A Spaces-Based Platform Enabling Responsive Environments

Daniela Micucci, Alessio Vertemati, Francesco Fiamberti, Diego Bernini, and Francesco Tisato

Department of Informatics, Systems and Communication
University of Milano - Bicocca
Milano, Italy
Email: {daniela.micucci, alessio.vertemati, francesco.fiamberti, diego.bernini, francesco.tisato}@disco.unimib.it

***Abstract*—Space Integration Services is a software communication platform that enables the seamless integration of sensors, actuators, and application-logic components through a multi-space model and a spaces-based publish/subscribe mechanism. The underlying model is based on finite spaces only, thus the application-logic components that need to reason on spaces like the geodetic or the Cartesian ones, are obliged to use a discretized version of those spaces and to maintain the correspondence between "real" locations and their discretized representation if they are interested in exploiting the services offered by the platform. To compensate for these limitations, the platform has been extended in order to add support for spaces with an infinite number of locations, (e.g., spaces described by continuous coordinates, such as geodetic or Cartesian spaces). Such extension is essential for the use of SIS also as enabling platform for outdoor pervasive computing systems thanks also to the wide spread of outdoor localization systems. This paper presents the new conceptual model that results from a generalization of the existing one valid for finite spaces only. To encompass both finite and infinite spaces, the new model moves the primitive concept of elementary position from *location* to *spatial context* (as a set of locations). Several kind of spatial contexts have been introduced to offer a (quite) complete set of localization typologies, for example, from a single position to a set of positions described by a function. The model has been turned in a prototypal implementation realized by means of an additional layer on top of the finite-space version. Such an implementation has been experimented in a real case scenario dealing with a parcel distribution company. Finally, the performance of the prototype is then compared to the one of the finite-space version in a series of experimental tests.**

Keywords–infinite spatial models; spaces-based communication; software architecture; location-aware; responsive environments.

I. INTRODUCTION

Space Integration Services (SIS) [1] is a platform that enables pervasive computing [2], supporting information flows between devices and applications. Pervasive computing aims at simplifying the everyday life through digital environments that are sensitive, adaptive, and responsive to the user's needs. A pervasive computing system requires the perception of the context in which the user operates to provide a richer and expanded mode of interaction, in addition to an intelligence for performing actions on the environment. From a technological point of view, pervasive computing relies on responsive environments. The term responsive environment [3] refers to physical environments enhanced by input devices (e.g., sensors or cameras) and output devices (e.g., displays, lights, motors). Input devices capture stimuli from the environment, whereas output devices execute actions on the environment given a predefined set of commands.

Responsive environments are, therefore, able to perceive and respond to users thanks to the presence of a computer system that receives data from the sensors (input stream) and sends commands to the actuators (output stream). For example, an application may locate users onto the cartographic representation of the city and may also receive data from light sensors (input stream) about their state (e.g., on and off). On the basis of established rules, an application could send commands to the street lights (output stream). A rule can state that in nighttime, if a user is close to a street light (within a distance of ten meters), the street light should be turned on. When no one is close to the street light, then the street light should be turned off. This simple example emphasizes how responsive environments require establishing information flows among the devices and the applications. SIS fulfills the above requirement by enabling a seamless integration of sensors, actuators, and application-logic components through a multi-space model and a spaces-based publish/subscribe mechanism. It provides various spatial models that can be used by applications to represent location-related information in order to support complex representations of the environment with a focus on location-aware systems. In this regard, different spatial representations can be put in appropriate correspondence to describe the localization according to different visions of the environment. This allows devices and end-user applications to reason on an own representation of the environment and thus are not obliged to share a common representation. For example, an application that localizes persons inside a building reason in terms of rooms and passages among them, whereas the devices reason in terms of a Cartesian representation of the controlled area. From the knowledge of the authors, platforms offering interoperability rely on a shared view of the space. Moreover, most of them provide spatial representations of the physical environment only. On the opposite, SIS allows to model any kind of space, be it physical (e.g., a geographical area) or logical (e.g., an organization chart).

SIS originates from a preliminary version that supported finite spaces only (symbolic models and grid models) [4]. Such kind models, even if discrete, were nevertheless sufficient to create indoor pervasive computing applications. For example, switching on and off of lights depending on the presence of people in the rooms. Recognition of people and then setup the environment on the basis of their optimal configurations, and so on. With the increase of accuracy and precision of localization sensors [5] and with the need to include geographical-related spatial models to support outdoor applications also, a revision of the platform was required. The revision aims at in-

troducing mechanisms to deal with spatial representations that contain a potentially infinite number of locations, for example those described by continuous coordinates, such as geodetic and Cartesian spaces, but also unbounded grids described by discrete coordinates. For example, such a platform will support applications that require to locate persons outdoor (they need to reason on a geographical space) or that require to define the fine-grained trajectory of a mobile robot (they need to reason on an Euclidean space).

Finally, the new version of SIS was designed from the previous one in order to reuse the base mechanisms that support information flows and that constitute the core of the platform. Those mechanisms, which are based on finite spaces, are then very efficient. The idea was then to yield the same mechanisms, properly managing flows between infinite spaces as if they were flows between finite spaces.

The main contributions of this article are a deep description of both the conceptual model and the platform reifying it. Moreover, since the implementation has been completely revised and completed with respect to the results presented in [1], some of its interesting issues will be discussed. Finally, tests have been reformulated in order to consider different configurations with different hardware.

The paper is organized as follows. Section II presents the overall SIS conceptual model. Section III summarizes the principal services that SIS should provide to applications. Section IV discusses the use of the model in an application scenario. Section V presents some issues related to the actual implementation of the platform. Section VI presents the results of several tests aimed at estimating the performance of the proposed extension with respect to the existing SIS implementation considering different configurations with different hardware. Section VII reviews related works. Finally, conclusions and future developments are presented in Section VIII.

II. Conceptual model

The conceptual model of the finite space-based SIS model was based on the assumption that *spaces* are finite sets of *locations* built from *spatial models* (e.g., graph spatial model, name spatial model, and grid spatial model). Non- empty sets of locations belonging to a space are named *spatial contexts*. In such a model, locations play a crucial role since the existence of both spaces and contexts is subject to the existence of the set of locations that constitutes them. In other words, spaces and spatial contexts are defined as the collection of all the locations that constitute them. This structural constraints is clearly impossible to fulfill when dealing with infinite spaces, that is, spaces that contain a potentially infinite number of locations. Thus, the conceptual model has been revised around a new definition of spatial context that becomes the elementary localization building block.

A. Space and Spatial Model

A *space* is a set of *potential location*s, that are all the locations that could be theoretically considered in that space. For example, if the Milan subway lines are described by means of a graph in which nodes model stops, then the potential locations are all the stops of all the lines. On the other hand, if a Cartesian space is used to localize entities within a room,

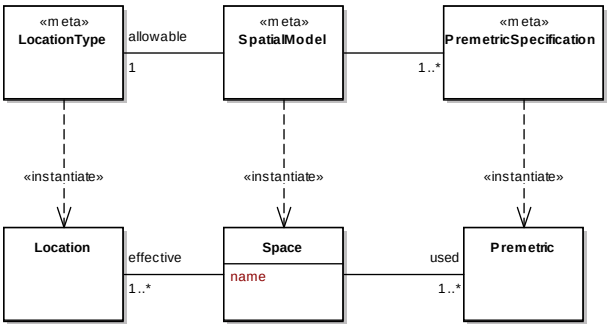


Figure 1: Core concepts: meta representation and corresponding instances

then the potential locations are every point in $\mathbb{R}^2$ of the area delimited by the room perimeter.

Applications, when dealing with a space, explicitly manage *effective location*s, which are a subset of space's potential locations. For example, an application that monitors the position of trains traveling on the Milan Red Line subway will only deal with locations that represent the stops on the Red Line. On the other hand, an application that calculates the trajectory of a mobile entity will only explicitly consider a finite number of locations in the Cartesian space, that is, the locations belonging to the trajectory.

Spaces and effective locations are respectively built from spatial models and location types. A *spatial model* specifies the type of *allowable location*s (*location type*), the way in which locations are arranged, and at least one premetric that can be applied to a pair of locations (*premetric specification*). The premetric defines the distance between two locations as a positive, non-zero number if the two locations are distinct, and zero if the locations are the same. A *location type* specifies the structure of potential locations (e.g., a couple of double values).

The key aspect, shown in Figure 1, is that location types, spatial models, and premetric specifications are meta-level concepts (≪meta≫ stereotype) because, according to them, it is possible to instantiate locations, spaces, and premetrics respectively, that are base-level concepts (i.e., they are the element an application manages when dealing with localization issues). The direction of arrows in Figure 1 emphasizes that base elements originate from meta elements. In an object oriented language, it is possible to compare meta-level elements to classes and base-level elements to objects instantiated from classes. In other words, space and spatial model, location and location type, premetric and premetric specification respectively belong to different levels of abstraction.

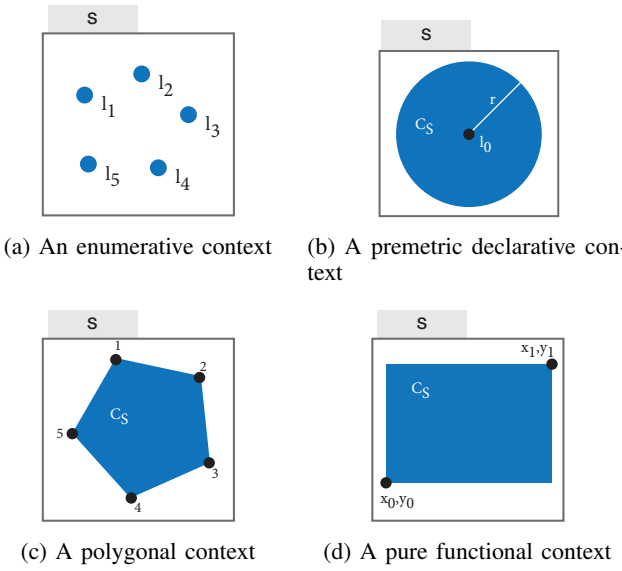
The definition of space applies both to cases in which the set of locations is infinite and to cases in which it is finite. In fact, with the term *potential* locations we refer to all the locations that can be defined according to the spatial model. Therefore, we distinguish potential locations from effective locations that are the actual ones used by applications.

Two spatial models representing spaces with potentially infinite number of locations have been defined: the geodetic and the n-dimensional Cartesian. The geodetic spatial model

defines locations as geodetic coordinates (i.e., latitude and longitude) in a geodetic coordinate reference system [6]. An applicable premetric is the orthodromic distance [7], which is the shortest distance between two points expressed by geodetic coordinates on the surface of a sphere. The orthodromic distance is also a metric. The n-dimensional Cartesian spatial model represents an Euclidean space $\mathbb{R}^n$. In this model, locations are represented by ordered tuples of real numbers in an orthogonal Cartesian reference system. A premetric for the Cartesian spatial model is the Euclidean distance, which is also a metric.

These examples highlight that a location in an infinite space can be identified with a potentially infinite precision, as in the nature of the coordinate system based on real numbers.

The spatial models defined in the previous finite space-based SIS model (i.e., graph, grid, and name spatial models) are declinable as special cases of the new definition of space in which the allowable locations are finite in number and are represented with finite precision. For example, in a grid space, allowable locations are the cells within the limits of the grid and are represented by their respective indexes defined on the set of integers.



(a) An enumerative context (b) A premetric declarative context

(c) A polygonal context (d) A pure functional context

Figure 2: Different types of spatial context

B. Spatial Context

In the finite space-based SIS model spatial contexts have been introduced in order to handle the selection of subsets of locations. The definition of spatial context has been refined to be also applicable in spaces with potentially infinite number of locations.

A *spatial context* C_S (simply *context* in the following) is a subset of potential locations of a space S. It is defined by a set of effective locations termed *characteristic locations* in S and by a *membership function* that states if a given location of S belongs to the context. Essentially, the membership function is a boolean function that is true when a location is in the spatial context. According to the membership function used, the following kinds of contexts have been identified: *enumerative*, *premetric declarative*, *polygonal*, and *pure functional*.

An *enumerative context* is a context in which the set of characteristic locations is non-empty and the membership function is based on the standard belonging relationship defined in set theory. The locations belonging to an enumerative context are identified through the enumeration of the characteristic locations. For example, given a space S, an example of enumerative context is defined by the set $C_S = \{l_1, l_2, ..., l_5\}$ of characteristic locations (see Figure 2a). For example, such kind of context can be useful to specify the room in which a person has been localized.

A *premetric declarative context* is a context in which the set of characteristic locations contains one location only and the membership function is a premetric one. Thus, the locations belonging to this kind of context are all the locations within a given distance from the characteristic location in terms of the premetric function (see Figure 2b). For example, such kind of context can be useful to specify the detection area of a RFID reader.

A *polygonal context* is a context in which the characteristic locations are the vertexes of a polygon, and the membership function indicates the inclusion of a location in the region corresponding to the polygon itself (see Figure 2c). The locations belonging to a polygonal context are all the locations within the area delimited by the polygon. For example, such kind of context can be useful to specify a delimited area inside a building or the field of view of a camera.

Finally, a *pure functional context* is a context in which the set of characteristic locations is empty and the membership function is defined by using mathematical expressions defined in terms of the space coordinate system. For example, consider a Cartesian bi-dimensional space S. A context can be defined by the following membership function: for all locations (x, y) in S and for any given location (x_0, y_0) and (x_1, y_1),

$$f(x,y) = \begin{cases} \text{true} & \text{if } x_0 < x < x_1 \text{ and } y_0 < y < y_1 \\ \text{false} & \text{otherwise} \end{cases} \quad (1)$$

(see Figure 2d). Obviously, this example aims at easily explaining the context idea. Indeed, it can be transformed into a polygonal one by interpolating the coordinates of the points that represent the extremes of the allowable values. Such a context should be used to face situations in which contexts are more complex and cannot be approximated with a polygonal one.

C. Projection

Related to the concept of space, is the concept of projection, which is widely used in geometry and in algebra.

The *projection function* allows the transformation of locations belonging to a space, called the source, in locations belonging to another space, called target. The target space can be defined according to the same spatial model of the source, or to a different one.

The transformation of the geodesic representation of the Earth to any other cartographic representation, the application

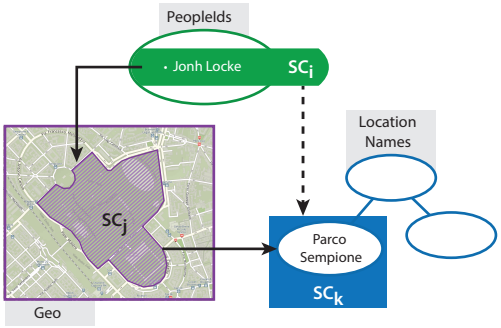


Figure 3: Explicit and implicit mappings

of a scale factor to a Cartesian representation, the transformation of a tri-dimensional environment onto a bi-dimensional surface, are all examples of the application of a projection function.

It is possible to observe how the application of the projection function to a region of the space may be useful in providing the representation of that region in agreement with the spatial model of the target space.

The projection function is used to define another kind of context termed *projective context*: given a source spatial context C_S defined in a source space S and given a target space T, the result of a projection is a spatial context C_T on the target space T containing the locations that are obtained by applying a projection function f to all the locations in the source context C_S.

D. Mapping

Mappings relate different spaces. For example, a mapping can relate an area of a Cartesian space (representing the plant of a building) to a node of a graph (representing a synoptic view of the same building). Mappings are key concepts because, as it will be explained afterward, they enable the communication among components, even if they rely on different spaces. Three kinds of mapping have been defined: *explicit*, *projective*, and *implicit*.

An *explicit mapping* is an ordered pair of contexts defined in different spaces (possibly based on different spatial models): given the two spaces S_1 and S_2 with $S_1 \neq S_2$, the ordered pair (SC_1, SC_2) is an explicit mapping between the contexts $SC_1 \subseteq S_1$ (source) and $SC_2 \subseteq S_2$ (target).

Figure 3 shows two examples of explicit mappings between the context SC_j of the *Geo* space (a geodetic spatial model) and the context SC_k of the *LocationNames* space (a graph spatial model), and between the context SC_i of the *PeopleIds* space (a name spatial model) and the context SC_j of *Geo*.

The target context may be defined independently of the source context. But when the target context is the result of the application of a projection to the source context, the mapping is termed *projective mapping* and is fully determined by the source context and the projection function.

Let SM be the set of all the defined explicit and projective mappings, and let SC_a and SC_b be contexts defined in different spaces. The *implicit mapping* (SC_a, SC_b) is derived if there exist n contexts $SC_1, \ldots, SC_n$ such that $(SC_a, SC_1), (SC_1, SC_2), \ldots, (SC_n, SC_b) \in SM$ for $n >= 1$.

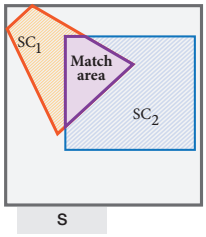


Figure 4: Direct matching

Figure 5: Indirect matching

In Figure 3 the dotted arrow represents the implicit mapping (SC_i, SC_k) between the contexts SC_i of *PeopleIds* and SC_k of *LocationNames*. It is derived from (SC_i, SC_j) and (SC_j, SC_k) mappings.

E. Matching

A *direct matching* occurs when the intersection between two spatial contexts defined in the same space is not empty. For example, given a space S defined in $\mathbb{R}^2$ that models a room in a building, a pure functional context SC_1, and a polygonal context SC_2 both defined in S. A direct match occurs since the intersection between SC_1 and SC_2 is not empty (see Figure 4).

Let SC_1 and SC_2 be spatial contexts defined in different spaces. An *indirect match* between SC_1 and SC_2 occurs when there exists a mapping (explicit or implicit) (SC_a, SC_b) such that the intersections between SC_1 and SC_a and between SC_2 and SC_b are both not empty (see Figure 5).

III. Services

The concepts introduced in Section II are the basis of several services that SIS offers to applications. They can be divided into two main groups: 1) management and inspection of spaces' configurations; 2) communication support through spatial contexts (spaces-based publish/subscribe).

A. Space Structure Support

Before applications may reason on spatial contextualizations, spaces must be defined and properly configured. For this purpose, a set of services is provided.

When defining a space, applications must initially choose the appropriate spatial model (i.e., the spatial model that best fits the needed space). For example, if an application needs to plan the movements of a mobile entity inside a building in terms of sequences of rooms it has to traverse before reaching

final destinations, then such an application may rely on a synoptic representation of the building built according to the graph spatial model; on the opposite, if another application is in charge of fine tuning the movement of the mobile entity, then it may rely on a Cartesian representation of the building built according to the Cartesian spatial model.

Once the spatial model has been chosen, it must be configured so that the resulting space will contain all the potential locations (i.e., all the locations an application possibly needs to use). For example, the synoptic view of the building can be obtained by specifying the list of nodes and arcs constituting the building; the Cartesian representation can be obtained by specifying the boundaries of the area containing the building.

From the above consideration, space definition is supported by the primitive

```
defSpace(SpaceName, Model, Parameters)
```

where `SpaceName` is the unique name for the new space, `Model` specifies the spatial model from which building the space, and `Parameters` are configuration information that depends on the spatial model. For example

```
defSpace(U14, GraphSpatialModel,
    <{r1, r2, r3}, {r1−r2, r1−r3}>)
```

defines a graph space named `U14` whose nodes are r_1, r_2, and r_3, and whose arcs are r_1−r_2 and r_1−r_3.

Once spaces exist, explicit mappings among spatial contexts can be specified through the

```
defMapping(Source, Target)
```

primitive, where `Source` and `Target` are respectively the source and the target contexts. Each context definition is specified by `<SpaceName, Structure>` where `SpaceName` is the unique name of the space in which the context is being defined, and `Structure` specifies the context typology, the set of characteristic locations, and the membership function. For example `<U14, <ENUM, {r1, r2}, ∈>>` defines an enumerative context in the `U14` space whose characteristic locations are r_1 and r_2 and the membership function is the belonging function of the set theory. Suppose that a Cartesian space named `1St` has been defined representing the first floor of the U14 building, then

```
defMapping(
    <U14, <ENUM, {r1}, ∈>>,
    <1St, <POLYG, {p1, p2, p3, p4, p5}, fin>>)
```

specifies a direct mapping. It may be interpreted as follows: room r_1 in U14 building is represented by the context `<U14, <ENUM, {r1}, ∈>>` in the graph space `U14` and by `<1St, <POLYG,` p_1, p_2, p_3, p_4, p_5, f_{in}`>>`, a polygonal context defined in the Cartesian space `1St`.

The above primitives can be used both when configuring SIS spaces before any application starts using the services, and during the normal execution of the applications. Moreover, SIS provides a set of primitives that allow applications to reason about spatial configurations. For examples:

- `getSpaces`: returns the list of all the defined spaces.
- `getSpaceInfo`: given the name of a space, returns all the related information (i.e., space typologies and parameters).
- `getMappings`: given two contexts, returns (if any) all the defined mappings.
- `getMapped`: given a context, returns (if any) all the defined mappings in which the context is involved.
- `getProjection`: given a context, a target space, and a projective function, returns the projected context.

The complete set of primitives that a platform reifying SIS model should provide is presented in the Figure 9 in Section V, which discusses the implementation we developed.

B. Communication Support

As previously introduced, SIS enables information flows relying on the publish and subscribe mechanism: applications *publish* information on spatial contexts, and *subscribe* on spatial contexts for the asynchronous reception of the published information. The published information is called *thematic information*, which is domain dependant and is not in any way interpreted by SIS.

Contexts of any kind may be defined on any spatial model: from the enumerative to the pure functional one. The choice of the context type depends on application domain issues. For example, in a situation of emergency, the State Forestry Department may report the area affected by a fire by publishing a thematic information (specifying information dealing with the fire) in a polygonal context (representing the affected area) defined in the geodesic space. On the opposite, if a room is equipped with a very accurate localization system (e.g., Ubisense), it may publish entity positions (the thematic information) exploiting enumerative contexts that contain the single position inside the Cartesian space modeling the room.

An application performs a publication via

```
publish(info, <C1, C2, ...>)
```

where `info` is the thematic information and `<`C_1, C_2, `...>` is a list of spatial contexts in which the information has been localized.

An application performs a subscription via

```
subscribe(C1, C2, ...)
```

where `<`C_1, C_2, `...>` is the list of spatial contexts in which the application declares its interests.

When an application performs a publication, the enclosed thematic information is received by all the applications that previously performed a subscription such that at least one of its contexts *matches*, either directly or indirectly, a context of the publication. The thematic information is enriched with all the contexts that contributed to the matching.

Referring to the example in Figure 6a, $\texttt{Component}_2$ subscribes to a context defined in the `Floor2D` space built from a Cartesian spatial model. The subscription context is polygonal (the hexagon in Figure 6a). $\texttt{Component}_1$ makes

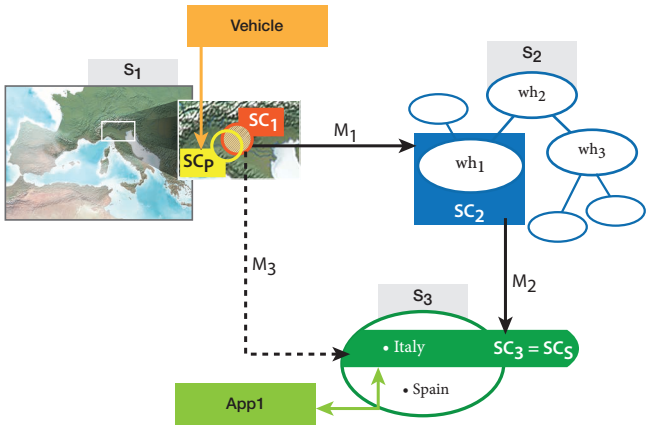


Figure 7: Example scenario

a publication on the same space Floor2D. The context is premetric (the circle in Figure 6a) and intersects the polygonal one. Thus, a direct match occurs and $Component_2$ receives the thematic information info enriched with additional information that includes, in addition to the time instant in which the match occurred, the complete contextualization, that is, all the contexts contributing the matching (the polygonal and the premetric).

In Figure 6b $Component_2$ and $Component_1$ reason on different spaces. $Component_2$ subscribes to an enumerative context (the highlighted node labeled "Living room" in Figure 6b) in the space FloorGraph space built from a graph spatial model. $Component_1$ makes the same publication as in 6a. Since a mapping between the publication and subscription contexts has been defined, an indirect match occurs so that $Component_2$ receives the thematic information info enriched with all the additional information (the matching time instant and the complete contextualization).

IV. Application Scenario

This section will apply the concepts introduced in both Section II and Section III considering an exemplified scenario, in particular the geodetic space and the mappings between infinite spaces and finite ones. Consider a parcel distribution company with warehouses distributed in Europe (for the sake of simplicity, we consider only six warehouses distributed in Italy and Spain). The company exploits vehicles to distribute parcels. Each vehicle is equipped with a device (mounting a GPS) that periodically notifies its position. Before reaching the final warehouse, vehicles can pass through intermediate warehouses. Each time a vehicle enters a warehouse, different operations have to be performed according to the country's rules, including the decision of the next warehouse the vehicle has to reach.

As depicted in Figure 7, the required spaces are: S_1, a geodetic spatial model covering the involved countries; S_2, a graph spatial model where each node corresponds to a specific warehouse (identifiers wh_1, wh_2, ..., wh_6) and each arc connects the warehouses that can be reached without any intermediate stop; finally, S_3, a name spatial model containing the identifiers of the two countries (Italy, Spain).

Explicit mappings are required from S_1 to S_2 with the aim of localizing each warehouse in the geodetic space. Mappings are in the form

$$M = \langle\langle S_1, \langle PREM, \langle [lat, long], radius, d\rangle\rangle\rangle, \langle S_2, \langle ENUM, \{wh_i\}, \in\rangle\rangle\rangle$$

where the target is an enumerative context containing a node of the S_2 space (i.e., a warehouse wh_i) and the source is a premetric declarative context specifying the area in S_1 where the warehouse wh_i is located. For example, the mapping $M_1 = \langle SC_1, SC_2\rangle$, where $SC_1 = \langle S_1, \langle PREM, \langle [45.523653, 9.219436], 50, d\rangle\rangle\rangle$ and $SC_2 = \langle S_2, \langle ENUM, \{wh_1\}, \in\rangle\rangle$. In Figure 7 spatial contexts on S_1 have been hugely enlarged for visualization purposes. Moreover, explicit mappings are required from S_2 to S_3 with the aim of localizing each warehouse in its country. Mappings are in the form

$$M = \langle\langle S_2, \langle ENUM, \{wh_i\}, \in\rangle\rangle, \langle S_3, \langle ENUM, \{country\}, \in\rangle\rangle\rangle$$

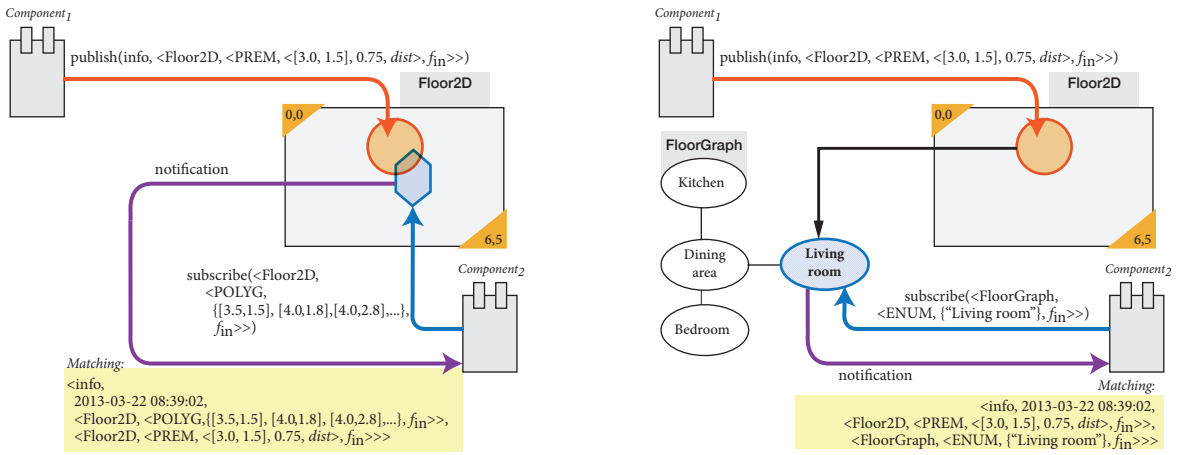
where the source is an enumerative context containing a node of the graph (i.e., the identifier of a warehouse) and the target is an enumerative context containing the identifier of the country. For example, the mapping $M_2 = \langle SC_2, SC_3\rangle$, where $SC_2 = \langle S_2, \langle ENUM, \{wh_1\}, \in\rangle\rangle$ and $SC_3 = \langle S_3, \langle ENUM, \{Italy\}, \in\rangle\rangle$. Finally, six indirect mappings are derived: their source contexts are the source contexts of the explicit mappings defined between S_1 and S_2, and their target contexts are the target contexts of the explicit mappings defined between S_2 and S_3. For example, $M_3 = \langle SC_1, SC_3\rangle$.

Two applications are required (App_1 for Italy and App_2 for Spain), each implementing the local rules. Each application subscribes to the appropriate country to be notified when a vehicle enters a warehouse in the country of competence. For example, App_1 performs a subscription to the enumerative context SC_S in S_3 containing the location Italy (i.e., $SC_S = \langle S_3, \langle ENUM, \{Italy\}, \in\rangle\rangle$). Periodically, the vehicles make publications in the S_1 space, thus sharing their position with all the interested applications. Publications are in the form

$$Pub = \langle vehicleID, \langle S_1, \langle PREM, \langle [lat, long], radius, d\rangle\rangle\rangle\rangle$$

where vehicleID is the thematic information that identifies the vehicle, and $\langle S_1, \langle PREM, \langle [lat, long], radius, d\rangle\rangle\rangle$ is a premetric declarative context specifying the position of the vehicle identified by vehicleID in the geodetic space. For example, publication $Pub_i = \langle 12345, SC_P\rangle$, where 12345 is the identifier of the vehicle that performs the publication and $SC_P = \langle S_1, \langle PREM, \langle [45.51788, 9.214071], 20, d\rangle\rangle\rangle$ is the location in which it has been localized.

When the vehicle 12345 makes the publication Pub_i, App_1 is notified because an indirect match occurs. Indeed, the following conditions result true: SC_P intersects SC_1, SC_1 is indirectly mapped to SC_3 (M_3 mapping), and SC_S intersects SC_3. When notified, App_1 receives the thematic information 12345 enriched with all the contexts that enabled the matching, that is, SC_P, SC_1, SC_2, SC_3, and SC_S. This way, App_1 is aware of the warehouse in which the vehicle 12345 is, and, if it is able to manage S_2, it can inspect the graph in order to



(a) Publish and subscribe in the same space

(b) Publish and subscribe in different spaces

Figure 6: Publication and subscription services

decide the next stop for the vehicle.

The finite-space version of SIS could not handle this application scenario for the presence of a geodetic space that is infinite. In any case, the operations used in this scenario are exactly the same as they would be used with the finite-space version to maintain backwards compatibility. What differs is how these operations are managed in the case involving infinite spaces. Obviously, using the finite-space version of SIS to handle this applicative scenario would have involved the use of a grid space to represent the geodetic space. The application would have had to manage all the correspondences between geodetic positions and locations on the grid space.

V. Implementation

This section will present issues related to the implementation of the current version of SIS, focusing on the new aspects that allow SIS to manage both finite and infinite spaces.

A. Layering

The developed prototype is based on the implementation of the finite-space version of Space Integration Services.

The infinite-space extension of SIS is organized according to two different software layers [8] on top of the finite-space SIS Core layer. Figure 8 shows the resulting structure.

The *Distributed Access* layer exhibits three different mechanisms allowing the interaction with the platform. The *Web Services* interface provides applications access to the platform features by means of the Representational State Transfer (REST) paradigm. The Web Services offers the SIS services through the `SISManager` class that implements the interfaces shown in the class diagram presented in Figure 9:

- `SpacesManagement` allows the creation, update, and deletion of spaces, as well as the management of the projections defined between spaces.

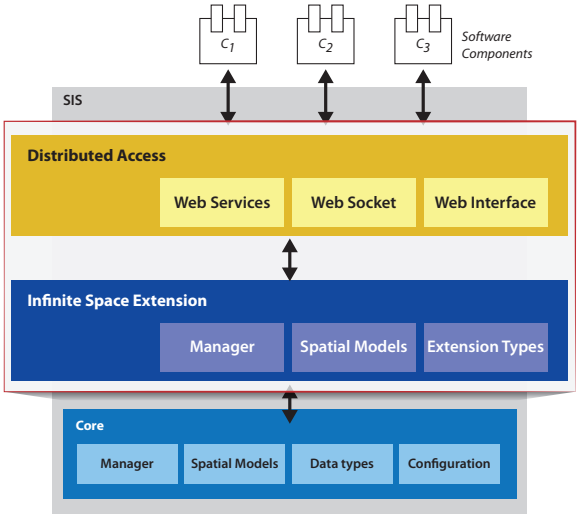


Figure 8: The extended SIS Structure

- `SpacesInspection` allows an application to explore the currently configured spaces, as well as the projections defined on those spaces.
- `MappingsManagement` provides an application with the necessary primitives for creating, editing, and removing mappings.
- `MappingsInspection` exposes the primitives for the exploration of the mappings (explicit and implicit) defined on the SIS instance; similarly to `SpacesInspection`, it offers a notification when a mapping is changed.
- `SpacesPublishSubscribe` exposes the primitives for the publication and subscription operations, as well as the necessary methods to enable the push notifications towards the applications.
- `MatchingNotifier` enables the asynchronous no-

tification (to applications) of new thematic information located in the subscribed contexts through the observer-observable pattern.

- `MappingChangeNotifier` enables the asynchronous notification (to applications) of mapping changes through the observer-observable pattern.

Web Sockets [9] are a recent W3C standard for two-way asynchronous communication in the context of web applications; this technology makes available asynchronous communication of notifications to the applications. Finally, the *Web Interface* visually exposes all the primitives of the platform and allows for the configuration of the users with their access permissions.

The *Infinite Space Extension* layer encloses the management of the new spatial models (*Spatial Models*), the new data structures (*Extension Types*), and the business logic to handle publications and subscriptions, the definition of contexts, and the creation of mappings between contexts of different spaces (*Manager*).

The *Core* layer contains the finite spatial models (i.e., graph, name, and grid) with related primitive types and the manager in charge of monitoring instances of SIS itself. In particular, the *Spatial Models* manages the finite spatial models, the *Data Types* manages the data structures, the *Manager* handles the subscription, publication, and the mappings, finally the *Configuration* handles the startup of the SIS instance. The *Core* layer uses the JESS rule engine in order to handle the operations of transitive closure and matching. This layer grants the full compatibility of the SIS extension with the previous versions of SIS.

The prototype has been developed in Java because the current Core layer is implemented in this language.

B. Inherited Basic Classes

The current developed prototype uses the base classes already defined in SIS. In particular the `Space`, `SpaceParameters`, and `Location` classes that serve as the basic foundation for describing the supported spatial models.

Every spatial model is strictly related to a `SpaceParameter` subtype, which defines the characteristic configuration parameters of that kind of space. As an example, a geodetic space will have the starting and ending coordinates defining the boundary of the region as parameters.

The `Space` class defines a set of common operations that could be performed on every kind of space instance; these operations include the calculation of the distance between two valid locations, the listing of the supported metrics, and the check of validity of a given location. The `Space` class also specifies that a space needs a name which will be used as the unique identifier.

C. Geodetic and Bi-dimensional Cartesian Spaces

In order to verify the suitability and applicability of the conceptual model described in Section II, two spatial models that are inherently infinite have been implemented: the Cartesian two-dimensional and the geodetic. The Cartesian space implementation provides as basic premetric the Euclidean metric and the related space parameter has been defined so that it specifies the boundary of the 2D region.

Two different models of the geodetic space have been provided according to two standard terrestrial representations defined by the European Petroleoum Survey Group (EPSG): the first called EPSG:3857 [10] (also known as Pseudo-Mercator) and the second called EPSG:4326 [11]. Both the representations use the WGS84 reference ellipsoid. The EPSG:3857 model is principally used to support tiled map representations of the World ([12], [13], [14]) as found in Google Maps, Bing Maps, OpenStreetMaps, and Nokia Here; whereas the EPSG:4326 model is used by the GPS satellite navigation system and for NATO military geodetic surveying.

The main difference between the two representations is that in EPSG:3857 the coordinate system is projective (i.e., the satellite coordinates are projected using the Mercator projection or the spherical projection Mercator), whereas EPSG:4326 is a reference geographic system [6] where the coordinates are not projected.

Both the geodetic spaces provide as basic premetric the orthodromic distance. The orthodromic distance is calculated using the haversine formula [15].

D. Infinite Space Management

Since the finite-space SIS Core has been used as the foundation of the prototypical implementation, some techniques have been required to manage infinite spaces in terms of finite spaces without incurring in the limitations of discretization. The use of a finite representation also enables a backward compatibility with all the already defined spatial models (i.e., name, grid, and graph spaces).

Techniques in the area of Geographic Information Systems (GIS) have been analyzed, in particular, the tessellation technique [16]. Tessellation is the process of tiling a plane using one or more geometric shapes (called tiles) with no overlaps and no gaps. Although the tessellation technique is well known and there exist implementations that overcome known problems of efficiency [17], this technique is difficult to apply since it is based on individual locations as the base unit of ”reasoning“ on spaces. Beside being in contrast with the model that considers the spatial context as the base unit, its exploitation will result in a huge memory occupancy because when defining mappings among spatial contexts, all the locations involved will be stored.

The second analyzed technique is partially inspired by [18] and [19] and overcomes the problems that tessellation introduces. The proposed technique is based on the following considerations that directly derive from the conceptual model:

- The base unit used is the spatial context; each operation the platform supports is based on spatial contexts; each operation on a space, either finite or infinite, can be reduced to a sequence of operations on spatial contexts.
- A spatial context may contain one location only.
- The number of effective spatial contexts in a space is finite.

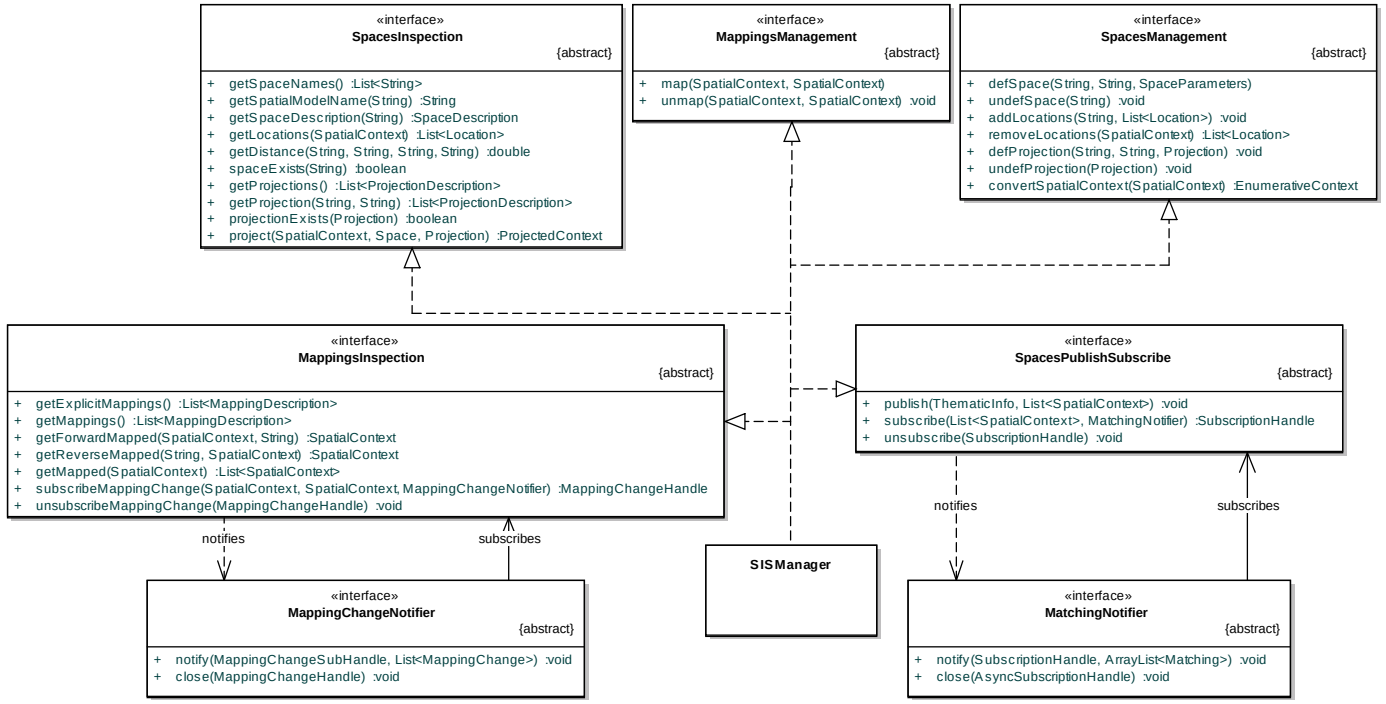


Figure 9: SIS services

- Each spatial context can have a unique identifier; if two contexts share the same identifier then they describe the same region of space.

Given the above considerations, the key idea is to use a finite space (in particular a name space) in the finite-space SIS Core to represent an infinite space. The proposed approach makes use of two concepts: (1) *spatial context fingerprint* (SCF) and (2) *finite support space*. The spatial context fingerprint is a unique identifier of a spatial context that is built from its characteristic locations and membership function. In particular, current implementation uses hash functions [20] to generate the identifier. Each SCF is maintained as a location of a space built form a name spatial model and termed finite support space. Each infinite space has its own finite support space containing the SCFs of all the defined contexts in that space. Obviously, a relation between the original spatial context and the fingerprint inserted in the support space must be maintained.

As an example consider Figure 10. Suppose that the application needs to create the spaces S_1 and S_2 as specified in the application scenario presented in Section IV. They are respectively a geodetic and a graph space. To define the spaces, the application relies on the `defSpace` primitive as follows:

```
defSpace(S1, GeodeticSpatialModel,
    Boundary.WORLD)

defSpace(S2, GraphSpatialModel,
    <{wh1, wh2, wh3, ...},
     {wh1-wh2, wh2-wh3, ...}>)
```

In order to represent the geodetic space S_1 in the finite SIS Core, a new space (`support_`S_1) based on the name spatial

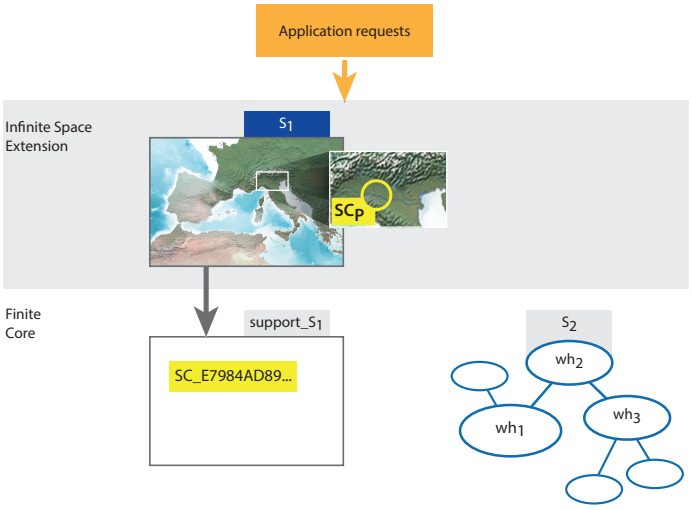


Figure 10: Definition of infinite spaces: an example

model is created. For each spatial context defined in S_1, a corresponding SCF is defined as location in the support space `support_`S_1. In the example, the SC_P context defined in S_1 will be represented by the location `SC_E7984AD89...` in the `support_`S_1 space. On the opposite, the graph space is directly created in the finite SIS Core.

In conclusion, an infinite space is managed through a finite space representing the collection of the defined spatial contexts. This approach maintains compatibility with all the originally supported finite spaces and does not impact on the publish and subscribe mechanism offered by the SIS Core.

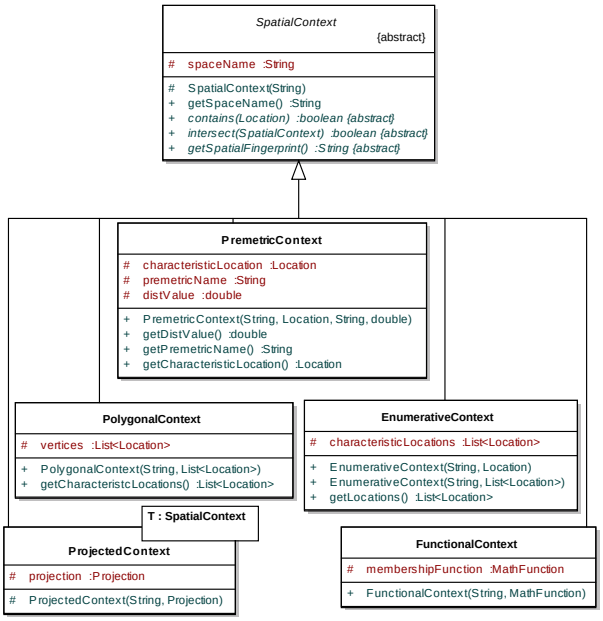


Figure 11: Hierarchy of spatial contexts

E. Spatial Contexts and Projections

As explained in the Section II, each spatial context is identified by a set of characteristic locations and a membership function. Accordingly, the spatial contexts supported by the current implementation are those shown in Figure 11.

An enumerative context (`EnumerativeContext` class) is just a list of locations, whereas a premetric context (`PremetricContext` class) is specified by means of a single location and a radius. Both enumerative and premetric contexts can be used on infinite and finite spaces.

In the prototypical implementation, the polygonal (`PolygonalContext` class), functional (`FunctionalContext` class), and projected (`ProjectedContext` class) contexts are limited to infinite spaces. The polygonal context represents a polygonal closed region of the space. It is defined by providing the vertexes of the region from which the boundaries are linearly interpolated. Vertexes should be provided so that the resulting edges do not overlap. The intersection test between polygonal contexts is done using state of the art techniques in the field of Computational Geometry. In the proposed prototype the general polygon clipper technique [21] has been used.

The functional context represents a region of the space using a mathematical function, which currently is limited to a two variable inequality. This function is internally represented by the `MathFunction` class, which makes use of the exp4j library [22] to enable the evaluation of the mathematical function. The evaluation is done with the Dijkstra version of the shunting yard algorithm ([23], [24]). The usage of this approach enables the specification of the mathematical function using a string representation that is also suitable for the web service interaction.

Finally, the projected context derives from the application of a projection function on a source spatial context. Projections (that are instances of the `Projection` class) are specified by a mathematical function (that operates on locations) and by a source and a target spatial model. The mathematical function is realized by means of the `MathFunction` class. A projection may specify the same spatial model both for the source and the target. It is useful, for example, to define a projection that reifies a rotation defined on a Cartesian space or, more trivially, to define a projection that realizes the identity operation on a space. The characteristic locations of a projected context are those obtained by applying the projective function of the `Projection` to all the characteristic locations of the source spatial context. The membership function for a location in a projected context relies on the membership function for the corresponding (i.e., inverse projected) location in the source space, thus the invertibility constraint on the projection function. To obtain a `ProjectiveContext`, an application can rely on the `project` primitive defined in the `SpacesInspection` interface (see Figure 9).

Finally, to support mappings, subscriptions, and publications, the three following methods have been defined in the base class `SpatialContext`:

- `contains`, which checks if a specified location in the space is included in the context.
- `intersect`, which enables the matching operation on contexts defined on infinite spaces.
- `getSpatialFingerprint`, which returns the spatial context fingerprint (SCF).

F. Mapping Management

As in the finite-space case, the definition of a mapping enables the creation of one or more relations between regions of different spaces. The creation of a mapping involving infinite spaces includes the following two steps:

1) The spatial context fingerprint is computed for each context and the corresponding locations are added to the corresponding finite support spaces (unless they have already been defined).
2) A finite-mapping request is sent to the finite-space SIS Core for the actual creation of the mapping.

This solution allows both the new spatial models and the already existing finite spatial models to be deal with in a similar way, so that both types can be used as source and target contexts for the mapping.

In order to clarify on how the mappings are handled, the mapping M_1 defined in the application scenario presented in Section IV will be deeply discussed. To define the mapping, the application relies on the `defMapping` primitive as follows:

```
defMapping(SC1, SC2)
```

where

```
SC1 =
   <S1, <PREM, <[45.523653, 9.219436], 50, d>>>
SC2 =
   <S2, <ENUM, {wh1}, ∈>>
```

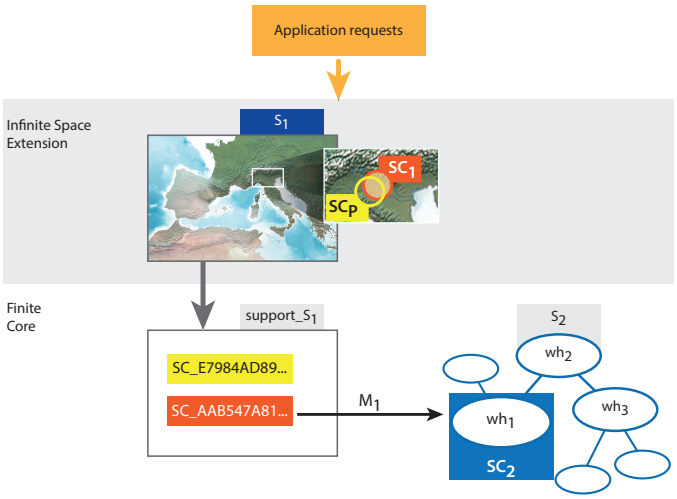


Figure 12: Definition of mappings: an example

As sketched in Figure 12 (that enriches Figure 10), the infinite extension defines a new location SC_AAB547A81... in the support space support_S_1 for the spatial context SC_1 (that is defined on the infinite space S_1). Successively, the mapping M_1 is created between the contexts <support_S_1, <ENUM, SC_AAB547A81..., ∈>> and <S_2, <ENUM, wh_1, ∈>>.

G. Publication

A generic publication on an infinite space (in the following, publication space) is converted into a publication on the corresponding finite support space as follows:

1) An intersection test is performed between the publication context and the source context of each mapping defined in the publication space. Only the source contexts are explicitly considered thanks to the fact that the finite-space SIS Core already contains all the implicit mappings. The result is a set (possibly empty) of intersecting contexts.
2) A new publication context is defined with all the fingerprints of the intersected contexts. The context is an enumerative one defined in the support finite space of the original publication space. Such a new context will be used to define a new publication whose thematic info contains the original thematic info plus the original infinite publication context so that, when a notification will be performed, the original context can be restored.
3) The newly created publication is delivered to the finite-space SIS Core to the reasoning process.

The above operations are performed for each spatial context of the publication that has been defined in an infinite space.

H. Subscription and Notification to Applications

A subscription is made on one or more spatial contexts. It involves the creation of a communication channel for receiving the notifications (for example, a communication channel could be created using Web Sockets). Each subscription is stored until the application removes it.

A subscription request is handled in a way similar to the publication case. For each spatial context defined in a infinite space (in the following, subscription space), the following operations are performed:

1) An intersection test is performed between the subscription context and the target context of each mapping defined in the subscription space. The result is a set (possibly empty) of intersecting contexts.
2) A new subscription context is defined with all the fingerprints of the intersected contexts. The context is an enumerative one defined in the support finite space of the original subscription space.
3) The newly created subscription is delivered to the finite-space SIS Core to be managed.

The deletion of a subscription closes the communication channel and removes all the support subscriptions created.

The notification process is performed only if the spatial contexts involved in a publication are in direct matching with the subscription contexts already stored. Only the direct matchings are considered because the finite-space SIS Core calculates the indirect matching in the transitive closure.

The construction of a notification is a bit more complicated because the contexts reported by the matching process can contain both finite and infinite spaces. Since the core implementation internally operates only on finite spaces, a check on the defined infinite spaces needs to be done in order to convert the involved contexts in the finite support spaces into the original contexts on infinite spaces.

The infinite-space layer of the SIS platform acts as the receiver for all the notifications raised by the finite-space SIS Core. Each notification includes the list of the corresponding matchings. A matching contains the published thematic information and all the contextualization information (i.e., all the contexts that have been found during the matching operation). Each notification is then checked to find possible contexts defined in finite support spaces. Each of these contexts, identified by its fingerprint, is then substituted with the original finite one. The resulting notification is then made available to the subscribed applications through the communication channel.

VI. Performance Evaluation

Several performance tests have been conducted aimed at comparing the mean reasoning times registered by the SIS prototype (SIS 2 in the following) and by the previous version based on finite spaces only (SIS 1 in the following). The mean reasoning time is the mean time between the reception of a publication and the moment at which notifications are made available to interested applications. The mean reasoning time is evaluated with respect to both the number of mappings and the size of the context. Since the contexts involved in the tests are defined over infinite spaces, they have been approximated in SIS 1 as it will be discussed in the following subsections describing the setup of the experiments and the achieved results.

The experimental tests have been performed on an SIS 1 and a SIS 2 instances running on a desktop PC equipped

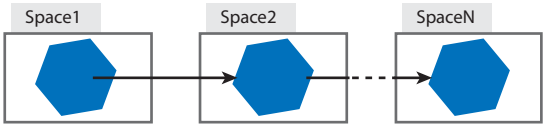


Figure 13: Space configuration for the mapping test

with an Intel Core i5 2.8 GHz, 4GB of RAM, Windows(R) 7 64bit, and the 64bit version of the Java Runtime Environment 1.6.33. Publications were generated by a client and sent to the SIS server. Moreover, performance tests have been also conducted on an Intel(R) Next of Computing Unit (NUC) equipped with an Intel(R) Core i3 1.8 GHz processor, 8GB of RAM, Windows(R) 8.1 64bit, and the 64bit version of the Java Runtime Environment 1.7.0 build 45. Indeed, such an hardware platform can be easily integrated in an actual pervasive computing environment because of its characteristics that make it a good compromise between performance and dimensions.

A. *Mean Reasoning Time vs. Mappings*

The first experimental setup allows studying the dependence of the mean reasoning time on the number of mappings. In this test, the mappings are created in the configuration phase and do not change dynamically. In particular, we analyzed the mean reasoning time in two different configurations: the mappings between polygonal contexts, and mappings between premetric declarative contexts.

In both the configurations, SIS 2 includes n Cartesian spaces, each containing a single context. Every context is directly mapped onto a context in the next space, thus realizing a chain of $(n-1)$ explicit mappings, as shown in Figure 13 for the polygonal contexts. On SIS 1 the Cartesian space has been approximated using a grid space and each context is approximated with a number of cells equals to the area occupied by context itself.

Considering the implicit mappings, the total number of mappings is therefore equal to $n(n-1)/2$. Publications occur on the context in the first space, whereas the subscription is made on the context in the last space. With this generic configuration, the mean reasoning time for a publication can be measured as a function of the number of spaces n. In the experimental tests, n varies from 2 to 150. Obviously, in actual applications, a chain of 150 mappings is to be considered an exceptional case: the objective of this configuration was to stress the system only.

In the first configuration, each Cartesian space in SIS 2 contains a single hexagonal polygonal context inscribed in a circle with a radius of two units. On SIS 1, each context is approximated with a number of cells equals to the area occupied by the polygonal context (i.e., 16 cells).

Figure 14 and Figure 15 show the mean reasoning time (expressed in milliseconds), using polygonal spatial contexts, registered by both the implementations (SIS 1 and SIS 2) when running on the desktop PC and the NUC respectively. As expected, the mean reasoning time obtained from SIS 2 deployed on the desktop PC is lower than that obtained when

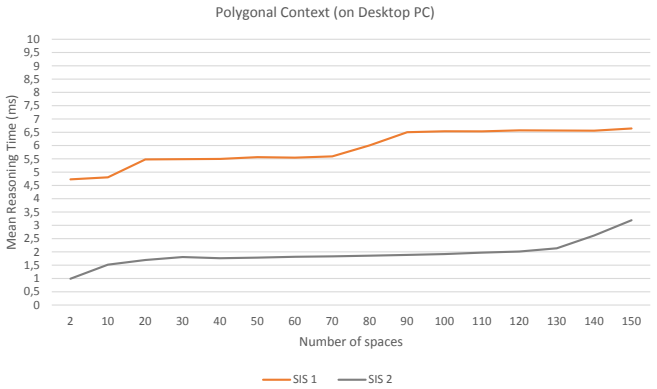


Figure 14: Mean reasoning time vs number of spaces on the desktop PC - Polygonal context

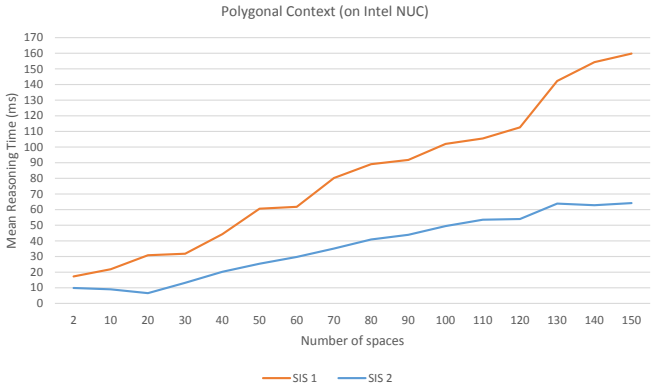


Figure 15: Mean reasoning time vs number of spaces on the Intel NUC - Polygonal context

deployed on the Intel NUC: an average of 5.91 ms for SIS 2 on desktop PC vs an average of 81.63 ms for SIS 2 on Intel NUC. Moreover, the mean reasoning time on the desktop PC is almost constant during the execution of the test, whereas on the NUC is almost sublinear. Finally, as pointed out by both the figures 14 and 15, SIS 2 is faster that SIS 1 using a grid approximation. In particular, SIS 2 on desktop PC is 3.99 ms faster than SIS 1, whereas SIS 2 on Intel NUC is 45.27 ms faster than SIS 1.

In the second configuration, each Cartesian space in SIS 2 contains a single premetric declarative context with a radius of two units. On SIS 1, each context is approximated with a number of cells equals to the area occupied by the premetric declarative context (i.e., 16 cells).

Figure 16 and Figure 17 show the mean reasoning time (expressed in milliseconds), using premetric spatial contexts, registered by both the implementations (SIS 1 and SIS 2) when running on the desktop PC and the NUC respectively. As in the previous configuration, the mean reasoning time obtained from SIS 2 deployed on the desktop PC is lower than that obtained when deployed on the Intel NUC: an average of 5.91 ms for SIS 2 on desktop PC vs an average of 81.63 ms for SIS 2 on Intel NUC. Again, the mean reasoning time on the

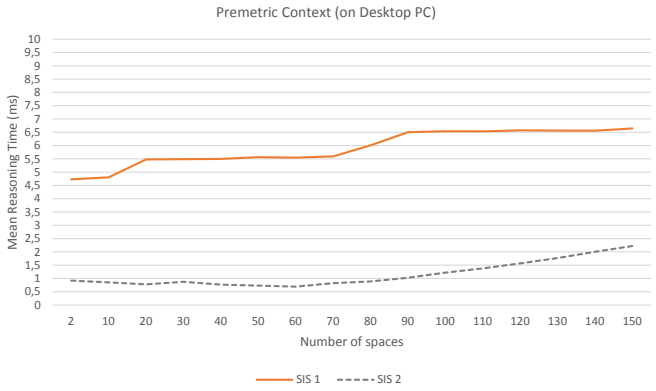


Figure 16: Mean reasoning time vs number of spaces on the desktop PC - Premetric context

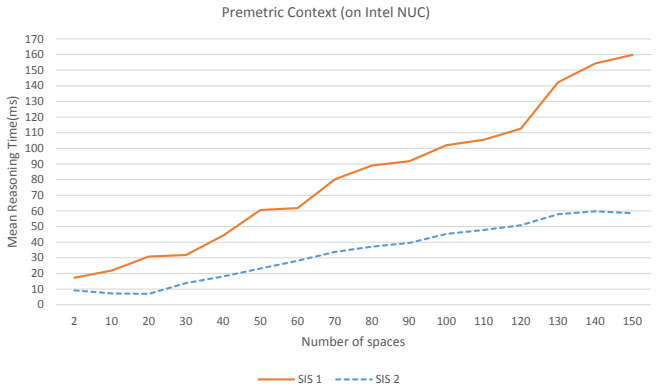


Figure 17: Mean reasoning time vs number of spaces on the Intel NUC - Premetric context

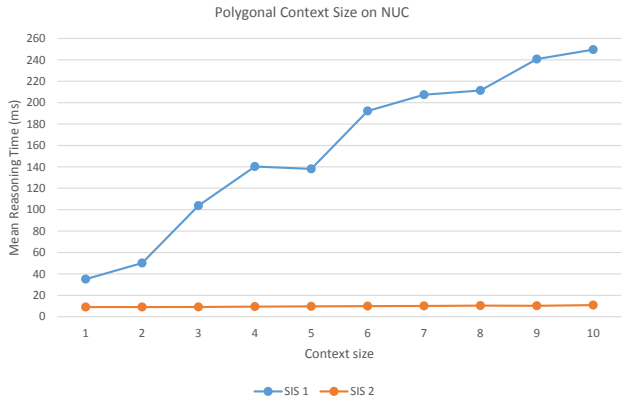


Figure 18: Mean reasoning time vs. size of context on the Intel NUC

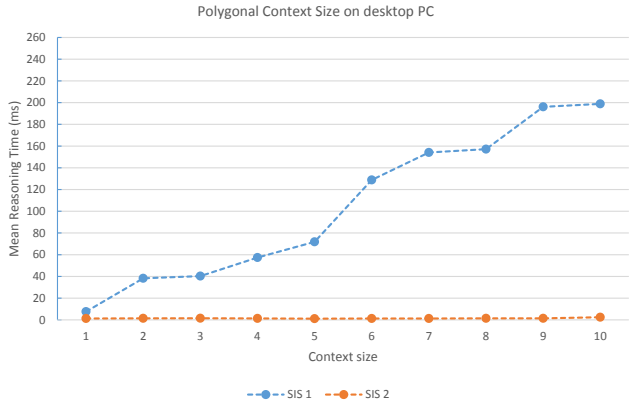


Figure 19: Mean reasoning time vs. size of context on the Desktop PC

desktop PC is almost constant during the execution of the test, whereas on the NUC is almost sublinear. The mean reasoning time for premetric contexts and for polygonal contexts are equal because the underlying implementation is the same as discussed in Section V: it consists in a single spatial context fingerprint for each defined spatial context. Finally, as in the previous configuration, SIS 2 is faster that SIS 1 using a grid approximation. In particular, SIS 2 on desktop PC is 4.07 ms faster than SIS 1, whereas SIS 2 on Intel NUC is 46.06 ms faster than SIS 1.

B. Mean Reasoning Time vs. Context Size

The second test aims to investigate the dependence of the mean reasoning time on the size of the contexts.

The space configuration in this case includes two Cartesian spaces for SIS 2 and two bi-dimensional grids for SIS 1. One hexagonal polygonal context was defined in each space, the first one for publication and the second one for subscription. As in the previous test, the hexagonal polygon has been approximated with a number of cells equals to its area on the grids. During the test, the number of mapped spaces has been maintained fixed at two, whereas the radius of the circle used to calculate the vertices of the inscribed polygonal contexts was varied according to the values between 1 and 10.

As pointed out by Figure 18 and Figure 19, the SIS 2 prototype has a constant mean reasoning time; about 2 ms on the desktop PC and around 9ms on the Intel NUC, while the SIS 1 implementation behaves like $O(24n)$ on the Intel NUC and $O(20n)$ on the desktop pc.

As expected, the SIS 2 prototype presents a constant trend independent from the context size because every context is represented using a single entity.

C. Discussion

The experimental tests show how the implemented techniques allow for a better management of different context sizes in infinite spaces, maintaining a constant time of reasoning. Moreover, the prototype has a better scaling behavior with respect to the finite-space implementation, both for increasing number of mappings and for increasing size of publication and subscription contexts. Such improved scalability can be understood in terms of the number of facts that form the knowledge base of the rule engine managing the operations

of transitive closure and matching; the infinite-space implementation reduces the number of facts to one per context, and thus to two facts for a single mapping, whereas the finite-space implementation creates a fact for each location involved in the mapping.

Finally, the execution of the first test of SIS 2 (number of mappings vs mean reasoning time) states that the Desktop PC has been faster than the NUC regardless of the type of context. This is obviously true, but it is reasonable to assume that in actual situations, the chain of mappings will contain a number of mappings that is substantially lower than 150. By considering a chain of 10 mappings (a plausible value), the mean reasoning time is approximately equal to 9 ms (about 3,5 times compared to the PC). Although each application has its own time constants, we consider a delay of 9 ms to be acceptable for pervasive computing applications.

VII. Related work

Location-aware computing has been an active area of research. Different platforms at the state of art enable location-aware applications focusing on sensor fusion and reasoning with the help of a multi-spatial model, or hybrid model (as called by Becker et al. [25]).

Location Stack [26] defines a layered model for fusing location information from multiple sensors and reasoning about an object's location. It, however, does not incorporate a spatial model of the physical world and does not support representations of immobile objects. This leads to a lack of support for spatial reasoning relative to stationary entities such as rooms or corridors.

Loc8 [27], on the other hand, extends the Location Stack layered architecture by considering only high level position and data instead of low level sensor data. Reasoning is applied to that position data, enriched by the knowledge given by a base ontology, to infer additional spatial relationships.

The *Aura Space Service* [28] combines coordinate and hierarchical location models into a single hybrid model, which supports spatial queries. The focus of the Aura Space Service is only on modeling the physical space and supporting spatial queries. It does not address location inferencing and does not provide a framework for spatial reasoning.

MiddleWhere [29] uses the hybrid location model introduced by the Aura Space Service and enables the fusion of different location sensing technologies. MiddleWhere introduces also probabilistic reasoning techniques to resolve conflicts and deduce the location of people given different sensor data. The model of the world is stored in a spatial-enabled database.

Semantic Spaces from Microsoft Research decomposes the physical environment into a hierarchy of spaces. The locations of moving users or devices are correlated to actual physical spaces, thus it is capable of answering "containment" queries. However, because of its inherent lack of metric attributes and precision, it is unable to compute distance accurately or represent locations precisely, which are requirements for some ubiquitous computing applications [30].

Semantic Spaces and Location Stack lack any support for infinite spaces and in general spaces with a coordinate system, while Loc8 and MiddleWhere have at least one spatial model with a Cartesian coordinate system and can handle different levels of precision on that space model. These two platforms substantially treat infinite spaces by using different granularities for location representation on a local and a global coordinate system.

VIII. Conclusion and Future Work

The paper proposed an extension to the conceptual model of the SIS platform. This refinement comes in order to enable the use of infinite spatial models like the geodetic or the Cartesian ones.

The approach that has been presented involves the extension of the concept of spatial context and the use of that concept as the elementary unit at the basis of all the spatial operations enabled by the platform itself (i.e., mapping and matching).

With the help of a prototypal implementation the revised conceptual model has been tested for performance evaluation. The tests that have been conducted showed an overall increase of performance and capacity to handle spatial contexts with large extent as needed when using the geodetic space. Moreover, the tests have also highlighted that the actual prototypal implementation can be installed into a computing machine (e.g., an Intel NUC) that can be easily plunged in an environment: the mean reasoning time with respect to both the number of mapped spaces and the dimension of the contexts can be considered negligible with respect to the timing requirements of pervasive computing applications.

The main future work consists in the deep integration of the Infinite Space Extension layer in the Core layer. This will enable a more efficient use of the rule engine. Moreover, the platform will be exploited to provide spatial localization to properties related to domain entities as inferred from stimuli acquired by sensing devices. This results in a framework that will support the implementation of pervasive computing applications. Preliminary results can be found in [31].

References

[1] D. Bernini, F. Fiamberti, D. Micucci, F. Tisato, and A. Vertemati, "Spaces-based communication: an extension to support infinite spatial models," in UBICOMM 2013: The Seventh International Conference on Mobile Ubiquitous Computing, Systems, Services and Technologies, 2013, pp. 92–97.

[2] D. Saha and A. Mukherjee, "Pervasive computing: a paradigm for the 21st century," Computer, vol. 36, no. 3, 2003, pp. 25–31.

[3] L. Bullivant, Responsive environments: architecture, art and design. Victoria & Albert Museum, 2006.

[4] D. Bernini, F. Fiamberti, D. Micucci, and F. Tisato, "Architectural abstractions for spaces-based communication in smart environments," Journal of Ambient Intelligence and Smart Environments, vol. 4, no. 3, 2012, pp. 253–277.

[5] C. A. Patterson, R. R. Muntz, and C. M. Pancake, "Challenges in location-aware computing," Pervasive Computing, IEEE, vol. 2, no. 2, 2003, pp. 80–89.

[6] Spatial referencing by geographic identifiers, ISO Standard ISO 19 112, 2003, accessed: 30.5.2014. [Online]. Available: http://www.iso.org/iso/iso_catalogue/catalogue_tc/catalogue_detail.htm?csnumber=26017

[7] C. Cauvin, "A systemic approach to transport accessibility. A methodology developed in Strasbourg: 1982-2002," Cybergeo: European Journal of Geography, Systems, Modelling, Geostatistics, no. 311, 2005.

[8] D. Garlan and M. Shaw, "An Introduction to Software Architecture," Advances in Software Engineering and Knowledge Engineering, vol. 1, 1993, pp. 1–40.

[9] I. Fette and A. Melnikov. The WebSocket Protocol - RFC 6455. Accessed: 30.5.2014. [Online]. Available: http://datatracker.ietf.org/doc/rfc6455/?include_text=1 (2011)

[10] H. Butler, C. Schmidt, D. Springmeyer, and J. Livni. Epsg geodetic parameters: 6864. Accessed: 30.5.2014. [Online]. Available: http://spatialreference.org/ref/sr-org/6864/ (2013)

[11] ——. Epsg geodetic parameters: 4326. Accessed: 30.5.2014. [Online]. Available: http://spatialreference.org/ref/epsg/4326/ (2013)

[12] A. Aitchison. The Google Maps/Bing Maps Spherical Mercator Projection. Accessed: 30.5.2014. [Online]. Available: http://alastaira.wordpress.com/2011/01/23/the-google-maps-bing-maps-spherical-mercator-projection/ (2011)

[13] OpenLayers. Spherical Mercator. Accessed: 30.5.2014. [Online]. Available: http://docs.openlayers.org/library/spherical_mercator.html/ (2008)

[14] P. P. Klokan. Tiles la Google Maps: Coordinates, Tile Bounds and Projection. Accessed: 30.5.2014. [Online]. Available: http://www.maptiler.org/google-maps-coordinates-tile-bounds-projection/ (2008)

[15] D. F. R. Sinnott, R.W; Olson, "Virtues of the haversine," Sky and Telescope, vol. 68, no. 2, 1984, p. 159.

[16] R. Laurini and D. Thompson, Fundamentals of spatial information systems, ser. The A.P.I.C. series. London, San Diego, New York: Academic press, 1992.

[17] K.-T. Chang, Introduction to geographic information systems (4. ed.). McGraw-Hill, 2008.

[18] H. Alt and L. Guibas, "Discrete Geometric Shapes: Matching, Interpolation, and Approximation," in Handbook of Computational Geometry. Amsterdam: Elsevier Science Publishers B.V. North-Holland, 1999, pp. 121–153.

[19] M. T. Goodrich and K. Ramaiyer, "Geometric Data Structures," in Handbook of Computational Geometry. Amsterdam: Elsevier Science Publishers B.V. North-Holland, 1999, pp. 463–489.

[20] C. E. Leiserson, R. L. Rivest, C. Stein, and T. H. Cormen, Introduction to algorithms. The MIT press, 2001.

[21] A. Murta. A general polygon clipping library. Accessed: 30.5.2014. [Online]. Available: http://www.cs.man.ac.uk/~toby/alan/software/ (2000)

[22] F. Asseg, "exp4j: a simple mathematical expression parser for java," 2012, accessed: 30.5.2014. [Online]. Available: http://www.objecthunter.net/exp4j/index.html

[23] E. W. Dijkstra, Primer of Algol 60 programming. Academic Press, Inc., 1962.

[24] R. R. Redziejowski, "On arithmetic expressions and trees," Communications of the ACM, vol. 12, no. 2, 1969, pp. 81–84.

[25] C. Becker and F. Dürr, "On location models for ubiquitous computing," Personal and Ubiquitous Computing, vol. 9, no. 1, 2005, pp. 20–31.

[26] D. Graumann, W. Lara, J. Hightower, and G. Borriello, "Real-world implementation of the location stack: The universal location framework," in Proceedings of the Fifth IEEE Workshop on Mobile Computing Systems and Applications, 2003, pp. 122–128.

[27] G. Stevenson, J. Ye, S. Dobson, and P. Nixon, "Loc8: a location model and extensible framework for programming with location," Pervasive Computing, IEEE, vol. 9, no. 1, 2010, pp. 28–37.

[28] C. Jiang and P. Steenkiste, "A hybrid location model with a computable location identifier for ubiquitous computing," in Proceedings of the 4th International Conference on Ubiquitous Computing, ser. UbiComp '02. London, UK, UK: Springer-Verlag, 2002, pp. 246–263.

[29] A. Ranganathan, J. Al-Muhtadi, S. Chetan, R. Campbell, and M. Mickunas, "Middlewhere: a middleware for location awareness in ubiquitous computing applications," in Proceedings of the 5th ACM/IFIP/USENIX international conference on Middleware. Springer-Verlag New York, Inc., 2004, pp. 397–416.

[30] B. Brumitt and S. Shafer, "Topological world modeling using semantic spaces," in Proceedings of the Workshop on Location Modeling for Ubiquitous Computing, UbiComp, 2001, pp. 55–62.

[31] F. Fiamberti, D. Micucci, M. Mobilio, and F. Tisato, "A layered architecture based on previsional mechanisms," in ICSOFT: International Joint conference on Software Technologies, 2013, pp. 354–359.