Query Language for Location-Based Services: A Model Checking Approach

Christian HOAREAU†, ‡, Nonmember and Ichiro SATOH††, Member

SUMMARY We present a model checking approach to the rationale, implementation, and applications of a query language for location-based services. Such query mechanisms are necessary so that users, objects, and/or services can effectively benefit from the location-awareness of their surrounding environment. The underlying data model is founded on a symbolic model of space organized in a tree structure. Once extended to a semantic model for modal logic, we regard location query processing as a model checking problem, and thus define location queries as hybrid logic-based formulas. Our approach is unique to existing research because it explores the connection between location models and query processing in ubiquitous computing systems, relies on a sound theoretical basis, and provides modal logic-based query mechanisms for expressive searches over a decentralized data structure. A prototype implementation is also presented and will be discussed.

1. Introduction

Of the many applications of ubiquitous computing, context-aware and location-aware services, in particular, have become the most typical and popular, entering into increasingly more mobile devices and application domains. To support these services, pervasive systems must be able to capture, maintain, and process location information related to various physical entities. Therefore, we believe that location query processing should be regarded as one of the major requirements of ubiquitous computing systems, being closely related to the way we model and organize location information.

Current research on ubiquitous computing has focused on the design and implementation of application-specific location-aware services, like smart rooms and navigation systems. These have often been designed for particular tracking systems like GPSs or RFID tags that capture location information about moving users and objects. As a result, the task of managing data for location information has attracted scant attention thus far. In fact, most existing location-based services maintain and process location information in an ad-hoc manner and/or rely on centralized and inadequate data infrastructures. The underlying connection between location models and query processing for ubiquitous computing systems has yet to be explored.

This paper aims to provide a query language for location-based services, where the data model and core mechanisms are founded on the very structure of the location model. In our opinion, the theoretical foundations of the language itself should be investigated. Indeed, modern database systems and standard query languages such as SQL rely on first-order logic, efficiently implemented using relational algebra [14]. Our approach is based on a well-founded symbolic location model, along with the hierarchical containment relation between places. We propose to extend this model to a Kripke structure, i.e., a transition graph with labeled nodes. Location query processing can then be regarded as a dynamic search over decentralized data structures, and be addressed using model checking techniques. We provide an expressive location query language that inherits syntax and semantics from hybrid logic [3].

The remainder of this paper presents the connection between the location model, query processing and model checking (Sect. 2). We then formally extend the hierarchical symbolic model to a Kripke structure, suitable for model checking (Sect. 3), over which we define the formal syntax and semantics of the query language (Sect. 4). We next give some examples to demonstrate how expressive the language is (Sect. 5) and outline our implementation of the location query processing framework (Sect. 6). After describing a real-world experiment we carried out in a museum (Sect. 7), we outline related work (Sect. 8), make some concluding remarks, and discuss some future research directions (Sect. 9).

2. Background

This section outlines our approach to adopting model checking techniques as a foundation for query mechanisms over symbolic location models.

2.1 Location Query Processing

As described by Becker et al. [5], queries over the location model encompass simple position queries ("Where is the conference room?") , nearest-neighbor queries ("Where is the closest sushi restaurant to my house?") , navigation queries ("How can I reach the bus terminal"), and range queries ("Where are the convenience stores located in my neighborhood?"). Figure 1 describes the system model. Query processing corresponds to a mobile search over the location model, in which the user issuing the query (i.e., its...
provenance) has his/her own representation.

We would also like to point out the crucial role played by the location model. Updated by the tracking (or positioning) system and by interacting with many applications, the location model not only stores representations of the physical world, but also aims to identify and locate users and objects (Fig. 1).

2.2 Location Modeling

Leonhardt [12] proposed a brief taxonomy of existing models and identified two principal categories: geometric and symbolic models.

Geometric Models

Geometric models simply represent the positions of people and objects as geometric coordinates, captured by positioning sensors (e.g., NEXUS [4], [11]). The most prominent reference-coordinate system for outdoor environments is the Global Positioning System (GPS), widely used for navigation-based applications.

However, such raw coordinate-based information is meaningless in human interactions and lacks any semantics for describing the relations between locations. Therefore, geometric coordinates are usually contextualized with secondary human-readable information. For example, in navigation-based applications, GPS coordinates are represented by moving points on city maps.

Symbolic Models

Most context-aware services (e.g., RAUM [6]) require higher-level information that is captured by a symbolic location model, i.e., the notion of place. As defined by Hightower [10]:

“A place is a human-readable labeling of positions. A more rigorous definition is an evolving set of both communal and personal labels for potentially overlapping geometric volumes. An object contained in a volume is reported to be in that place”.

Location information is then defined by all the interrelations between places. We believe that such intrinsic abstraction and expressiveness suits the need for formalization extremely well. In addition, the symbolic model we assume in this paper provides a spatial hierarchy that can be used as a scheme for distributing location information among multiple database servers.

Figure 2 outlines a hierarchical space tree. The nodes represent places, whose interrelations are indicated by the edges. This interrelation is the hierarchical (or containment) relation between places. For example, the “Surgery” is on “Floor 1”, itself contained inside the “Hospital”. Nodes representing container places are called parents, while nodes representing contained places are called children. Since our model does not handle overlapping spaces, a child has one parent at most; however, a parent may have multiple children. A hierarchical tree structure does not exclusively define static places (e.g., “Hospital” and “Surgery”), but can represent mobile objects (e.g., “Elevator” and “Helicopter”), people (e.g., “Nurse” and “Paramedic”), and computing devices (e.g., “PDA-1” and “PDA-2”). Satoh [15] provides a mobile agent-based implementation of a hierarchical model that has such a unified view of location information.

2.3 Query Processing via Temporal Model Checking

Model checking [7] is a well-founded approach and is a widely used technique to enable various types of computing systems to be automatically analyzed, ranging from hardware to software [13], [16]. A system is verified if it satisfies all the specifications defined by its designers. A specification is called a property in the model checking terminology. For example, an automatic lighting system in a conference room could have the following property: to switch on when at least one person is present in the room.

Inputs to a model checking algorithm are a representation of the system S to be analyzed and a property P. In more formal terms, S is defined as a finite state machine, i.e., a Kripke Structure. A Kripke structure is a directed graph, where nodes represent all the possible states of the system, and where edges represent the transitions (or triggers) between states. In our previous example, someone entering or leaving the room could be feasible triggers. Property P is written in a suitable temporal logic formula.

This interrelation information is not necessarily predefined or static, but may be computed from dynamic geometric locations, e.g., the distance between a moving car and a walking pedestrian in an application to prevent collisions.
The connection between model checking and query processing has been extensively studied in the literature, as both evaluate logic-based formulas over finite data structures. With the wide adoption of eXtensible Markup Language (XML) as the standard format to describe the content of documents (and their hierarchical structures), researchers have investigated model checking approaches to query semi-structured data [1], and especially those based on temporal logic like CTL [2].

3. Extending Hierarchical Location Model

3.1 From Temporal to Hybrid Logic

Although temporal logic provides many interesting features for querying graphs, their expressiveness is too restricted to meet our requirements. We have highlighted two major limitations:

- **Unnamed states**: As mentioned by Franceschet et al. [8], temporal logic lacks any mechanism for naming individual nodes in the Kripke structure, whereas location-related queries need such mechanisms to handle identified places. Moreover, temporal logic cannot express relative nodes that are required when users only have local knowledge about the system. Figure 1 shows an example of a relative reference, i.e., the user queries for some place close to his own location here.

- **Top-to-bottom query routing**: Because temporal logic defines the operators for current and future states, location queries can only be routed down the containment hierarchy. Due to innate decentralized information on location and the numerous query sources and targets, our query language should provide both downward and upward operators to search inside the space tree.

As a consequence, from now on we will assume an extension of temporal logic called Hybrid Logic [3]. Hybrid logic introduces the concept of nominals. Nominals are propositional variables that are true at only one node in the Kripke structure, i.e., if p is a nominal, formula p holds if and only if the current node is called p. Thus, it is easy to capture the notion of place: a place corresponds to a nominal.

3.2 Hierarchical Tree Graph

Let us now define the hierarchical location model as a Kripke structure that will define the semantics of the query language. One can also view the hierarchical tree graph as a data model.

**Definition 1 (Hierarchical tree graph)**: A hierarchical tree graph is a 4-tuple \( G = (N, R_l, R_r, L : N \to \mathcal{P}(N)) \) where:

- \( N \) is the set of tree nodes. A node has the same name as the place it corresponds in the location model. Each node is thus uniquely identified.

- \( R_l \) and \( R_r \) are irreflexive, intransitive, and asymmetric, \( R_l \) and \( R_r \) are irreflexive, intransitive, and asymmetric, \( \forall p_1, p_2 \in N : (p_1, p_2) \in R_l \iff (p_2, p_1) \notin R_l \) and \( \forall p_1, p_2, p_3 \in S : (p_1, p_2) \in R_l \land (p_3, p_2) \in R_r \implies p_1 = p_3 \).

- \( L : N \to \mathcal{P}(N) \) labels the tree nodes with elements from \( N \), where \( \mathcal{P}(N) \) is the powerset of \( N \).

Figure 3 has the hierarchical tree graph corresponding to the location model in Fig. 2. The nodes can be identified by place names. The relations \( R_l \) (resp. \( R_r \)) defines reachability from parents (resp. children) to children (resp. parents).

**Definition 2 (Transition relations)**: The transition relations \( R_l \) and \( R_r \) are defined as:

1. \( R_l \) and \( R_r \) are irreflexive, intransitive, and asymmetric.
2. \( \forall p_1, p_2 \in N : (p_1, p_2) \in R_l \iff (p_2, p_1) \notin R_l \) and \( \forall p_1, p_2, p_3 \in S : (p_1, p_2) \in R_l \land (p_3, p_2) \in R_r \implies p_1 = p_3 \).

The properties in (1) mean that a place cannot be its own container (irreflexivity), i.e., a place cannot be its own parent or child. Intransitivity restricts the routing sequence of a query from skipping intermediate locations. Asymmetry is obvious. Property (2) simply defines the containment relationship, by stating that if place \( p_1 \) contains place \( p_2 \), then \( p_2 \) is contained in \( p_1 \), and vice versa. With property (3), a place cannot have more than one container, as areas are assumed not to overlap.

Readers may have noticed that locations on the graph are labeled by their children's identifiers. There is actually an equivalence between these labels and the atomic propositions that are associated with the Kripke graph's nodes in model checking. Atomic propositions are Boolean variables that refer to the set of properties that hold at a given node. As far as the location information is concerned, we believe that this is entirely satisfactory since we have not assumed any quantitative information. When positions (and movements) of people and objects are detected by sensing systems within
the symbolic location model, they are translated on the hier-
archical side into Boolean information. Let us consider the
place “Elevator” in Fig. 3. “Nurse” is a Boolean variable:
she is either present (true) or absent (false). At place “Floor
1”, the “Elevator” is either detected (true) or not (false). On
the “Roof”, the “Helicopter” returning from a rescue opera-
tion has either landed (true) or not (false).

4. Hybrid Logic-Based Query Language

In the previous section, we defined a semantic model to ex-
press logic-based queries over symbolic location models.
“Where is the nurse?” consists of exploring the hierarchical
graph structure, and finding the location where the atomic
proposition “Nurse” will hold. The answer is the “Eleva-
tor”.

4.1 Syntax

Definition 3 (Language components): Our hybrid logic-
based query language contains the following components:

- a countable set $N = \{ l_1, l_2, \ldots, l_n \}$ of locations,
- a countable set of variables $X = \{ x_1, x_2, \ldots, x_n \}$,
- standard logical symbols $\land$, $\lor$, $\neg$,
- spatial modalities $E_{I}$ and $E^{I}$, and
- hybrid logic operators $\downarrow x$ (binding reference) and $\@ l$
  (direct access operator).

The first point we want to draw to the readers’ attention
is that places in the model, like “Hospital” and “Elevator”
(see Fig. 3), are uniquely identified. Semantically speaking,
this is the most important ingredient of hybrid logic in com-
parison with temporal logic. Given these identifiers, opera-
tor $\@ l$ gives direct access to the location $l$.

Moreover, the binding reference $\downarrow x$ creates a brand
new name $x$ and assigns it to the current location. For ex-
ample, $\downarrow$ here instantiates the name here and binds it to
“Elevator”, as long as the current location is identified as
“Elevator”. The binding reference allows us to express rel-
ative locations. However, the relative locations are local to
a query. For the sake of formalization, assignment has been
defined as follows:

Definition 4 (Assignment): An assignment $b$ for $G$ is a
mapping $b : X \rightarrow N$. Given assignment $b$, a variable
$x \in X$, and a location $l \in N$, we define $b_1^x$ by letting
$b_1^x(x) = l$ and $b_1^x(y) = g(y)$ for all $y \neq x$.

The operator $E_{I}$ is the spatial counterpart of the tem-
poral operator $EF$, commonly used in CTL, and $E_{I}$ is its
backward looking analogues. They correspond to routing
triggers that pass the query along the location hierarchy, and
in both directions. Indeed, a query can be passed to its chil-
dren or parent locations.

Finally, our location query language is defined as fol-
definition:

$$\Phi ::= \top \mid p \mid \neg \varphi \lor \varphi_1 \land \varphi_2 \mid E_{I} \varphi \mid \downarrow x \varphi,$$

where $p \in N$, $x \in X$ and $\varphi, \varphi_1, \varphi_2 \in \Phi$.

4.2 Semantics

We will now provide the formal semantics of the hybrid
logic-based query language, by giving its interpretation in
terms of hierarchical tree graphs. The exact meaning of a
formula is given by a satisfaction relation that connects hier-
archical tree graphs and formulas. That satisfaction relation
is written as: $G, b, l \models \varphi$

Readers who are unfamiliar with such notations should
regard the left side of the satisfaction relation as the context
of the query, while the right side corresponds to the query
itself.

- **Context**: This defines all the parameters to evaluate
  the query. The context encompasses the hierarchical
tree structure $G$ relative to the location model (i.e., the
data structure), the binding names locally created by
the query (stored by the assignment $b$), and the location
$l$, where the query is evaluated.

- **Query**: This is expressed by a hybrid logic-based for-
  mula. In the following, we will define its formal se-
mantics and provide informal comments and illustra-
tions.

1. $G, b, l \models \top$

   This is a basic assumption. True ($\top$) is valid ever-
where in the location model.

2. $G, b, l \models p$ iff $p$ is true at $l$

   Location $l$ is the parent location of $p$. This means that
   $p$ is true at location $l$.

3. $G, b, l \models \neg \varphi$ iff $G, b, l \not\models \varphi$

   This is the standard definition for the $\neg$ operator.

4. $G, b, l \models \varphi_1 \lor \varphi_2$ iff $G, b, l \models \varphi_1 \lor G, b, l \models \varphi_2$

   This is the standard definition for the $\lor$ operator.

5. $G, b, l \models \varphi_1 \land \varphi_2$ iff $G, b, l \models \varphi_1 \land G, b, l \models \varphi_2$

   This is the standard definition for the $\land$ operator.

6. $G, b, l \models E_{I} \varphi$ iff $\exists l' \in N$, such that $l'$ is reachable
   from $l$ following $R_{I}$ transitions and $(G, b, l' \models \varphi)$

   Operator $E_{I}$ "routes" the query upward, until a satisfac-
tory location is found. This operator also exhibits the
paths between the query provenance and the resulting
location.

7. $G, b, l \models E_{I} \varphi$ iff $\exists l' \in N$, such that $l'$ is reachable
   from $l$ following $R_{I}$ transitions and $(G, b, l' \models \varphi)$
Operator $E_1$ "routes" the query downward, until a satisfactory location is found. This operator also exhibits the paths between the query provenance and the resulting location.

8. $G, b, l \models @p \varphi$ iff
   $$G, b, p \models \varphi \quad \text{for} \quad p \in N$$
   $$G, b, g(p) \models \varphi \quad \text{for} \quad p \in X$$
Operator $@p$ directly forwards the query to specific location $p$, where it will be evaluated. Location $p$ is either a real place or a local reference bound to a real place.

9. $G, b, l \models \downarrow x. \varphi$ iff $G, b^x, l \models \varphi$
Operator $\downarrow x$ binds the local name $x$ to the actual location where the query is evaluated. This operator provides a mechanism for creating location aliases. Note that aliases are local to a given query and cannot be reused by other queries to specify location identifiers.

5. **Expressiveness**

Let us now demonstrate the expressiveness of the proposed query language, by giving some examples based on the location model in Fig. 2, which is an extended hierarchical tree graph (i.e., the semantic model) of that depicted in Fig. 3.

1. A simple query would be to ask if a given entity is located at a given place, e.g., if we want to know whether the nurse is located in the elevator. The corresponding formula is $G, b, elevator \models nurse$, and is satisfied, as the the Boolean variable "Nurse" is true at place "Elevator". As the query provenance is set at "Elevator", the query is not routed to any other place in the location model.

2. As the language comes with basic Boolean operators, we can enrich the queries to express multiple conditions within a single formula. For example, one can ask if both the victim and the paramedic are inside the helicopter that has just returned from its rescue operation. The corresponding formula is $G, b, helicopter \models victim \land paramedic$. This is obviously satisfied, as both variables "Victim" and "Paramedic" are true at place "Helicopter".

3. Modal-logic operator $E_1$ routes the query upward in the location graph, searching for a place that satisfies the initial formula (see Fig. 4). For example, query $G, b, nurse \models E_1.floor.1$ searches for place "Floor 1" upward in the location model, by routing the query from the place where it was issued, i.e., "Nurse". As place "Nurse" is not labeled by "Floor 1", the query is evaluated again at the container place, "Elevator", and other places, until a satisfactory place is found, or not. As "Floor 1" is contained under "Hospital", and thus "Floor 1" is labeled by "Hospital", all places along the routing path ("Hospital" and "Floor 1") are returned by the query as satisfactory locations. As in the previous example, this set of places can be viewed as an itinerary from the query provenance to the requested location. In our example, one needs to be on the first floor of the hospital before entering the surgery.

4. Modal operator $E_1$ routes the query downward in the location graph, searching for a place that satisfies the initial formula (see Fig. 5). For example, the query $G, b, hospital \models E_1.surgery$ searches for the place "Surgery" downward in the location model, by routing the query from the place where it was issued, i.e., "Hospital". As place "Hospital" is not labeled by "Surgery", the query is evaluated again at the contained place, "Floor 1", and other places, until a satisfactory place is found, or not. As "Surgery" is contained under "Floor 1", and thus "Floor 1" is labeled by "Surgery", all places along the routing path ("Hospital" and "Floor 1") are returned by the query as satisfactory locations.

5. Modal-logic operator $@p$ directly forwards the query to specific place $p$. In other words, $@p$ gives random access to the place named $p$, where it will then be evaluated (see Fig. 6). Name $p$ is either a place identifier or a bound name. In the latter case, a prior name resolution has to be performed. This is particularly useful when a specific place needs to be checked independently of the query provenance. For example, consider the query $G, b, floor.1 \models @roof.helicopter$, issued on the first floor of the hospital. There, a surgical room is available for the victim who has been rescued by the helicopter. The query is forwarded to the roof's sublocation model and this checks if the helicopter has already landed. As the answer is positive, the medical staff can start preparing the intervention.

6. Modal-logic operator $\downarrow x$ binds variable $x$ to place $p$, where the query is evaluated. In other words, it creates
a brand new label, $x$, and assigns it to the current place. The formula $\downarrow x.\phi$ holds if and only if $\phi$ holds whenever the current place has been named $x$ (see Fig. 7). Within the same query, $x$ can then be used as an alias (or a tag) for $p$. For example, consider the following query: $(G, b, hospital \models (E_1 \downarrow emergency.victim) \land (@\text{emergency paramedic}))$. It combines all the operators we have described so far. The first component of the formula $(E_1 \downarrow emergency.victim)$ routes the query downward, searching for the place where the victim is located, namely “Helicopter”. Then $\downarrow emergency$ assigns the label “Emergency” to “Helicopter”, that can be used as a relative reference to the place “Helicopter”. After this, the second component (@emergency paramedic) retrieves the place “Emergency” and checks if the paramedic is there. Consequently, the formula is satisfied. This means that there is an emergency location in the hospital where a paramedic is assisting a victim, “Emergency” being an alias for “Helicopter”.

6. Design and Implementation

To evaluate the expressiveness of our query language, we built a prototype implementation of the framework, called Chequery. The query language itself is independent of any programming languages but the current implementation uses OCAML (version 3.09 or later).

6.1 System Functionalities

Our system provides query analysis for the underlying location model. The query evaluation’s algorithms were derived from hybrid logic-based model checking algorithms.

Chequery works in interactive mode, where the user can move along the model with basic navigation operators, and thus modify the context of the query, i.e., its provenance. Readers may relate this navigation-based system to the navigation inside tree-based file systems (i.e., enter or leave a folder, etc.) and execute location queries, based on the syntax specified in Sect. 4. The overall architecture for the Chequery system is outlined in Fig. 8.

6.2 Architecture

Tree-Based Database

The major issue with model checking is the so-called state-explosion problem. Because the Kripke structure represents the state space of the whole system, it is exponential in size to the system description. There have been several approaches to tackling the state explosions, in particular symbolic techniques where the Kripke structure is defined by binary decision diagrams. We did not need such techniques because we directly built the hierarchical tree graph with a mapping strategy. The first version of our system supports networked-file tree structures, where files correspond to places or entities in the location model and where links correspond to containment relationship between these places and/or entities.

Hierarchical Tree Graph

The Model Builder takes the input files and extracts their spatial organization to produce a corresponding modal structure, the hierarchical tree graph. Following the definition in Sect. 3, the modal structure is a Kripke system which:

1. Associates the files with places,
2. Links the places with two transition relations according to the containment structure, and
3. Labels all places with the basic logical propositions that hold at corresponding places, i.e., the contained places. Therefore, the edges of the graph can be seen as augmented places (with a logical view of their descendants), and we call these worlds. This is the appropriate term in model checking terminology.

\[
\text{type world} =
\begin{cases}
\text{mutable containe} : \text{place} ;
\text{mutable valid_prop} : \text{place list} ;
\text{mutable nested_loc} : \text{place list} ;
\end{cases}
\]

The Kripke form is $((\text{place world} (\text{worlds})) \ldots )$. The data structure is implemented in a Hashtable where the keys are the world’s labels and the values represent the world itself. Note that in the current implementation, after Model Builder has been executed, the supporting logic functions implemented in the HL Model Checker will use this modal system as the basis of all location queries.

Query Evaluation

The query language is defined by a recursive grammar
where syntax is analyzed is performed by the Query Parsing module. All query evaluations calls functions of the HL Model Checker, which maintains and updates two data structures, i.e., the context and the current query expression. Without the binding operator, the result of a query is simply a Boolean value. For example, consider the following query related to the hierarchical tree graph $G$ depicted in Fig. 3:

$$G, b, Hospital \models E_{\text{Nurse}}$$

The query evaluation is actually a matching procedure between graph $G$ and the hybrid logic-based expression $E_{\text{Nurse}}$. "Is there a nurse in the hospital ?". The answer would be either "Yes" (true) or "No" (false). More than the presence data which may be regarded as low-level information (and usually captured by a sensing system involving RFID tags), a location-based service aims to locate people and objects. Formulas are enriched by the binding operator and the result for the model checking algorithm is either a failure or an association with bound places. For example,

$$G, b, Hospital \models E_{\{\text{place}.\text{Nurse}\}}$$

we obtain, bound to the location place, whether the Nurse is present or not. "Is the nurse somewhere in the hospital ?". By returning place as the query result, the query becomes "Where is the nurse ?". The module Binders maintains the binding variables in a list of pairs variable/real location. Real locations can be directly matched or determined after a computation step, i.e., a sub-routine of the query evaluation.

We have so far shown that model checking brings interesting features to tackle some of the salient issues raised by query processing in ubiquitous computing systems, where a symbolic location model encapsulates the data model. Indeed, model checking is a powerful tool that provides sound theoretical foundations to reason about queries and helps us to implement a complete framework with built-in search algorithms performing over a decentralized data model. However, our approach points out the chasm between the theoretical design of the query language and its actual and pragmatic implementation in a real environment.

7. Experience

This section presents the contributions made by our query language from a software-engineering perspective. We have a project that provides a location/user-aware system to several museums to assist visitors. Although the current version has no query mechanisms, a prototype implementation of the next generation of the system is based on the query language presented in this paper, and we discuss how our query language can enhance the overall system.

7.1 Application to Indoor, RFID-Based Location-Aware Systems

Although the proposed query processing framework is general and may be adjusted to cope with many different location-based services (e.g., through language extensions), this section highlights its application to indoor environments, along with an RFID-based tracking system. Due to the symbolic location model, the query language obviously deals with qualitative information about the location-tree structure, rather than quantitative information (e.g., distances between physical entities). From our point of view, RFID-based tracking systems are a convenient application domain because they inherently handle the symbolic notion of place, through RFID readers’ coverage areas.

Such systems detect the location of physical entities, and usually deploy services bound to the entities at appropriate computing devices near to where they are located. A pair made of an RFID reader and an RFID tag thus define the containment relation presented in this paper, the former being the parent, and the latter, the child. When an RFID tag is detected inside an RFID reader’s coverage area, the underlying hierarchical tree graph is updated: the tag becomes the reader’s child, and it labels its parent (see Fig. 9). However, the language could be applied to geometric-based systems, where a prior refinement step derives symbolic relations from geometric coordinates.

7.2 Location/User-Aware System for Museums

The system is deployed at the National Museum of Nature and Science in Ueno (Japan). It was constructed with the framework presented by Satoh [15]. This system:

- Plays audio-annotations at specified spots when visitors stand at these, and
- Provides compound document-based GUIs for selecting audio-annotations according to visitors and spots. These GUIs visualize the positions of visitors.

All visitors are provided with a hat equipped with embedded active RFID tags to track their locations. The exhibition space is augmented with RFID tag readers that detect the presence of visitors within it. When visitors come sufficiently close to some objects (e.g., fossils of dinosaur), located at several spots throughout the exhibition, the system selects and plays sounds about dinosaurs according to
the combination of the spot, the visitors and their routes in the room. The upper right image (2) in Fig. 10 depicts the equipment used during the experiment, i.e., a loud speaker and an RFID reader were positioned in front of the fossils.

The query processing engine is independent of any hardware or computing devices. Within the current system in the museum, the devices are not overloaded with query processing tasks, as query processing is done on proxies where the application is deployed. Our solution does not make any assumptions about the deployment of the application, as that is beyond the scope of this paper.

7.3 Query Language for Middleware Support

Even if the application is fully functional, its future extensions are made more difficult from an engineering point of view. The current location-based service (i.e., the appropriate audio-based information) is delivered to the users (i.e., the visitors) without any query mechanisms, in a reactive manner, i.e., the fact that a user has been detected entering a spot triggers the corresponding loud speaker. This event-based mechanism is dependent on the underlying sensing systems and it is specifically implemented for that application. Instead, our query processing framework could be integrated as a component available at the middleware level. In the current version of the system, developers need to manually construct the query mechanisms, based on simple events, e.g., entering or leaving a spot. Our query processing framework is useful when we want to extend the functionalities of such a location-based system, without increasing its complexity.

Monitoring

Collecting information about the activities of visitors is an ongoing process that is relevant to museum curators. The information delivered by monitoring is analyzed and used to improve and enhance their collections. In our location-based service, the query processing framework could be used to determine visitors’ itineraries inside museums. For example, visitors may not wish or be able to visit all the exhibits, or visit some of these more than once during their stay. Our query language can then enhance the existing information system of the museum, by providing access to location information.

More Complex Triggers

The reactive location-based service, based on simple location triggers, points out the lack of a query language. Let us consider the following example. When a visitor enters spot $S$ at time $t_0$, the corresponding speaker delivers some audio-based information for a given amount of time $T$, i.e., the event trigger is pulled. If a second visitor enters the same spot $S$ at time $t_1 < t_0 + T$, nothing happens because the event triggers have not yet been released. The information that a spot has already been visited can be accessed by simple location queries using our framework. Our language provides a higher level of control to manage the event triggers, and makes it easier to develop more complex behaviors. In the system that is presented, this management is done in a centralized way, by a software component that knows the entire location model. Thus, individual spots are only dedicated to delivering location information. Our query language could be used by the spots themselves to obtain and access location-based information to provide more local views, and to trigger specific actions.

Performance Evaluation

Although the system has not yet been tested in a real environment yet, we simulated three scenarios, each of which corresponded to a given number of visitors inside the exhibition spaces of the museum (i.e., the number of places in the hierarchical tree graph). We generated nine queries of different sizes, based on the language presented in Sect. 4. Queries were evaluated locally on a Pentium III/800-MHz laptop. The results are shown in Fig. 11.

The current system was not optimized for performance; however, the cost of handling each query was shorter than $20\text{ms}$. We believe that this cost is acceptable for a

\[\text{Fig. 11} \quad \text{Ability of the system to evaluate nine queries of different sizes.}\]
location-aware system used in rooms or buildings.

Application Domain

Location-aware systems can provide numerous services, and we presented one possible application of the proposed query language to a user-aware guide system in museums. However, the language was not limited to this single application, and could be used in a variety of fields, e.g., (1) stock management in warehouses and (2) in home appliances. We have given one example query below for each of these two applications:

1. To find a given item (serial number 2007KBP0029) in a warehouse, use the following query, that returns the stock to where the item is located:
   \[ G, t, Warehouse \models E \downarrow stock.2007KBP0029 \]

2. Hybrid logic-based queries can be used to send commands to control home appliances. To save energy, the following query can turn off the lights in a kitchen, if nobody is there:
   \[ G, b, House \models @Kitchen((\neg someone \land light.on) \Rightarrow turn\_light\_off) \]

8. Related Work

As we propose a logic-based approach to building query processing mechanisms for ubiquitous computing, we investigated other logic-based approaches, and focused on the underlying connection between location modeling and query processing. Database researchers handle this connection by modeling moving objects and their dynamic locations in a database. They provide spatial data management systems, moving object databases, and query mechanisms that support geometric location models. Moving object databases integrate spatio-temporal data that has been derived from moving objects, e.g., people, cars, and hurricanes.

In the DOMINO project [17], Wolfson et al. focused on the trajectories of moving objects and proposed a set of operators, both spatial and temporal, to query their locations. However, their data model was built on a centralized architecture and could not represent any relations between objects and places as in the containment relation. Still, within the context of moving object databases, Gütting and Schneider [9] proposed a modal logic-based approach to query processing. They defined a Future Temporal Logic (FTL) to reason about future events and thus retrieve objects in the future\(^1\) using some SQL-like query language extensions. However, we believe they had four drawbacks: (1) the location data were stored in a single data repository, (2) their approach seemed to be dedicated to outdoor environments, along with the Global Positioning System (i.e., a geometric location model), (3) object trajectories had to essentially be known by the query processor, and (4) relative locations could not be expressed.

\(^{1}\)By considering estimates about their trajectories.

9. Conclusion and Future Works

We presented a novel approach to handling location query processing in ubiquitous computing environments. Starting from the symbolic location model, along with the spatial containment relation between places, we defined an extended Kripke structure as a semantic and data model for a hybrid logic-based query language. We demonstrated that our logic, which is defined by a small set of operators, provides a great deal of expressiveness to common location queries. We also designed and implemented a prototype system based on the framework. The approach we presented in this paper is general and may not be limited to location query processing. Actually, it has a broad application domain and can be applied to any hierarchical data structure.

Finally, we would like to point out further issues that need to be resolved. Although there are many directions for future work, we will only mention some of these. First, from a technical viewpoint, we assumed a static topology for the location model when issuing queries. As a result, we focused on queries related to the spatial organization of data. However, users, objects, and/or services are mobile in ubiquitous computing environments, adding further dynamicity to the data model, decentralized by nature. The topology of the location model itself, i.e., the spatial configuration, is dynamic. Thus, we need to extend our language to cope with these temporal changes. Second, privacy is an important issue in all information-based applications, and is a fortiori in any ubiquitous computing application where the need for information about people’s locations is increasing. Our query language especially deals with global location references and direct access that may lead to privacy violations. Privacy should be regarded as part of the core of the query processing framework, and our model should be able to support access control mechanisms.

References


Christian Hoareau earned his B.Sc. and M.Sc. in Computer Science from the University of La Réunion, France (2003) and from Pierre and Marie Curie University, France (2005), respectively. Since 2006, he has been a Ph.D. student at the Graduate School for Advanced Studies, National Institute of Informatics, Japan. His research interests include model checking, location-dependent query processing and system software for ubiquitous computing.

Ichiro Satoh received his B.E., M.E, and Ph.D. degrees in Computer Science from Keio University, Japan in 1996. From 1996 to 1997, he was a research associate in the Department of Information Sciences, Ochanomizu University, Japan and from 1998 to 2000 was an associate professor in the same department. From 2001 to 2005, he was an associate professor in National Institute of Informatics (NII), Japan. Since 2006, he has been a professor of NII. His current research interests include, distributed and ubiquitous computing. He received IPSJ paper award, IPSJ Yamashita SIG research award, and JSSST Takahashi research award. He is a member of six learned societies, including ACM and IEEE.