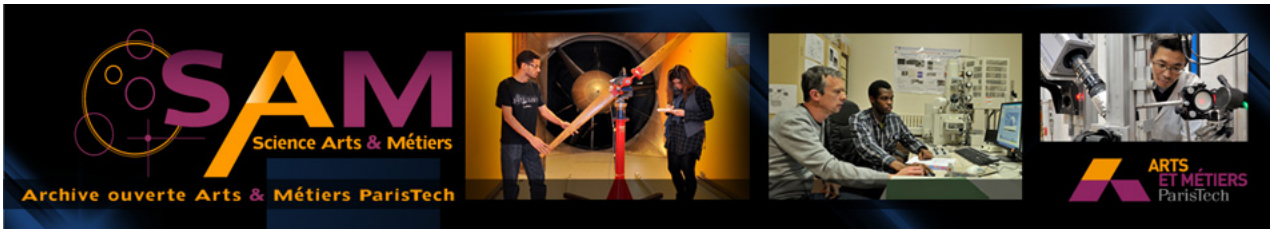
Context-Aware Processing of Continuous Location-Dependent Queries in Indoor Environments

Imad Afyouni, Sergio Ilarri, Cyril Ray, Christophe Claramunt



Context-Aware Modelling of Continuous Location-Dependent Queries in Indoor Environments

Imad Afyouni [a],Sergio Ilarri [b], Cyril Ray [a] and Christophe Claramunt [a]
[a] *Naval Academy Research Institute, 29240 Brest Cedex 9, France*
E-mail: {imad.afyouni,cyril.ray,christophe.claramunt}@ecole-navale.fr
[b] *Department of Computer Science and Systems Engineering, University of Zaragoza, Maria de Luna 1, 50018 Zaragoza, Spain*
E-mail: silarri@unizar.es

Abstract
Emerging and continuing advances in ambient systems and localization techniques have brought novel opportunities to develop context-aware navigation services in indoor environments. Diverse kinds of services delivered to the users can be provided by enabling real-time integration of contextual dimensions. In particular, continuous location-dependent queries can be considered as key elements for the development of different categories of context-aware services. However, most work on location-dependent query processing has been mainly oriented towards outdoor environments. This paper introduces a generic architecture for continuous location-dependent and navigation-related queries in indoor environments. A multi-level model of space is designed by taking into account contextual information and the hierarchical layout of an indoor environment. The semantics of a query language for continuous location-dependent queries are introduced, along with some motivating sample queries.

Keywords: Context-aware services, location-dependent queries, indoor spatial data model, query language, continuous processing architecture

1. Introduction

Location-based services [43] have recently attracted extensive research attention, as their development is expected to have a significant impact for end users in both indoor and outdoor environments. Such services should offer customized access to information by taking into account the location of mobile users. More generally, *context-aware systems* exploit contextual dimensions such as user-centred dimensions (e.g., user profile, user's physical/cognitive capabilities), environmental context (e.g., location, light, etc.), temporal context, and the context of execution (e.g., network connectivity, nearby resources, etc.). This allows to anticipate user's needs and to customize the user's experience [5,9,14].

A successful integration of indoor spaces (e.g., houses, commercial malls, etc.) and context-aware systems still requires the development of dynamic and flexible spatial models that provide appropriate services to mobile users acting in the environment. In a previous work [2], we introduced a context-dependent multi-granular spatial model that embeds different levels of granularity and represents: (i) all the *features* that populate an indoor environment, where a feature can refer to either a person (i.e., a mobile user or any other social

entity of interest[1]) or an object of interest (e.g., sensors, exits, tables, continuous phenomena such as a fire, etc.); (ii) their spatial properties (e.g., location and extent); and (iii) the behaviours that emerge from them (i.e., how these objects can interact and communicate within the environment). It should be noted that an *object of interest (OOI)* may or may not have communication capabilities, be mobile or static, physical or virtual, and attractive or repellent (i.e., depending on whether the user may want to reach it or to avoid it, respectively)[2].

A wide range of location-aware services can be applied to indoor environments. The main goal of these services is to provide the user with the ability to interact with his/her physical surroundings in order to achieve some objectives. Location-aware and user-centred services can be distinguished according to two modes of data access: *pull* mode and *push* mode. These access modes have been studied in the field of distributed and mobile databases [39,49]. For indoor context-aware services, they can be summarized as follows:

- Pull-based location-aware services comprise explicit requests triggered by the user with the aim of "pulling" some location-dependent information from the service provider. As an example, a user in a building may submit a request to locate the nearest exit.
- Conversely, push-based services imply communications initiated by the service provider without having been explicitly requested by the user. The service provider takes into account the location information of subscribed users to trigger alerts or contextual advertising and push the information to the user's device. As an example, one can imagine a user who is subscribed to a service that automatically alerts him/her when one of his/her colleagues is nearby.

Location-dependent queries [27,58] are typical examples of pull-based services needed in context-aware systems, as well as a key building block to detect situations of interest for push-based services. The location-dependent character of these queries means that any change of the locations of objects that are involved in the query may significantly affect the answer. For example, if a user wants to find out his/her friends within a range of 100 meters while navigating a shopping centre, the answer depends on both the user's current position and the location of the nearest friends. This type of query is particularly challenging because, in most cases, the user and entities relevant for the query (e.g., the friends of the user) are moving. Location-dependent queries have been surveyed in [27]; some particularly relevant queries in an indoor context are briefly described:

1. *Position queries* determine the locations of mobile and static objects, and are processed according to either a geometric or symbolic model of space (cf., [3]). Location-dependent queries cannot be carried out without up-to-date information on the locations of objects of interest [7].
2. *Navigation queries* encompass all the queries that directly help the users to find and reach points of interest by providing them with navigational information while optimizing some criteria such as the total traversed distance or travel time. Examples of such queries are: (i) discovering *optimal* paths to a nearest point of interest (e.g., landmark, place, etc.), (ii) planning a path to a destination, etc.
3. *Range queries* are used to find and retrieve information about objects of interest or places within a user-specified range or area [52,57]. Those queries support navigation by continuously updating relevant details according to users' movements. Ranges may be characterized by a circular or rectangular-shaped window in which objects of interest must be located. In addition, range queries may be static or dynamic according to whether or not the query point is in a static location. Similarly, a range query can be applied on static or dynamic data, depending on whether the target objects are moving or not.
4. *k nearest neighbour (kNN) queries* search for the k closest qualifying objects to the moving user with respect to his/her current location [46,58]. As opposed to range queries, kNN queries are range-independent, except in the case of *constrained nearest neighbour queries* [20], where the search is constrained to a given region. The user initiates a request by specifying some characteristics about

[1] Human beings that are located in the vicinity and are of interest to the query are referred to as social entities.

[2] The distinction between attractive and repellent events is similar to the one suggested in [16] regarding attraction and repulsion events.

objects of interest, so that the k closest objects whose specifications meet these characteristics are retrieved (e.g., the closest available colour printer or the k nearest friends).

Appropriate management of static and dynamic data is a key issue for processing these queries, since the result of a query is only valid for a particular location of the query issuer and for certain locations of the objects of interest. As those queries are time-sensitive and location-dependent, they may be valid only for a given period of time and within a given area (i.e., data returned are only spatio-temporally valid). So, they must be processed as *continuous queries* [47], which means that the system should continually keep the answers up-to-date, until the query is explicitly cancelled by the user. However, regularly updating those query answers may imply significant communication overhead and additional processing cost at the server side. Several approaches to alleviate this problem have been proposed. For example, the concept of *validity region*, introduced in [58], determines a safe area around the initial user location in which the result of the query is always valid. Many variants of continuous location-dependent queries are summarized in [27]. Although many research studies have discussed location-dependent queries and location-based services, a few works have discussed the problem of incorporating contextual dimensions, particularly those related to the *user-centric and environmental context*, into query processing. Indeed, this may significantly affect the answer to a query even if the locations of the query issuer and other involved objects have not changed.

The research presented in this paper studies location- and context-aware services and queries in indoor environments, with a special focus on navigation-related queries (i.e., mainly path search, range, and nearest neighbour queries). A unique combination of challenges arises, as the proposal must be able to represent different kinds of location-dependent queries in a flexible manner, and to take into account additional context information, time-dependency, and the hierarchical layout of the indoor environment. The remainder of the paper is organised as follows. Section 2 introduces a hierarchical indoor data model, and emphasizes its interest for location-aware queries and services. Section 3 firstly presents an architecture for continuous query processing, and discusses the integration of the indoor data model with a decentralized query processing and data management architecture. Secondly, a query language that models continuous location-dependent queries in an indoor environment is proposed. Section 4 reviews related works, while Section 5 draws some conclusions and outlines further work.

2. Modelling Approach

Indoor spatial models have been studied and developed in many areas, ranging from mobile robot mapping to Geographic Information Systems (GIS) and ubiquitous computing [1,50]. In a previous work, preliminary requirements for the development of indoor spatial models have been introduced from a context-aware system perspective [3]. Those are classified into two categories: service-oriented and efficiency-related requirements. Based on these requirements, a modelling approach of an indoor-oriented system that takes into account different levels of spatial granularity is introduced. The modelling approach developed in this paper is an extension of our preliminary work reported in [2], and introduces a multi-granular spatial representation of an indoor system that can be integrated into a context-aware system architecture. This model represents: (i) all the features that populate the environment, (ii) their spatial properties, and (iii) the behaviours or actions that emerge from them (see Fig. 1). The model is hierarchically organised and can be viewed as a tree structure in which location information is represented at different levels of abstraction. This hierarchical design can support a large spectrum of applications that can be developed at different levels of abstraction, and offers a promising solution to alleviate performance and scalability issues in location-dependent query processing.

Let us formally present the main concepts of the indoor data model. This multi-granular context-dependent model represents an indoor environment with three complementary components $\langle \mathcal{S}, \mathcal{F}, \mathcal{A} \rangle$, where:

- The spatial component $\mathcal{S} = \bigcup_{i=1...|\mathcal{S}|} \mathcal{S}_i$ is made of a set of layers ($\mathcal{S}_i$) hierarchically organised and representing the indoor space, and thus defining the multi-granular spatial structure of the model.

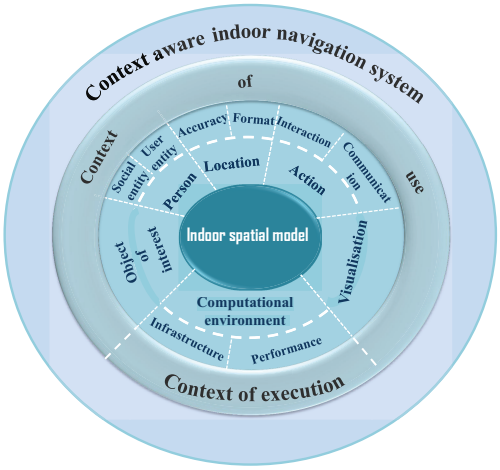


Figure 1. Indoor spatial model as a basic component for the design of context-aware information systems

- The feature component $\mathcal{F} = \bigcup(\mathcal{P}, \mathcal{OOI})$ encompasses the features (i.e., persons ($\mathcal{P}$) and objects of interest ($\mathcal{OOI}$)) located in the environment.
- The action component $\mathcal{A} = \bigcup(\mathcal{FA}, \mathcal{SA})$ represents actions that are either predefined and triggered automatically by the system in form of informative, context-aware messages ($\mathcal{SA}$), or generated by a given feature acting in the environment ($\mathcal{FA}$).

These three components are hereafter discussed in more detail.

2.1. Spatial Component

A spatial component $\mathcal{S}$ contains a set of spatial layers hierarchically organized. The illustration in Fig. 2 describes a part of a scenario where a user navigates inside a three-story laboratory building. The ground floor of the building comprises two teaching labs and some administrative staff offices. Offices of the Linguistics Department are located on the fist floor. Specifically, Fig. 2 illustrates the second floor of the building where the Computer Science Department is located. The core layer ($\mathcal{S}_1$) is firstly presented. Then, other coarser layers that can be incorporated into the hierarchical data model are discussed.

2.1.1. Core Spatial Layer

The core layer $\mathcal{S}_1$ (referred to as $\mathcal{S}_{micro}$) of the indoor data model is made of a fine-grained, context-dependent graph $G_{micro} = (V_{micro}, E_{micro}, W_{length}, W_{time})$ embedded within a spatial grid and which covers the indoor space[3] (Fig. 2). Consequently, vertices of the base graph represent cells within the grid, and connections between cells are materialized by edges.

In the definition of $\mathcal{S}_{micro}$, $V_{micro} = \{v_i\}$ is the set of vertices and $E_{micro} \subseteq V_{micro} \times V_{micro}$ is the set of edges. For each edge $e = (v_i, v_j) \in E_{micro}$, there exist two time-dependent cost functions $\omega l_{i,j}(t) \in W_{length}$ and $\omega t_{i,j}(t) \in W_{time}$ that compute the *length* and *travel-time* from v_i to v_j, respectively, if traversal is started at instant t. Besides time, this model also takes into account other contextual dimensions such as *user profiles* and *real-time events*, to further associate impedances with edge weights. User profiles are handled by considering generic graphs that are derived from the base graph G_{micro} and which correspond to predefined categories of users. Effects of real-time events on edge weights will be discussed later in Section 2.2.3.

Figure 2. A fine-grained graph of a floor plan: first level of the hierarchical spatial data model

Each node $v \in V_{micro}$ has a set of attributes that describe its physical location or state (i.e.,

[3]The motivation behind the use of a fine-grained graph as the core spatial layer is provided in more detail in previous work [2,32].

whether it is accessible or not). A node v is formally defined by the tuple $\langle v_{id}, \mathtt{x}_v, \mathtt{y}_v, \mathtt{s}_v, \mathtt{L}_v, \mathtt{A}_v \rangle$. v_{id} is the node identifier, $(\mathtt{x}_v, \mathtt{y}_v)$ denotes the geometric location of v according to a reference system, and $\mathtt{s}_v \in \{free, occupied\}$ determines whether or not the node v is physically occupied by an object at that moment. We assume that nodes which are occupied by static objects are inaccessible for path planning. Let $\Sigma_{label} = \{\Sigma_{fine-grained} \cup \Sigma_{room} \cup \Sigma_{floor} \cup \Sigma_{building}\}$ be a set of labels or symbolic values that consists of all the identifiers of the topological hierarchy (i.e., local identifiers of nodes at the fine-grained level, as well as room, floor, and building identifiers) for a given space. Hence, $\mathtt{L}_v \subset \Sigma_{label} = \{$*local-id*, *room-id*, *floor-id*, *building-id*$\}$ is a set of labels assigned to v, where *local-id* denotes its local identifier at the fine-grained level, and the others are associated according to their belonging to the topological hierarchy. We assume at this level that $\mathtt{v}$ belongs to one and only one room, and one building. In contrast, *floor-id* is a subset of the set of floor identifiers since, for instance, a node located on a staircase may belong to several floors. Finally, $\mathtt{A}_v \subset \mathcal{A}$ is the set of triggered actions, i.e., contextual messages or notifications that are predefined and can be executed according to some contextual constraints (e.g., to remind a user navigating a shopping centre to buy some food or fruit stock when he/she is located next to a supermarket).

An edge $e \in E_{micro}$ is defined by a tuple $\langle (v_i, v_j), \mathtt{L}_e, \omega l_{i,j}(t), \omega t_{i,j}(t) \rangle$, where $v_i, v_j \in V_{micro}$, $v_i \neq v_j$, and $\mathtt{L}_e \subset \Sigma_{label}$ is a subset of the set of labels (e might have multiple labels when it intersects several spatial units -e.g., rooms-). $\omega l_{i,j}(t)$ and $\omega t_{i,j}(t)$ are time-dependent functions associated with the traversal of e. The traversal of some edges may be constrained by a temporal interval defined at the application level, and within which the traversal is possible; otherwise the corresponding edge cannot be traversed. These functions are defined as follows:

$$\omega l_{i,j}(t) = \begin{cases} Ed(v_i, v_j) & if\ \mathtt{t} \in [t_{start}, t_{end}] \\ \infty & otherwise \end{cases}$$

where $Ed(v_i, v_j)$ is the Euclidean distance between v_i and v_j, and t_{start} and t_{end} are defined at the application level (for example, $[08:00, 17:00]$ can be specified for an office building).

$$\omega t_{i,j}(t) = \begin{cases} f(\omega l_{i,j}(t)) & if\ \mathtt{t} \in [t_{start}, t_{end}] \\ \infty & otherwise \end{cases}$$

where $f(\omega l_{i,j}(t))$ is a length-dependent time function that further associates impedances to compute the travel time between v_i and v_j.
Consequently, the network distance and the travel time from v_s to v_d are computed as indicated in Definitions 1 and 2, respectively.

Definition 1 ***Fine-grained and time-dependent network distance:*** *Let $p = \{v_{start}{=}v_1 \rightarrow v_2 \rightarrow \ldots \rightarrow v_k{=}v_{goal}\}$ be a path that contains a sequence of nodes $v_i \in V_{micro}$, i=1, ..., k. The fine-grained network distance of p is given by $length_{start,goal}(t_{start}) = \sum_{i=1}^{k-1} \omega l_{i,i+1}(t_i)$, where $t_i = t_{i-1} + \omega t_{i-1,i}(t_{i-1})$ represents the estimated time instant at node v_i, $\forall$ i=2, ..., k, and $t_1 = t_{start}$.*

Definition 2 ***Fine-grained and time-dependent travel time:*** *Let $p = \{v_{start}{=}v_1 \rightarrow v_2 \rightarrow \ldots \rightarrow v_k{=}v_{goal}\}$ be a path that contains a sequence of nodes $v_i \in V_{micro}$, i=1, ..., k. The fine-grained travel time of p is given by $time_{start,goal}(t_{start}) = \sum_{i=1}^{k-1} \omega t_{i,i+1}(t_i)$, where $t_i = t_{i-1} + \omega t_{i-1,i}(t_{i-1})$ represents the estimated time instant at node v_i, $\forall$ i=2, ..., k, and $t_1 = t_{start}$.*

The core spatial layer is built in an offline phase and a subsequent online phase is in charge of updating potential changes and time-dependent data. For instance, in the offline phase, nodes that are covered by static objects (e.g., a wall, a table, etc.) are marked as occupied whereas the rest are considered initially free. Furthermore, the state of a node depends also on the user profile, since different kinds of users may have a completely different set of accessible nodes (e.g., a certain node may be apparently free but correspond to a room that can only be entered with a key card). This can also be statically managed with the use of a user access model, as discussed in Section 2.2.2.

2.1.2. Coarser Spatial Layers

A node v at a coarser layer $\mathcal{S}_i \in \{\mathcal{S}_2, \ldots, \mathcal{S}_{|\mathcal{S}|}\}$ is defined as an aggregation of a subgraph of the finer graph, and is denoted by $\langle \mathtt{L}_v, \mathtt{A}_v \rangle$, where $\mathtt{L}_v \subset \Sigma_{label}$ is the set of labels assigned to v, which is adapted accordingly to fit the corresponding level of abstraction, and $\mathtt{A}_v \subset \mathcal{A}$ comprises the set of triggered actions that are predefined at the cor-

responding node. Cost functions are derived and processed based on the edge weights of the fine-grained level. In the following, the relevant layers considered in the data model are described, along with an explanation of how these layers can be derived.

Exit hierarchy. An exit is an important element of the data model used for query processing, through which a user can leave or enter a place (e.g., doorways or staircases). An exit is represented as an abstract node that belongs to two different spatial units, and is derived by aggregating boundary nodes of both units whose adjacent node lists contain at least one neighbour that belongs to the other spatial unit (Fig. 3). By means of these exits, optimal network distances and travel times between relevant pairs of exits are pre-processed and cached in order to reduce on-the-fly computation of hierarchical path searches. An exit e' is relevant for a given exit e if and only if e' is directly reachable from e (i.e., there is an accessible passageway for pedestrians from e to e' which does not involve any other exit). An exit hierarchy is constructed at a higher level of abstraction, which allows computing optimal distances between locations to be used later for processing diverse kinds of queries.

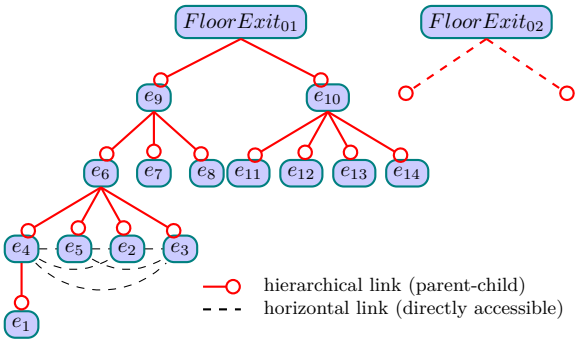


Figure 3. Part of the exit hierarchy[4] derived from the fine–grained graph (Floor-01, Building-1)

More formally, let r, $r' \in \Sigma_{label}$ be the labels of two connected rooms, the exit representing the doorway between r and r' is given by: $e_{r,r'} = \{v_i, v_j \in V_{micro} \mid \exists e \in E_{micro}, e = (v_i, v_j) \wedge r \in e.\mathtt{L}_e \wedge r' \in e.\mathtt{L}_e\}$. Regarding its belonging to the topological hierarchy, an exit is also characterised by: $\mathtt{L}_{e_{r,r'}} = \{$*local-id*, $\{r, r'\}$, *floor-id*, *building-id*$\}$. $FloorExit_{01}$ is an example of an exit depicted in Fig. 3, which belongs to two structural units: $Stair_{01}$ and HW_{03} (see Fig. 4). Therefore, $\mathtt{L}_{FloorExit_{01}} = \{$*FloorExit-01*, $\{Stair_{01}, HW_{03}\}$, *Floor-01*, *Building-1*$\}$. In a similar way, one can derive the abstract nodes of the second layer representing all exits on a given floor. An abstract edge $(e_{r,r'}, e_{r',r''})$ in the exit hierarchy is a path made of a sequence of nodes and edges of the fine-grained level that compose the optimal network distance from a node $v_{start} \in e_{r,r'}$ to a node $v_{goal} \in e_{r',r''}$. An edge of the exit hierarchy is referred to as *exit-path* and is denoted by $\langle source\_exit\_id, target\_exit\_id, length, time\rangle$. The optimal network distance and travel time are computed by applying $length_{start,goal}(t_{start})$ and $time_{start,goal}(t_{start})$, and the resulting values are associated with each edge of the exit hierarchy, thus forming the second layer of the data model.

Moreover, exits are organised in a hierarchical manner since a flat graph does not reflect their significance from a semantic navigation point of view [23]. As illustrated in Fig. 3, this hierarchical structure allows to distinguish between a room exit and a floor exit, which is represented at a higher level of abstraction due to its importance, so that a direct path from a current position to the nearest floor/building exit can be easily determined. Other edges between exits of the same level are also materialized according to their connectivity (horizontal links illustrated as dashed lines in Fig. 3)[5]. Consequently, the final representation of this layer preserves the connectivity between directly accessible exits while emphasizing their importance for navigation purposes. A generalisation of this hierarchy that covers a multi-storey building is used for path planning. Consequently, an exit of a ground floor has a *building exit* as a parent node, and a first-floor exit as a child node since both are parts of a staircase.

It should be emphasized that exit-paths in this hierarchy are assumed to be undirected. However, this model can be adapted for specific scenarios where a one-way access to several areas is required. This can be done by either replicating edges in the opposite direction or associating a property to each edge that adds impedances to the path

[4] The hierarchy is not fully illustrated in Fig. 3, since the right part rooted at $FloorExit_{02}$ is developed similarly.

[5] For clarity's sake, not all the edges that depict connectivity between exits are shown in Fig. 3.

weight depending on the travel direction. Adjustments of the query processing algorithms will be needed accordingly.

Location hierarchy. Incorporating information about exits into the topological hierarchy enables the modelling of optimal paths at an abstract layer. Those are used to facilitate hierarchical path searches and to alleviate performance issues raised while traversing the fine-grained graph. Although connectivity relationships between those structural units can be computed from the exit hierarchy, an adjacency relationship, for instance, needs to be associated to each unit in a separate abstraction layer. Therefore, such topological semantics are not explicitly materialised in the exit hierarchy, even though information representing their belonging to the topological hierarchy has been incorporated. Consequently, a location hierarchy that is based on a connectivity graph, which represents rooms as nodes and doorways as edges, can be derived as an additional layer in order to preserve topological relationships (Fig. 4).

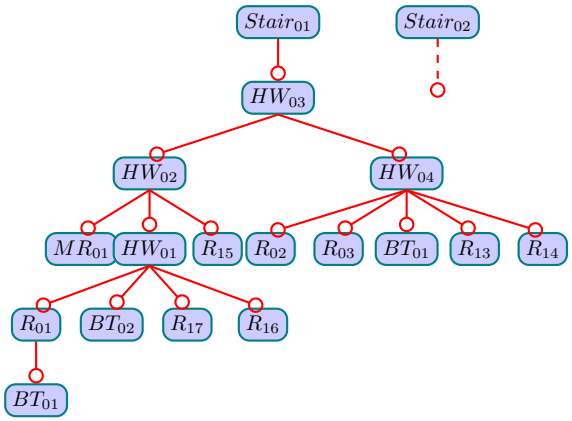


Figure 4. Part of the location hierarchy derived from the fine-grained graph; "HW" stands for Hallway, "MR" for Meeting Room, "R" for Room, and "BT" for Bathroom

A room in the location hierarchy is characterized by $\langle room\_id, room\_type, Adj\_room\_list, \mathtt{L}_r, \mathtt{A}_r \rangle$, where $room\_type$ describes whether this unit is a room, a meeting room, a hallway, etc., Adj_room_list denotes the list of identifiers of the adjacent units, and $\mathtt{L}_r$, $\mathtt{A}_r$ are introduced in a similar way as in the fine-grained level. Such a location hierarchy can be directly derived from the fine-grained layer, but can also be generated from the exit hierarchy since information about the belonging of exits to their respective structural units is stored. A staircase that connects a given floor to another is represented as a room that belongs to the two corresponding floors, and which is bounded by two floor exits. On the other hand, an elevator is represented in a similar way to stairs. A multi-floor elevator consists of several stages that correspond to the number of floors of the building. Each stage of the elevator is modelled as a room that belongs to the two corresponding floors and bounded by exits/entrances to/from the corresponding floors.

From the fine-grained graph, a typical clustering process results in an abstract layer as illustrated in Fig. 4. Graph partitioning is thus carried out based on the set of room labels associated to the nodes of the base graph. Consequently, this process consists of: (1) extracting and aggregating nodes whose room labels are identical to form the new abstract nodes of the location hierarchy; and (2) creating abstract edges between connected structural units, thus favouring topology-based queries. These steps are as follows:

- ***Step 1.*** Based on the set of room labels, the fine-grained graph is partitioned into subgraphs. Let $\varphi = \bigcup_{i=1\ldots|\Sigma_{room}|} \varphi_{\ell_i}$ be the set of subgraphs of S_{micro} such that $\ell_i \in \Sigma_{room}$, and where $\forall i \in \{1, \ldots, |\Sigma_{room}|\}$, $\varphi_{\ell_i} = (\mathtt{V}_{\ell_i}, \mathtt{E}_{\ell_i}) \subset S_{micro}$ is a subgraph extracted from the fine-grained graph according to node and edge labels, and where $\bigcap_{\ell_i \in \Sigma_{room}} \mathtt{V}_{\ell_i} = \emptyset$. An abstract node that represents each subgraph is then created, having ℓ_i as its *local-id*.
- ***Step 2.*** The set of outgoing edges between connected subgraphs is defined by: $\mathtt{E}_{\ell_i,\ell_j} = (\varphi_{\ell_i}, \varphi_{\ell_j})$ $\forall i, j \in \{1, \ldots, |\Sigma_{room}|\}$, $i \neq j$. It should be noted that, for geometric-based queries (e.g., navigation, range, and nearest neighbour queries), the exit hierarchy is more likely considered, as it lends itself to more accurate and more realistic pre-processing techniques. On the other hand, the location hierarchy is more suitable for topology-based queries (e.g., connectivity, adjacency, etc.) or when one looks for the optimal path that contains the smaller number of rooms. Therefore, there is no need to associate precomputed network distances to each edge in the location hierarchy.

Similarly, there exists a relationship between exit and location hierarchies since exits belong to structural units. For instance, by retrieving the list of room labels associated to all exits, one can derive connected rooms and rebuild the corresponding location hierarchy. Accordingly, switching between a location hierarchy and an exit hierarchy is always possible, thus covering a larger range of queries (Fig. 5). Three spatial layers at two levels of abstraction (i.e., the fine-grained layer at the first level, and the exit and location hierarchies at the second level) are employed and used in this work. However, the data model can be generalized to introduce higher levels of abstraction in order to cover a wider range of applications, and with more flexibility.

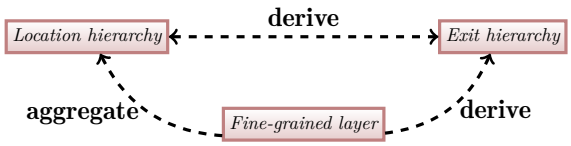


Figure 5. Links between neighbouring layers of the hierarchical data model

2.2. Feature Component

This section firstly presents the feature component principles, and secondly highlights the management of user profiles and real-time events.

2.2.1. Principles

A feature component $\mathcal{F}$ models persons and objects of interest in an indoor space. These features are either attached to the infrastructure (e.g., static objects like tables, doors, walls, fixed sensors, etc.) or dynamic, that is, they evolve in the environment (e.g., mobile users, continuous phenomena). An *entity* may be static or dynamic and is modelled as an object. An *object* is identified and characterized by *static properties* (i.e., attributes) and potentially other *dynamic properties* such as the *interaction spaces* attached to it [10]. In addition, an object can perform a selected list of *actions* that can be triggered according to some contextual constraints which are application-dependent.

Formally, a *feature* $\mathtt{f} \in \mathcal{F} = \bigcup(\mathcal{P}, \mathcal{OOI})$ is defined by a tuple $\langle \mathtt{Id}, \mathtt{Ct}, \mathtt{S}, \mathtt{FD}, \mathtt{A}, \mathtt{IS} \rangle$, where: $\mathtt{Id}$ is the feature identifier, $\mathtt{Ct}$ denotes the feature class type, $\mathtt{S}$ is the set of states a feature can hold, $\mathtt{FD}$ is the set of values that describe $\mathtt{f}$ (i.e., typically, a set of *string* values), $\mathtt{A} \subset \mathcal{A}$ is the list of context-dependent actions associated with $\mathtt{f}$, and $\mathtt{IS}$ is the list of interaction spaces associated with the feature [2,10] (explained below). The sets of states and actions available for a given feature are specified depending on the feature class type. A feature class type $\mathtt{Ct}$ is associated with a pair $\langle \mathtt{S}, \mathtt{A} \rangle$ where $\mathtt{S} = \bigcup_{i=1\ldots|\mathtt{S}|} \mathtt{s}_i$ and $\mathtt{A}$ denote the set of states and actions, respectively. As an example, a user $\mathtt{u} \in \mathcal{P}$ may have some static descriptions about the user profile and some predefined preferences. Besides, $\mathtt{u}$ can execute an action $a \in \mathtt{A}_u$ at an instant t when, for instance, he/she is in state $s \in \mathtt{S}$ and located on a node v. In contrast, an object of interest can be characterized by some qualitative and quantitative descriptions (e.g., its spatial extent), and boolean parameters that determine whether the object is able to communicate or not, whether it is mobile or static, physical or virtual, and attractive or repellent.

Moreover, as indicated above, each feature is associated with special dynamic properties referred to as *interaction spaces* (firstly introduced in [10] and extended in a previous work [2]) that cover some semantic information used for interaction purposes. The component $\mathtt{IS}$ is a quadruple $\langle \mathtt{ps}, \mathtt{os}, \mathtt{fs}, \mathtt{rs} \rangle$ that refers to the *physical*, *operational*, *functional*, and *range* space. At the fine-grained level, the interaction spaces are formally defined as sets of nodes dynamically updated in real-time (see Fig. 6):

- The *physical space* is represented by the set of nodes covered by the feature at a given time instant. For a mobile user, the physical space corresponds to the node where the user is currently located.
- The *functional space* denotes the nodes on which another feature can physically interact with the considered feature.
- The *range space* is a specific parameter only assigned to sensor objects and designates the set of nodes covered by the sensor (i.e., detectable nodes).

Furthermore, the notion of *operational space* has been introduced to cover all features of the space (static and/or moving objects). However, the definition of the operational space varies significantly depending on whether this feature is a (pseudo)-

static[6] or a mobile object. The difference between the two definitions is emphasized as follows:

- The operational space of a (pseudo-)static object can be represented by the union of all the potential nodes and edges an object may cover when it performs an action in the environment. For example, the operational space of a window comprises all the potential nodes this window may cover when opening and closing.
- The operational space of a mobile user denotes the set of nodes accessible to the user at a given time instant. The operational space of a mobile user strongly depends on the contextual information gathered. For example, the user profile directly affects the operational space according to whether the user is a security guard, a firefighter, a user with special needs, a normal user, etc. Time is another important dimension that might have an impact when visiting a shopping centre or entering a laboratory building (i.e., if the current time is in the morning, at night, during the weekend, etc.). Continuous phenomena such as a gas leak or a fire that breaks out inside a building may also have a significant impact on the operational space of the user.

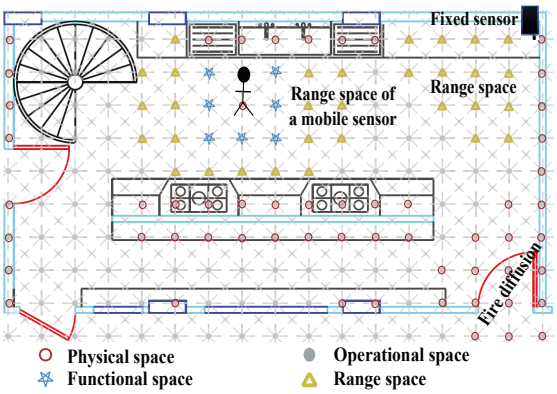


Figure 6. Interaction spaces of features evolving or located in space

It is worth noting that functional, range, and operational spaces are computed based on a user request, and are considered as specific continuous location-dependent queries. In particular, the range space of a given mobile sensor is considered as a typical range query, by taking into account the sensor range as the maximum threshold needed by the query. Moreover, a continuous evaluation of the operational space for a given user requires to retrieve all the spatial units accessible to this user at a given instant; this is typically done in the case of reachability queries (presented in Section 3.3.4).

2.2.2. User Profiles

One assumption of this approach is that the user model, which encapsulates knowledge about the users' personal data and preferences, allows classifying users into groups according to their privileges to access restricted areas. The aim of this classification is to derive an adaptive representation of space based on access control information associated with the user. This filtering process allows to derive *adaptive graphs* from the generic *base graph* by eliminating the set of nodes that are actually inaccessible for a specific type of user, thus reducing the amount of data that need to be processed in real-time for each query and supporting the retrieval of more accurate answers based on user profiles. A similar process takes place to update the time-dependent accessibility of some nodes, for instance, abstract nodes corresponding to rooms that are closed at specific times.

Users are therefore classified into three main categories: (i) unrestricted user, (ii) restricted user, and (iii) user with special needs. Additional types could be obtained by referring to these basic categories and incorporating further restrictions, thus yielding different configurations (see Fig. 7).

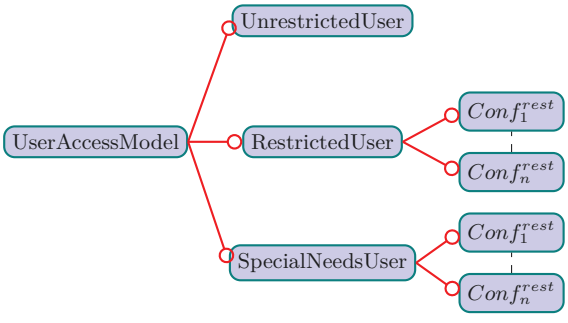


Figure 7. Classification of user profiles based on access control information

An *unrestricted user* has full access privileges and so he/she can navigate the complete map of

[6] A door or a window is an example of a pseudo-static object, since it can either open or close (as illustrated in Fig. 8), but it cannot move elsewhere.

the building, that is, the generic graph representing all the floors of the building. A example of an unrestricted user is a firefighter or a security guard, who should have a complete knowledge of all the emergency exits in a building. A *restricted user* category includes *staff members*, *guests*, and *visitors*. Usually, staff users have premium member privileges, but with some restricted areas, and can also have different configurations, whereas visitors have access to all the public places in a building. *Users with special needs* follow the same rules as restricted users except that additional penalties might be added to edge weights so that the most appropriate routes can be selected (e.g., using the elevator instead of the stairs for wheelchair users). Access control information is subject to very few changes over time, and thus it can be processed statically.

When there are no clearly defined privileges for certain areas inferred from a given user model, which may be due to unavailable information or lack of attention when deploying a context-aware system, the closest upper-level category of user profiles (from the ones discussed above) is considered. This upper-level category is less restrictive. Therefore, there might be some inaccessible areas whose corresponding data could be considered for query processing. Consequently, the system may generate some answers which are not well adapted to a specific user, such as a route that passes through a restricted area.

2.2.3. Real-Time Event Management

The occurrence of real-time events may have a significant impact on the nodes accessibility. For example, when considering a fire that is spreading in the Computer Science Department, fire alarms are expected to detect this event and communicate it to the system. With periodical updates performed automatically, the system is capable of representing the growing spatial extent of the fire, thus marking nodes of that physical space as inaccessible to users. This subset of nodes will be temporarily blocked when computing the operational space of the users. Instead, other subsets of nodes which correspond to emergency exit routes will be favoured to build the new operational space.

The distinction between attractive and repellent events is embodied by associating negative or positive impedance values to edge weights, respectively. Therefore, unscheduled or unexpected events are characterized by a triple: *event* ⟨`info_source`, `event_ps`, ±`value`⟩[7]. Common sources of information about events (`info_source`) include the system supervisor (if any), users and the social entities situated in the environment, and the communicating sensors. Their main task is to gather information about changes in the environment and to communicate that information to the system. The physical space of the event (`event_ps`) should also be determined in real-time in order to change edge weights as well node states accordingly. Finally, depending on the nature of each event, a positive or negative value (±`value`) is assigned to edge weights, so that adapted paths can be recomputed. Therefore, algorithms for handling continuous location-dependent queries are adapted in order to deal with these dynamic factors and with the information uncertainty.

2.3. Action Component

The action component $\mathcal{A} = \bigcup(\mathcal{FA}, \mathcal{SA})$ models the set of actions that are either triggered automatically by the system ($\mathcal{SA}$) or performed by a given feature acting in the environment ($\mathcal{FA}$). System actions ($\mathcal{SA}$) denote context-aware notifications that are mainly triggered based on users' locations and implement a publish/subscribe approach; this means that events are published by service providers to address their subscribers. This also includes *geocast* messaging [6,37], which can be described as a location-based multicast where messages are delivered to users located in a specific area instead of those subscribed to a given group.

Feature actions ($\mathcal{FA}$) encompass static and moving object actions, as well as continuous phenomenon actions (see Fig. 8). When considering objects, actions specify whether and how objects of a given type change their states in order to behave in a certain way. For instance, objects can adapt their behaviour and properties according to some contextual changes in the environment. This model implicitly builds semantic and topological relations among the features situated in space, by establishing relations between interaction spaces of different features.

[7] Temporal events are, on the other hand, regularly evaluated by means of the time-dependent functions previously described, and so they do not belong to this category of evens.

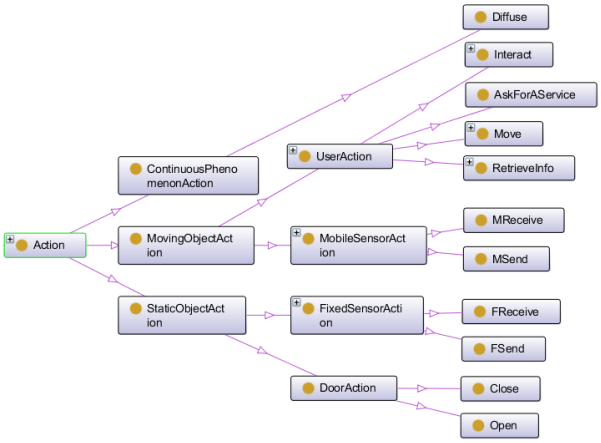


Figure 8. A set of actions performed by different types of features

Actions are context-dependent; this means that, at a given time instant and for a certain feature, only a specific list of possible actions is valid, which can then be performed according to some execution constraints. For a mobile user, actions comprise a sequence of movements, interactions with other neighbouring entities and artefacts, and requests for some services in order to achieve a predefined goal (Fig. 8). This approach allows to represent artefacts of interest located in the environment, so that users who are engaged in a certain activity can gather knowledge and understand their physical surroundings, as well as reconfigure and manipulate physical artefacts (e.g., a chair, a door, a heating, etc.) or virtual artefacts (e.g., a 2D/3D image of a physical artefact, a digital user interface, some recommendation/information, etc.) in order to produce changes in the environment. Moreover, a user can communicate with any fixed or mobile sensor located in the range space of a (mobile) sensor integrated in his/her device or attached to him/her (e.g., a MEMS sensor, an RFID tag, etc.). When considering continuous phenomena, their actions can materialize the way a given phenomenon diffuses in space.

The following sections will more closely consider typical user requests and services by integrating this modelling approach in the general system architecture for query processing.

3. Continuous Query Processing Architecture and Motivating Queries

A reasonable assumption of the approach, typically adopted in other related works, is that moving objects cooperate with the system by providing up-to-date location data (and possibly other information) when needed. Thus, a minimum intervention of a user device is required for query processing by communicating the location of the user to the system according to a certain location update policy [51]. It is worth noting that no constraints are imposed on the movements and directions of the *reference* and *target objects* (following the terminology used in [26]). Accordingly, the object that represents the reference for a location-dependent constraint (i.e., the *reference* object) is assumed to be either in a static location or moving freely in space. Similarly, a location-dependent query can request information about static or dynamic data, depending on whether the *target objects* (objects of interest to the query) are moving or not. An additional challenging issue is how to deal with dynamically changing edge weights, as described earlier. Therefore, a unique combination of challenges arises, as the proposed architecture must be able to continuously process different kinds of location-dependent queries, and to take into account additional context information, such as time-dependency and user profiles, as well as the hierarchical layout of the indoor environment.

This section presents the general architecture proposed for the continuous processing of several types of location-dependent queries in indoor environments. First, the main elements and components considered in the architecture are introduced, and then some distribution issues are discussed. Secondly, a query language, which will be used to express location-dependent queries, is presented. Finally, some motivating sample queries are illustrated.

3.1. Architecture Principles

The general query processing flow is illustrated in Fig. 9. Navigation-related queries are processed in accordance with this flow, and are executed continuously while the request is not explicitly cancelled by the user. Unlike many query processing approaches that focus on specific types of que-

ries and specific scenarios, this architecture has the advantage of supporting many different types of queries without making any restrictive assumption. The features that are managed in the environment are: (i) *mobile persons*, each of them carrying a mobile device that allows computing their current location and communicating with other entities, and (ii) *objects of interest*, which contribute to enrich the context of the query and are used by the user to provide his/her preferences and constraints (e.g., by using a digital user interface). These features are managed by a set of fixed servers, each of them in charge of: (1) maintaining a part of the hierarchical spatial graph that represents the environment (i.e., a part of the graph covering a certain spatial area); (2) managing data and communicating with objects located within its area; and (3) executing queries or parts of queries whose data are locally available.

3.1.1. Architecture Overview

The main phases of the query processing architecture are illustrated in Fig. 9 and can be described as follows:

Phase 1 and 2. A user interacts with the system interface to issue a query. The system transforms the query expressed in a natural or high-level language into an SQL-like format, as proposed in [26]. We assume that an expert user can also directly issue an SQL-like query based on the syntax described in Section 3.2.

Phase 3. Parsing a query implies lexical, syntactic, and semantic analysis of the query expressed in an SQL-like format in order to derive a valid internal representation (e.g., a query graph [28]).

Phase 4. A query plan is prepared that is composed of all the operations that are needed to appropriately answer the user request. This not only includes typical relational operations (e.g., selection, projection, join, etc.), but also external calls to specific functions that implement new query operators that are defined and discussed in the next section. For optimization purposes, some typical transformations can take place, such as the removal of redundant predicates, the simplification of complex expressions, etc. In addition, for each constraint in the query, the reference object and its target classes are obtained. Furthermore, information regarding the *location granules* (defined in [25] and discussed in the next section) of the reference and target objects is retrieved, if the use of location granules is specified in the query.

Phase 5. All navigation-related queries that need to expand routes either towards a specified target object (e.g., an optimal path search towards a destination) or in all directions with a maximum threshold (in the case of range queries), are directed to the route manager in charge of determining the candidate routes based on user-defined preferences and context data (e.g., information about user profiles and descriptions of objects of interest). The main tasks performed by the route manager are explained in Section 3.1.2.

Phase 6. Obtaining *standard SQL queries* from an SQL-like query is needed, since data elements are assumed to be stored in relational databases, which only accept standard SQL. A location-dependent query is broken up into standard queries and operations that are organised in an execution plan to optimise system resources. Not all the operators are necessarily translated to equivalent standard queries; for instance, operators related to route computation are directly handled at the algorithmic level. The candidate routes obtained by applying such operators could, however, be used as the input data to complete the construction of some standard queries.

Phase 7. In this phase, candidate routes along with an execution plan including standard queries and operations arrive at the *query execution engine*. Timestamped data about locations of relevant objects as well as other context data are associated with operations, so that the query engine can execute these queries appropriately. The continuous processing of a query means that the execution of simple queries and operations is kept alive until receiving an explicit request from the user to cancel that query. Therefore, the engine must repeatedly perform the following tasks: (1) update simple queries with the locations of relevant objects and with the new set of relevant routes, if needed; (2) execute standard queries; (3) correlate the results of the different subqueries; and finally (4) present the answer to the user.

3.1.2. Route Management

Two main tasks are performed by the *route manager* in order to execute navigation-related queries:

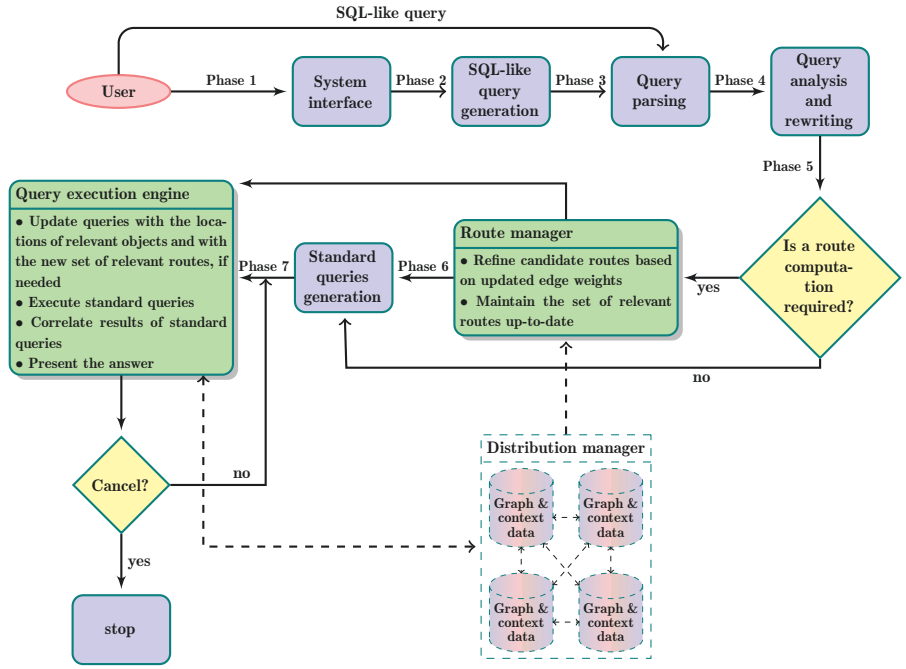


Figure 9. Execution of location-dependent queries in indoor environments

Task 1: *Obtaining an initial answer.* Depending on whether a target object is specified or not, different strategies are applied. A specified target implies expanding a directed tree routed at the node where the reference object is located, and oriented towards the target object. For a static shortest path problem, this can typically be solved using Dijkstra's or A*'s algorithm [18,22]. A more complex and challenging scenario for estimating the route cost and computing the optimal path arises when considering parameters such as dynamic edge weights, a hierarchical graph structure and, most importantly, the need for an incremental approach for continuous path search with moving reference and target objects.

In a range query, a maximum threshold or a radius is specified instead of a target object. Therefore, all the qualifying objects located within this radius are retrieved. A slightly different strategy consists of expanding all the routes whose network distance from the source node is less than or equal to the specified radius. Once again, this typical problem becomes significantly more complex when incorporating the aforementioned elements.

We are currently evaluating extensions of classical path search algorithms that can appropriately deal with all these challenges [4].

Task 2: *Maintaining the answer up-to-date.* Incremental search algorithms are required in order to execute continuous location-dependent queries, without having to solve each search problem independently from scratch [45,57]. Incremental search implies reusing information from previous searches in order to obtain the current result adaptively. In the case of navigation and range queries, a route planner needs to maintain the set of relevant routes up-to-date, especially when dealing with moving objects. For a navigation query, this means transforming the search tree to an updated tree depending on the movements of objects and other changes in the environment. In a range query, this implies either expanding new sub-trees from boundary nodes (i.e., leaves) or eliminating some of them if the new network distance exceeds the specified threshold [4].

3.1.3. Distribution Management

Another component that contributes to the process of refining candidate routes and to the execution of queries is the *distribution manager*. The architecture for processing continuous location-dependent queries over a large space should consider scalability and performance requirements. When considering a large indoor space, a decentralised approach should therefore be proposed to

alleviate performance problems when answering continuous queries and managing the corresponding data about moving objects [21,26]. This approach makes no assumptions about either the number (one or many) of computers deployed in the environment or the geographic area that should be managed by each computer. Therefore, decisions about data distribution management can be taken at the application level.

Let us consider the sample scenario described in Section 2.1. This three-storey building could be managed by deploying three servers, one on each floor, so that a hierarchical graph is created for each floor and stored in the corresponding server[8]. Consequently, (sub)queries and operations whose data are locally available are computed independently and results are communicated to other servers, if needed, or to the user if no other dependencies exist. Therefore, two challenging tasks must be performed by the distribution manager in order to support a distributed query processing:

1. *Keeping track of the relevant servers required to execute a given continuous query.* One can observe that the set of relevant servers changes depending on the locations of the reference and target objects. In the scenario mentioned above, a user from the Linguistics Department located on the first floor may want to reach his/her colleague currently located on the second floor. This scenario needs the first and second servers to be involved in the path planning query. If the target colleague moves down to the ground floor, the distribution manager must detect that the target leaves the area that is currently being watched and then, based on the new location of the target, the third server should be added to the set of relevant servers.
Therefore, each server is considered to be in charge of keeping (and providing) information about moving objects located within a fragment of the hierarchical data model. Then, for a given query in an indoor environment, a path or a set of paths, that can change dynamically along time, is computed. In this particular scenario, a path between the reference object (i.e., the query issuer in this example) and the target colleague is computed. On the contrary, in the case of a range query (which requests the objects that satisfy the specified constraints and are located within a certain radius from the reference object), the set of all the potential paths within the radius specified is computed. From this set of paths, the set of relevant servers for the query is derived. Therefore, queries that refer to a specific spatial area only require the service of a small subset of servers.
This optimisation phase has more important effects in large spaces (e.g., a campus with several buildings or in scenarios with a high number of moving objects), where using only the relevant servers can significantly reduce the query processing overhead. It should be noted that the information about the hierarchical graph itself is also distributed[9]. In that case, some nodes in the subgraph stored in a server actually represent entry nodes in another subgraph cached in a different server. Such nodes store the information needed to contact the other server.
2. *Computing the answer in a distributed environment.* Once the relevant servers have been identified, each server is queried in parallel to retrieve the objects located within the relevant fragments of the graph. This query has to be executed as a continuous query, as the relevant objects may move continuously. Moreover, objects' movements and changes in the environment can lead to changes in the set of relevant paths, which may in turn modify the set of relevant servers. Therefore, the query processing is assumed to be executed according to a certain *refreshment period* (as in other works, such as [26]), since the answer can change continuously.

Thanks to the distribution management, this architecture is generic and can be easily adapted to meet the requirements of a specific scenario. It works in small scenarios where a single server is enough as well as in scenarios that require a higher number of servers.

3.2. A Language for Location-Dependent Queries in Indoor Environments

As previously mentioned, several types of queries, such as navigation, range, and nearest neigh-

[8] Nodes corresponding to a staircase between two floors could be assigned to the server of either of such floors.

[9] Nevertheless, it could also be stored in a centralised manner, as the proposal is general enough to support any scheme.

bour queries, are of interest in an indoor environment. In order to improve query expressiveness, a query grammar is introduced to present those queries. This grammar is illustrated in Fig. 10, which has been extended from a previous work [25] to support navigation queries (of key importance in context-aware indoor navigation systems), and to incorporate some other preferences and semantics in the query model. For example, this grammar includes operators (e.g., *All-routes*) and constraints (e.g., *Stop-vertices*) used for navigation queries and inspired by [11], where the authors have introduced an approach for query processing in multimodal transportation systems based on the definition of a query language that provides users with the ability to choose between different modes of transportation and define spatio-temporal restrictions and preferences on the resulting path. In the grammar presented in Fig. 10, non-terminals start in upper-case and literals are in italics (reserved words) or in lower-case. The following description of the query language highlights the main elements involved in the definition of the queries.

In the general structure of the query language, two kinds of queries are identified: the former typically represents an SQL standard structure (*Standard-query*) along with specific kinds of location-dependent constraints, which are mainly used to express range and nearest neighbour queries. The latter (*Navigation-query*) represents navigation queries that incorporate route computation into the query processing while optimising distance/time criteria. For a navigation query, the ⟨ FROM clause ⟩ contains an external call to the *All-routes* operator, which has a general syntax as follows: *All-routes(Loc-Ref, Loc-Target)*. This operator returns a non-materialised set of tuples representing valid routes between the current locations of the reference and target objects. A route is a sequence of nodes and edges that can belong to different levels of granularity, which is determined by taking into account the context-dependent data integrated into the hierarchical data model. As a non-materialised table, the set of tuples (routes) obtained as a result of this operator are generated at runtime, and used only to execute the corresponding query. Each generated route is defined as: *route (route-id, source-vertex-id, target-vertex-id, ⟨v1-id, e1-id, v2-id, e2-id, ..., vk-id⟩, length, time)*, where route-id is a route identifier automatically assigned by the system when the route is computed. The *Loc-Ref* and *Loc-Target* arguments may correspond to either a "Vertex-id" or to the locations of the reference and target objects, respectively, but they can also be interpreted as the location granules that contain the corresponding objects, as discussed below. Moreover, *Loc-Target* can refer to the class name of objects of interest (target objects); this is used, for instance, in an inside constraint to retrieve all the objects of a given type.

The ⟨ WITH Stop-vertices ⟩ clause is an optional statement that expresses a user preference implying that the route must go through some place(s) that is(are) of interest to the user. Several *Stop-vertices* can be specified within a single query, and it is assumed that vertices are processed in the order they appear in the query. Furthermore, two different optimisation criteria are applied: *time* and *distance*, which can be considered based on the time-dependent functions defined in Section 2.1.1. In the standard structure of the query, two kinds of location-dependent conditions can be expressed in the ⟨ WHERE clause ⟩: *inside(Radius, Loc-Ref, Loc-Target)* and *nearest(K, Loc-Ref, Loc-Target)*. A constraint *inside* is applied when performing a continuous range query processing, which takes into account the radius as a maximum threshold to consider, and is used to build the set of paths (around the reference object), whose network distances are less than the radius. The *nearest* constraint is expressed to process continuous K nearest neighbour queries, by specifying the class name of objects of interest in Loc-Target, so that the K objects of interest that are the closest to the current location of the reference object are retrieved.

The concept of *location granule* proposed in [25] is used. A location granule identifies a set of fine-grained geographic locations (i.e., geometric coordinates of vertices in the base graph) under a common name. This is completely consistent with the hierarchical spatial graph proposed in Section 2. The use of location granules allows to formulate queries with a location resolution which is appropriate for the intended application. With them, it is possible to formulate queries using the location terminology required by the user (e.g., vertices at the fine-grained level, rooms, floors, buildings, etc.). For example, a user may be interested in persons that are near the room where another (moving) object is currently located (see Example

General query structure

Query	→	(Standard-query | Navigation-query)
Standard-query	→	*select* (Attr-Projections | '*') *from* Class-names (*where* Conds)?
Navigation-query	→	*select* (Attr-Projections | '*') *from* All-routes-expression (',' Class-names)* (*with* Stop-vertices)? (*where* Conds)? (*optimization-criteria*)?
Attr-Projections	→	Attr-Loc-Select (',' Attr-Loc-Select)*
Attr-Loc-Select	→	attribute | Loc-Select
attribute	→	Qualified-attr | Unqualified-attr
Qualified-attr	→	Class-name ':' Unqualified-attr
Loc-Select	→	Object-id ':' 'loc' | *gr* '('Map-id ',' Class-name ')' | *gr* '('Map-id ',' Route-id ')'
Class-names	→	Class-name (',' Class-name)*
All-routes-expression	→	*All-routes* '(' Loc-Ref ',' Loc-Target ')'
Loc-Ref	→	Object-id (':' 'coord')? | *gr* '('Map-id ',' Object-id ')' | *gr-map* '('Map-id ',' Gr-id ')' | Vertex-id
Loc-Target	→	Class-name | Object-id | Vertex-id ':' 'coord' | *gr* '(' Map-id ',' Class-name ')'
Stop-vertices	→	Stop-vertex (',' Stop-vertex)*
Stop-vertex	→	Vertex-id
optimization-criteria	→	(*minimize* | *maximize*) Measure
Measure	→	*time* | *distance*

Conditions can be standard conditions on attributes or location-dependent conditions

Conds	→	Cond ((*and* | *or*) Cond)*
Cond	→	(Bool-Cond | LDQ-Cond)
Bool-Cond	→	attribute Comp Value | *intersect* '(' Vertex-set ',' Vertex-set ')' | Value IN Vertex-id ':' POI

Location-dependent conditions

LDQ-Cond	→	*inside* '(' Args-Inside ')' | *nearest* '(' Args-Nearest ')' | ...
Args-Inside	→	Radius ',' Loc-Ref ',' Loc-Target
Args-Nearest	→	K ',' Loc-Ref ',' Loc-Target
Radius	→	Real Units

Basic grammar productions

String	→	([a-z] | [A-Z] | [0-9])+
Real	→	([0-9]+) (':' [0-9]+)?
K	→	[1-9] [0-9]*
Class-name	→	" String " /* Name of a class of objects */
Unqualified-attr	→	" String " /* Name of a class attribute */
Object-id	→	" String " /* Identifier of an object */
Map-id	→	" String " /* Identifier of a granule map */
Gr-id	→	" String " /* Identifier of a granule */
Vertex-id	→	" String " /* Identifier of a vertex */
coord	→	'(' Real ',' Real ')' /* Two dimensions are assumed */
Units	→	*meters* | *kilometres* | ...
Comp	→	'=' | '>' | '<' | '>=' | '<=' | '<>'
Value	→	([0-9]+) | " String "
POI	→	" String " /* A point of interest */

Figure 10. Query grammar for location-dependent queries in an indoor environment

1 in Section 3.3.2). In such a case, the location granule is set to the room level. The operator *gr* is a shorthand for *granule* and returns the location granule associated with a certain object according to a specified *granule map* (i.e., a named set of granules).

As depicted in Fig. 10, the location granule operator can be referenced in the SELECT clause, the FROM clause and/or the WHERE clause of a query, depending on whether the granules are used for the visualisation of the results and/or for the processing of constraints or routes. For visualisation purpose, a location granule operator can be used in a *Loc-Select* projection in the SELECT clause, according to the request submitted by the user, to show the result at the desired level of granularity; for example, SELECT gr('room-level', Person) can be used to project the rooms where the persons retrieved by the query are located. In addition, the *gr* operator can be applied on a route, which is the result of a navigation query, to show the sequence of nodes and edges obtained in the route at one chosen abstraction level. For instance, SELECT gr('room-level', Routes.id) could be used to illustrate the sequence of rooms of the valid route, which is made of nodes and edges of different levels (e.g., fine-grained and exit hierarchy levels). In this case, nodes and edges of the resulting route are abstracted to the room level, and the corresponding nodes of this chosen level are shown.

On the other hand, the same *gr* operator can be specified for processing-oriented uses as a *Loc-Ref* and/or *Loc-Target* argument within the FROM clause (i.e., in an *All-routes-expression*), and/or within the location-dependent query constraints (i.e., inside and nearest constraints), in reference to the locations of the reference and target objects, so that they can be interpreted as granules according to a given granule map (i.e., a given level of granularity). For instance, *inside(100 meters, gr('room-level', 'o1'), Person)* is a constraint satisfied by the persons within 100 meters around the room where object o1 is located (Example 1 in Section 3.3.2); similarly, *inside(100 meters, gr('room-level', 'room12'), Person)* is satisfied by the persons within 100 meters around room 12 (note that, in contrast to the previous example, the reference object here is not moving). On the contrary, *inside(100 meters, 'o1', Person)* would be used when the desired range is determined around object *o*1 itself; it should be clarified that *gr('micro-level', 'o1')* is equivalent to o1, that is, a fine-grained granule corresponding to the current fine-grained location of the object is considered by default when the *gr* operator is not explicitly expressed.

3.3. Location-Dependent Queries in Indoor Environments

This section presents typical examples of location-dependent queries. These examples follow the scenario introduced in Section 2.1. In particular, we consider navigation queries, range queries, nearest neighbour queries, and also other specific types of queries.

3.3.1. Navigation Queries

The continuous processing of navigation queries is based on a hierarchical path search that relies on a bottom-up technique with different levels of abstraction (i.e., fine-grained, room, floor, and building). The hierarchical path search starts from a user-specified level of granularity, depending on the location granule specified in the request and which contains the initial query point. The main steps of the process can be summarized as follows:[10].

1. Find the optimal path within the initial granule until reaching the nearest exit.
2. Search at the abstract level (exit hierarchy) for the optimal path from the exit of the initial granule to the granule containing the target object.
3. Find the optimal path within the last granule to the target object starting from the corresponding entrance of the granule.
4. Start a continuous path search by taking into account updated locations of reference and target objects (considering moving targets). This implies transforming an initial search tree rooted at the previous v_{start} to an updated tree rooted at the current v_{start}. The process continues either by expanding new sub-trees from the leaves towards the target and/or by removing sub-trees that are no longer needed.

[10]Steps 1 to 3 represent the first iteration that performs the hierarchical path search, while step 4 addresses the continuous processing of the navigation query.

Below are some typical examples of navigation queries:

1. A user identified by 'userID' wants to find the *fastest* path from his/her current location to the meeting room 'MR01' of the Computer Science Department[11] that goes through a break-room, showing the result at the *room* level:

```
SELECT gr('room-level', RO)
FROM Room AS R, Person AS P,
All-routes(gr('micro-level',P.id),R)
  AS RO
WITH Stop-vertices v1
WHERE R.id = 'MR01' AND P.id ='userID'
AND 'break-room' IN v1.POI
MINIMIZE time(RO)
```

where time(RO) = $time_{start \rightarrow goal}(t_{start})$ is the estimated time to traverse the path *RO* from 'userID' located at v_{start} to 'MR01'. As previously mentioned, the *gr* operator used in the SELECT statement returns an ordered set of nodes of the optimal route at the room level.

2. Find the *shortest* route from person 'userID1' to person 'userID2', showing the results at the *room* level:

```
SELECT gr('room-level', RO)
FROM Person AS P1, Person AS P2
All-routes(gr('micro-level', P1.id),
    gr('micro-level', P2.id)) AS RO
WHERE P1.id = 'userID1'
AND P2.id = 'userID2'
MINIMIZE length(RO)
```

where length(RO) = $length_{start \rightarrow goal}(t_{start})$ is the time-dependent network distance from 'p1' located at v_{start} to 'p2' located at v_{goal}.

3. Retrieve the time needed by all my colleagues of the Computer Science Department to reach the room where I am located:

```
SELECT MAX(t)
FROM Person As P,
(SELECT RO.time
FROM All-routes(gr('micro-level',P.id)
     , gr('room-level', 'myID')) AS RO
MINIMIZE time(RO)) AS t
WHERE 'C.S. Department member' IN P.FD
```

[11] MR01 is a unique identifier of the structural unit specified by the user and which belongs to the Computer Science Department.

A similar query could be "Retrieve the time needed to evacuate the building", which could be computed as the estimated time needed for the evacuation of the slowest person in the building.

3.3.2. Range Queries

Range queries are used to retrieve information about objects or places within a specified range or area. Some range queries have a static reference object and others have a moving reference object. Similarly, the target objects of the queries can be static or moving. The continuous processing of range queries consists in hierarchically expanding all routes whose *network distance* from the source node is less than or equal to the specified radius. A hierarchical network expansion is performed once for the first iteration so that all visited nodes that compose the range around the reference object are stored. To continuously process a range query, the set of parent nodes is maintained up-to-date when changing the root of the sub-tree (i.e., when the reference object moves). Boundary nodes are checked to decide, for each of them, whether to further expand that node or to perform a reverse search towards the source to remove nodes that are not relevant any more. Examples of such queries are:

1. Retrieve the identifiers of persons accessible at a network distance smaller 100 meters of the room where object $o1$ is located:

```
SELECT Person.id
FROM Person
WHERE inside(100 meters,
      gr('room-level', 'o1'), Person)
```

2. Retrieve all the communicating entities accessible at a network distance smaller than 100 meters of the user identified by 'userID' and with a communication range of at least 200 meters:

```
SELECT CO.id
FROM Object AS CO
WHERE inside(100 meters,
      gr('micro-level','userID'), CO)
AND CO.communicate = true
AND CO.commRange >= 200
```

3. Retrieve all the persons who belong to the Computer Science Department and that are accessible at a network distance smaller than 100 meters of the user identified by 'userID':

```
SELECT P.id
FROM Person AS P
WHERE inside(100 meters,
      gr('micro-level','userID'), P)
AND 'C.S. Department member' IN P.FD
```

In the previous query examples, the query issuer does not play the role of the reference object of the query, which shows the generality of the types of queries supported. On the other hand, the reference object can certainly be the query issuer himself/herself.

3.3.3. K Nearest Neighbour Queries

A (K) nearest neighbour query retrieves the (K) objects that meet certain specifications and which are the closest to a certain object or location. As in the case of other location-dependent queries, this kind of queries can also be issued by either a static or a dynamic reference object, and applied to either static or dynamic data. Let us show some examples:

1. Find the nearest available bathroom to the user identified by 'userID':

```
SELECT BR.id
FROM Bathroom AS BR
WHERE nearest(1, gr('micro-level',
      'userID'), BR)
AND BR.state = 'free'
```

2. Find the two nearest colour printers to each member of the C.S. department:

```
SELECT Pr.id, P.id
FROM Printer AS Pr, Person AS P
WHERE
nearest(2, gr('micro-level',P.id), Pr)
AND 'C.S. Department member' IN P.FD
AND Pr.type = 'ColourPrinter'
```

3. Retrieve the nearest extinguisher to each person located in buildings where a fire alarm is beeping:

```
SELECT P.id, Ex.id
FROM Sensor AS Ex,
(SELECT Person.id
FROM Person, Sensor AS FS
WHERE
inside(0,gr('building-level',FS.id),P)
AND FS.state = 'active'
AND FS.type = 'Fire-Sensor'
) AS P
WHERE
nearest(1,gr('micro-level',EX.id),P)
AND Ex.type = 'Extinguisher'
```

3.3.4. Reachability Queries and Reverse Range Queries

Finally, examples that show the relevance of the proposed query language and implicitly embed interaction spaces associated with each object are hereafter illustrated. Reachability queries check for places and/or objects that are reachable from the current position of the reference object. Indeed, those queries are implicitly processed as range queries by assigning a default threshold value θ to the inside constraint, which is large enough to determine whether the target object/place is reachable or not.

1. *Reachability queries:*

– Is the room where the object 'objID' is currently located accessible to the user identified by 'userID'? This request should check whether the room that contains 'objID' is within the range of 'userID' having a maximum threshold value set to θ:

```
SELECT Room.id
FROM Room
WHERE inside(θ meters,
gr('micro-level','userID'),
gr('room-level', Room))
AND inside(0 meters,Room.id,'ObjID')
```

– Retrieve all the rooms that are accessible to the user 'userID':

```
SELECT Room.id
FROM Room
WHERE inside(θ meters,
gr('micro-level', 'userID'),
gr('room-level', Room))
```

– Retrieve all the floors of the building that have at least one room accessible to the user 'userID':

```
SELECT DISTINCT gr('floor-level',
    Room)
FROM Room
WHERE inside(θ meters,
gr('micro-level', 'userID'),
gr('room-level', Room))
```

2. *Continuous reverse range queries.* Retrieve all the entities of type 'Sensor' that are covering the user 'userID' can be considered as an example of a *continuous reverse range query*, according to the definition provided in [54], since it continuously checks whether a moving object is inside the range of some sensor:

```
SELECT S.id
FROM Sensor AS S, Person AS P
WHERE inside(S.radius, S.id, P)
AND P.id = 'userID'
```

3.4. Discussion

The architecture presented provides a continuous query processing approach that can be applied on both static and moving objects, and proposes a generic execution flow for different kinds of location-dependent queries in indoor environments. The query language grammar supports navigation-related queries and incorporates other preferences and semantics into the query model. This language handles the granularity of moving objects' locations, thus favouring the hierarchical indoor data model previously presented.

Although the management of location granules during query processing introduces a certain overhead due to some extra geometric computations, this cost is limited and affordable. Indeed, the use of location granules together with incremental processing help reducing the communication overhead. Moreover, dealing with coarse location granules reduces the number of location updates that must be communicated to the mobile device. Similarly, efforts needed to keep track of the current positions of the reference and target objects are also smaller when coarse location granules are specified in the query constraints.

A stand-alone platform based on an extensible DBMS (Postgresql [35] with extensions: PostGIS [38]/Hermes [41], TelegraphCQ [13]) is currently under development. The main parts that are being developed in this platform are:

- The data model that represents the hierarchical network.
- The operators and location-dependent constraints introduced in Section 3.2. Those are designed as PL/pgSQL functions applied on user-defined types.
- The algorithms to process continuous location-dependent queries. A potential use of the TelegraphCQ extension is expected for handling continuous spatial streams as an input to the algorithms.

This implementation effort supports open source spatio-temporal databases and data stream management systems to handle continuous queries. Moreover, the hierarchical data modelling, i.e, abstract layers generation, is carried out by means of the supplied spatio-temporal functionality. Furthermore, the extensible query language allows for developing new data types, functions, and operators required to express the location-dependent queries presented in Section 3.2. An experimental evaluation of these algorithms as well as of the whole system with respect to the most related work in the literature will be performed in order to ensure scalability and efficiency of the proposed solutions.

4. Related Work

This section presents related work in three different areas: location-dependent query processing, context-aware indoor navigation, and query languages for location-dependent queries.

Location-dependent query processing

Most work on location-dependent query processing has been developed with an outdoor environment in mind (cf., [27] for a recent survey). However, indoor environments have brought special features and constraints that should be considered during query processing. As an example, the Euclidean distance is meaningless to compute routes in indoor spaces, due to path constraints. Therefore, approaches for query processing based on the network distance are preferred and more realistic. However, existing approaches for network-based query processing usually assume an outdoor environment (e.g., [17,30,40,44]), where for example hierarchical networks do not appear and there are no accessibility rules based on user profiles.

Nevertheless, recent works have studied location-dependent queries in indoor environments [55,56, 57]. Specific graph data models that represent the indoor space have been designed in these ap-

proaches, thus allowing a better processing of specific kinds of queries on top of the generated spatial network. In [57], the authors have introduced an approach to support range queries based on a virtual cell-based network generated for each query. Besides, an extension of this method has been proposed in the same paper to continuously process range queries whenever the query point moves. However, this approach is designed to address only one kind of queries, and is only applied to static data (i.e., static points of interest). Moreover, for each query, a new virtual network that connects the query point to the predetermined points of interest is required, and additional computations are also needed to update the network each time the query point leaves its *safe area*.

Other solutions for continuous range query processing as well as k nearest neighbour query processing over moving objects have been provided in [55] and [56], respectively. Both methods are developed on top of the same graph data model. The former deploys a set of sensors to continuously monitor users movements, thus maintaining the query result up-to-date, while the latter uses a probability estimation mechanism to prune unqualified candidates from the candidate set, so that the most probable k nearest neighbours are retrieved. The results show that the provided data model is flexible, since it allows for different kinds of queries to be performed, and the solutions on top of these foundations are efficient and scalable. However, the model underneath is based on sensor-range-based positioning techniques, which is not perfectly suitable for navigation queries that may require fine-grained location information. Moreover, other context dimensions such as the time and user profiles are not considered in the query processing. Furthermore, in the case of large indoor spaces, a generic architecture that allows distributing and managing data over several pieces of a database would still be required.

Context-aware indoor navigation

Graph-based data models support practical solutions to compute an optimal and realistic route to a destination by taking into consideration architectural constraints and dynamic changes in the environment [3]. A context-aware navigation task should comprise a next-step selection algorithm that keeps continuous track of the user's location and tries to adapt possible route deviations. Although very few works have discussed the integration of such a dynamic mechanism, several studies have incorporated dynamic changes into querying tasks by introducing time-dependent and length-dependent optimal routing in a spatial network [15,19]. Others have proposed algorithms for shortest and fastest path search with improved tracking strategies [8,45,53]. The main focus of these algorithms is to keep real-time tracking of moving objects. However, each of them deals with either time or distance constraints without incorporating other elements such as the user preferences or events that may significantly influence the answer. Hierarchical spatial data models have also been proposed to deal with performance and scalability issues when performing path searches over large graphs [12]. A *bottom-up* path finding approach [12], similar to the one presented in this paper, has shown to be up to 10-times faster for a 1% degradation in the path quality.

From a context-aware systems perspective, an indoor data model should support activity-oriented interactions by representing artefacts of interest located in the environment, as well as location-aware communication (e.g., communication between users and sensors, as suggested in [42]). Unfortunately, most of the existing data models are not designed for that purpose and thus do not support interactions with these artefacts and the tasks they might participate in. Indeed, the evaluation of mobile indoor navigation systems presented in [24] shows that most of the existing systems do not support context-awareness. There are only a few works that integrate some context dimensions other than location, especially the semantics behind the user profiles, and provide context-dependent adaptation according to these dimensions [29,33,48]. C-NGINE [29] supports an ontology-based modelling approach along with a rule-based reasoning technique to develop a navigation system adapted to the user's needs and preferences. The major shortcoming of such a semantic approach is the lack of geometric details about objects of interest and places represented in space. On the other hand, OntoNav [48] is based on a hybrid data model, which combines an indoor navigation ontology with a geospatial model (i.e., GIS layers representing a building blueprints), and a user model that helps processing path queries adapted to the user context. CoINS [33] is another

indoor navigation system that supports navigation queries. It integrates a hybrid (i.e., symbolic and geometric) spatial data model, as well as a user model with access permissions to enable adaptive pathfinding. Nevertheless, these three systems aim at providing a navigation service, and thus they do not support other location-dependent queries such as range and nearest neighbour queries. Moreover, none of these systems has proposed a general architecture for the continuous processing of location-dependent queries.

Query languages for location-dependent queries

The query language grammar presented in Section 3.2, has been extended from a previous work [25] and adapted to the context of indoor environments, by adding support for navigation queries and incorporating some other preferences and semantics in the query model. To the best of the authors' knowledge, no other work in the literature supports enhancing the expressiveness of location-dependent queries by considering the granularity of moving objects' locations. The approach presented in [25] covers the use of location granules from both a query processing as well as a result visualisation points of view. The results shown from using the location granules in query processing are considered very satisfactory. Indeed, the experiments show that the advantages of location granules do not come at the expense of performance. Moreover, the distributed approach increases the performance and scalability of the query processing [26].

Similar semantically enriched query languages for path planning in outdoor environments have been proposed in [11,36]. A query model for multimodal transportation systems has been presented in [11], which provides users with the ability to choose between different modes of transportation and applies spatio-temporal restrictions adapted to the user's preferences. As explained in Section 3.2, some concepts have been adopted from that query model. Another approach based on fuzzy logic theory that helps identifying ambiguous and possibly contradictory preferences have been proposed in [36]. However, the authors do not provide an architecture for continuous processing of location-dependent queries.

5. Conclusions and Future Work

The research introduced in this paper presents an approach to model continuous location-dependent queries in indoor environments. A generic architecture for continuous query processing has been introduced, along with a specific query language to enhance query expressiveness. This architecture proposes a generic execution flow applied to different kinds of location-dependent queries in indoor environments, and allows for managing data in a distributed manner. This architecture is built on top of a hierarchical and context-dependent data model, which leads to the consideration of other context dimensions besides the location of the involved entities, such as time and user profiles. The main advantage of the indoor data model relies on its hierarchical and context-dependent design, which allows adaptive and appropriate processing of location-dependent queries. The query language grammar allows to represent a variety of location-dependent queries and incorporates user preferences and other semantics into the query model. Moreover, the use of location granules in the query language significantly increases the query expressiveness, and is perfectly consistent with the hierarchical layout of the environment.

Future work is oriented towards: (1) taking advantage of this proposal to deal with the continuous processing behind the operators defined in Section 3.2 (i.e., the *All-routes* operator, and the *inside* and *nearest* constraints). Two algorithms for continuous processing of navigation and range queries on top of the modelling approach presented in this paper have already been developed, but experimental evaluation and comparison with related works are still under progress; (2) integrating an extended context model that incorporates users' activities as well as content generated by other social entities into location-dependent query processing; and (3) generalising the hierarchical data model to higher levels of abstraction (floor and building levels), thus building a nested-graph model similar to the Hypernode/Master-node data model described in [31] and [34], respectively.

Acknowledgements

This research was partially supported by a Short Term Scientific Mission performed by the first au-

thor at the University of Zaragoza and funded by the COST Action IC0903 on "Knowledge Discovery from Moving Objects" (MOVE project). We would also like to acknowledge the support of the CICYT project TIN2010-21387-C02-02 and DGA-FSE.

References

[1] G.D. Abowd and E.D. Mynatt. Charting past, present, and future research in ubiquitous computing. *ACM Transactions on Computer-Human Interaction*, 7(1):29–58, 2000.

[2] I. Afyouni, C. Ray, and C. Claramunt. A fine-grained context-dependent model for indoor spaces. In *Proceedings of the 2nd ACM SIGSPATIAL International Workshop on Indoor Spatial Awareness*, pages 33–38. ACM, 2010.

[3] I. Afyouni, C. Ray, and C. Claramunt. Spatial models for indoor and context-aware navigation systems: A survey. *Journal of Spatial Information Science*, 4(1):85–123, 2012.

[4] I. Afyouni, C. Ray, S. Ilarri, and C. Claramunt. Algorithms for continuous location-dependent and context-aware queries in indoor environments. In *Proceedings of the 20th ACM SIGSPATIAL International Conference on Advances in Geographic Information Systems, 10 pp., accepted.* ACM, 2012.

[5] M. Baldauf, S. Dustdar, and F. Rosenberg. A survey on context-aware systems. *International Journal of Ad Hoc and Ubiquitous Computing*, 2(4):263–277, 2007.

[6] S. Basagni, I. Chlamtac, and V.R. Syrotiuk. Geographic messaging in wireless ad hoc networks. In *Proceedings of the 49th Vehicular Technology Conference (VTC)*, volume 3, pages 1957–1961. IEEE, 1999.

[7] C. Becker and F. Durr. On location models for ubiquitous computing. *Personal and Ubiquitous Computing*, 9(1):20–31, 2005.

[8] A. Berger, M. Grimmer, and M. Müller-Hannemann. Fully dynamic speed-up techniques for multi-criteria shortest path searches in time-dependent networks. In *Proceedings of the 9th International Symposium on Experimental Algorithms (SEA)*, pages 35–46. Springer, 2010.

[9] C. Bettini, O. Brdiczka, K. Henricksen, J. Indulska, D. Nicklas, A. Ranganathan, and D. Riboni. A survey of context modelling and reasoning techniques. *Pervasive and Mobile Computing*, 6(2):161–180, 2009.

[10] M. Bhatt, F. Dylla, and J. Hois. Spatio-terminological inference for the design of ambient environments. In *Proceedings of the 9th International Conference on Spatial Information Theory (COSIT)*, pages 371–391. Springer, 2009.

[11] J. Booth, P. Sistla, O. Wolfson, and I.F. Cruz. A data model for trip planning in multimodal transportation systems. In *Proceedings of the 12th International Conference on Extending Database Technology (EDBT)*, pages 994–1005. ACM, 2009.

[12] A. Botea, M. Muller, and J. Schaeffer. Near optimal hierarchical path-finding. *Journal of Game Development*, 1(1):7–28, 2004.

[13] S. Chandrasekaran, O. Cooper, A. Deshpande, M.J. Franklin, J.M. Hellerstein, W. Hong, S. Krishnamurthy, S.R. Madden, F. Reiss, and M.A. Shah. Telegraphcq: Continuous dataflow processing. In *Proceedings of the 2003 ACM SIGMOD International Conference on Management of Data*, pages 668–668. ACM, 2003.

[14] G. Chen and D. Kotz. A survey of context-aware mobile computing research. Technical Report TR2000-381, Dartmouth College Hanover, NH, USA, 2000.

[15] D. Delling. Time-dependent SHARC-routing. *Algorithmica*, 60(1):60–94, 2011.

[16] T. Delot, S. Ilarri, N. Cenerario, and T. Hien. Event sharing in vehicular networks using geographic vectors and maps. *Mobile Information Systems*, 7(1):21–44, 2011.

[17] K. Deng, X. Zhou, H.T. Shen, S. Sadiq, and X. Li. Instance optimal query processing in spatial networks. *The VLDB Journal*, 18(3):675–693, 2009.

[18] E.W. Dijkstra. A note on two problems in connexion with graphs. *Numerische mathematik*, 1:269–271, 1959.

[19] B. Ding, J.X. Yu, and L. Qin. Finding time-dependent shortest paths over large graphs. In *Proceedings of the 11th International Conference on Extending Database Technology (EDBT)*, pages 205–216. ACM, 2008.

[20] H. Ferhatosmanoglu, I. Stanoi, D. Agrawal, and A. El Abbadi. Constrained nearest neighbor queries. In *Proceedings of the 7th International Symposium on Advances in Spatial and Temporal Databases (SSTD)*, pages 257–278. Springer, 2001.

[21] B. Gedik and L. Liu. MobiEyes: A distributed location monitoring service using moving location queries. *IEEE Transactions on Mobile Computing*, 5(10):1384–1402, 2006.

[22] P.E. Hart, N.J. Nilsson, and B. Raphael. A formal basis for the heuristic determination of minimum cost paths. *IEEE Transactions on Systems Science and Cybernetics*, 4(2):100–107, 1968.

[23] H. Hu and D.L. Lee. Semantic location modeling for location navigation in mobile environment. In *Proceeding of the IEEE International Conference on Mobile Data Management (MDM)*, pages 52–61. IEEE, 2004.

[24] H. Huang and G. Gartner. A survey of mobile indoor navigation systems. In *Proceedings of the 1st ICA Symposium on Cartography in Central and Eastern Europe*, pages 305–319. Springer, 2010.

[25] S. Ilarri, C. Bobed, and E. Mena. An approach to process continuous location-dependent queries on moving objects with support for location granules. *Journal of Systems and Software*, 84(8):1327–1350, 2011.

[26] S. Ilarri, E. Mena, and A. Illarramendi. Location-dependent queries in mobile contexts: Distributed processing using mobile agents. *IEEE Transactions on Mobile Computing*, 5(8):1029–1043, 2006.

[27] S. Ilarri, E. Mena, and A. Illarramendi. Location-dependent query processing: Where we are and where we are heading. *ACM Computing Surveys*, 42(3):1–73, 2010.

[28] D. Kossmann. The state of the art in distributed query processing. *ACM Computing Surveys*, 32(4):422–469, 2000.

[29] M. Kritsotakis, M. Michou, E. Nikoloudakis, A. Bikakis, T. Patkos, G. Antoniou, and D. Plexousakis. Design and implementation of a semantics-based contextual navigation guide for indoor environments. *Journal of Ambient Intelligence and Smart Environments*, 1(3):261–285, 2009.

[30] D.L. Lee, M. Zhu, and H. Hu. When location-based services meet databases. *Mobile Information Systems*, 1(2):81–90, 2005.

[31] M. Levene and G. Loizou. A graph-based data model and its ramifications. *IEEE Transactions on Knowledge and Data Engineering*, 7(5):809–823, 1995.

[32] X. Li, C. Claramunt, and C. Ray. A grid graph-based model for the analysis of 2D indoor spaces. *Computers, Environment and Urban Systems*, 34(6):532–540, 2010.

[33] F. Lyardet, D.W. Szeto, and E. Aitenbichler. Context-aware indoor navigation. In *Proceedings of the 2nd European Conference in Ambient Intelligence (AmI)*, pages 290–307. Springer, 2008.

[34] M. Mainguenaud. Modelling of the geographical information system network component. *International Journal of Geographical Information Systems*, 9(6):575–593, 1995.

[35] N. Matthew and R. Stones. *Beginning databases with postgreSQL: From novice to professional*. Apress, 2005.

[36] Amine Mokhtari. *Système personnalisé de planification d'itinéraire : Une approche basée sur la théorie des ensembles flous*. PhD thesis, Université de Rennes 1 - IRISA - France, 2011.

[37] J.C. Navas and T. Imielinski. GeoCast - Geographic addressing and routing. In *Proceedings of the 3rd Annual ACM/IEEE International Conference on Mobile Computing and Networking*, pages 66–76. ACM, 1997.

[38] R. Obe and L. Hsu. *PostGIS in Action*. Manning Publications Co., 2011.

[39] M.T. Özsu and P. Valduriez. *Principles of Distributed Database Systems, Third Edition*, chapter 1 "Introduction", pages 1–40. Springer, 2010.

[40] D. Papadias, J. Zhang, N. Mamoulis, and Y. Tao. Query processing in spatial network databases. In *Proceedings of the 29th International Conference on Very Large Data Bases (VLDB)*, pages 802–813. VLDB Endowment, 2003.

[41] N. Pelekis, Y. Theodoridis, S. Vosinakis, and T. Panayiotopoulos. Hermes–a framework for location-based data management. In *Proceedings of the 10th International Conference on Extending Database Technology (EDBT)*, pages 1130–1134. Springer, 2006.

[42] I. Satoh. A location model for smart environments. *Pervasive and Mobile Computing*, 3(2):158–179, 2007.

[43] J.H. Schiller and A. Voisard. *Location-Based Services*. Morgan Kaufmann, San Francisco, CA, USA, 2004.

[44] D. Stojanovic, A.N. Papadopoulos, B. Predic, S. Djordjevic-Kajan, and A. Nanopoulos. Continuous range monitoring of mobile objects in road networks. *Jounrnal of Data & Knowledge Engineering*, 64(1):77–100, 2008.

[45] X. Sun, W. Yeoh, and S. Koenig. Efficient incremental search for moving target search. In *Proceedings of the 21st International Joint Conference on Artifical Intelligence (IJCA)*, pages 615–620. Morgan Kaufmann, 2009.

[46] Y. Tao, D. Papadias, and Q. Shen. Continuous nearest neighbor search. In *Proceedings of the 28th International Conference on Very Large Data Bases (VLDB)*, pages 287–298. VLDB Endowment, 2002.

[47] D. Terry, D. Goldberg, D. Nichols, and B. Oki. Continuous queries over append-only databases. In *Proceedings of the ACM SIGMOD International Conference on Management of Data*, pages 321–330. ACM, 1992.

[48] V. Tsetsos, C. Anagnostopoulos, P. Kikiras, and S. Hadjiefthymiades. Semantically enriched navigation for indoor environments. *International Journal of Web and Grid Services*, 2(4):453–478, 2006.

[49] A.B. Waluyo, B. Srinivasan, and D. Taniar. Research in mobile database query optimization and processing. *Mobile Information Systems*, 1(4):225–252, 2005.

[50] M. Weiser. Some computer science issues in ubiquitous computing. *Communications of the ACM*, 36(7):75–84, 1993.

[51] O. Wolfson, A.P. Sistla, S. Chamberlain, and Y. Yesha. Updating and querying databases that track mobile units. *Distributed and parallel databases*, 7(3):257–387, 1999.

[52] K.L. Wu, S.K. Chen, and P.S. Yu. Incremental processing of continual range queries over moving objects. *IEEE Transactions on Knowledge and Data Engineering*, 18(11):1560–1575, 2006.

[53] M. Xu, Z. Pan, H. Lu, Y. Ye, P. Lv, and A. El Rhalibi. Moving-target pursuit algorithm using improved tracking strategy. *IEEE Transactions on Computational Intelligence and AI in Games*, 2(1):27–39, 2010.

[54] Z. Xu and A. Jacobsen. Adaptive location constraint processing. In *Proceedings of the ACM SIGMOD International Conference on Management of Data*, pages 581–592. ACM, 2007.

[55] B. Yang, H. Lu, and C.S. Jensen. Scalable continuous range monitoring of moving objects in symbolic indoor space. In *Proceeding of the 18th Conference on Information and Knowledge Management (ICIKM)*, pages 671–680. ACM, 2009.

[56] B. Yang, H. Lu, and C.S. Jensen. Probabilistic threshold k nearest neighbor queries over moving objects in symbolic indoor space. In *Proceedings of the 13th International Conference on Extending Database Technology (EDBT)*, pages 335–346. ACM, 2010.

[57] W. Yuan and M. Schneider. Supporting continuous range queries in indoor space. In *Proceedings of the 11th International Conference on Mobile Data Management (MDM)*, pages 209–214. IEEE, 2010.

[58] J. Zhang, M. Zhu, D. Papadias, Y. Tao, and D.L. Lee. Location-based spatial queries. In *Proceedings of the ACM SIGMOD International Conference on Management of Data*, pages 443–454. ACM, 2003.