) because the Euclidean distance does not involve any information about the staircases or elevators which are critical for the movement between different floors. Another optimization technique could be used. For the start and end locations, the height above ground level can be retrieved. Then, during searching on the graph all doors whose height values are out of the range can be pruned.

smallest number of rooms: This case is simple. The total cost is achieved by aggregating the number of edges in the path because each edge corresponds to a general room.

minimum traveling time: Let v_{walk} be the average speed of a pedestrian walking in *indoor*. As for each edge we already have the shortest path connecting two doors inside a *groom*, the time for passing an edge can be obtained by v_{walk} and the length of the path. This can apply for passing *grooms* like office rooms, chambers, corridors. But it is not available for the time spent on an elevator which contains two parts: (1) time for waiting until the lift door is *open*; (2) time on moving in the elevator. The time for the first part is uncertain because it is unknown at which floor the elevator is located for a given time instant. The second part can be easily acquired if the height between two floors and the elevator speed is given. We process the two parts as follows. The elevator speed can be defined which is a constant value and d_i^{lift}, d_j^{lift} ($\in D_{\underline{door}}$) be two entries for a lift on the floor i and j. The whole time spent (including waiting) on moving from d_i^{lift} to d_j^{lift} depends on the time arriving at d_i^{lift} and the path where the lift moves along to reach d_j^{lift}. Without loss of generality, we assume $0 \leq i < j$. In the best case, the lift happens to stay at floor i and it directly moves up to floor j. On the contrary, when it arrives at d_i^{lift} the lift just leaves from floor i and moves up so that it has to wait until it goes down to the bottom floor and comes up again. Let h be the height between two floors and n be the number of floors, then the length of the shortest and longest path between floor i and j is $(j-i)*h$ and $(j-i)*h+2*n*h$, respectively. The two values are the lower and upper bounds and the set of all possible values is $\{(j-i)*h, (j-i)*h+h, ..., (j-i)*h+2*n*h\}$.

Each value is associated with a membership probability depending on the elevator schedule. We do not focus on elevator schedule algorithm but modeling the time cost. A possible solution would be the uniform distribution for the probability of each path length. With the length of the path and the elevator speed, the time cost on an elevator can be obtained.

Some indoor graph models [31, 24] represent a *groom* as a node and a door as an edge. But in our model, we do it the other way round. For the edge which corresponds to a *groom*, it explicitly defines the shortest path connecting two doors inside the *groom*. It has some advantages due to the following reasons:

1. To get the shortest path defined on each edge, the cost is small as the calculation is limited to a *groom*, and normally a straight line connecting two doors can suffice.

2. It is meaningful to express how people follow the route. Explicitly showing the route line for each room is more helpful than just describing which rooms you should pass through, especially when the room is very large and has complex structures, e.g., an airport hall. If the room is simply abstracted as a node, the route inside from one door to another can not be well described. Also the weight value assigned to the door (edge) can not be defined. Additionally, there can be some obstacles inside a groom so that the straight line does not work. In this case, some part of the shortest path may be not visible from the position where the traveler is currently located. This motivates us to describe the route explicitly for better guiding.

3. We can express three kinds of optimal routes but other models cannot express all of them. This is because if the node represents a room and the edge denotes a door, then one can only represent a shortest path as a sequence of rooms but not the precise geometry of the path.

The *Doors Graph* [55] is similar to our graph, but there are still some differences. First, they do not take into account the temporal state for doors: $open$ or $closed$. That is the connectivity in *Doors Graph* is constant, while the state of nodes (doors) in our graph is time-dependent so that we can model when offfice doors are open or locked (e.g. according to some regular schedule). Second, the *Doors Graph* does not define the data types representing doors and rooms. We believe the geometric representation for rooms is important because one has to explicitly describe the route from one room to another for navigation. The minimal indoor walking distance in [55] can be applied because the shortest distance between two doors is also computed and assigned to the edge in our model. Third, the concept of doors in our model is general which includes the entrances/exits for elevators and entrances for staircases, but *Doors Graph* is only for normal office doors and hallways.

3.3 Region-based Outdoor

For this environment, we employ the method in [36, 37] that the space is partitioned into a set of disjoint zones where each is represented by a region. The location of a moving object is represented by *referencing* to these zones. The partition algorithm is beyond the scope of the present paper. [36, 37] only represents the location in an approximate way, that is it is known which zone the object is located in, but the precise location inside is not represented. Here, we give the representation for precise location. It is known which zone the moving object is located in as well as the relative position inside. As there are already infrastructures for public transportation modes (Section 3.1) and modes like Car and $Taxi$ (introduced in the next subsection) where all of them are for $outdoor$ movements, the infrastructure here

only serves for the mode $Walk$. Let S be the overall outdoor space for walking and it is partitioned into a set of zones $\{S_1, S_2, ..., S_n\}(n \in D_{\underline{int}})$ where (1) $\forall i \in [1, n], S_i \neq \oslash$; (2) $\forall i, j \in [1, n], i \neq j \Rightarrow S_i \cap S_j = \oslash$; (3) $\bigcup S_i = S$.

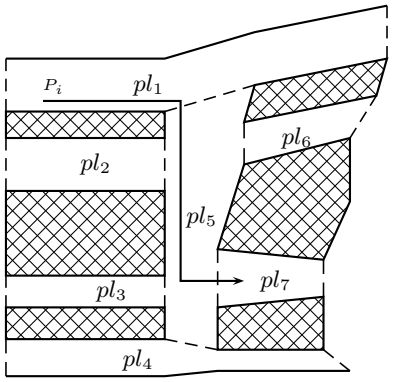

Figure 6: Outdoor Space Partition

Each zone represents an area in space for people walking, i.e., the pavement or footpath. Figure 6 shows an example of a partition on the city map where the polygons drawn by crosshatching represent the buildings and the others represent the area that people can walk in. We only consider the outdoor movement, thus the space for walking (ignoring the building areas) consists of seven disjoint polygons $\{pl_1, ..., pl_7\}$. The shape of a polygon can be regular, e.g., pl_2, pl_3, or irregular, pl_4, pl_6.

The location is represented in two steps: first, we map the location to a polygon; second, the relative position in the polygon is identified. We use IO_{rbo} to represent an *infrastructure object* in I_{rbo} and let $u_{rbo} = (i, oid, i_{loc_1}, i_{loc_2}, m)$ denote a temporal unit for moving objects with the transportation mode $Walk$ where

(1) $oid \rightarrow IO_{rbo}(oid, \textit{REGION}, \beta, name)(\beta \in D_{\underline{region}})$;

(2) $i_{loc_1} \rightarrow (x, y)$;

(3) $i_{loc_2} \rightarrow (x, y)$;

(4) $m \rightarrow Walk$.

During time interval i, the position between i_{loc_1} and i_{loc_2} is obtained by *linear interpolation*. Using Definition 2.14, we can also represent the location in a low resolution way. That is, the location only goes to the level of a polygon identification, while the accurate location inside is ignored ($i_{loc_1} \rightarrow (\perp, \perp), i_{loc_2} \rightarrow (\perp, \perp)$). Consider an example movement of a pedestrian in Figure 6, denoted by $traj_1$. With the low resolution, we can represent it by $P_i =< (pl_1, Walk), (pl_5, Walk), (pl_7, Walk) >$ (for brevity, we ignore the time interval i and location description i_{loc_1}, i_{loc_2}). One application for the low resolution representation is mobility pattern query [36].

3.4 Road Network

For this part, we directly transfer the method in [17] where the street, highway and road is described by a $\underline{line}$ with an unique identifier. As objects move on these $\underline{line}$ objects, the location is represented by a two-tuple $(rid, pos)(rid \in D_{\underline{int}}, pos \in D_{\underline{real}})$ where rid is the route id and pos denotes the relative position in that route. More details can be found by referring to that paper. Applying the framework of *generic temporal units*, let IO_{rn} denote a

street or highway and $u_{rn} = (i, oid, i_{loc_1}, i_{loc_2}, m)$ be a temporal unit for moving objects in road network. Specifically,

(1) $oid \rightarrow IO_{rn}(oid, \textit{LINE}, \beta, name)(\beta \in D_{\underline{line}})$;
(2) $i_{loc_1} \rightarrow (pos, \perp)$;
(2) $i_{loc_2} \rightarrow (pos, \perp)$;
(4) $m \rightarrow \{Car, Taxi, Bicycle\}$.

To identify a position along a $line$, only one dimensional data is needed. Thus, for the second attribute δ_2 in i_{loc_1} and i_{loc_2}, we let it be undefined. The positions between $i_{loc_1}.pos$ and $i_{loc_2}.pos$ are achieved by *linear interpolation*.

3.5 Free Space

In this infrastructure, there is no element inside. That is, the location is directly represented by the precise position in space (geographical coordinates) where no *infrastructure object* is *referenced* to. Using the location representation of [13, 19], let $u_{fs} = (i, oid, i_{loc_1}, i_{loc_2}, m)$ be a temporal unit for moving objects in free space and the location between i_{loc_1} and i_{loc_2} is obtained by *linear interpolation*. Specifically,

(1) $oid \rightarrow \perp$;
(2) $i_{loc_1} \rightarrow (x, y)$;
(3) $i_{loc_2} \rightarrow (x, y)$;
(4) $m \rightarrow Free$.

4 Operations

In this section, we introduce the operators defined on the proposed data types. It would not make sense to start from scratch here. A well-defined type system and systematically designed framework of operations is available from earlier work in [19] which was later extended for network-constrained movement in [17]. These designs consider aspects such as systematic construction of the type system, definition of generic operations ranging over large collections of data types, and consistency between non-temporal and temporal (i.e. time dependent) operations.

We carefully define the new generic model to be consistent with these earlier designs. In fact, the previous models for free space and network-constrained movement are integrated as specific infrastructures into the new model. As a result, the expressive power of the already available generic operations is available here, as will be demonstrated in Section 6 by a comprehensive set of example queries ranging over different infrastructures and transportation modes.

The now model is obtained by extending the earlier type system with the new types arising from the infrastructures developed in the previous sections. We keep the names of the standard operators in [19], but the signatures and semantics are changed as the input data is equipped with novel knowledge.

Section 4.1 introduces the type system in the general model. Operations on non-temporal and temporal generic data types are discussed in Sections 4.2 and 4.3, respectively. Section 4.4 discusses decomposition of multi-component objects. Section 4.5 considers operators on transportation modes and *infrastructure objects*. Section 4.6 defines subtype relationships and the mapping from different infrastructures into free space.

Some notations are used to define operator semantics. u, v are single values of a data type, and correspondingly U, V are generic sets of values of a type. By default, u or U refers to the first argument, and v or V refers to the second argument in an operator. We focus on operator

semantics for data types in the generic model where those on data types such as $\underline{point}$, $\underline{line}$, $\underline{region}$ are previously defined in [19].

We give the *full* space by the set of all possible infrastructure objects and let w be one element of $Space$. As the location is represented by referencing to the underlying space and the operators require the information from the space, by default the space is available to operators (i.e. is not needed as an explicit argument).

$$Space = \{(oid, s, \beta, name) | oid \in D_{\underline{int}}, s \in IOSymbol, \beta \in D_{\mu(s)}, name \in string\}$$

We extend the definition of function **geodata** which will be used for defining semantics. The function is to get the global position in space for the input arguments, where the possible domains and ranges are listed in Table 2. The first three are already introduced in Section 3.1. Given a $\underline{gline}$ and a value describing the relative position along that line, **geodata** returns the point of that position. For a region and a point (line) where the coordinates of the point (line) are represented by referencing to that region, the global location in space is returned. Similarly, we can get the global curve in space for a subline bus route. The last one maps the indoor location to free space. This is done by ignoring the height information, i.e., projecting the room to the ground floor of the building.

For $\alpha \in \{\underline{point}, \underline{line}\}$:	
geodata	$\underline{busroute} \rightarrow \underline{line}$
	$\underline{busroute} \times \underline{busstop} \rightarrow \underline{point}$
	$\underline{busroute} \times \underline{busloc} \rightarrow \underline{point}$
	$\underline{gline} \times \underline{real} \rightarrow \underline{point}$
	$\underline{region} \times \alpha \rightarrow \alpha$
	$\underline{busroute} \times \underline{line} \rightarrow \underline{line}$
	$\underline{groom} \times \alpha \rightarrow \alpha$

Table 2: Domains and Ranges for **geodata**

4.1 Type System

In general, the type system is specified as a signature which consists of *sorts* and *operations*. The *sorts* are *kinds* and represent sets of data types, and *operations* are *type constructors*. Type constructors may take arguments or not, in the latter case they are *constant types*. The terms generated by the signature describe the types available in the type system. For more background refer to [15].

	$\rightarrow BASE$	$\underline{int}$, $\underline{real}$, $\underline{string}$, $\underline{bool}$
	$\rightarrow SPATIAL$	$\underline{point}$, $\underline{points}$, $\underline{line}$, $\underline{region}$
	$\rightarrow TIME$	$\underline{instant}$
$BASE \cup SPATIAL$	$\rightarrow TEMPORAL$	$\underline{moving}$, $\underline{intime}$
$BASE \cup TIME$	$\rightarrow RANGE$	$\underline{range}$

Table 3: Type System in [19]

Tables 3 and 4 provide the type system for moving objects in free space and road network, respectively. We summarize the data types proposed in this paper in Table 5 where they are

	$\rightarrow BASE$	$\underline{int}$, $\underline{real}$, $\underline{string}$, $\underline{bool}$
	$\rightarrow SPATIAL$	$\underline{point}$, $\underline{points}$, $\underline{line}$, $\underline{region}$
	$\rightarrow GRAPH$	$\underline{gpoint}$, $\underline{gline}$
	$\rightarrow TIME$	$\underline{instant}$
$BASE \cup SPATIAL \cup GRAPH$	$\rightarrow TEMPORAL$	$\underline{moving}$, $\underline{intime}$
$BASE \cup TIME$	$\rightarrow RANGE$	$\underline{range}$

Table 4: Type System in [17]

classified into three groups. The first group contains data types relevant for all infrastructures. The second and third groups are specific data types for infrastructures I_{ptn} and I_{indoor}. The type system for the general model is given in Table 6. We compare the type system in the general model with [19, 17] as follows.

- Data types for free space and road network are still available for their own infrastructures. Two groups of new data types *PTN (Public Transportation Network)* and *INDOOR* are added for specific infrastructures.
- The generic types for location and sets of locations $\underline{genloc}$ and $\underline{genrange}$ are available in all infrastructures. To represent a location we can apply $\underline{genloc}$ in all cases while [19] used $\underline{point}$ in free space and [17] introduced $\underline{gpoint}$ for road network. Similarly, to represent a set of possible locations we use $\underline{genrange}$ while [19] used $\underline{points}$, $\underline{line}$, or $\underline{region}$ and [17] added $\underline{gline}$. With $\underline{genloc}$ and $\underline{genrange}$, we can cover the locations in all cases instead of defining different data types according to each specific environment.
- For the moving types except those *lifting* from *BASE*, e.g., $\underline{int}$, $\underline{real}$, $\underline{genloc}$ applies in all cases. That is the type $\underline{genmo}$ = ($\underline{moving}$($\underline{genloc}$)) applies for moving objects in all infrastructures no matter whether the environment is free space, road network, or indoor. The type $\underline{mpoint}$ is used for free space [19] and $\underline{mgpoint}$ is for road network [17]. $\underline{mpptn}$ is proposed for moving buses, but it is also covered by $\underline{genmo}$.

	Name	**Meaning**
generic data types	$\underline{tm}$	transportation modes
	$\underline{genloc}$	generic locations
	$\underline{genrange}$	generic sets of locations
	$\underline{genmo}$	generic moving objects
public transportation network	$\underline{busstop}$	bus stops
	$\underline{busroute}$	bus routes
	$\underline{busloc}$	locations on bus routes
	$\underline{mpptn}$	moving buses
indoor	$\underline{groom}$	general rooms
	$\underline{door}$	doors

Table 5: A Summary of Proposed Data Types

	$\rightarrow BASE$	$\underline{int}$, $\underline{real}$, $\underline{string}$, $\underline{bool}$
	$\rightarrow TIME$	$\underline{instant}$
$BASE \cup TIME$	$\rightarrow RANGE$	$\underline{range}$
	$\rightarrow SPATIAL$	$\underline{point}$, $\underline{points}$, $\underline{line}$, $\underline{region}$
	$\rightarrow GRAPH$	$\underline{gpoint}$, $\underline{gline}$
	$\rightarrow PTN$	$\underline{busstop}$, $\underline{busroute}$, $\underline{busloc}$
	$\rightarrow INDOOR$	$\underline{groom}$, $\underline{door}$
	$\rightarrow GENLOC$	$\underline{genloc}$
	$\rightarrow SPACE$	$\underline{genrange}$, $\underline{space}$
	$\rightarrow TM$	$\underline{tm}$
$BASE \cup SPATIAL \cup GRAPH \cup GENLOC$	$\rightarrow TEMPORAL$	$\underline{moving}$, $\underline{intime}$
	$\rightarrow TEMPORAL$	$\underline{mpptn}$

Table 6: Type System in General Model

4.2 Operations on Non-Temporal Types

The non-temporal operators applicable to generic data types are collected in Table 7. For defining semantics, let $IOSymbol' = \{LINE, REGION, BUSROUTE, GROOM\}$ denote the set of symbols for all static infrastructure objects, which are to be called by the function **geodata**. We believe the meaning for the predicates should be obvious and give some brief comments. For predicates **intersects, inside**, the processing is more efficient if the result is *false*, because with our method one can first check two objects' identifiers. If they are not equal, it can terminate and return *false*. The previous methods may involve costly geometry computation. As a $\underline{genrange}$ object is a set object, **card** returns the number of elements, e.g., how many streets are covered or how many rooms are included. The operator **length** gets the geometry line length in space. For the operator **distance** between two generic locations, the meaning of it should be clear if the two input arguments belong to the same infrastructure, e.g., Euclidean distance for free space and network distance for road network. If the two arguments belong to different infrastructures, we define the distance by the minimum length of a trip connecting them. Section 4.5 introduces the operator to get the trip for two locations. Operator **ref_id** returns the referenced object identifier. For a bus route, all bus segments have the same route id.

There are three cases in Def. 4.1. Cases (i) and (ii) are clear as both locations are in free space or in the same infrastructure object. In case (iii) and (iv), if one location is in free space and the other belongs to another infrastructure, one can compare them by loading the referenced infrastructure object.

Definition 4.1 $\underline{genloc} \times \underline{genloc} \rightarrow \underline{bool}$ =
Let u, v be the two locations and the result is TRUE iff
(i) $u.oid = \perp \wedge v.oid = \perp \wedge u.i_{loc} = v.i_{loc}$*; or*
(ii) $u.oid = v.oid \wedge u.i_{loc} = v.i_{loc}$*; or*
(iii) $u.oid = \perp \wedge$
$(\exists w \in Space : w.oid = v.oid \wedge w.s \in IOSymbol' \wedge u.i_{loc} =$ ***geodata***$(w.\beta, v.i_{loc}))$*; or*
(iv) $v.oid = \perp \wedge$
$(\exists w \in Space : w.oid = u.oid \wedge w.s \in IOSymbol' \wedge v.i_{loc} =$ ***geodata***$(w.\beta, u.i_{loc}))$

Name	Signature	Semantic
isempty	$\underline{genloc} \rightarrow \underline{bool}$	$u = \perp$
	$\underline{genrange} \rightarrow \underline{bool}$	$U = \emptyset$
$=, \neq$	$\underline{tm} \times \underline{tm} \rightarrow \underline{bool}$	$u = v, u \neq v$
	$\underline{genloc} \times \underline{genloc} \rightarrow \underline{bool}$	see Def. 4.1
inside	$\underline{genloc} \times \underline{genrange} \rightarrow \underline{bool}$	see Def. 4.2
intersects	$\underline{genloc} \times \underline{genrange} \rightarrow \underline{bool}$	the same as Def. 4.2
	$\underline{genrange} \times \underline{genrange} \rightarrow \underline{bool}$	see Def.4.3
card	$\underline{genrange} \rightarrow \underline{int}$	$\mid U \mid$
length	$\underline{genrange} \rightarrow \underline{real}$	see Def. 4.4
distance	$\underline{genloc} \times \underline{genloc} \rightarrow \underline{real}$	see Def. 4.5
ref_id	$\underline{genloc} \rightarrow \underline{int}$	$u.oid$
	$\underline{genrange} \rightarrow \underline{set}(\underline{int})$	$\{u.oid \mid u \in U\}$
	$\underline{busroute} \rightarrow \underline{int}$	$u_1.bs_1.rid$

Table 7: Non-Temporal Generic Operators

In Def. 4.2, case (i) represents a single location and a set of locations in free space. If u, v reference to the same object w (case (ii)), one can directly compare the location inside w or load the referenced object to retrieve the location in the global space. Similarly, if either u or v corresponds to free space and the other does not, the referenced object is loaded to determine the location in space. These are cases (iii) and (iv).

Definition 4.2 $\underline{genloc} \times \underline{genrange} \rightarrow \underline{bool}$ ***inside***

Let u be the single location and V be the set of locations. The result is TRUE iff $\exists v \in V$ such that

(i) $u.oid = \perp \wedge v.oid = \perp \wedge u.i_{loc} \in v.l$*; or*

(ii) $u.oid = v.oid \wedge (u.i_{loc} \in v.l \vee$
$(\exists w \in Space : v.oid = w.oid \wedge$ ***geodata***$(w.\beta, u.i_{loc}) \in$ ***geodata***$(w.\beta, v.l)))$*; or*

(iii) $u.oid = \perp \wedge$
$(\exists w \in Space : v.oid = w.oid \wedge w.s \in IOSymbol' \wedge u.i_{loc} \in$ ***geodata***$(w.\beta, v.l))$*; or*

(iv) $v.oid = \perp \wedge$
$(\exists w \in Space : u.oid = w.oid \wedge w.s \in IOSymbol' \wedge$ ***geodata***$(w.\beta, u.i_{loc}) \in v.l)$

Definition 4.3 $\underline{genrange} \times \underline{genrange} \rightarrow \underline{bool}$ ***intersects***

The result is TRUE iff $\exists u \in U, \exists v \in V$ such that

(i) $u.oid = \perp \wedge v.oid = \perp \wedge u.l$ ***intersects*** $v.l$ *; or*

(ii) $u.oid = v.oid \wedge u.l$ ***intersects*** $v.l$ *; or*

(iii) $u.oid = \perp \wedge$
$(\exists w \in Space : v.oid = w.oid \wedge w.s \in IOSymbol' \wedge$
$u.l$ ***intersects geodata***$(w.\beta, v.l))$*; or*

(iv) $v.oid = \perp \wedge$
$(\exists w \in Space : u.oid = w.oid \wedge w.s \in IOSymbol' \wedge$
$v.l$ ***intersects geodata***$(w.\beta, u.l))$

Definition 4.4 $\underline{genrange} \rightarrow \underline{real}$ ***length***

if $\forall u_i \in U, u_i.l \in D_{\underline{line}}$ *then* $\sum_{i=0}^{|U|} \parallel u_i.l \parallel$

else undefined

Definition 4.5 $\underline{genloc} \times \underline{genloc} \rightarrow \underline{real}$ ***distance***

length(***trajectory***(***trip***(u, v))) *(operators* ***trajectory*** *and* ***trip*** *are defined in Sections 4.3 and 4.5, respectively)*

4.3 Operations on Temporal Types

Table 8 lists operators on temporal generic data types. Because we represent the location data in a multiresolution way where both imprecise and precise data are managed, all the operators defined in [19, 17] can still be applied, e.g., **passes**, **present**, but with the modification that in the signature $\underline{mpoint}$ and $\underline{mgpoint}$ are replaced by $\underline{genmo}$. **Trajectory** projects a moving object into the infrastructure space and **deftime** yields the time intervals when the object is defined. The type $\underline{periods}$ is introduced in Section 2.1 which is a set of time intervals. **duration** returns the time span of a period and we assume the result unit is given in *minutes*.

Present checks whether the object is defined at the given time parameter. **Initial** and **final** yields the first and last value. The object can be restricted by a time instant or a set of time intervals by **atinstant** and **atperiods**. **val** returns $\underline{genloc}$ from an $\underline{intime}(\underline{genloc})$ object and **inst** returns the time.

At restricts the object within the given space. The space can be a point in space, then the movement is only temporal, e.g., t_{move} for buses introduced in Section 3.1.2. One can also restrict the movement to a generic region given in a $\underline{genrange}$ argument. A formal definition of the semantics in this case is a bit lengthy and omitted.

Operator **passes** checks whether the moving object passes a place or not. The place could be a single point or an infrastructure object. For example, "Did *Bobby* visit the Sparkasse Bank in city center?" or "Did *Bobby* pass *Alexander* street?".

Section 2.4 introduces the method of representing moving objects' location in an imprecise way, so we can get the moving object with coarse location representation by **lowres**. And then we can apply operator **trajectory** to get the rough trajectory.

Let mo be a generic moving object and $u_i(\in mo)$ denote a temporal unit. We use notation $f(t)$ (Def. 2.9) to denote the generic location for a moving object at time t. The result has different meanings according to the infrastructure object the location maps to. For the sake of clarity, we use subscripts to denote each kind of location mapping, listed in Table 9.

In Def. 4.6, the trajectory of a moving object is a set of movement projections on infrastructure objects, each of which is taken from $u_i \in mo$. The meaning for (i) and (ii) should be clear. In case (iii), the value l' is the global line in space and needs to be converted according to the infrastructure object. For case (iv), the location in u_i maps to a bus. Then, the trajectory is the bus movement, which goes to case (iii).

Definition 4.6 $\underline{genmo} \rightarrow \underline{genrange}$ ***trajectory***

The result is a set of (oid, l, m) *where* $m = u_i.m$*. In the following, we define the values for* oid *and* l*. Let function* $\gamma(l_1, l_2)(l_1, l_2 \in D_{\underline{line}} \wedge l_1 \subseteq l_2)$ *return the relative location of* l_1 *according to* l_2*.*

(i) $u_i.oid = \perp \Rightarrow oid = \perp \wedge l = \cup f_{\perp}(t)(t \in u_i.i)$*; or*

(ii) $\exists w \in Space : u.oid = w.oid \wedge (w.s = REGION \vee w.s = GROOM)$*, then* $oid = u.oid, l = \cup f_r(t).i_{loc}(t \in u_i.i)$*; or*

(iii) $\exists w \in Space : u.oid = w.oid \wedge (w.s = LINE \vee w.s = BUSROUTE)$*, then* $oid = u.oid, l = \gamma(l', w.\beta), l' = \cup$ ***geodata***$(w.\beta, f_{l(br)}(t).i_{loc})(t \in u_i.i)$*; or*

(iv) $\exists w_1, w_2 \in Space : u.oid = w_1.oid \wedge w_1.s = MPPTN \wedge$ ***ref_id***$(w_1) = w_2.oid$*, then* $oid = w_2.oid, l =$ ***trajectory(atperiods***$(w_1, u_i.i))$

Name	Signature	Semantic
ref_id	$\underline{mpptn} \rightarrow \underline{int}$	$u_i.bs_1.rid$
	$\underline{genmo} \rightarrow set(\underline{int})$	$\{u_i.oid \mid u_i \in mo\}$
trajectory	$\underline{genmo} \rightarrow \underline{genrange}$	see Def. 4.6
deftime	$\underline{genmo} \rightarrow \underline{periods}$	
duration	$\underline{periods} \rightarrow \underline{real}$	
present	$\underline{genmo} \times \underline{intime} \rightarrow \underline{bool}$	
	$\underline{genmo} \times \underline{periods} \rightarrow \underline{bool}$	
initial, final	$\underline{genmo} \rightarrow \underline{intime}(\underline{genloc})$	
atinstant	$\underline{genmo} \times \underline{instant} \rightarrow \underline{intime}(\underline{genloc})$	see [19]
atperiods	$\underline{genmo} \times \underline{periods} \rightarrow \underline{genmo}$	
val	$\underline{intime}(\underline{genloc}) \rightarrow \underline{genloc}$	
inst	$\underline{intime}(\underline{genloc}) \rightarrow \underline{instant}$	
at	$\underline{genmo} \times \underline{genloc} \rightarrow \underline{genmo}$	see Def. 4.7
	$\underline{genmo} \times \underline{genrange} \rightarrow \underline{genmo}$	omitted
passes	$\underline{genmo} \times \underline{genloc} \rightarrow \underline{bool}$	$\mathbf{at}(mo, v) \neq \emptyset$
	$\underline{genmo} \times \underline{genrange} \rightarrow \underline{bool}$	$\mathbf{at}(mo, v) \neq \emptyset$
lowres	$\underline{genmo} \rightarrow \underline{genmo}$	$< u_1, u_2, ..., u_n >$ *where* $u_i.i_{loc_1} = \perp \wedge u_i.i_{loc_2} = \perp$

Table 8: Temporal Generic Operators

Location Mapping	Symbols for Referenced Infrastructure Objects
$f_\perp(t) = (\perp, (\delta_1, \delta_2))$	$\perp$
$f_r(t) = (oid, (\delta_1, \delta_2))$	$REGION, GROOM$
$f_l(t) = (oid, (\delta_1, \perp))$	$LINE$
$f_{br}(t) = (oid, (\delta_1, \delta_2))$	$BUSROUTE$

Table 9: Different Location Mappings

In Def. 4.7, the meaning is clear if the second argument represents a precise location (case (i)). If a rough location is given (case (ii)), i.e., only the object id, then the units in the result could have the same reference id as the input or the location of units map to the places covered by the given infrastructure object.

Definition 4.7 $\underline{genmo} \times \underline{genloc} \rightarrow \underline{genmo}$ ***at***

Let $mo =< u_1, u_2, ..., u_n >$ *be the moving object and* v *denote the location.*

(i) $v.i_{loc} \neq \perp$, *the result is* $< u'_1, ..., u'_k >$ *where* $u'_i \in mo \wedge \forall t \in u'_i.i : \ f(t) = v$; *or*

(ii) $v.oid \neq \perp \wedge v.i_{loc} = \perp$, *then the result is* $< u'_1, ..., u'_k >$ *where* $u'_i \in mo \wedge$

(a) $u'_i.oid = v.oid$; *or*

(b) $u'_i.oid = \perp \wedge$

$(\exists w \in Space$ *such that*

$w.oid = v.oid \wedge \ \forall t \in u'_i.i : \ f_\perp(t) \in w.\beta \wedge w.s \in IOSymbol')$

4.4 Sets and Decomposition

The design of types and operations in [19] emphasized compatibility with a relational model and therefore proposed only "atomic" data types, i.e., types suitable as attribute types in a

relation. Presumably for this reason, in [19] there is no $\underline{set}$ type constructor applicable to atomic types so that for example, a type $\underline{set}(\underline{periods})$ would be available.

However, it was recognized that a tool for decomposing values of a data type into components was needed. For example, for a $\underline{region}$ value consisting of several disjoint components ("faces") it should be possible to obtain each face as an independent $\underline{region}$ value; similarly for a $\underline{moving}(\underline{point})$ one would like to have a decomposition into continuous pieces, each as a separate $\underline{mpoint}$ value. It is obvious that a natural operation to perform such decompositions would have a signature

components: $\alpha \rightarrow \underline{set}(\alpha)$

for any atomic type α consisting of several components. Due to the lack of a $\underline{set}$ constructor, in [19] decomposition was described as a relation operation in a somewhat complicated manner.

In this paper we leave this restriction behind and do introduce a $\underline{set}$ constructor. In the implementation of the model of [19] it has turned out that one does not even need an explicit data structure to represent such sets but can at the executable level of the system handle sets of values as streams of values. Of course, it is also possible to introduce an explicit data structure for sets. Together with the set constructor we define a few generic operations:

contains: $\underline{set}(\alpha) \times \alpha \rightarrow \underline{bool}$
card: $\underline{set}(\alpha) \rightarrow \underline{int}$

Furthermore, the operation **components** is offered for decomposition:

For $\alpha \in \{\underline{range}(\beta), \underline{points}, \underline{line}, \underline{region}, \underline{moving}(\gamma)\}$:
components: $\alpha \rightarrow \underline{set}(\alpha)$

For a formal definition of the semantics of the **components** operator see [19], Section 4.4. Note that type $\underline{periods}$ is just an abbreviation for $\underline{range}(\underline{instant})$, hence the signature

components: $\underline{periods} \rightarrow \underline{set}(\underline{periods})$

is also available.

4.5 Transportation Modes and Infrastructure Objects

Table 10 lists the proposed operators. Given a moving object, we can get its transportation modes. Using *Bobby*'s trajectory M_1 as an example, **get_mode** returns $\{Walk, Car, Indoor\}$. With operators **get_mode** and **contains** (Section 4.4), one can examine whether the moving object includes a specific transportation mode. For example, one can issue "Does *Bobby* use public transportation vehicles during his trip?"

Given a transportation mode, the moving object can be restricted to a sub movement with respect to that mode which is done by **at** (in Section 4.3, **at** restricts the object to a given space and now it is extended). Before introducing the operators for referenced *infrastructure objects*, we first define a reference data type named $\underline{ioref}$.

Definition 4.8 *Reference Data Type*
$D_{\underline{ioref}} = \{(oid, ref) | oid \in D_{\underline{int}}, ref \in IOSymbol\}$

The values of type $\underline{ioref}$ have two components where oid is the referenced object id and ref is the symbol of a data type (Def. 2.5). The data type is a compact representation for *infrastructure objects* which have different data types according to the infrastructures and the value of which may need large storage space, e.g., region. With the reference type, we can

describe the result in a light way. Operator **ref_id** returns the referenced object id. When the underlying information is needed, one can get the full data by operator **ref_obj**. One possibility could be

$(oid,$ *GROOM*$) \rightarrow \underline{groom}$.

Operator **get_ref** gets the *referenced infrastructure objects*. See Query 1 and 2 in Section 6.

Finally, we define the operator **trip** for route planning. It takes two locations and a query instant where the locations are general which can be in any *infrastructure*. The result is described in the form of $\underline{genmo}$. The operator is to answer the query where the trip covers multiple environments, for example, "*find a trip from my office room to my home with minimum traveling time*". The infrastructures covered could be *indoor, road network, region-based outdoor* ($Indoor \rightarrow Car \rightarrow Walk$) or *indoor, region-based outdoor, public transportation network* ($Indoor \rightarrow Walk \rightarrow Bus \rightarrow Walk$). A formal definition of the semantics would be complex and is omitted.

Name	Signature	Semantic
get_mode	$\underline{genmo} \rightarrow \underline{set}(\underline{tm})$	$\{u_i.m \mid u_i \in mo\}$
at	$\underline{genmo} \times \underline{tm} \rightarrow \underline{genmo}$	$< u_1, u_2, ..., u_n >, u_i.m = v$
ref_id	$\underline{ioref} \rightarrow \underline{int}$	$u.oid$
ref_obj	$\underline{ioref} \rightarrow \alpha\ (\alpha \in IOType)$	$\mu(u.ref)$ *where* $\exists w \in Space : u.oid = w.oid$
get_ref	$\underline{genloc} \rightarrow \underline{ioref}$	$(u.oid, w.s)$ *where* $\exists w \in Space : u.oid = w.oid$
	$\underline{genrange} \rightarrow \underline{set}(\underline{ioref})$	$\{(u.oid, w.s) \mid u \in U \wedge (\exists w \in Space : u.oid = w.oid)\}$
	$\underline{genmo} \rightarrow \underline{set}(\underline{ioref})$	$\{(u_i.oid, w.s) \mid u_i \in mo \wedge (\exists w \in Space : u_i.oid = w.oid)\}$
trip	$\underline{genloc} \times \underline{genloc} \times \underline{instant} \rightarrow \underline{genmo}$	omitted

Table 10: Operators on Transportation Modes and Infrastructure Objects

4.6 Subtype Relationships and Conversions Between Generic and Specific Types

Sometimes one needs to apply generic operations (defined for $\underline{genloc}$, $\underline{genrange}$ or $\underline{genmo}$) to objects of more specific types. This is possible through subtype relationships. In Section 3, data types for the different infrastructures have been shown to fit into the generic framework; hence arguments of the specific types can be substituted in operations for the generic types. The valid subtype relationships are shown in Table 11.

Infrastructure	Generic Types		
	$\underline{genloc}$	$\underline{genrange}$	$\underline{genmo}$
I_{ptn}	$\underline{busstop}$, $\underline{busloc}$	$\underline{busroute}$	$\underline{mpptn}$
I_{indoor}		$\underline{groom}$, $\underline{door}$	
I_{rbo}			
I_{rn}	$\underline{gpoint}$	$\underline{gline}$	$\underline{mgpoint}$
I_{fs}	$\underline{point}$	$\underline{points}$, $\underline{line}$, $\underline{region}$	$\underline{mpoint}$

Table 11: Subtype Relationships

Furthermore, sometimes it is necessary to compare locations or moving objects that belong to different infrastructures. For example, consider the problem of determining whether a person riding a bicycle was passed by a public bus (a similar example query is Query 12 in Section 6). To be able to evaluate such relationships between objects from different infrastructures, we might try to provide mappings between all pairs of infrastructures. However, it is simpler to introduce a generic mapping that converts an object from any infrastructure into its free space counterpart. We introduce such an operation called **freespace**. Essentially it maps values of any of the types in Table 11 to the corresponding type in free space, hence offers the signatures shown in Table 12. For the mapping of indoor locations into free space we ignore the height

Operator	Signature
freespace	$\underline{genloc} \rightarrow \underline{point}$
	$\underline{busstop} \rightarrow \underline{point}$
	$\underline{busloc} \rightarrow \underline{point}$
	$\underline{gpoint} \rightarrow \underline{point}$
	$\underline{busroute} \rightarrow \underline{line}$
	$\underline{groom} \rightarrow \underline{region}$
	$\underline{door} \rightarrow \underline{line}$
	$\underline{gline} \rightarrow \underline{line}$
	$\underline{genmo} \rightarrow \underline{mpoint}$
	$\underline{mpptn} \rightarrow \underline{mpoint}$
	$\underline{mgpoint} \rightarrow \underline{mpoint}$
freespace_p	$\underline{genrange} \rightarrow \underline{points}$
freespace_l	$\underline{genrange} \rightarrow \underline{line}$
freespace_r	$\underline{genrange} \rightarrow \underline{region}$
genloc	$\underline{int} \times \underline{real} \times \underline{real} \rightarrow \underline{genloc}$

Table 12: Conversion Operators

information associated with a $\underline{groom}$ and so project into the (x, y) plane (or the ground level of the building). This maps rooms from different floors into the same locations. However, the purpose of the mapping is to relate locations between different infrastructures which is still possible. For example, one can determine whether a person was in a room within a building located in a thunderstorm area given as a $\underline{region}$ value.

For the mapping of $\underline{genrange}$ values into free space we need to introduce three operators **freespace_p**, **freespace_l**, and **freespace_r** returning $\underline{points}$, $\underline{line}$, or $\underline{region}$ values, respectively. This is because $\underline{genrange}$ is a union type. The operators return the parts of the argument that can be mapped into the respective result type. For example, one can obtain the trajectory of a trip as a $\underline{line}$ value.

Finally, for querying a construction operator **genloc** for generic locations is needed that builds a generic location from an infrastructure object identifier and two real "coordinates".

5 A Relation Interface

5.1 Space: A Relational View

We provide an interface to exchange information between values of the proposed data types and a relational environment. To manage moving objects, one first has to supply *infrastructure objects* that moving objects *refer* to. Table 13 shows the available infrastructures each of which

may have several components. For each component, we create a relation where each tuple corresponds to an infrastructure object. The respective relation schemas are shown in Table 14.

Infrastructure	Infrastructure Component
I_{ptn}	BUSSTOP BUSROUTE BUS
I_{indoor}	ROOM DOOR ROOMPATH
I_{rbo}	OUTDOOR
I_{rn}	ROAD
I_{fs}	

Table 13: Infrastructures and Their Components

$rel_{busstop}$	`(BusStopId: int, Stop: busstop, Name: string)`
$rel_{busroute}$	`(BusRouteId: int, Route: busroute, Name: string, Up: bool)`
rel_{bus}	`(BusId: int, Bus: mpptn, Name: string)`
rel_{room}	`(RoomId: int, Room: groom, Name: string)`
rel_{door}	`(DoorId: int, Door: door)`
$rel_{roompath}$	`(RoomPathId: int, Door1: int, Door2: int, Weight: real, Room: groom, Name: string, Path: line)`
rel_{rbo}	`(RegId: int, Reg: region, Name: string)`
rel_{rn}	`(RoadId: int, Road: line, Name: string)`

Table 14: Infrastructure Relations

For each relation, the data type representing *infrastructure objects* is embedded as an attribute. To have a unique identifier for each object, for each kind of infrastructure we define a range of $\underline{int}$ values for the object id such that there is no overlap between the ranges for different infrastructures.

With the relations for all infrastructures, we can construct the space in two steps:

1. create a space with one infrastructure relation by the operator:

   createspace: $\underline{rel} \rightarrow \underline{space}$

2. add more infrastructure relations to the space by the command:

   put_infra: $\underline{space} \times \underline{rel} \rightarrow \underline{space}$

In the first step, operator **createspace** takes in the relevant information for one infrastructure described by a relation. The input relation can be empty. In this case, the space equals to free space. If the input relation stores roads, then the space is road network. In the second step, more infrastructure relations can be added into the space so that the space can cover multiple environments. With the two operators, one can create a full or non-full space according to the application requirement, e.g., road network plus bus network or only indoor.

With the relational interface, we can exchange information between values of the proposed data types and a relational environment. We can get each infrastructure relation information by:

get_infra: $\underline{space} \times \underline{int} \rightarrow \underline{rel}$

The value of the second argument is from the set {BUSSTOP, BUSROUTE, BUS, ROOM, DOOR, ROOMPATH, OUTDOOR, ROAD}, i.e., the names of infrastructure components from Table 13, whose elements are assumed to be available as integer constants. Thus, one can retrieve the required infrastructure objects by operations of a relational environment for each infrastructure.

Suppose we have the relations for the *infrastructure* data of a city called *Gendon*. In this paper, we consider all infrastructures. Applying the two steps above (**createspace** and **put_infra**) we create a full space for *Gendon* and let `SpaceGendon` ($\in D_{\underline{space}}$) denote the result. The following query illustrates how to access infrastructure relations.

"Show me the information of *Alexander* street."

```
SELECT *
FROM get_infra(SpaceGendon, ROAD) as road
WHERE road.Name = "Alexander"
```

Assume we also have some trajectory data of citizens living and working in *Gendon*. We use the object-based approach [12] to store moving objects where the complete history is kept together. The trip is stored as an attribute in a relation named `MOGendon` with the schema:

```
MOGendon(Mo_id: int, Traj: genmo, Name: string)
```

Each tuple has three attributes: Mo_id being the moving object identifier, $Traj$ storing the trajectory data and a string value describing the name. In the next section, we show queries on the model based on the above example relations.

5.2 Graph Model for Indoor Navigation

We use two relations rel_{door} and $rel_{roompath}$ to store graph nodes and edges for a building, respectively. Recall that according to Section 3.2.2 doors correspond to nodes and door connections in rooms to edges of the graph. Their schemas are included in Table 14. We can create an indoor graph by the following operator.

createindoorgraph: $\underline{rel} \times \underline{rel} \rightarrow \underline{indoorgraph}$

Then, we can run shortest path queries using operator

indoornavigation: $\underline{genloc} \times \underline{genloc} \times \underline{instant} \times \underline{int} \times \underline{indoorgraph} \rightarrow \underline{genrange}$

where the first two arguments are the start and end locations, the third is the query issue time and the fourth argument of type $\underline{int}$ denotes the type of the shortest path (e.g., distance, time).

6 Query Examples

In this section, we evaluate our data model by performing a group of queries. We show how to use an SQL-like language to formulate the queries expressed by natural language. We retain the basic syntax and structure of SQL with some extensions. The operators presented in Section 4 can be registered as functions in the DBMS such that they are available for users. The earlier operators in [19, 17] are all available. Before describing the queries, we first recall some notations introduced in [19] to achieve a smooth integration with abstract data types and their operations.

The command `LET <name> = <query>` creates a new database object `name` whose value is given by the query expression. A query can thus be formulated in several steps (separated by `;`), defining intermediate results by `LET`. The last expression determines the result of the query.

`ELEMENT(<query>)` is used to convert a relation with a single tuple having a single attribute to a typed atomic value. For example, given a relation `r(Name: string)` with only one tuple `("Bobby")`, `ELEMENT(r)` returns `"Bobby"` as a $\underline{string}$ value.

Conversely, the notation `SET(<name>, <value>)` constructs a relation with a single tuple and attribute from an atomic value. For example, `SET(Name, "Bobby")` constructs the relation r of the previous paragraph.

In this paper, we extend the notation further to handle also sets of atomic values produced by some operators. An example is the **components** operator with signature $\underline{periods} \rightarrow \underline{set}(\underline{periods})$. Then the notation `SET(Time, components(p))` will produce a relation with schema `(Time: periods)` having one tuple for each distinct time interval in the $\underline{periods}$ value p.

Further, we use some auxiliary operations introduced in [17] to specify instants or time intervals, e.g. `instant(2010, 12, 5, 8)` (8am on December 5, 2010) or `minute(2010, 12, 5, 8, 30)` (the minute beginning at 8:30 on this day).

Using the example data of Section 5.1, the queries we take into consideration are listed and formulated as follows. We classify them into three groups – the classification may be not so strict because some queries cover more than one group.

- Queries on Infrastructures and Infrastructure Objects
- Queries on Transportation Modes
- Interaction between Different Transportation Modes and Infrastructures

The Appendix provides useful information to check the following query formulations in detail. It provides the schemas of the involved infrastructure relations and for each query an index showing where the used operations have been defined.

6.1 Queries on Infrastructures and Infrastructure Objects

- **Query** 1. Where is *Bobby* at 8:00 am?

```
LET qt = instant(2010, 12, 5, 8);

SELECT val(mo.Traj atinstant qt)
FROM MOGendon AS mo
WHERE mo.Name = "Bobby"
```

We give some comments for the query expression. First, after filtering the moving object by name we restrict the movement to the given time instant by operator **atinstant**. Second, we retrieve the value of an $\underline{intime}(\underline{genloc})$ object by **val**. Let loc be the general location from the query expression. If the location references to an infrastructure object, we can use **ref_obj**(**get_ref**(loc)) to get the referenced object. Combining the general location and the referenced object, the global position in space can also be obtained.

- **Query** 2. Find all people taking "Bus527".

```
SELECT mo.Name
FROM MOGendon AS mo, get_infra(SpaceGendon, BUS) AS bus
WHERE bus.Name = "Bus527" AND
  get_ref(mo.Traj at Bus) contains bus.BusId
```

We restrict the movement to the mode Bus and get the referenced objects (represented in a light way) and then we check whether the selected bus is referenced.

- **Query** 3. Find who passed the room r_{312} at the university between 8:00 am and 9:00 am.

```
LET qt = hour(2010, 12, 5, 8);
```

Assume the name of the room is "Uni-312" and first we get the room id by

```
LET room_id = ELEMENT(
  SELECT room.RoomId
  FROM get_infra(SpaceGendon, ROOM) AS room
  WHERE room.Name = "Uni-312");

SELECT mo.Name
FROM MOGendon AS mo
WHERE
  get_ref((mo.Traj atperiods qt) at Indoor) contains room_id
```

To define this query, several operators are needed. First, we restrict moving objects to the given period by **atperiods** and to indoor movement by **at**. Second, we get the referenced infrastructure objects by **get_ref**. The result is represented by the reference type, i.e., as $\underline{set}(\underline{ioref})$. Third, we check whether the room id is included by **contains**.

- **Query** 4. Find all people passing zone A and zone B as well as a location p in space. (An interesting query could be: find all people passing the Christmas Market Area and the city center area as well as the cinema.)

  Assume the names for the zones are "Zone-A" and "Zone-B". Let $p = (\perp, (x, y))(\in D_{\underline{genloc}})$ denote the location. First, we get the object ids for zones A and B.

```
LET A_Id = ELEMENT(
  SELECT outdoor.RegId
  FROM get_infra(SpaceGendon, OUTDOOR) AS outdoor
  WHERE outdoor.Name = "Zone-A");

LET B_Id = ELEMENT(
  SELECT outdoor.RegId
  FROM get_infra(SpaceGendon, OUTDOOR) AS outdoor
  WHERE outdoor.Name = "Zone-B");
```

Second, we create two *genloc* objects for the zones.

```
LET ZoneA = genloc(A_Id, undef, undef);
LET ZoneB = genloc(B_Id, undef, undef);
```

Third, we formulate the final query.

```
SELECT mo.Name
FROM MOGendon AS mo
WHERE mo.Traj passes ZoneA AND mo.Traj passes ZoneB AND
  mo.Traj passes p
```

- **Query** 5. How long does *Bobby* wait for bus 512 at the bus stop "Uni"?

```
SELECT duration(deftime((mo.Traj at Walk) at bs.Stop))
FROM MOGendon AS mo, get_infra(SpaceGendon, BUSSTOP) AS bs
WHERE mo.Name = "Bobby" AND bs.Name = "University"
```

First, we get the sub movement for the mode $Walk$. And then, the movement is restricted to the bus stop. Finally, **deftime** gets the time period at the place and **duration** returns the time span.

6.2 Queries on Transportation Modes

- **Query** 6. At 8:00 am, who sits in the same bus as *Bobby*.

```
LET qt = instant(2010, 12, 5, 8);

SELECT mo1.Name
FROM MOGendon AS mo1, MOGendon AS mo2
WHERE mo2.Name = "Bobby" AND
  val(mo1.Traj atinstant qt) = val(mo2.Traj atinstant qt)
```

- **Query** 7. Find all people using public transportation vehicles.

```
SELECT mo.Name
FROM MOGendon AS mo
WHERE get_mode(mo.Traj) contains Bus OR
  get_mode(mo.Traj) contains Train) OR
  get_mode(mo.Traj) contains Metro)
```

- **Query** 8. How long does *Bobby* walk during his trip?

```
SELECT duration(deftime(mo.Traj at Walk))
FROM MOGendon AS mo
WHERE mo.Name = "Bobby"
```

- **Query** 9. Where does *Bobby* walk during his trip?

```
SELECT trajectory(mo.Traj at Walk)
FROM MOGendon AS mo
WHERE mo.Name = "Bobby"
```

- **Query** 10. Find all people staying at office room 312 at the university for more than 2 hours.

```
SELECT mo.Name
FROM MOGendon AS mo, get_infra(SpaceGendon, ROOM) AS room
WHERE room.Name = "Uni-312" AND EXISTS
  SELECT *
  FROM SET(Piece,
    components(deftime(mo.Traj at room.Room)))
  WHERE duration(Piece) > 120
```

The trajectory of mo is restricted to the times when he/she was at room 312 which due to the subtype relationship can be used as a $\underline{genrange}$ value. Obviously the query refers to a single stay at this office rather than the aggregated time over many visits. Hence we decompose the definition time interval into disjoint intervals using **components**. This is transformed into a relation from which we select tuples for which the duration of the stay is more than two hours. If the relation is not empty for a given mo, then this mo qualifies for the result.

6.3 Interaction between Different Transportation Modes and Infrastructures

- **Query** 11. Find all buses passing the city center area.

```
SELECT bus.Name
FROM get_infra(SpaceGendon, BUS) AS bus,
  get_infra(SpaceGendon, OUTDOOR) AS outdoor
WHERE outdoor.Name = "CityCenter" AND
  freespace(bus.Bus) passes outdoor.Reg
```

To be able to compare the bus movement (of type $\underline{mpptn}$) with the city center region (which is a $\underline{region}$ value in free space) we must map it into the free space using operation **freespace**, i.e. convert it to an $\underline{mpoint}$.

- **Query** 12. Did bus 527 pass any traveler going by bicycle?

  We define *pass* to mean that the distance between the two objects is less than 3 meters.

```
SELECT mo.Name
FROM get_infra(SpaceGendon, MPPTN) AS mbus, MOGendon AS mo
WHERE mbus.Name = "Bus527" AND
  sometimes(distance(freespace(mbus.Vehicle),
    freespace(mo.Traj at Bicycle)) < 3.0)
```

Here we need to determine the distance between two moving objects, namely, the bus and a traveler going by bicycle. We assume the bus passes the bicycle if at some time their distance is small enough. To be able to determine the time dependent distance, both moving objects have to move within the same infrastructure. Therefore we first map both of them into free space, i.e., convert them to $\underline{mpoint}$. The application of the **distance** operator returns a moving real. Comparing this with the constant 3.0 returns an $\underline{mbool}$ value. Finally, **sometimes** yields true, if the $\underline{mbool}$ ever assumes the value true.[4]

- **Query** 13. Who entered bus 527 at bus stop "FernUni"?

[4] **sometimes** is a derived operation, **sometimes**(mb) = **not**(**isempty**(**deftime**(mb **at** true). See [20], Exercise 4.5.

```
SELECT mo.Name
FROM MOGendon AS mo, get_infra(SpaceGendon, BUS) AS bus,
  get_infra(SpaceGendon, BUSSTOP) AS busstop
WHERE bus.Name = "Bus527" AND busstop.Name = "University" AND
  val(initial(mo.Traj at genloc(bus.BusId, undef, undef)))
  = bus.Stop
```

- **Query** 14. Did anyone who was at the University on floor H-2 between 4:30pm and 5pm take a bus to the main (train) station?

  To be on floor H-2 means to be in any of the rooms of floor H-2. We assume a table is available associating rooms with the floors they belong to:

  ```
  University(Floor: string, RoomName: string)
  ```

  Then the query can be formulated as follows:

  ```
  LET qt1 = period(instant(2010, 12, 5, 16, 30),
              instant(2010, 12, 5, 17));
  LET qt2 = period(instant(2010, 12, 5, 16, 30),
              instant 2010, 12, 5, 19));

  SELECT mo.Name
  FROM MOGendon AS mo, University AS u,
    get_infra(SpaceGendon, ROOM) AS r,
    get_infra(SpaceGendon, BUSSTOP) AS bs1,
    get_infra(SpaceGendon, BUSSTOP) AS bs2,
  WHERE u.Floor = "H-2" AND
    u.RoomName = r.Name AND
    (mo.Traj atperiods qt1) passes r.Room AND
    bs1.Name = "University" AND
    bs2.Name = "Main station" AND
    val(initial((mo.Traj atperiods qt2) at Bus))
      = bs1.Stop AND
    val(final((mo.Traj atperiods qt2) at Bus))
      = bs2.Stop
  ```

The query implicitly requires that the bus was taken to the main station a little later than when the person was on floor H-2 (rather than on some other day). Hence we define a second time period from 4:30pm to 7pm during which the bus must be taken. We then ask for trajectories of people passing through the generic locations of rooms on floor H-2 within period $qt1$ as well as having a bus trip during $qt2$ that starts at the University bus stop and ends at the main station bus stop.

- **Query** 15. Who arrived by taxi at the university in December?

  To arrive by taxi at the university means that the final location of the passenger within the taxi belongs to some driveway area close to the university. We assume such a part of the road network has been entered into the database as an object $UniDriveway$ of type $\underline{gline}$.

  ```
  LET December = month(2010, 12);
  ```

```
SELECT mo.Name
FROM MOGendon AS mo
WHERE EXISTS
  SELECT *
  FROM SET(Trip,
    components((mo.Traj atperiods December) at Taxi))
  WHERE val(final(Trip)) inside UniDriveway
```

We reduce the trajectory of a person to the taxi trips in December. There may be several such trips (present in the resulting trajectory), hence we apply **components** to get the continuous pieces, i.e., the individual taxi trips. On the relation containing these trips, we check whether there is one with final destination within the university driveway area.

7 Implementation

The earlier work in [19, 13, 17] has been implemented in an extensible database system SECONDO [16] where the infrastructures there are free space and road network. The implementation of the model of this paper is underway within the same environment. The implementation strategies used are similar to those presented in the earlier papers [13, 29, 17]; so we do not describe such techniques here. Here we report on the current status of the implementation.

The generic model represents moving objects in multiple environments which covers the above two infrastructures, and the representation for locations and moving objects is consistent with the earlier ones. Currently, the data types for generic location and moving objects are already implemented as well as the data types for different kinds of infrastructure objects. Besides free space and road network, the other infrastructures (region-based outdoor, public transportation network and indoor) also exist and are managed by the database system.

Besides implementing the model, our goal is to create a benchmark that provides realistic data for infrastructures and generic moving objects as a basis for testing the implementation. Motivated by the lack of real data for all infrastructures in a consistent environment, we create all outdoor infrastructures based on real roads data which are available from many public resources. [54] demonstrates the infrastructures that we create showing roads, pavement areas, and bus routes.

The indoor environment is generated using public floor plans, and the indoor graph is created so that indoor navigation is available. In addition, the SECONDO user interface has been extended by a 3D visualization viewer, which is to display 3D indoor objects including rooms, staircases, 3D lines for indoor shortest paths and the animation of indoor moving objects.

Due to the difficulty of getting the real data of generic moving objects, we create them based on the result of trip plannings where the start and end locations can be located anywhere in the defined infrastructures. A complete navigation system through all environments is available supporting queries like "*find the shortest path from the office room to my apartment*". The system can generate moving objects with different transportation modes, e.g., $Indoor \rightarrow Walk \rightarrow Car, Walk \rightarrow Bus \rightarrow Walk \rightarrow Indoor$.

Some operators on generic moving objects are already implemented like **ref_id**, **trajectory**, **get_mode** so that one can do some manipulation on the data.

8 Related Work

The related work includes the following three parts:

Modeling Moving Objects

In the database literature, there has been a large body of recent works [42, 53, 19, 13, 46, 52, 21, 45, 7, 11, 36, 17, 28, 38, 26, 24] on modeling moving objects, all of which only consider one specific environment and do not discuss transportation modes. In addition, the location data is not represented in a multiresolution way. The closest to our work is [7, 36, 17] where the infrastructure is considered for representing location data. A semantic model for trajectories is proposed in [7] where the background geographic information is considered. It is restricted to a specific application domain where an algorithm is developed to map the positions of vehicles into a road network. Thus, the spatial aspect of trajectories is modeled in terms of a network, which consists of edges and nodes. A model for mobility pattern queries [36] is proposed which relies on a *discrete* view of spatio-temporal space. The movement of moving objects is based on a discrete representation of underlying space, called *reference* space. The space is partitioned into a set of zones each of which is uniquely identified by a label. Afterwards, the location is represented by mapping it into zones and a trajectory is defined as a sequence of labels. A so-called *route-oriented* model [17] is presented for moving objects in networks. It represents the road network by routes and junctions, and trajectories are integrated with road networks. As moving objects in a road network are located on roads and streets, the position can be represented by a route identifier and the relative position in that route, where a $line$ is used to describe the geometrical property of a route. However, the three aforementioned work share three common drawbacks: first, the *infrastructure* there is single, *free space* in [36] and *road network* in [7, 17], making the model not general and only feasible for one environment; second, the location representation is not in a multiresolution way where [36] is in a coarse level, which is identified by a symbol corresponding to a zone in space (if a precise location query is requested, for example, what is the relative position in that zone, the model can not answer), [7] is also in a rough level (road segment) and [17] only represents the precise location (without rough description); third, transportation modes are not handled. Besides, in [36] the trajectory is represented in a discrete way (a sequence of timestamps) instead of continuous.

GPS is the dominant positioning technology for outdoor settings, but for the indoor environment new techniques are required. A graph model based approach [24] is proposed for indoor tracking moving objects using the technology $RFID$. They assume the $RFID$ readers are embedded in the indoor space in some known positions and the indoor space is partitioned into cells corresponding to vertices in the graph. An edge in the graph indicates the movement between cells which is detected by $partitioning\ readers$. The raw trajectory is a sequence of RFID tags and from the raw trajectory they construct and refine the trajectory. The goal is to improve the indoor tracking accuracy, which is different from the intention of this paper, modeling moving objects. $Jensen\ et\ al.$ [25] present an index structure for moving objects in symbolic indoor space. The trajectory model is composed of records in the format $(oid, symbolicID, t)$ where oid is the moving object identifier, $symbloicID$ is the identifier for a specific indoor space region and T indicates time. They do not give the data type representing a space region and the location there is imprecise. It is known in which room the object is located, but the precise location inside is not represented.

Transportation Modes and Semantic Trajectory

A data model presented in [6] gives the framework of a transportation system which can provide a trip consisting of several transportation modes, e.g., $Bus, Walk, Train$. It integrates moving objects and graph-based databases to facilitate trip planning in urban transportation networks. They take into consideration how to provide a path connecting an origin and destination including multiple transportation modes so that the $shortest\_path$ has more constraints and choices, like different motion modes, number of transfers, etc. They define a graph model where each vertex corresponds to a place in a transportation network which has a name and a geometric representation, e.g., a point or a region. Each edge builds a connection between

two vertices and is associated with a kind of transportation mode attribute, for example, *pedestrian*, *auto*. Edges with different modes can be incident on the same vertices indicating that a transfer between different modes can happen. A trip is defined as a sequence of *legs*, where each *leg* represents a path with a kind of transportation mode. The model is for planning trips with various constraints, e.g., distance, duration, number of transfers. It is different from our work that considers modeling moving objects. They do not focus on representing the location data for moving objects in various environments, but the constraints with the trip. It does not give the data type to represent the *leg* and *trip*, but describes them conceptually and abstractly. We focus on representing moving objects in different environments and manage the trips in a database system so that users can query them in a system. Besides, it does not include the *indoor* environment. An interesting query called *Isochrones* is considered in [4] which is to find the set of all points on a road network so that a specific point of interest can be reached within a given time span. They do not focus on modeling moving objects and only take into account two transportation modes, *Walk* and *Bus*.

The work in Microsoft's *GeoLife* [60, 58] is different from ours. It focuses on discovering and inferring transportation modes from raw GPS trajectory data. The procedure comprises three phases. First, it partitions a GPS trajectory into several segments of different transportation modes, while maintaining a segment of one mode as long as possible. It identifies a set of features being independent of the velocity. Second, these features are fed into a classification model and output the probability of each segment being different transportation modes. Third, a graph-based post-processing algorithm is proposed to further improve the inference performance. We concentrate on representing moving objects in various environments with different transportation modes instead of inferring the modes. The results of their work can be used as the raw data for our model. In addition, they only take into consideration *outdoor* movement because they infer based on GPS data where a GPS receiver will lose signal indoors.

Recently, semantic trajectory [3, 43, 5] receives more attention in the literature which complements the recording of moving position. It concerns enriching trajectories with semantic annotations and allowing users to attach semantic data to specific parts of the trajectory. They define two parts called *stop* and *move* where a *stop* is defined as a specified spatial location according to the application and a *move* is the part of a trajectory delimited by two consecutive stops. The work does not propose a method for modeling moving objects but involves a data preprocessing model to add semantic information to moving objects trajectory before query processing. That is, the semantic data is not represented by the data model. Another difference is the semantic data there is an interesting place, e.g., a hotel or a touristic place, while ours is transportation modes.

Multi-Scale Representation

Paper [35] studies on multi-scale classification of moving object trajectories. They propose a model that allows to classify trajectories with respect to a multi-scale representation of a domain area, each scale level being a partition of the domain in zones. But the location represented there is only at the zone level, while the precise location inside is unknown. Although it is a multi-scale representation, the location data is still approximate. They focus on a specific query called *trajectory pattern*, while we address modeling moving objects in various environments as well as transportation modes. A multi-scale model [50] is proposed for vector maps which can identify the level of detail of entities forming in the map and allow for map storage and processing with an acceptable trade-off between cost and performance. It is different from our work that considers modeling moving objects.

9 Conclusions and Future Work

In this paper, we propose a generic data model to manage moving objects in multiple environments including public transportation network, *indoor*, region-based outdoor, road network and free space. With this model, the database system can manage the complete movement of objects in different environments. We consider the *space* where an object moves as a collection of so-called *infrastructures*. Each *infrastructure* corresponds to one moving environment and consists of a set of elements, called *infrastructure objects*. Using the *object-based* approach, we represent the location of moving objects by *referencing* to these *infrastructure objects*. A generic model is presented where the location is expressed by a function from time to space which can be applied for all environments. Using the method of *sliced representation*, we define a framework for generic moving objects. It is expressed in a multiresolution way where both approximate and accurate locations are supported. In addition, semantic data (transportation modes) is encapsulated for moving objects making the data expressive which can be used for knowledge analysis.

For each kind of *infrastructure*, we define the corresponding data types representing the space it covers and the relative location inside, as well as the location expression for objects moving in that *infrastructure*. A graph model is proposed for *indoor* navigation where three types of optimal routes are supported: shortest distance, smallest number of rooms and minimum traveling time. The generic model encapsulates the existing work for free space and road network, and they are consistent. A set of operators are defined in the type system for generic model and a relational interface is provided for exchanging data between values of proposed data types and a relational environment. Finally, a group of example queries are formulated on the model.

We envision several future directions. The first is to implement the data model in an extensible database system Secondo including data types, operators, and visualization technology (3D data for *indoor*). This implementation has already been partially be completed (see Section 7. Another topic is to conduct trajectory pattern mining and analysis as semantic data is involved. It is also interesting to develop a benchmark for generic moving objects as sketched in Section 7.

Acknowledgment

The first author gratefully acknowledges the financial support by Chinese Scholarship Committee.

References

[1] http://conversations.nokia.com/2008/09/23/indoor-positioning-coming-to-life/.

[2] http://research.microsoft.com/en-us/projects/geolife.

[3] L.O. Alvares, V. Bogorny, B. Kuijpers, J. Macedo, B. Moelans, and A. Vaisman. A model for enriching trajectories with semantic geographical information. In *ACM GIS*, 2007.

[4] V. Bauer, J. Gamper, R. Loperfido, S. Profanter, S. Putzer, and I. Timko. Computing isochrones in multi-modal, schedule-based transport networks. In *ACM GIS, Demo*, 2008.

[5] V. Bogorny, B. Kuijpers, and L. Alvares. St-dmql: A semantic trajectory data mining query language. *International Journal of Geographical Information Science*, 23(10):1245–1276, 2009.

[6] J. Booth, P. Sistla, O. Wolfson, and I.F. Cruz. A data model for trip planning in multimodal transportation systems. In *EDBT*, 2009.

[7] S. Brakatsoulas, D. Pfoser, and N. Tryfona. Modeling, storing and mining moving object databases. In *IDEAS*, 2004.

[8] X. Cao, G. Cong, and C.S. Jensen. Mining significant semantic locations from gps data. In *VLDB, Journal Track*, 2010.

[9] L. Chen, M.T. Özsu, and V. Oria. Robust and fast similarity search for moving object trajectories. In *SIGMOD*, 2005.

[10] Z. Chen, H.T. Shen, X. Zhou, Y. Zheng, and X. Xie. Searching trajectories by locations - an efficiency study. In *SIGMOD*, 2010.

[11] Z. Ding and R.H. Güting. Managing moving objects on dynamic transportation networks. In *SSDBM*, 2004.

[12] C. Düntgen, T. Behr, and R.H. Güting. Berlinmod: A benchmark for moving object databases. *VLDB Journal*, 18(6):1335–1368, 2009.

[13] L. Forlizzi, R.H. Güting, E. Nardelli, and M. Schneider. A data model and data structures for moving objects databases. In *SIGMOD*, pages 319–330, 2000.

[14] S. Grumbach, P. Rigaux, and L. Segoufin. Manipulating interpolated data is easier than you thought. In *VLDB*, 2000.

[15] R.H. Güting. Second-order signature: A tool for specifying data models, query processing and optimization. In *SIGMOD*, 1993.

[16] R.H. Güting, V. Almeida, D. Ansorge, T. Behr, Z. Ding, T. Höse, F. Hoffmann, and M. Spiekermann. Secondo: An extensible dbms platform for research prototyping and teaching. In *ICDE, Demo Paper*, 2005.

[17] R.H. Güting, V.T. de Almeida, and Z.M. Ding. Modeling and querying moving objects in networks. *VLDB Journal*, 2006.

[18] R.H. Güting, T. Behr, and J. Xu. Efficient k-nearest neighbor search on moving object trajectories. *VLDB Journal*, 2010.

[19] R.H. Güting, M.H. Böhlen, M. Erwig, C.S. Jensen, N.A. Lorentzos, M. Schneider, and M. Vazirgiannis. A foundation for representing and querying moving objects. *ACM TODS*, 25(1):1–42, 2000.

[20] R.H. Güting and M. Schneider. *Moving Objects Databases*. Morgan Kaufmann, 2005.

[21] C. Hage, C.S. Jensen, T.B. Pedersen, L. Speicys, and I. Timko. Integrated data management for mobile services in the real world. In *VLDB*, 2003.

[22] G.S. Iwerks, H. Samet, and K. Smith. Continuous k-nearest neighbor queries for continuous moving points with updates. In *VLDB*, 2003.

[23] C.S. Jensen, A. Kligys, T.B. Pedersen, and I. Timko. Multidimensional data modeling for location-based services. *VLDB Journal*, 13:1–21, 2004.

[24] C.S. Jensen, H. Lu, and B. Yang. Graph model based indoor tracking. In *MDM*, 2009.

[25] C.S. Jensen, H. Lu, and B. Yang. Indexing the trajectories of moving objects in symbolic indoor space. In *SSTD*, 2009.

[26] H. Jeung, Q. Liu, H.T. Shen, and X. Zhou. A hybrid prediction model for moving objects. In *ICDE*, 2008.

[27] H. Jeung, M.L. Yiu, X. Zhou, C.S. Jensen, and H.T. Shen. Discovery of convoys in trajectory databases. In *VLDB*, 2008.

[28] B. Kuijpers and W. Othman. Trajectory databases: Data models, uncertainty and complete query languages. In *ICDT*, 2007.

[29] J.A. Lema, L. Forlizzi, R.H. Güting, and M. Schneider. Algorithms for moving objects databases. *The Computer Journal*, 46(6):680–712, 2003.

[30] Y. Liu, O. Wolfson, and H. Yin. Extracting semantic location from outdoor positioning systems. In *MDM*, 2006.

[31] B. Lorenz, H.J. Ohlbach, and E.P. Stoffel. A hybrid spatial model for representing indoor environments. In *W2GIS*, 2006.

[32] K. Mouratidis, M. Hadjieleftheriou, and D. Papadias. Conceptual partitioning: An efficient method for continuous nearest neighbor monitoring. In *SIGMOD*, 2005.

[33] K. Mouratidis, Y. Lin, and M.L. Yiu. Preference queries in large multi-cost transportation networks. In *ICDE*, 2010.

[34] K. Mouratidis, M.L Yiu, D. Papadias, and N. Mamoulis. Continuous nearest neighbor monitoring in road networks. In *VLDB*, 2006.

[35] C. Mouza and P. Rigaux. Multi-scale classification of moving object trajectories. In *SSDBM*, 2004.

[36] C. Mouza and P. Rigaux. Mobility patterns. *GeoInformatica*, 9(4):297–319, 2005.

[37] C. Mouza, P. Rigaux, and M. Scholl. Efficient evaluation of parameterized pattern queries. In *CIKM*, 2005.

[38] R. Praing and M. Schneider. Modeling historical and future movements of spatio-temporal objects in moving objects databases. In *CIKM*, pages 183–192, 2007.

[39] S. Reddy, M. Mun, J. Burke, D. Estrin, M.H. Hansen, and M.B. Srivastava. Using mobile phones to determine transportation modes. *TOSN*, 6(2), 2010.

[40] P. Scarponcini. Generalized model for linear referencing in transportation. *GeoInformatica*, 6(1):35–55, 2002.

[41] S. Shekhar, M. Coyle, B. Goyal, D.R. Liu, and S. Sarkar. Data models in geographic information systems. In *Commun ACM 40*, volume 4, pages 103–111, 1997.

[42] P. Sistla, O. Wolfson, S. Chamberlain, and S. Dao. Modeling and querying moving objects. In *ICDE*, pages 422–432, 1997.

[43] S. Spaccapietra, C. Parent, M.L. Damiani, J. Macedo, F. Porto, and C. Vangenot. A conceptual view on trajectories. *Data Knowledge Engineering*, 65:126–146, 2008.

[44] L. Speicys and C.S. Jensen. Enabling location-based services – multi-graph representation of transportation networks. *GeoInformatica*, 2008.

[45] L. Speicys, C.S. Jensen, and A. Kligys. Computational data modeling for network-constrained moving objects. In *ACM-GIS*, 2003.

[46] J. Su, H. Xu, and O.H. Ibarra. Moving objects: Logical relationships and queries. In *SSTD*, 2001.

[47] Y. Tao, D. Papadias, and Q. Shen. Continuous nearest neighbor search. In *VLDB*, 2002.

[48] A. Thiagarajan and S. Madden. Querying continuous functions in a database system. In *SIGMOD*, 2008.

[49] M. Vazirgiannis and O. Wolfson. A spatiotemporal model and language for moving objects on road networks. In *SSTD*, 2001.

[50] R. Viana, P. Magillo, E. Puppo, and P.A. Ramos. Multi-vmap: A multi-scale model for vector maps. *Geoinformatica*, 10:359–394, 2006.

[51] A. Voisard and B. David. A database perspective on geospatial data modeling. *TKDE*, 14(2):226–242, 2002.

[52] O. Wolfson, S. Chamberlain, K. Kalpakis, and Y. Yesha. Modeling moving objects for location based services. In *IMWS*, 2001.

[53] O. Wolfson, B. Xu, S. Chamberlain, and L. Jiang. Moving objects databases: Issues and solutions. In *SSDBM*, pages 111–122, 1998.

[54] J. Xu and R.H. Güting. Infrastructures for research on multimodal moving objects. In *MDM, Demo Paper*, 2011.

[55] B. Yang, H. Lu, and C.S. Jensen. Probabilistic threshold k nearest neighbor queries over moving objects in symbolic indoor space. In *EDBT*, 2010.

[56] J. Zhang, D. Papadias, K. Mouratidis, and M. Zhu. Spatial queries in the presence of obstacles. In *EDBT*, 2004.

[57] J. Zhang, M. Zhu, D. Papadias, Y. Tao, and D.L. Tee. Location-based spatial queries. In *SIGMOD*, 2003.

[58] Y. Zheng, Y. Chen, X. Xie, and W.Y. Ma. Understanding transportation mode based on gps data for web application. *ACM Transaction on the Web*, 4(1):1–36, 2010.

[59] Y. Zheng, Q. Li, Y. Chen, X. Xie, and W.Y. Ma. Understanding mobility based on gps data. In *UbiComp*, 2008.

[60] Y. Zheng, L. Liu, L. Wang, and X. Xie. Learning transportation mode from raw gps data for geographic applications on the web. In *WWW*, 2008.

[61] Y. Zheng, L. Zhang, X. Xie, and W.Y. Ma. Mining interesting locations and travel sequences from gps trajectories. In *WWW*, 2009.

Appendix: Example Relations and Query Signatures

The following relations are used in the queries of Section 6. Infrastructure relations are accessed by the **get_infra** operator.

```
MOGendon(Mo_id: int, Traj: genmo, Name: string)

rel_busstop  (BusStopId: int, Stop: busstop, Name: string)
rel_busroute (BusRouteId: int, Route; busroute, Name: string,
              Up: bool)
rel_bus      (BusId: int, Bus: mpptn, Name: string)
rel_room     (RoomId: int, Room: groom, Name: string)
rel_door     (DoorId: int, Door: door)
rel_roompath (RoomPathId: int, Door1: int, Door2: int, Weight: real,
              Room: groom, Name: string, Path: line)
rel_rbo      (RegId: int, Reg: region, Name: string)
rel_rn       (RoadId: int, Road: line, Name: string)
```

In the following tables we show the signatures of operations used in the queries. Operator **get_infra** is omitted as it occurs in almost every query.

Query No.	Operator	Signature		Definition
Query 1	**instant**	$\underline{int} \times \underline{int} \times \underline{int} \times \underline{int}$	$\rightarrow \underline{instant}$	Intro Section 6
	atinstant	$\underline{genmo} \times \underline{instant}$	$\rightarrow \underline{intime}(\underline{genloc})$	Table 8
	val	$\underline{intime}(\underline{genloc})$	$\rightarrow \underline{genloc}$	Table 8
	get_ref	$\underline{genloc}$	$\rightarrow \underline{ioref}$	Table 10
	ref_obj	$\underline{ioref}$	$\rightarrow \alpha$	Table 10
Query 2	**at**	$\underline{genmo} \times \underline{tm}$	$\rightarrow \underline{genmo}$	Table 10
	get_ref	$\underline{genmo}$	$\rightarrow \underline{set}(\underline{ioref})$	Table 10
	contains	$\underline{set}(\underline{ioref}) \times \underline{int}$	$\rightarrow \underline{bool}$	Section 4.4
Query 3	**hour**	$\underline{int} \times \underline{int} \times \underline{int} \times \underline{int}$	$\rightarrow \underline{periods}$	Intro Section 6
	atperiods	$\underline{genmo} \times \underline{periods}$	$\rightarrow \underline{genmo}$	Table 8
	at	$\underline{genmo} \times \underline{tm}$	$\rightarrow \underline{genmo}$	Table 10
	get_ref	$\underline{genmo}$	$\rightarrow \underline{set}(\underline{ioref})$	Table 10
	contains	$\underline{set}(\underline{ioref}) \times \underline{int}$	$\rightarrow \underline{bool}$	Section 4.4
Query 4	**genloc**	$\underline{int} \times \underline{real} \times \underline{real}$	$\rightarrow \underline{genloc}$	Table 12
	passes	$\underline{genmo} \times \underline{genloc}$	$\rightarrow \underline{bool}$	Table 8
Query 5	subtype	$\underline{busstop}$	$< \underline{genloc}$	Table 11
	at	$\underline{genmo} \times \underline{tm}$	$\rightarrow \underline{genmo}$	Table 10
	at	$\underline{genmo} \times \underline{genloc}$	$\rightarrow \underline{genmo}$	Table 8
	deftime	$\underline{genmo}$	$\rightarrow \underline{periods}$	Table 8
	duration	$\underline{periods}$	$\rightarrow \underline{real}$	Table 8
Query 6	**instant**	$\underline{int} \times \underline{int} \times \underline{int} \times \underline{int}$	$\rightarrow \underline{instant}$	Intro Sect 6
	atinstant	$\underline{genmo} \times \underline{instant}$	$\rightarrow \underline{intime}(\underline{genloc})$	Table 8
	val	$\underline{intime}(\underline{genloc})$	$\rightarrow \underline{genloc}$	Table 8
	=	$\underline{genloc} \times \underline{genloc}$	$\rightarrow \underline{bool}$	Table 7
Query 7	**get_mode**	$\underline{genmo}$	$\rightarrow \underline{set}(\underline{tm})$	Table 10
	contains	$\underline{set}(\underline{tm}) \times \underline{tm}$	$\rightarrow \underline{bool}$	Section 4.4
Query 8	**at**	$\underline{genmo} \times \underline{tm}$	$\rightarrow \underline{genmo}$	Table 10
	deftime	$\underline{genmo}$	$\rightarrow \underline{periods}$	Table 8
	duration	$\underline{periods}$	$\rightarrow \underline{real}$	Table 8

Query No.	Operator	Signature		Definition
Query 9	**at**	$genmo \times tm$	$\rightarrow genmo$	Table 10
	trajectory	$genmo$	$\rightarrow genrange$	Table 8
Query 10	subtype	$groom$	$< genrange$	Table 11
	at	$genmo \times genrange$	$\rightarrow genmo$	Table 8
	deftime	$genmo$	$\rightarrow periods$	Table 8
	components	$periods$	$\rightarrow set(periods)$	Section 4.4
	duration	$periods$	$\rightarrow real$	Table 8
Query 11	**genloc**	$int \times real \times real$	$\rightarrow genloc$	Table 12
	freespace	$mpptn$	$\rightarrow mpoint$	Table 12
Query 12	**distance**	$mpoint \times mpoint$	$\rightarrow mreal$	Paper [19]
	freespace	$mpptn$	$\rightarrow mpoint$	Table 12
	freespace	$genmo$	$\rightarrow mpoint$	Table 12
Query 13	subtype	$busstop$	$< genloc$	Table 11
	genloc	$int \times real \times real$	$\rightarrow genloc$	Table 12
	at	$genmo \times genloc$	$\rightarrow genmo$	Table 8
	initial	$genmo$	$\rightarrow intime(genloc)$	Table 8
	val	$intime(genloc)$	$\rightarrow genloc$	Table 8
	=	$genloc \times genloc$	$\rightarrow bool$	Table 7
Query 14	subtype	$groom$	$< genrange$	Table 11
	passes	$genmo \times genrange$	$\rightarrow bool$	Table 8
	subtype	$busstop$	$< genloc$	Table 11
	atperiods	$genmo \times periods$	$\rightarrow genmo$	Table 8
	at	$genmo \times tm$	$\rightarrow genmo$	Table 10
	initial	$genmo$	$\rightarrow intime(genloc)$	Table 8
	final	$genmo$	$\rightarrow intime(genloc)$	Table 8
	val	$intime(genloc)$	$\rightarrow genloc$	Table 8
	=	$genloc \times genloc$	$\rightarrow bool$	Table 7
Query 15	subtype	$gline$	$< genrange$	Table 11
	atperiods	$genmo \times periods$	$\rightarrow genmo$	Table 8
	components	$periods$	$\rightarrow set(periods)$	Section 4.4
	final	$genmo$	$\rightarrow intime(genloc)$	Table 8
	val	$intime(genloc)$	$\rightarrow genloc$	Table 8
	at	$genmo \times tm$	$\rightarrow genmo$	Table 10
	inside	$genloc \times genrange$	$\rightarrow bool$	Table 7

Verzeichnis der zuletzt erschienenen Informatik-Bericht

[343] Güting, R. H.:
Operator-Based Query Progress Estimation

[344] Behr, T., Güting, R. H.:
User Defined Topological Predicates in Database Systems

[345] vor der Brück, T.; Helbig, H.; Leveling, J.:
The Readability Checker Delite Technical Report

[346] vor der Brück, T.:
Application of Machine Learning Algorithms for Automatic Knowledge Acquisition and Readability Analysis Technical Report

[347] Fechner, B.:
Dynamische Fehlererkennungs- und –behebungsmechanismen für verlässliche Mikroprozessoren

[348] Brattka, V., Dillhage, R., Grubba, T., Klutsch, A.:
CCA 2008 - Fifth International Conference on Computability and Complexity in Analysis

[349] Osterloh, A.:
A Lower Bound for Oblivious Dimensional Routing

[350] Osterloh, A., Keller, J.:
Das GCA-Modell im Vergleich zum PRAM-Modell

[351] Fechner, B.:
GPUs for Dependability

[352] Güting, R. H., Behr, T., Xu, J.:
Efficient *k*-Nearest Neighbor Search on Moving Object Trajectories

[353] Bauer, A., Dillhage, R., Hertling, P., Ko K.I., Rettinger, R.:
CCA 2009 Sixth International Conference on Computability and Complexity in Analysis

[354] Beierle, C., Kern-Isberner, G.
Relational Approaches to Knowledge Representation and Learning

[355] Sakr, M.A., Güting, R.H.
Spatiotemporal Pattern Queries

[356] Güting, R. H., Behr, T., Düntgen, C.:
SECONDO: A Platform for Moving Objects Database Research and for Publishing and Integrating Research Implementations

[357] Düntgen, C., Behr, T., Güting, R.H.:
Assessing Representations for Moving Object Histories

[358] Sakr, M.A., Güting, R.H.
Group Spatiotemporal Pattern Queries

[359] Hartrumpf, S., Helbig, H., vor der Brück, T. , Eichhorn, C.
SemDupl: Semantic Based Duplicate Identification