Traitement continu des requêtes dépendantes de la localisation dans des environnements intérieurs

Imad Afyouni [1,3], **Cyril Ray** [1], **Christophe Claramunt** [1], **Sergio Ilarri** [2]

1. *Institut de recherche de l'École Navale, 29240 Brest Cedex 9, France*
   cyril.ray,christophe.claramunt@ecole-navale.fr
2. *University of Zaragoza, Maria de Luna 1, 50018 Zaragoza, Spain*
   silarri@unizar.es
3. *KACST GIS Technology Innovation Center, KSA*
   iafyouni@gistic.org

ABSTRACT. *This paper develops a data and knowledge representation of an indoor environment that takes into account user-centred contextual dimensions and mobile data management issues. We introduce a hierarchical, context-dependent, and feature-based indoor spatial data model in which location information is represented at different levels of abstraction. A query language supporting continuous location-dependent queries and taking into account user preferences at execution time is developed and implemented on top of that model. This modelling approach is complemented by implemented algorithms allowing path searches and range queries applied to both static and moving objects. Several experiments have been conducted to evaluate benefits of the hierarchical design as well as the scalability and performance of the whole framework.*

RÉSUMÉ. *Cet article développe une représentation de données spatiales d'un environnement intérieur dit "indoor" qui tient compte des dimensions contextuelles centrées sur l'utilisateur et aborde les enjeux de gestion de données mobiles. Un modèle de données "indoor" hiérarchique et sensible au contexte est proposé. Cette conception hiérarchique favorise un traitement adaptatif et efficace des requêtes dépendantes de la localisation. Un langage de requêtes continues est développé et illustré par des exemples de requêtes. Cette approche de modélisation est complétée par le développement d'algorithmes de traitement continu des requêtes de recherche de chemin hiérarchique et des requêtes de zone sur des objets mobiles en "indoor". Une étude expérimentale des solutions développées a été menée pour évaluer la performance et le passage à l'échelle à l'égard des propriétés intrinsèques des solutions proposées.*

KEYWORDS: *Indoor data models, context-aware systems, mobile data management, continuous location-dependent queries, moving objects.*

MOTS-CLÉS : *Modèles de données indoor, systèmes sensibles au contexte, gestion de données mobiles, requêtes continues dépendantes de la localisation, objets mobiles.*

1. Introduction

L'augmentation de la population mondiale, des besoins énergétiques et de leurs coûts, le changement des modes de vies entraînent d'une part un accroissement continu des mobilités, un allongement des distances et d'autre part une diversification des modes de déplacements. Cette évolution s'accompagne d'un besoin accru de connectivité. Durant leurs déplacements les individus, souhaitent de plus en plus être connectés à tout, tout le temps, quelque soit leur localisation. Ce paradigme défini au début des années quatre vingt dix (Weiser, 1991), souvent qualifié d'intelligence ambiante (AMI), est celui de la puissance de l'informatique ubiquitaire, qui avec une capillarité massive et une diversité de solutions de communication permet d'offrir des services durant la mobilité.

Parmi ces services, les *services géolocalisés* ont récemment fait l'objet de recherches approfondies. De tels services qui reposent essentiellement sur la localisation de l'utilisateur offre une palette applicative infinie et supposent un impact significatif pour les utilisateurs quel que soit leur environnement de mobilité (i.e., indoor et outdoor) (Schiller, Voisard, 2004; Krisp, 2013). De tels services offrent un accès personnalisé à l'information en tenant compte de la localisation des utilisateurs mobiles. Plus généralement, ces services tendent vers des systèmes *sensibles au contexte* qui utilisent les différents aspects du contexte telles que les dimensions centrées utilisateur (e.g., profil, capacités physiques et cognitives, etc.), le contexte environnemental (e.g., position, entités sociales, lumière, etc.), le contexte temporel et le contexte d'exécution (e.g., topologie du réseau, ressources à proximité, etc.), afin d'anticiper les besoins de l'utilisateur et de personnaliser son expérience (Bettini *et al.*, 2009; Petit, 2010).

Le principal objectif de la recherche présentée dans cet article est d'examiner la gestion et l'interrogation des données dans les systèmes de navigation sensibles au contexte implantés au sein de différents environnements "indoor" (e.g., des bâtiments, des centres commerciaux). Malgré le développement et les améliorations apportés dans l'informatique mobile et la variété des technologies qui peuvent être utilisées pour favoriser les environnements intérieurs ambiants, de nombreux défis de recherche restent posés. La représentation et la gestion de données spatiales et le traitement des requêtes dépendantes de la localisation sont parmi les enjeux actuels à aborder de sorte qu'un système de navigation sensible au contexte et suffisamment flexible puisse être conçu. Une intégration réussie des espaces "indoor" dans les systèmes sensibles au contexte requiert le développement de structures appropriées de données spatiales et d'un modèle spatial dynamique et flexible qui aidera à fournir des services appropriés aux utilisateurs mobiles se déplaçant dans un tel environnement.

Les requêtes dépendantes de la localisation (RDL) (Ilarri *et al.*, 2010; Zhang *et al.*, 2003) (en anglais "*location-dependent queries*") sont des exemples typiques de services "push-based" nécessaires pour ces systèmes sensibles au contexte. Dans ce type de requêtes, tout changement de position des objets qui sont impliquées dans la requête peut influencer de manière significative la réponse. Par exemple, si un utilisateur demande à connaître les positions de ses amis dans une zone de 100 mètres tout

en se déplaçant dans un centre commercial, la réponse dépendra à la fois de la position actuelle de l'utilisateur ainsi que des positions de ses amis les plus proches. Ce type de requête est particulièrement difficile car, dans la plupart des cas, les positions de l'utilisateur et des entités impliquées dans la requêtes évoluent dans le temps. Une étude exhaustive sur les requêtes RDL a été présentée dans (Ilarri *et al.*, 2010); certains types de requêtes particulièrement pertinents dans notre contexte sont brièvement décrits comme suit:

1. *Les requêtes de localisation* déterminent les positions des objets d'intérêt, et sont traitées selon un modèle géométrique (e.g., coordonnées cartésiennes ou latitude-longitude-altitude) ou symbolique (e.g., identifiant d'une pièce, d'un étage ou d'un bâtiment) de l'espace (Afyouni, Ray, Claramunt, 2012). Ce type de requête est essentiel puisque d'autres requêtes RDL ne peuvent pas être réalisées sans être alimentées par les positions actuelles des objets d'intérêt (Becker, Durr, 2005).

2. *Les requêtes de navigation* aident les utilisateurs à trouver et à atteindre des points ou objets d'intérêt en leur fournissant des informations de navigation tout en optimisant certains critères tels que la distance totale parcourue ou le temps de déplacement.

3. *Les requêtes de zone* (en anglais "*Range ou Window queries*") sont utilisées pour trouver et récupérer des informations sur des points/objets d'intérêt dans une zone spécifiée par l'utilisateur (Wu *et al.*, 2006).

4. *Les requêtes de recherche des K voisins les plus proches* ("*K-Nearest Neighbours (KNN) queries*") permettent de sélectionner les K voisins les plus proches de la position actuelle de l'objet mobile (Tao *et al.*, 2002). Contrairement aux requêtes de zone, les objets sélectionnés dans les requêtes KNN satisfont une certaine spécification (e.g., les deux imprimantes couleur les plus proches disponible dans le bâtiment).

La gestion efficace des données statiques et dynamiques est un enjeu clé du traitement des requêtes RDL, car le résultat d'une requête n'est valable que pour une position particulière de l'objet référence, et pour certaines positions des objets d'intérêt. Comme ces requêtes sont sensibles au temps et dépendantes de la localisation, elles doivent être traitées sous forme de *requêtes continues* (Terry *et al.*, 1992), ce qui signifie que le système doit maintenir les réponses à jour, jusqu'à ce que la requête soit explicitement annulée par l'utilisateur. Bien que de nombreuses études ont examiné les requêtes RDL, peu de travaux ont abordé l'intégration d'autres dimensions du contexte, particulièrement celles liées à l'*utilisateur* ainsi que le *contexte environnemental*, lors du traitement de la requête. En effet, cela peut complètement modifier la réponse à une requête, même si les positions des objets concernés n'ont pas changé.

Cet article étudie les systèmes sensibles au contexte ainsi que les requêtes RDL dans les environnements indoor, en accordant une attention particulière aux requêtes de navigation et celles d'intervalle ou de zone. Les différents aspects et composantes

fondamentales pour la réalisation d'un système de navigation indoor sensible au contexte sont introduits dans (Afyouni, Ray, Claramunt, 2012). Des recommandations pour la conception d'un modèle de données spatiales en "indoor" efficace et flexible sont présentées. Ces recommandations sont basées sur des critères *orientés service* ainsi que d'autres *liés à la performance*. Une combinaison unique de défis se pose, étant donné que l'approche doit être en mesure de représenter différents types de requêtes de manière flexible, et de prendre en compte des éléments de contexte supplémentaires, ainsi que la structure hiérarchique de l'environnement. La contribution de ce travail est quadruple : (1) un modèle de données spatiales hiérarchique et sensible au contexte qui représente un environnement "indoor"; (2) un langage de requête et une architecture générique utilisés pour l'expression et le traitement continu de requêtes dépendantes de la localisation dans les environnements intérieurs; (3) des algorithmes pour le traitement continu des *requêtes de recherche de chemin hiérarchique* et des *requêtes de zone* appliqués à des objets statiques et/ou en mouvement sont introduits; et enfin (4) un prototype encapsulant les différents algorithmes et méthodes pour la manipulation de ces requêtes en se basant sur le modèle de données mentionné.

La suite de cet article est organisée de la manière suivante. La section 2 présente le modèle de données spatiales, en mettant l'accent sur sa structure hiérarchique, et tout en affirmant son intérêt pour les services géolocalisés. La section 3 propose un langage de requête pour exprimer les requêtes RDL continues dans un environnement "indoor". Les algorithmes de traitement continu des requêtes de navigation et de zone sont introduits dans la section 4 . Une étude expérimentale des solutions développées est présentée dans la section 5. La section 6 conclut ce travail et expose les directions dans lesquelles nos travaux peuvent être poursuivis.

2. Approche de Modélisation

Les représentations spatiales "indoor" peuvent prendre différentes formes en fonction des besoins applicatifs. De tes modèles spatiaux en "indoor" ont été étudiés et développés dans de nombreux domaines : robotique mobile, systèmes d'information géographique, informatique ambiante) (Thrun, 2003; Becker, Durr, 2005; Nagel *et al.*, 2010). Dans (Afyouni, Ray, Claramunt, 2012), une taxonomie de ces modèles est présentée et évaluée en fonction de leur applicabilité pour la conception de systèmes de navigation en "indoor" prenant en compte les dimensions contextuelles. Les modèles de données symboliques orientés graphe fournissent des solutions pratiques pour calculer un itinéraire optimal et réaliste vers une destination en tenant compte des contraintes architecturales et des changements dynamiques dans l'environnement. Des modèles de graphes (réseaux spatiaux dynamiques sensibles au temps ont été récemment proposés dans (Delling, 2011; Ding *et al.*, 2008). Des modèles hiérarchiques de données spatiales ont également été présentés pour pallier aux problèmes de performance et de passage à l'échelle lors de l'exécution des recherches de chemin sur des graphes de grande taille. Les modèles de données spatiales hybrides favorisent les différents types d'applications à différents niveaux d'abstraction.

L'approche de modélisation proposée introduit une représentation hiérarchique de données spatiales dans laquelle des informations de localisation des objets mobiles peuvent être présentées à différents niveaux d'abstraction. Le principe sous-jacent est de considérer l'espace à différents niveaux de granularité. Ce modèle spatial hiérarchique est sensible au contexte et permet de pallier aux problèmes de performance et de passage à l'échelle lors du traitement des requêtes RDL. Ce modèle représente: (i) les entités situées ou évoluant dans l'environnement; une entité peut être une personne (i.e., un utilisateur mobile ou toute autre entité sociale[1]) ou un objet d'intérêt (e.g., capteurs, entrées/sorties, tables, phénomènes continus tels qu'un feu, etc.); (ii) leurs propriétés spatiales (e.g., position et emprise spatiale); et (iii) les comportements qui s'en dégagent (i.e., comment ces objets peuvent interagir et communiquer au sein de l'environnement). Ce modèle de données hiérarchique et sensible au contexte d'un environnement indoor se distingue par ses trois composantes complémentaires:

– La composante spatiale est constituée d'un ensemble de couches organisées hiérarchiquement et représentant l'espace indoor. Cette composante intègre aussi la gestion d'autres éléments du contexte, et plus particulièrement, le temps et les profils utilisateurs.

– La deuxième composante englobe les entités (i.e., personnes et objets d'intérêt) situées ou évoluant dans l'environnement.

– La composante des actions représente les actions qui sont soit prédéfinies et déclenchées automatiquement par le système d'information sous forme de messages informatifs contextuels, soit générées par des entités données agissant dans l'environnement.

Ces trois composantes sont examinées plus en détail dans les sections suivantes.

2.1. Composante Spatiale

Dans cette section, les propriétés de différentes couches de la composante spatiale sont décrites, ainsi que leurs utilités pour le traitement des requêtes RDL.

2.1.1. Couche Spatiale de Base

La couche de base du modèle est constituée d'un graphe sensible au contexte à un niveau micro G_{micro} = (V_{micro}, E_{micro},W_{length},W_{time}) généré à partir d'une grille qui couvre l'espace indoor (Fig. 1). Les noeuds du graphe représentent ainsi les cellules de la grille, et les connexions entre les cellules sont matérialisées par des arêtes.

Dans la définition de G_{micro}, V_{micro} = $\{v_i\}$ désigne l'ensemble de sommets et $E_{micro} \subseteq V_{micro} \times V_{micro}$ est l'ensemble des arêtes (Fig. 2). Pour chaque arête e =

1. Les êtres humains situés dans le voisinage de l'utilisateur et qui pourraient influencer la réponse à une requête sont considérés comme des entités sociales.

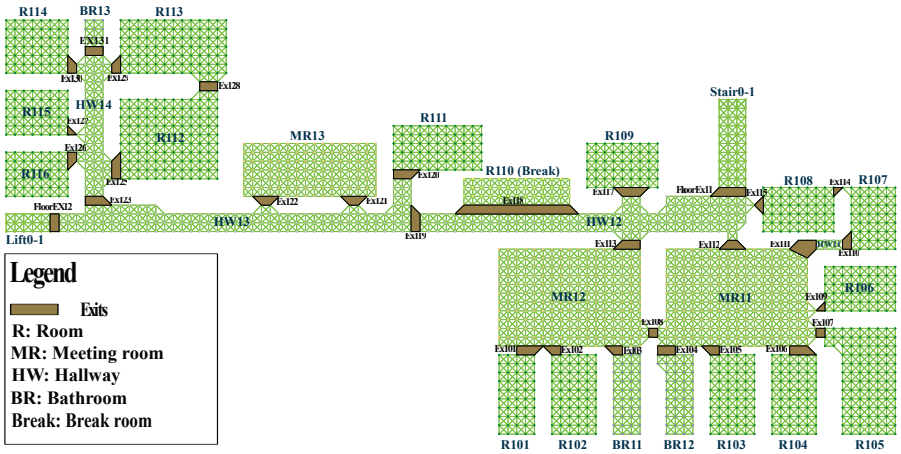


Figure 1. Un graphe à un niveau micro d'un plan d'étage: premier niveau du modèle hiérarchique

$(v_i, v_j) \in E_{micro}$, il existe deux fonctions dépendantes du temps $\omega l_{i,j}(t) \in W_{length}$ et $\omega t_{i,j}(t) \in W_{time}$ qui calculent la *distance* et le *temps de parcours* séparant la source v_i de la cible v_j, respectivement, si le parcours du graphe est lancé à l'instant t. En plus du temps, ce modèle prend également en compte d'autres dimensions telles que les *profils utilisateurs* pour ajuster les impédances des arêtes. Les profils utilisateurs sont gérés statiquement afin de dériver des graphes génériques provenant du graphe G_{micro} et qui correspondent à des catégories prédéfinies d'utilisateurs.

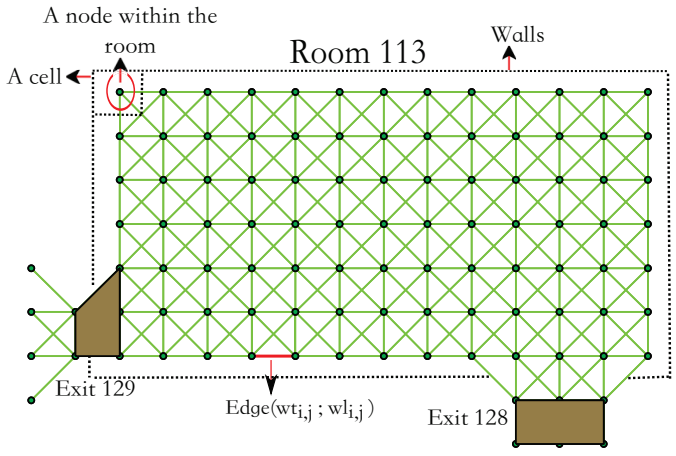


Figure 2. Une vue rapprochée de la pièce (R113) du graphe à granularité fine (V_{micro}, E_{micro})

Chaque noeud possède un ensemble de propriétés qui maintiennent des informations sur sa position, son statut actuel (i.e., s'il est accessible ou non), un ensemble

d'identifiants qui fait référence aux unités spatiales auxquelles le noeud appartient (i.e., identifiants de la pièce, de l'étage et du bâtiment) et l'ensemble d'actions déclenchées en temps réel (i.e., messages contextuels ou notifications) et qui peuvent être exécutées selon certaines contraintes contextuelles (par exemple, pour rappeler un utilisateur qui se déplace dans un centre commercial d'acheter des produits alimentaires ou de fruits quand il se trouve à côté d'un supermarché).

La couche de base est construite d'une première phase hors ligne, et d'une phase ultérieure en ligne qui est en charge de la mise à jour des changements potentiels de données dépendantes du temps. Lors de la première phase, les noeuds qui sont couverts par des objets statiques (e.g., un mur, une table, etc.) sont marqués comme étant occupés alors que le reste est initialement libre. En outre, le statut d'un noeud dépend aussi du profil de l'utilisateur, puisque différents types d'utilisateurs peuvent avoir un ensemble de noeuds accessibles complètement différent (e.g., un certain noeud peut ne pas être occupé physiquement, mais appartient à une pièce qui ne peut être accessible qu'avec une carte-clé). Le seul changement dynamique qui doit être détecté en temps réel fait référence à l'occurrence d'événements qui peuvent avoir un impact direct sur l'accessibilité de noeuds. Un exemple typique est le début d'incendie dans une pièce du bâtiment. Dans cette situation, les avertisseurs d'incendie sont censés détecter cet événement et le communiquer au système. Avec des mises à jour périodiques effectuées automatiquement, le système serait capable de représenter l'emprise spatiale croissante de l'incendie, et de marquer ainsi les noeuds de cet espace physique comme inaccessible aux utilisateurs.

2.1.2. Couches Abstraites

2.1.2.1. **Hiérarchie d'entrée/sortie**

Une entrée/sortie (en anglais *exit*) est un élément essentiel du modèle de données, à travers laquelle un utilisateur peut quitter ou entrer dans un lieu (e.g., portes ou escaliers). Une sortie est représentée comme un noeud abstrait qui appartient à deux unités spatiales différentes, et est dérivée en agrégeant les noeuds frontaliers de ces deux unités adjacentes (cf. Fig. 2). Les distances du réseau optimales et les temps de parcours entre les paires de sorties sont pré-calculés et mises en cache afin d'accélérer le calcul de recherche de chemin en temps réel. Au second niveau d'abstraction, une hiérarchie d'entrée/sortie est par conséquent construite. Elle est utilisée pour le traitement des requêtes RDL qui nécessitent un calcul précis de la distance.

Les sorties dans cette couche sont organisées de manière hiérarchique pour mieux refléter leur sémantique dans le contexte de la navigation indoor (Hu, Lee, 2004). Comme illustré dans Fig. 3, la structure hiérarchique permet de distinguer entre une sortie d'une pièce, *RoomExit*, et une sortie d'un étage, *FloorExit*, qui est représentée à un niveau d'abstraction supérieur en raison de son importance, de sorte qu'un trajet direct à partir d'une position actuelle vers la sortie de l'étage la plus proche peut être facilement déterminé. Il existe aussi d'autres arcs entre les sorties du même niveau en fonction de leur connectivité (liens horizontaux illustrés par des lignes pointillées dans Fig. 3). Il convient également de noter qu'une généralisation de cette hiérar-

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 (Figure 4.5). Accordingly, an exit can contain multiple nodes and edges at the fine-grained level (see Figure 4.3 and 4.4). Based on these exits, a coarser network (at a higher level of abstraction) can be designed, in which nodes depict those exits and links represents optimal navigational paths between directly reachable 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 ex' is relevant for a given exit ex if and only if ex' is directly reachable from ex (i.e., there is an accessible passageway for pedestrians from ex to ex' 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.

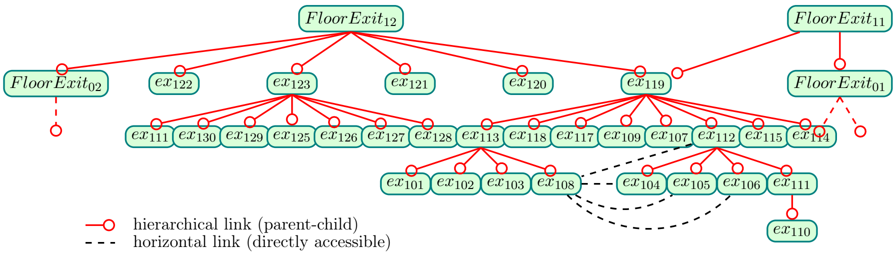


Figure 4.5: Part of the exit hierarchy derived from the fine-grained graph (Floor-01, Building-1)

A link between two directly reachable exits is represented by a path (i.e., a sequence of nodes and edges in (V_{micro}, E_{micro})) at the fine-grained layer. 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: $ex_{r,r'} = \{v_i, v_j \in V_{micro} \mid \exists ex \in E_{micro}, ex = (v_i, v_j) \wedge r \in ex.\mathtt{L}_{ex} \wedge r' \in ex.\mathtt{L}_{ex}\}$. Regarding its belonging to the topological hierarchy, an exit is also characterised by: $\mathtt{L}_{ex_{r,r'}}$ ={*local-id*, {*r*, *r'*}, *floor-id*, *building-id*}. $FloorExit_{11}$ is an example of an exit depicted in Figure 4.5, which belongs to two structural units: $Stair_{01}$ and HW_{12}. Therefore, $\mathtt{L}_{FloorExit_{11}}$ = {*FloorExit-11*, {$Stair_{01}$, HW_{12}}, *Floor-1*, *Building-1*}. Similarly, one can derive the abstract nodes of the second layer representing all exits on a given floor. An abstract edge $(ex_{r,r'}, ex_{r',r''})$ in the exit hierarchy is a path made of a sequence of nodes and edges of the fine-grained layer that … network distance from a node $v_{start} \in ex_{r,r'}$ to a node $v_{goal} \in ex_{r',r''}$. 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

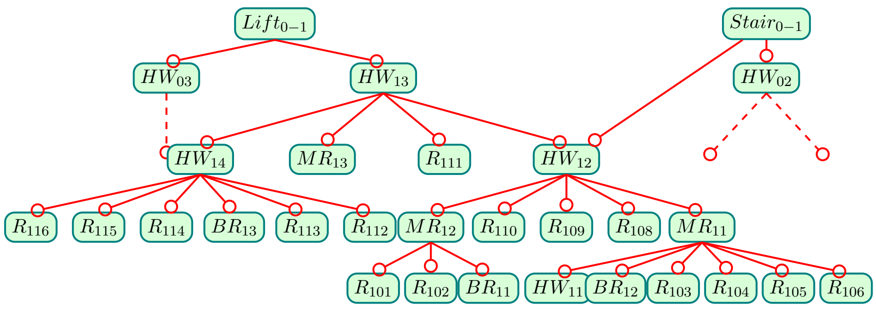


Figure 4.6: Part of the location hierarchy derived from the fine-grained graph (Floor-01, Building-1); "*HW*" stands for Hallway, "*MR*" for Meeting Room, "*R*" for Room, and "*BT*" for Bathroom.

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. 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.

Figure 3. Une partie de la hiérarchie d'entrée/sortie (Étage-01, Bâtiment-1)

chie qui couvre un bâtiment multi-étages est utilisée pour la planification de chemin. Par conséquent, une sortie du rez-de-chaussée fait référence à la sortie du bâtiment, *BuildingExit*, comme un noeud parent, et à la sortie du premier étage comme un noeud fils.

2.1.2.2. Hiérarchie d'unités structurelles

L'intégration des informations sur les sorties dans la hiérarchie topologique permet la modélisation explicite des chemins optimaux à un niveau d'abstraction supérieur. Celles-ci sont utilisées pour faciliter des parcours hiérarchiques du graphe et pour pallier les problèmes de performances lors de la traversée du graphe de base. Cependant, la sémantique topologique telle que la connectivité entre les unités structurelles n'est pas explicitement matérialisée dans la hiérarchie d'entrée/sortie, même si des informations qui représentent leur appartenance à la hiérarchie topologique ont été incorporées. Par conséquent, une hiérarchie d'unités structurelles qui repose sur un graphe de connectivité, représentant les pièces comme des noeuds et les connexions comme des arcs, peut être dérivée comme une couche supplémentaire afin de préserver les relations topologiques (Fig. 4).

Figure 4. Une partie de la hiérarchie des pièces (Étage-01, Bâtiment-1); "HW" signifie Hallway, "MR" représente une Meeting Room, "R" pour Room, et "BT" pour Bathroom

Une telle hiérarchie peut être directement dérivée à partir de la couche de base. En effet, le processus d'agrégation, *"clustering"*, consiste en: (1) l'extraction et l'agrégation des noeuds dont l'identifiant de la pièce est identique pour former les nouveaux noeuds abstraits de la hiérarchie; (2) la détermination des chemins optimaux entre les unités structurelles connectées en utilisant les fonctions dépendantes du temps préalablement définies et (3) la création des arcs abstraits entre les unités connectées en initialisant les poids aux valeurs calculées dans l'étape précédente.

Par conséquent, cette conception hiérarchique est très utile, car elle prend en charge une large gamme d'applications, et permet de pallier les problèmes de performance et de passage à l'échelle lors du traitement des requêtes RDL.

2.2. Composante des Entités

Cette composante représente les personnes et les objets d'intérêt de l'espace indoor. Ces entités sont soit fixées à l'infrastructure (i.e., objets statiques tels que des portes, des murs, des capteurs fixes, etc.) soit dynamique (e.g., utilisateurs mobiles, phénomènes continus). Une entité est caractérisée par des propriétés statiques (i.e., des attributs tels que son type, son statut et une description quantitative et qualitative) et potentiellement d'autres propriétés dynamiques telles que des *espaces d'interaction* (introduits dans (Bhatt *et al.*, 2009) et adaptés dans un travail précédent (Afyouni *et al.*, 2010)) qui couvrent des informations sémantiques utilisées à des fins d'interaction. Les espaces d'interaction identifiés sont: (1) *l'espace physique* représenté par l'ensemble de noeuds couverts par l'entité à un instant donné; (2) *l'espace fonctionnel* désigne les noeuds sur lesquels une autre entité peut interagir physiquement avec l'entité considérée; (3) *l'espace de zone* est un paramètre spécifique uniquement affecté aux capteurs et désigne l'ensemble des noeuds couverts par le capteur; et (4) *l'espace opérationnel* d'un utilisateur mobile est l'ensemble des noeuds qui lui sont accessibles à un instant donné. De plus, un objet peut effectuer une liste sélectionnée d'actions qui peuvent être déclenchées en fonction de certaines contraintes contextuelles dépendantes de l'application.

2.3. Composante des Actions

Cette composante modélise l'ensemble des actions qu'une entité donnée peut effectuer. Les actions sont dépendantes du contexte; ce qui signifie qu'à un instant donné et pour une certaine entité, seulement une liste spécifique d'actions possibles est valide. Pour un utilisateur mobile, les actions comprennent une séquence de mouvements, des interactions avec d'autres entités voisines et avec des objets situés dans l'environnement, un certain nombre d'interrogations sur des services en vue d'atteindre un objectif prédéfini. L'approche présentée permet de modéliser efficacement des objets d'intérêt situés dans l'environnement, afin que les utilisateurs qui sont engagés dans une activité donnée puissent recueillir des connaissances et mieux comprendre leur entourage. Un utilisateur peut ainsi reconfigurer et manipuler des objets physiques (e.g., une chaise, une porte, un chauffage, etc.) ou d'autres

objets virtuels (e.g., une image 2D/3D d'un objet physique, une interface utilisateur numérique, des recommandations/informations, etc.) afin de produire des changements dans l'environnement. Un utilisateur peut de plus communiquer avec tout capteur fixe ou mobile situé dans l'espace de zone du capteur mobile qui lui est rattaché ou qui est intégré dans son appareil (e.g., un capteur MEMS, une étiquette RFID, etc.). Lorsque l'on considère les phénomènes continus, leurs actions peuvent se matérialiser par la façon dont un phénomène donné se diffuse dans l'espace.

3. Langage de requêtes RDL dans les environnements indoor

De nombreuses applications mobiles innovantes doivent intégrer un mécanisme de traitement continu des requêtes dépendante de la localisation (RDL) sur les objets en mouvement. Des exemples de tels services comprennent la surveillance continue de la foule dans une zone donnée, les alertes basés sur la localisation (e.g., *"envoyer en permanence des e-coupons aux clients situés à moins de 200 mètres de mon magasin"*), la notification de la foule dans une situation d'urgence, et les applications de type "Friend Finder" (e.g., 1 - *"permettez-moi de savoir si je suis près d'un restaurant où un de mes amis est présent"*, et 2 - *"si je continue dans cette direction, quels seront les restaurants les plus proches dans les 10 prochaines minutes?"*).

Les approches qui permettent le traitement *adaptatif* et *incrémental* pour améliorer la performance des requêtes RDL sur des objets mobiles sont nécessaires (Hu *et al.*, 2005; Mokbel *et al.*, 2005). D'une part, le traitement adaptatif des requêtes aborde le problème de coûts imprévisibles et des données dynamiques afin d'optimiser l'exécution des requêtes d'une manière qui apporte des réponses appropriées (Deshpande *et al.*, 2007). D'autre part, un paradigme d'exécution incrémental aborde le traitement continu des requêtes grâce à la réutilisation d'informations provenant de recherches prétendantes pour accélérer les recherches courantes. Par conséquent, de nouvelles réponses pour des séries de problèmes de recherche similaires peuvent être calculées plus rapidement que de tout réévaluer pour chaque requête indépendamment (Sun *et al.*, 2009). Les paradigmes d'exécution adaptative et incrémentale sont nécessaires afin d'atteindre un traitement des requêtes RDL en continu d'une manière suffisamment souple et plus efficace.

Comme indiqué précédemment, certains types de requêtes RDL présentent un intérêt particulier dans les environnements indoor, telles que les requêtes de navigation, de zone et de recherche des plus proches voisins. Afin d'améliorer l'expressivité des requêtes, une grammaire est introduite dans cette section. Cette grammaire est adaptée au contexte des environnements indoor, et supporte les requêtes RDL tout en favorisant d'autres préférences spécifiques au modèle de requêtes (Fig. 5). Elle inclut, entre autres, des opérateurs (e.g., *All-routes*) et des contraintes (e.g., *Stop-vertices*) utilisées dans les requêtes de navigation et inspirées par (Booth *et al.*, 2009).

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
Loc-Select	→	Object-id '.' 'loc' | *gr* '('Map-id ',' Class-name ')' | *gr* '('Map-id ',' Route-id ')'
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)*
optimization-criteria	→	(*minimize* | *maximize*) Measure
Measure	→	*time* | *distance*

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

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

Figure 5. Une partie de la grammaire utilisée pour les requêtes RDL dans les environnements indoor

3.1. Principes

Dans la structure générale du langage de requête, deux types de requêtes sont identifiés: le premier représente typiquement une structure standard de requête SQL (*Standard-query*) ainsi que des types spécifiques de contraintes dépendantes de la localisation (*inside* et *nearest*), qui sont essentiellement utilisés pour exprimer les requêtes de zone et de recherche des voisins les plus proches. Le second représente les requêtes de navigation qui ont besoin d'intégrer un mécanisme de calcul d'itinéraire lors du traitement, tout en optimisant les critères de la distance et/ou du temps. Pour une requête de navigation, la clause FROM contient un appel externe à l'opérateur *All-routes*, qui a la syntaxe générale suivante: *All-routes (Loc-Ref, Loc-Target)*. Cet opérateur retourne un ensemble de routes valides entre la position actuelle de l'objet référence et celle de l'objet cible. Les arguments *Loc-Ref* et *Loc-Target* peuvent correspondre soit à un identifiant d'un noeud "Vertex-id" soit aux positions actuelles de l'objet référence et de l'objet cible, respectivement. *Loc-Target* peut aussi faire

référence au nom de la classe de l'objet cible; ceci est utilisé, par exemple, dans la contrainte *inside* afin de récupérer tous les objets d'un type donné.

La clause ⟨ WITH Stop-vertices ⟩ indique que la route doit passer par un endroit qui a un intérêt pour l'utilisateur. Par ailleurs, deux critères d'optimisation sont appliqués: *time* et *distance*, qui sont considérés en se basant sur les fonctions définies précédemment ($\omega l_{i,j}(t)$ et $\omega t_{i,j}(t)$). Dans la structure standard, deux types de conditions dépendantes de la localisation peuvent être exprimés dans la clause WHERE: *inside(Radius, Loc-Ref, Loc-Target)* et *nearest(K, Loc-Ref, Loc-Target)*. Une contrainte *inside* est appliquée lors de l'exécution d'une requête de zone, et prend en compte le seuil maximal à examiner, pour calculer l'ensemble de chemins (autour de l'objet référence). La contrainte *nearest* est exprimée pour exécuter les requêtes de *K* voisins les plus proches, en précisant le nom de la classe d'objets d'intérêt dans Loc-Target.

Le concept du *location granule* proposé dans (Ilarri *et al.*, 2011) est adopté. Un granule regroupe un ensemble de positions à un niveau plus fin (dans notre cas, les coordonnées des noeuds dans le graphe de base) sous un nom commun. L'utilisation des granules permet de formuler des requêtes avec une résolution spatiale appropriée à l'application prévue (e.g., des noeuds à un niveau micro, pièces, étages, bâtiments, etc.). L'opérateur *gr* est un raccourci de *granule* et retourne le granule associé à l'objet selon le niveau de granularité choisi. Ce concept de granule est étendu pour être utilisé au niveau de la clause SELECT afin d'illustrer les routes de navigation à un niveau d'abstraction choisi, ainsi qu'au niveau de la clause FROM pour gérer les requêtes de navigation.

Figure 5 montre que l'opérateur *gr* peut être référencé dans la clause SELECT, la clause FROM et/ou la clause WHERE de la requête, ceci en fonction de l'utilisation des granules pour la visualisation des résultats et/ou le traitement des contraintes ou le calcul des itinéraires. Pour la visualisation, cet opérateur est utilisé comme une projection *Loc-Select* dans la clause SELECT, selon la demande soumise par l'utilisateur, pour afficher le résultat au niveau de granularité désiré; par exemple, SELECT gr('room-level', Person) est utilisé pour projeter les pièces où les personnes récupérées par la requête sont situées. L'opérateur *gr* peut en plus être appliqué sur une route, qui est le résultat d'une requête de navigation, afin de montrer la séquence de noeuds et d'arêtes obtenues dans la route à un niveau d'abstraction choisi. Ainsi, SELECT gr('room-level', Routes.id) sera utilisé pour illustrer la séquence de pièces inférée de l'itinéraire composé initialement de noeuds et des arêtes de différents niveaux de granularité (e.g., niveau micro et la hiérarchie de sortie).

Le même opérateur *gr* peut être spécifié pour le traitement des requêtes comme un argument *Loc-Ref* et/ou *Loc-Target* de la clause FROM (i.e., dans l'expression *All-routes-expression*), et/ou dans les contraintes (i.e., inside et nearest), en référence aux positions des objets concernés. Par exemple, *inside(100 meters, gr('room-level', 'o1'), Person)* est une contrainte satisfaite par les personnes qui se situent sur une route valide (i.e., distance inférieure à 100 mètres) autour de la pièce où l'objet o1 est situé. Au contraire, *inside(100 meters, 'o1', Person)* serait utilisé lorsque la zone souhaitée est déterminée autour de l'objet $o1$ lui même.

3.2. Exemples de Requêtes Dépendantes de la Localisation

Cette section présente quelques exemples de requêtes RDL qui montrent le potentiel de la grammaire proposée. En particulier, nous considérons des requêtes de navigation, des requêtes de zone et des requêtes de recherche des voisins les plus proches.

3.2.1. Requêtes de Navigation

Les requêtes de navigation aident à découvrir des chemins optimaux vers un point/objet d'intérêt, tout en respectant les contraintes dynamiques de l'environnement. Un exemple de requête de navigation est :

– Trouver le plus court chemin de la personne 'userID1' vers la personne 'userID2'; Afficher le résultat à un niveau *room level*

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

où length(R) = $length_{start \rightarrow goal}(t_{start})$ est la distance de la route dépendante du temps de 'P1' situé au noeud v_{start} vers 'P2' situé au noeud v_{goal}.

3.2.2. Requêtes de Zone

Les requêtes de zone sont utilisées pour récupérer des informations sur les objets ou les lieux qui se trouvent dans la zone spécifiée. Un exemple d'une telle requête est :

– Récupérer toutes les entités communicantes dans le voisinage (à une distance inférieure à 100 mètres) d'un utilisateur identifié par 'userID' et avec une portée de communication d'au moins 100 mètres

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

3.2.3. Requêtes de K Voisins les Plus Proches

Une requête de (K) voisins les plus proches récupère les (K) objets qui satisfont certaines spécifications et qui sont les plus proches à un objet ou à un endroit. Par exemple :

– Trouver les deux imprimantes couleur les plus proches à chaque membre du département informatique

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

où FD "*Feature Description*" correspond aux propriétés statiques décrivant l'objet d'intérêt.

En résumé, le langage de requête proposé favorise les requêtes RDL orientées navigation et intègre d'autres préférences dans le modèle de requête. De plus, ce langage gère la granularité des positions des objets statiques et mobiles, favorisant ainsi le modèle hiérarchique de données présenté précédemment.

4. Algorithmes pour le traitement continu de requêtes RDL dans les environnements indoor mobiles

Cette section présente plusieurs solutions pour résoudre les problèmes algorithmiques rencontrés lors du traitement des requêtes RDL continues. Les algorithmes présentés constituent les fonctions essentielles derrière les opérateurs et les contraintes définies dans la grammaire de requête introduite dans la section 3. Des algorithmes pour le traitement continu des *requêtes de recherche de chemin hiérarchique* et des *requêtes de zone* appliqués à des objets statiques et/ou en mouvement sont développés en se basant sur le modèle de données hiérarchique décrit dans la section 2. Ces algorithmes profitent des différents niveaux d'abstraction du modèle de données, et développent un paradigme de traitement incrémentale afin de fournir une solution efficace et évolutive pour les systèmes de navigation indoor.

Contrairement aux requêtes RDL classiques qui considèrent que l'objet référence et/ou les objets cibles sont statiques, les algorithmes présentés dans cette section visent une généralité maximale en supposant que les objets cibles peuvent se déplacer librement dans l'espace. Par conséquent, une réévaluation continue de la requête est nécessaire tout en gardant une trace de l'information pertinente d'objets impliqués dans le traitement. Un paradigme de traitement incrémentale est développé afin de permettre la réévaluation continue de ces requêtes. Ce paradigme prend en compte les évaluations antérieures pour inférer la nouvelle réponse sans devoir tout recalculer (Afyouni, Ray, Ilarri, Claramunt, 2012). Cela nécessite de construire un arbre de

recherche spécifique adapté à chaque type de requêtes traitées dans cette section. Cet arbre de recherche est créé et maintenu de façon dynamique pour chaque requête, de sorte que les éléments soient ajoutés et/ou retirés selon une certaine politique de mise à jour.

4.1. Traitement continu des requêtes de recherche de chemin hiérarchique

Les requêtes de recherche de chemin couvrent toutes les requêtes qui aident directement les utilisateurs à trouver et atteindre des points/objets d'intérêt, en leur fournissant des indications de navigation et tout en optimisant certains critères tels que la distance parcourue ou le temps de déplacement. Cette section présente une approche pour le traitement continu de requêtes de chemin qui s'appuie sur une technique *"bottom-up"*, et qui utilise deux niveaux d'abstraction: La couche fine au premier niveau et la hiérarchie d'entrée/sortie au deuxième niveau d'abstraction. Les principales étapes du processus d'exécution peuvent être résumées comme suit:

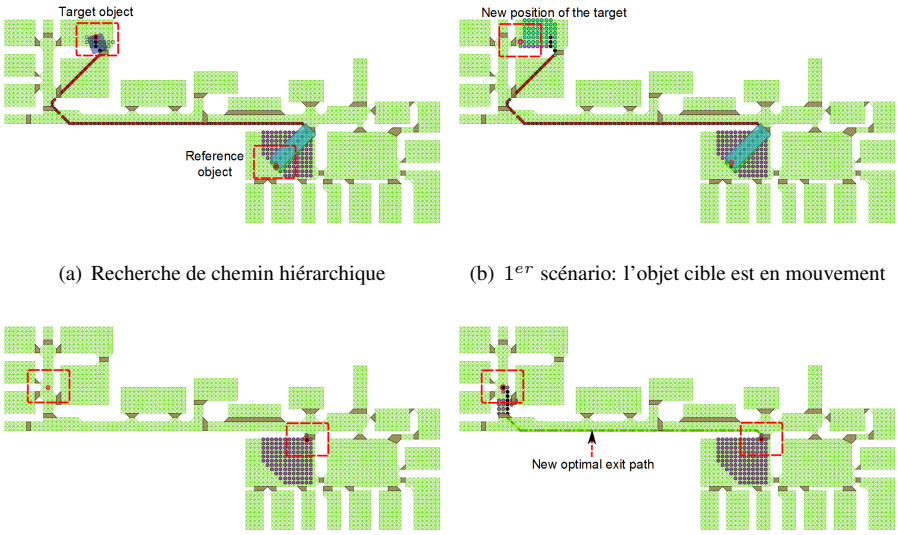
– **Étape 1:** Trouver le chemin optimal au sein du granule initial jusqu'à la sortie optimale.

– **Étape 2:** Effectuer la recherche au niveau abstrait (hiérarchie d'entrée/sortie) du chemin optimal à partir de la sortie du granule initial vers le granule contenant l'objet cible.

– **Étape 3:** Trouver le chemin optimal dans le dernier granule vers l'objet cible, à partir de l'entrée correspondante du granule (Fig. 6(a)).

– **Étape 4:** Démarrer une recherche continue du chemin en tenant compte des positions mises à jour à la fois de l'objet référence et d'objets cibles. Cela implique la transformation d'un arbre de recherche initial enraciné à l'emplacement précédent de l'objet référence à un arbre enraciné à son emplacement actuel. Le processus se poursuit soit par l'expansion de nouveaux sous-arbres à partir des feuilles vers la cible et/ou en supprimant les sous-arbres qui ne sont plus nécessaires (Fig. 6(b), 6(c) et 6(d)).

L'approche retenue pour le traitement continu des requêtes de chemin est affinée par deux algorithmes complémentaires. Les étapes 1 à 3 représentent la première itération du processus, qui effectue *la recherche de chemin hiérarchique* (Fig. 6(a)), présenté en détail dans l'algorithme 1 (cf. Annex A). L'étape 4 concerne le traitement continu de la requête, qui est présenté en détail dans l'algorithme 2 (cf. Annex A). Ce sont des étapes génériques qui peuvent ne pas être complètement exécutées selon que la référence et/ou les objets cibles sont statiques ou en mouvement. Ces deux algorithmes représentent la mise en oeuvre de l'opérateur *All-routes* défini dans la section 3.1.



(a) Recherche de chemin hiérarchique (b) 1^{er} scénario: l'objet cible est en mouvement

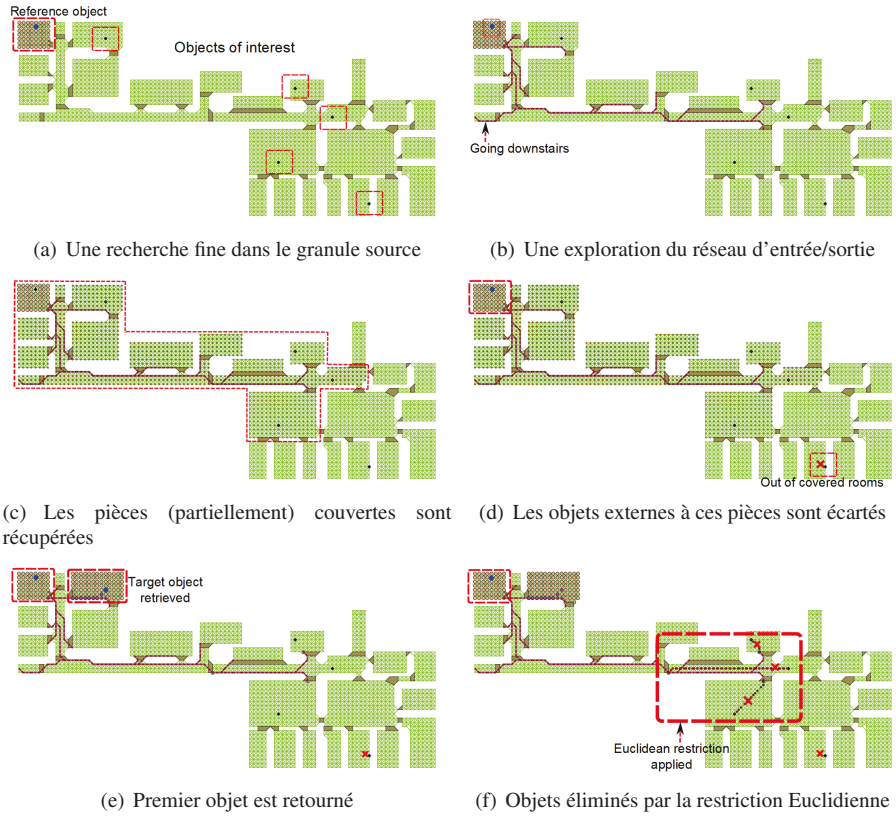
(c) 2^{eme} scénario: l'objet cible change son granule actuel (d) 2^{eme} scénario: adaptation de la paire de sorties

Figure 6. Algorithme de recherche de chemin hiérarchique et incrémental

4.2. Traitement continu et hiérarchique des requêtes de zone

Les requêtes de zone cherchent et récupèrent les objets ou lieux d'intérêt dans une zone définie par l'utilisateur. Dans nos scénarios, les zones sont caractérisées par un ensemble de routes valides sur lesquelles les objets d'intérêt doivent être situées. Cette section présente notre approche pour le traitement continu de requêtes de zone qui considère la mobilité à la fois de l'objet référence ainsi que les objets cibles. Cette approche est basée sur un mécanisme d'*expansion du réseau hiérarchique* (Algorithme 3 (cf. Annex B). Le principe de cette approche est de mettre à jour en permanence l'ensemble des noeuds visités qui composent les routes valides autour de l'objet référence (Fig. 7(a), 7(b) et 7(c)). En outre, un index spécifique est construit à la suite de l'expansion du réseau hiérarchique. Cette structure dispose de plusieurs propriétés associées aux noeuds générés, tels que la trajectoire optimale pour le noeud courant vers le noeud source. Cette propriété particulière offre un avantage important car elle permet au système de fournir non seulement des informations d'appartenance d'un objet dans la zone spécifiée, mais aussi de retrouver le chemin optimal vers cet objet cible. Par conséquent, le résultat d'une requête de zone continue englobe l'ensemble des identifiants d'objets qualifiés, leur chemin optimal vers l'objet de référence, ainsi que les distances du réseau correspondantes (Fig. 7(e) et 7(f)). Le traitement continu de requêtes de zone consiste à maintenir en permanence l'ensemble des noeuds parents à jour lorsque la racine de l'arbre de recherche change (par exemple, lorsque

l'objet référence se déplace). Les noeuds frontaliers sont vérifiés afin de décider, pour chacun d'eux, si un sous-arbre doit être exploré à partir de ce noeud, ou bien une *"recherche inversée"* vers la source est effectuée afin d'éliminer les noeuds qui ne sont plus pertinents (cf. Algorithme 4, Annex B).



(a) Une recherche fine dans le granule source

(b) Une exploration du réseau d'entrée/sortie

(c) Les pièces (partiellement) couvertes sont récupérées

(d) Les objets externes à ces pièces sont écartés

(e) Premier objet est retourné

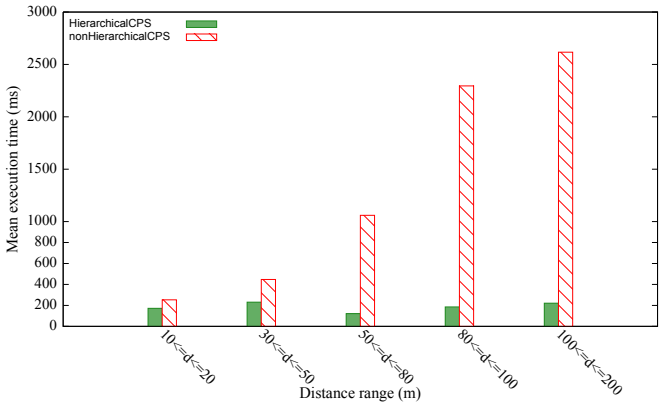
(f) Objets éliminés par la restriction Euclidienne

Figure 7. Algorithme Incrémental pour les requêtes de zone: un champ de 50 mètres est appliqué

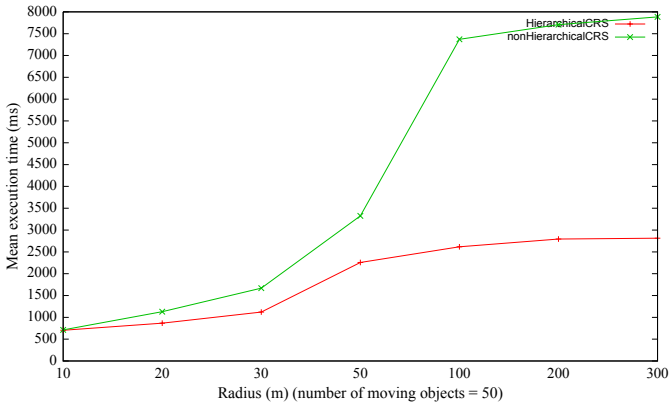
5. Une extension PostgreSQL pour le traitement continu des requêtes RDL

Cette section présente la conception et la mise en oeuvre d'une extension de base de données basée sur le SGBD Open Source PostgreSQL. Le prototype développé gère les recherches de chemin continues et les requêtes de zone en se basant sur notre modèle de données hiérarchique d'un environnement intérieur. Les principales parties du prototype développé comprennent: (i) un modèle de données basé sur un réseau hiérarchique des environnements intérieurs; (ii) les opérateurs et les contraintes dépendantes de la localisation introduits dans la grammaire de requête et (iii) Plusieurs algorithmes

de traitement continu des requêtes RDL sur des objets mobiles. Les résultats des expériences qui ont été menées pour étudier le passage à l'échelle et la performance de nos solutions sont également signalés.



(a) Recherche de chemin hiérarchique vs. non-hiérarchique



(b) Requêtes de zone hiérarchique vs. non-hiérarchique

Figure 8. Impact de la distance sur le temps d'exécution

Selon les résultats expérimentaux, le temps d'exécution de l'algorithme développé pour la recherche continue de chemin est satisfaisant, et permet le passage à l'échelle par rapport à la distance entre les objets d'intérêt et au nombre de noeuds développés (Fig. 8(a)). Il a été conçu pour être suffisamment évolutif pour les grands espaces intérieurs grâce au traitement de la requête hiérarchique. En outre, l'approche de traitement continu des requêtes de zone offre un passage à l'échelle satisfaisant en ce qui concerne le champs spécifié par l'utilisateur (Fig. 8(b)), et des performances acceptables avec un nombre d'objets en mouvement assez élevé (Fig. 9). En ce qui concerne les résultats expérimentaux pour la recherche continue de requêtes de zone,

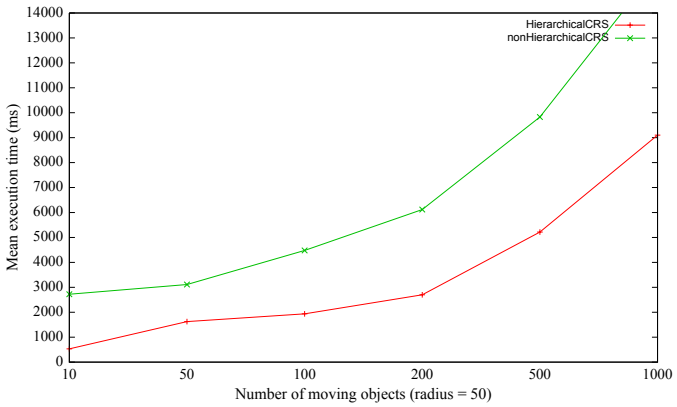


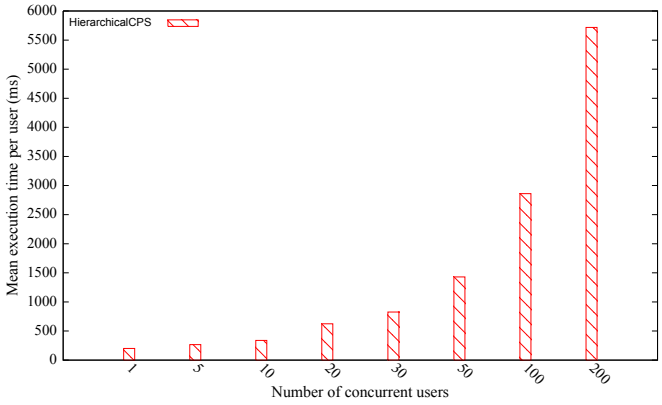
Figure 9. Impact du nombre d'objets mobiles sur le temps d'exécution des requêtes de zone

tous les objets mobiles examinés sont supposés être d'intérêt pour la requête correspondante. En effet, seuls les objets d'un certain type (ceux qui sont impliqués dans la requête) ont un impact direct sur la performance du traitement des requêtes, donc cela génère le scénario le plus pessimiste. Un pré-filtrage d'objets basés sur les propriétés statiques (par exemple, les gens dans ma liste d'amis) a un effet similaire, car cela réduit le nombre d'objets à considérer comme un candidat potentiel. Néanmoins, le nombre total d'objets en mouvement, indépendamment de leur type, a également un léger impact sur la performance du serveur en raison de la nécessité de gérer leurs mises à jour de localisation.

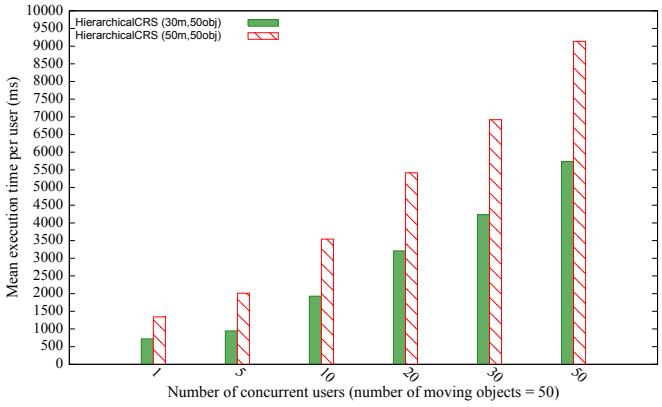
Figure 10(a) illustre le temps de réponse moyen pour une requête de chemin continu à un timestamp donné et pour un utilisateur donné. Le nombre d'utilisateurs simultanés qui interrogent le système en temps réel varie de 1 à 200. Les résultats des simulations montrent qu'avec 30 à 50 accès simultanés le temps de réponse moyen varie entre 1 et 1,5 secondes. Même avec 100 à 200 des requêtes de chemin concurrentes, le temps d'exécution d'une requête reste acceptable, et le nombre de requêtes simultanées a un impact linéaire sur la performance. Les résultats des expériences qui évaluent les requêtes de zone concurrentes sont illustrés dans Fig. 10(b). La simulation montre une diminution du nombre d'accès simultanés qui varie de 1 à 50.

6. Conclusions & Perspectives

Cet article examine la gestion des données et de la connaissance dans les systèmes de navigation sensibles au contexte implantés au sein de différents environnements indoor (e.g., des bâtiments, des centres commerciaux). Un modèle de données hiérarchique et sensible au contexte est présenté, qui mène à l'utilisation d'autres dimen-



(a) Passage à l'échelle des requêtes de chemin continues



(b) Passage à l'échelle des requêtes de zone continues

Figure 10. Impact des requêtes continues simultanées sur la performance du système

sions du contexte en plus de la localisation des entités concernées, telles que les profils utilisateurs et le temps. La structure hiérarchique de ce modèle favorise un traitement adaptatif et efficace des requêtes RDL. D'autre part, un langage de requêtes qui permet de représenter une variété de requêtes RDL est introduit. La grammaire proposée intègre les préférences des utilisateurs (e.g., points d'intérêts intermédiaires, choix d'optimiser le calcul du chemin selon le temps ou la distance) et d'autres sémantiques (e.g., génération des graphes adaptés aux profils utilisateurs). L'utilisation des granules dans le langage augmente de façon significative l'expressivité des requêtes.

Plusieurs algorithmes pour le traitement continu des requêtes dépendantes de la localisation ont été développés. Un prototype encapsulant les différents algorithmes et méthodes pour la manipulation de ces requêtes a été développé comme une extension

pour le SGBD Open Source PostgreSQL. Les futurs travaux seront orientées vers: (1) l'intégration d'un modèle de contexte étendu afin d'incorporer l'activité des utilisateurs ainsi que le contenu généré par d'autres entités sociales; (2) le développement d'un modèle d'évaluation des requêtes sensibles au contexte qui fournit un indicateur quantitatif et sémantique sur la pertinence et l'incertitude des résultats de la requête en temps réel; et (3) L'annotation sémantique des trajectoires hétérogènes.

References

Afyouni I., Ray C., Claramunt C. (2010). A fine-grained context-dependent model for indoor spaces. In *Proceedings of the 2nd acm sigspatial international workshop on indoor spatial awareness*, pp. 33–38. ACM.

Afyouni I., Ray C., Claramunt C. (2012). Spatial models for indoor and context-aware navigation systems: A survey. *Journal of Spatial Information Science*, Vol. 4, No. 1, pp. 85–123.

Afyouni I., Ray C., Ilarri S., Claramunt C. (2012). 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*, pp. 329–338.

Becker C., Durr F. (2005). On location models for ubiquitous computing. *Personal and Ubiquitous Computing*, Vol. 9, No. 1, pp. 20–31.

Bettini C., Brdiczka O., Henricksen K., Indulska J., Nicklas D., Ranganathan A. *et al. (2009). A survey of context modelling and reasoning techniques.* Pervasive and Mobile Computing*, Vol. 6, No. 2, pp. 161–180.*

Bhatt M., Dylla F., Hois J. (2009). Spatio-terminological inference for the design of ambient environments. In Proceedings of the 9th international conference on spatial information theory (cosit)*, pp. 371–391. Springer.*

Booth J., Sistla P., Wolfson O., Cruz I. (2009). A data model for trip planning in multimodal transportation systems. In Proceedings of the 12th international conference on extending database technology (edbt)*, pp. 994–1005.*

Delling D. (2011). Time-dependent SHARC-routing. Algorithmica*, Vol. 60, No. 1, pp. 60-94. Retrieved from http://dx.doi.org/10.1007/s00453-009-9341-0*

Deshpande A., Ives Z., Raman V. (2007). Adaptive query processing. Foundations and Trends in Databases*, Vol. 1, No. 1, pp. 1–140.*

Ding B., Yu J., Qin L. (2008). Finding time-dependent shortest paths over large graphs. In Proceedings of the 11th international conference on extending database technology (edbt)*, pp. 205–216.*

Hu H., Lee D. (2004). Semantic location modeling for location navigation in mobile environment. In Proceeding of the ieee international conference on mobile data management (mdm)*, pp. 52–61.*

Hu H., Xu J., Lee D. L. (2005). A generic framework for monitoring continuous spatial queries over moving objects. In Proceedings of the 2005 acm sigmod international conference on management of data*, pp. 479–490.*

Ilarri S., Bobed C., Mena E. (2011). An approach to process continuous location-dependent queries on moving objects with support for location granules. Journal of Systems and Software*, Vol. 84, No. 8, pp. 1327–1350.*

Ilarri S., Mena E., Illarramendi A. (2010). Location-dependent query processing: Where we are and where we are heading. ACM Computing Surveys*, Vol. 42, No. 3, pp. 1–73.*

Krisp J. M. (Ed.). (2013). Progress in location-based services*. Springer-Verlag.*

Mokbel M., Xiong X., Hammad M., Aref W. (2005). Continuous query processing of spatio-temporal data streams in place. GeoInformatica*, Vol. 9, No. 4, pp. 343–365.*

Nagel C., Becker T., Kaden R., Li K., Lee J., Kolbe T. (2010). Requirements and space-event modeling for indoor navigation*. Technical report No. OGC 10-191r1. Open Geospatial Consortium Discussion Paper.*

Petit M. (2010). Approche spatiale pour la caractérisation du contexte d'exécution d'un système d'information ubiquitaire*. Unpublished doctoral dissertation, Arts et Métiers ParisTech - Institut de Recherche de l'Ecole Navale.*

Schiller J., Voisard A. (2004). Location-based services*. San Francisco, CA, USA, Morgan Kaufmann.*

Sun X., Yeoh W., Koenig S. (2009). Efficient incremental search for moving target search. In Proceedings of the 21st international joint conference on artifical intelligence (ijca)*, pp. 615–620.*

Tao Y., Papadias D., Shen Q. (2002). Continuous nearest neighbor search. In Proceedings of the 28th international conference on very large data bases (vldb)*, pp. 287–298.*

Terry D., Goldberg D., Nichols D., Oki B. (1992). Continuous queries over append-only databases. In Proceedings of the acm sigmod international conference on management of data*, pp. 321–330. ACM.*

Thrun S. (2003). Robotic mapping: A survey. In Exploring artificial intelligence in the new millennium*, pp. 1–35.*

Weiser M. (1991). The computer for the 21st century. Scientific American*, Vol. 265, No. 3, pp. 94–104.*

Wu K., Chen S., Yu P. (2006). Incremental processing of continual range queries over moving objects. IEEE Transactions on Knowledge and Data Engineering*, Vol. 18, No. 11, pp. 1560–1575.*

Zhang J., Zhu M., Papadias D., Tao Y., Lee D. (2003). Location-based spatial queries. In Proceedings of the acm sigmod international conference on management of data*, pp. 443–454. ACM.*

Annex A. Traitement continu des requêtes de recherche de chemin hiérarchique

Algorithm 1: *hierarchicalPathSearch(locRef, locTarg)*

Data: $\mathcal{S} : \bigcup_{i=1,2} \mathcal{S}_i : G_i = (V_i, E_i)$: hierarchical graph data; q: path query.
Result: A sequence of nodes of the optimal path $outPath = \{v_{start}, v_2, \ldots, v_{goal}\}$ to the target object where $v_i \in \mathcal{S}_1 \cup \mathcal{S}_2$, and the resulting network distance $outLength$
// $locRef/locTarg$: Reference/target object location; $g(v) = length_{v_{start},v}(t)$, $h(v)$, $parent(v)$: a predecessor is associated with each node and for each query; $CLOSED$: set of expanded nodes; $OPEN$: set of boundary nodes;

begin
  $CLOSED \longleftarrow \emptyset$;
  $v_{start} = getNode(locRef)$; $SGranuleId = getDirectGranule(v_{start})$;
  $OPEN = \{v_{start}\}$;
  // A new position implies a new root of the tree;
  $v_{goal} = getNode(LocTarg)$; $TGranuleId = getDirectGranule(v_{goal})$;
  if $SGranuleId = TGranuleId$ **then**
    $outRecord = adaptedAstar(v_{start}, v_{goal})$;
    $outPath = outRecord.outPath$;
    $outLength = outRecord.outLength$;
  else
    // Retrieve the best pair of exits of the source and target granules, and the corresponding optimal path;
    $optimalExitPath = computeRefTarExits(vstart, vgoal)$;
    $sourceExit = optimalExitPath[1]$;
    // **Step 1:** Directed A* in G_{micro} from v_{start} to the source exit;
    select a node $v_{midway1}$ such that $\{v_{midway1} \in sourceExit.nodeListIds$ *and* $v_{midway1} \in SGranuleId\}$;
    $outPath = adaptedAstar(v_{start}, v_{midway1})$;
    // $adaptedAstar$ removes v with $f(v) = g(v) + h(v) = \min_{v' \in neighbours(v)} f(v')$ from $OPEN$;
    // And then inserts v into $CLOSED$;
    // **Step 2:** Insert all exits of $optimalExitPath$ into $OPEN$;
    $generate(sourceExit)$; $insert(sourceExit)$ into $OPEN$;
    $parent(sourceExit) = v_{midway1}$;
    foreach *exit* $e \in optimalExitPath$ **do**
      // All-pairs optimal network paths between exits are already precomputed;
      $generate(e)$; $insert(e)$ into $OPEN$; $parent(e) = e'$; // e' is the predecessor of e;
    end
    // **Step 3:** Directed A* in G_{micro} until reaching v_{goal};
    $targetExit = optimalExitPath[length(optimalExitPath)]$;
    select a node where $\{v_{midway2} \in targetExit.nodeListIds$ *And* $v_{midway2} \in TGranuleId\}$;
    $currentPath = append(outPath, optimalExitPath)$;
    // The final $outPath$ is obtained by applying a $reversePath$ procedure from v_{goal} to v_{start} following v_{goal}'s ancestors;
    $outRecord = adaptedAstar(v_{midway2}, v_{goal}, currentPath)$;
    $outPath = reversePath(v_{goal}, v_{start})$;
    $outLength = outRecord.outLength$;
  end
end

Algorithm 2: *continuousHPath(refObjId, tarObjId)*

Data: $\mathcal{S} : \bigcup_{i=1,2} \mathcal{S}_i : G_i = (V_i, E_i)$: hierarchical graph data; q: path query; up-to-date location data of ref/target objects.

Result: A continuous set of optimal paths $outPath$, and the resulting network distance $outLength$

// $refObjId/tarObjId$: Reference/target object identifier; $CLOSED$: set of expanded nodes; $OPEN$: set of boundary nodes.

begin
  $locRef = q.queryLocation(refObjId)$; $v_{start} = getNode(locRef)$;
  $locTarg = q.queryLocation(tarObjId)$;
  $[outPath,\ outLength] = hierarchicalPathSearch(locRef, locTarg)$;
  // At this stage, a complete search tree has been built and stored;
  // Continuous path search (keeping the initial answer up-to-date);
  while $v_{start} \neq v_{goal}$ **do**
    previous-$v_{start} = v_{start}$;
    $v_{start} = getNode(q.queryLocation(refObjId))$;
    $v_{goal} = getNode(q.queryLocation(tarObjId))$;
    if *previous-*$v_{start} == v_{start}$ and $v_{goal} \in CLOSED$ **then**
      // The answer is returned without extra computation;
      $[outPath, outLength] = \{reversePath(v_{goal}, v_{start}),\ g(v_{goal})\}$;
    else
      if *previous-*$v_{start} \neq v_{start}$ **then**
        // An updated search tree is being created with a new root v_{start};
        $updateTreeRootedAt(v_{start})$; // *g-values* are not affected;
        $deleteUnnecessaryNodes()$; // Unnecessary nodes from $CLOSED$ are deleted;
        $completeOPEN()$; // Nodes of the outer perimeter are added;
      end
      if $v_{goal} \notin CLOSED$ **then**
        // Tracking of v_{goal}, check the new optimal pair <exit/entrance>;
        $newOptimalExitPath = computeRefTarExits(vstart, vgoal)$;
        if $(newOptimalExitPath == optimalExitPath)$ **then**
          continue A* in G_{micro} with the same $OPEN$ and $CLOSED$ lists until reaching v_{goal};
        else
          // v_{goal} is either nearer to another exit within the same granule or has left the last granule;
          delete subtree rooted at $LastExit$ from $CLOSED$;
          insert not generated exits from $newOptimalExitPath$ into $OPEN$;
          repeat Step 3 of Algorithm 1 starting from $parent(newOptimalExitPath[n])$ until reaching v_{goal};
        end
      end
      $[outPath, outLength] = \{reversePath(v_{goal}, v_{start}),\ g(v_{goal})\}$;
    end
    $sleepUntilNextPositionUpdate(minWaitingTime)$; // The thread remains asleep while no location update is performed or the minimum waiting time between iterations (if specified) has not been consumed
  end
end

Annex B. Traitement continu et hiérarchique des requêtes de zone

Algorithm 3: *hierarchicalNetworkExpansion(refObjId, radius, objectIds[])*

Data: $\mathcal{S} : \bigcup_{i=1,2} G_i = (V_i, E_i)$: hierarchical graph data; q: range query; up-to-date location data.
Result: $ResultSet$: Returns a SETOF [targObjID, outPath, outLength] for the qualifying target objects
// $C \subseteq objectIds[]$: candidate set; $LocRef$; $g(v)$; $parent(v)$; $RANGE$: set of nodes around the reference object; $coveredRooms$: set of totally/partially covered rooms.

1 **begin**
2 $C \longleftarrow \emptyset$; $coveredRooms \longleftarrow \emptyset$;
// **Step 1:** At this stage, only nodes of the reference object's granule are expanded;
3 $v_{start} = getNode(q.queryLocation(refObjId))$;
$SGranuleId = getDirectGranule(v_{start})$;
4 $RANGE = networkExpansion(v_{start}, radius)$;// Network expansion only at the reference granule;
// A new search tree $RANGE$ is built after Step 1 and stored in the range queue;
// **Step 2:** Network expansion at the Exit Hierarchy;
5 **foreach** *accessible exit* $e \in$ *Exits of the reference granule* **do**
6 select an expanded node $v_{midway1}$ in e;
7 $RANGE = append(RANGE,\ networkExpansion(e, radius - g(v_{midway1}),\ v_{midway1}.path))$;
8 **end**
// **Step 3:** Search for the qualifying objects;
9 **foreach** $potentialQualifyingObject \in objectIds$ **do**
10 $v_{goal} = getNode(q.queryLocation(objectIds[i]))$;
$TGranuleId = getDirectGranule(v_{goal})$;
// Totally/partially covered rooms have at least one accessible exit;
11 **if** $objectIds[i] \in coveredRooms$ **then**
// Apply Euclidean restriction at the target granule;
12 $retrieve$ exit $e \in TGranuleId$ that minimizes $g(e) + Ed(e, v_{goal})$;
13 **if** $g(e) + Ed(e, v_{goal}) > radius - g(e)$ **then**
14 $display\ objectIds[i]$ is out of range;
15 **else**
// A fine-grained search at the target granule is required;
16 select an expanded node $v_{midway2}$ in e;
17 $RANGE = append(RANGE,\ networkExpansion(v_{midway2}, radius - g(e), e.path))$;
18 **if** v_{goal} *is expanded* **then**
19 $[targObjID,\ outPath,\ outLength] = \{objectIds[i],\ reversePath(v_{goal}, v_{start}),\ g(v_{goal})\}$;
20 **else**
21 $display\ objectIds[i]$ is finally out of range;
22 **end**
23 **end**
24 **else**
25 $display\ objectIds[i]$ is out of covered rooms;
26 **end**
27 **end**
28 **end**

Algorithm 4: : *continuousRangeSearch(refObjId, radius, objectIds[])*

Data: $\mathcal{S} : \bigcup_{i=1,2} G_i = (V_i, E_i)$: hierarchical graph data; q: range query; r: network distance; up-to-date location data; $NetDistanceSet$.
Result: $ResultSet$: Returns a SETOF [targObjID, outPath, outLength] for the qualifying target objects
// $C \subseteq objectIds$: candidate set; $locRef$; $g(v)$; $parent(v)$; $RANGE$: set of accessible nodes around the reference object; $N \subseteq RANGE$: set of boundary nodes, $tempSet$: temporary set of nodes.

begin
  $C \longleftarrow \emptyset$;
  $locRef = q.getRefObj.queryLocation(refObjId)$;
  C = $getObjectsInEuclideanRange(locRef, objectIds, radius)$;
  $RANGE = hierarchicalNetworkExpansion(refObjId, radius, C)$;
  while $NotCancel$ **do**
    if $locRef$!= $q.getRefObj.queryLocation(refObjId)$ **then**
      // A new position implies a new root of the tree;
      $locRef = q.getRefObj.queryLocation(refObjId)$;
      $C = getObjectsInEuclideanRange(locRef, objectIds, radius)$;
      foreach $v \in RANGE$ *and* $v \in getDirectGranule(getNode(locRef))$ **do**
        $UpdateParent(v)$;
        // After this step, all the nodes in $RANGE$ are rooted at the new $locRef$
      end
      foreach $v \in N$ **do**
        $length_{locRef,v}(t_v)$ = $updateLength(v)$; // reverse path search to $locRef$;
        if $length_{locRef,v}(t_v) \leq radius$ **then**
          $tempSet = networkExpansion$
          $(v, radius - length_{v,locRef}(t_v), v.path)$;
          $append(RANGE, tempSet)$;
        else
          $v_{current} = parent(v)$;
          $delete(RANGE, v)$;
          $length_{LocRef,v_{current}}(t_{v_{current}})$ = $updateLength(v_{current})$;
          while $length_{LocRef,v_{current}}(t_{v_{current}}) > radius$ **do**
            $v_{current} = parent(v_{current})$;
            $delete(RANGE, v_{current})$;
          end
        end
      end
    end
    foreach $o \in C$ **do**
      // $computePartOfPath$ will repeat steps similar to Step 3 in Algorithm 3;
      $v_{goal} = getNode(q.queryLocation(o))$;
      if $intersect(RANGE, getNode(o.LocTarg))$ *and* $computePartOfPath(e.path, getNode(o.LocTarg)) < radius$ **then**
        $[targObjID,\ outPath,\ outLength] =$
        $\{o,\ reversePath(v_{goal}, v_{start}),\ g(v_{goal})\}$; // e is the optimal target exit to o;
      else
        $display\ objectIds[i]$ is finally out of range;
      end
    end
  end
end