International Journal of Software Engineering and Knowledge Engineering
© World Scientific Publishing Company

Towards Composable Location Models in Ubiquitous Computing Applications

Esteban Robles Luna[*]
LIFIA, Facultad de Informatica, UNLP, 115th street between 49th and 50th street
La Plata, Buenos Aires (1900), Argentina
esteban.robles@lifia.info.unlp.edu.ar

Gustavo Rossi[†]
LIFIA, Facultad de Informatica, UNLP, 115th street between 49th and 50th street
La Plata, Buenos Aires (1900), Argentina
gustavo@lifia.info.unlp.edu.ar
http://www.lifia.info.unlp.edu.ar/en/rossi.htm

Silvia Gordillo[‡]
LIFIA, Facultad de Informatica, UNLP, 115th street between 49th and 50th street
La Plata, Buenos Aires (1900), Argentina
gordillo@lifia.info.unlp.edu.ar
http://www.lifia.info.unlp.edu.ar/en/silvia.htm

Received (Day Month Year)
Revised (Day Month Year)
Accepted (Day Month Year)

The state of the art in location models falls short to adapt to new requirements such as composition and integration of maps. Composing and integrating maps are typical operations we want to apply when we deal with ubiquitous computing applications because they evolve permanently (i.e. to add location information of new cities, buildings, means of transport, etc). We propose a novel approach that by abstracting some concepts such as located objects and locations, can result in more flexible models, therefore allowing dynamic composition and integration of maps. With this approach, it is also possible to combine different location representations, making applications easier to extend.

Keywords: Location modeling; Ubiquitous computing; GIS.

1. Introduction

Location modeling has attracted the interest of different areas in computer science, since it is a key application aspect in those applications in which we need either to

[*]Also CONICET, Argentina
[†]Also CONICET, Argentina
[‡]Also CICPBA, Argentina

track the user's position or when application objects need to feature some position (in relationship to the earth, to other objects or both). Geographic information systems [1, 2, 3, 4, 5], location base services [6, 7, 8], context aware software [9, 10], physical hypermedia [11, 12, 13] and ubiquitous computing [14, 15, 16, 17] are some examples of application areas in which dealing with locations is important and therefore as the applications are complex, having a high level and abstract view of locations is a must.

Similarly to other modeling activities, location modeling involves defining how locations are abstracted in an application, in order to be manipulated by application objects, how they affect other application features or objects, how they are related with each other, how operations related with locations are computed, etc. A particular location model may simplify the way some operations are computed and may also pose some constraints regarding which operations are meaningful or not. Some location models might be completely abstract (e.g. symbolic models like [18, 19, 6]) while others might have strong relationships with our common understanding of locations (this is the case of geometric and global geographic models).

In the particular case of ubiquitous computing, the inherent complexity of this kind of software, which grows "organically" as explained in [20], has motivated an important tenet of work to find adequate location models which support the variety and the evolution of application requirements. For those ubiquitous applications which are physically spread on multiple physical areas, the choice or development of the location model is crucial.

Let's suppose that a company has developed an application that helps salesmen traveling among different cities. By using the cartography of the city where their clients are located, the system provides information to give assistance during their trips. As the salesmen need to move inside huge buildings or campuses, the company decides to incorporate information, to help them moving inside those areas. For this new requirement, the company requests each customer a location model for its buildings. This task is done by using customized tools the company has developed so that each customer can give a map of some of their most important buildings. With this new information, the application can eventually compute paths between two different rooms of two different buildings.

In this scenario different kinds of location models appear, for example in terms of indoor or outdoor models. Each of these models is represented using different information structures, supports different operations and defines varied relationships. For example, while in an indoor model it may be useful to organize the space using a composition structure (e.g. a tree structure), this kind of relationship does not appear in a outdoor model which uses a global reference system. We might want to integrate a transport model for those salesmen who use public transport (such as the metro); here we have another kind of composition problem, because once the two models are integrated we should modify our path computing algorithms to take into account the new means of transport.

In this context, an additional integration effort is needed in order to make models compatible enough to work together, especially in those particular locations in which we have to switch from one model to another one. For example, the "borders" of a building like the location of the entrance door might be modified when we develop the integrated model. This is because the door may have some information about how it is connected with the corridors of the building, but we also need to show how it is connected with the outdoor part of the building. When integrating the location model submitted by a client, we need to manipulate the model to add this semantic to the location. For example, if the location of the entrance door is modeled as a pair ((45,50), *#entranceD*), where the first component is the geometric part and the second one is the symbolic, we need to express that the door is adjacent to the street segment that pass next to the building. This new feature involves the manipulation of the symbolic model, creating a link between two different symbolic locations.

While this integration problem has been recently studied [18, 19, 21, 22, 23], most approaches can not avoid polluting the original model, making it difficult to support further evolution or reuse.

In summary, when ubiquitous applications evolve, we face the following problems regarding to the integration of disparate location models:

- Expressing new physical relationships without modifying previous models. The evolution of models forces to integrate the location of an object to express new relationships, e.g. the entrance door and the metro stations adjacencies. This fact may imply working on previous models so that these relationships are taken into account. Like in the entrance door example, the addition of the relationships originated when we integrate these models should be completely decoupled from the existing relationships.
- The way paths are computed should be easily changed when new relationships appear. This means that, when we incorporate a new model like the metro model, the decision of guiding the user to use the metro should be seamless incorporated in the model without modifying the relationships between locations.

In this paper we present a novel approach for modeling locations which solves the problems found when we integrate disparate location models. The approach is materialized in a conceptual object-oriented framework in which we represent different concepts related with locations.

The main contributions of the paper are:

- From a conceptual point of view, we show that location is a more general concept than the way it is modeled (e.g. geometric figure, symbol, etc). Our approach proposes a way for decoupling the concept from how it is represented in a particular location model. Also, we propose an effective way for computing paths between two located objects in disparate location models.

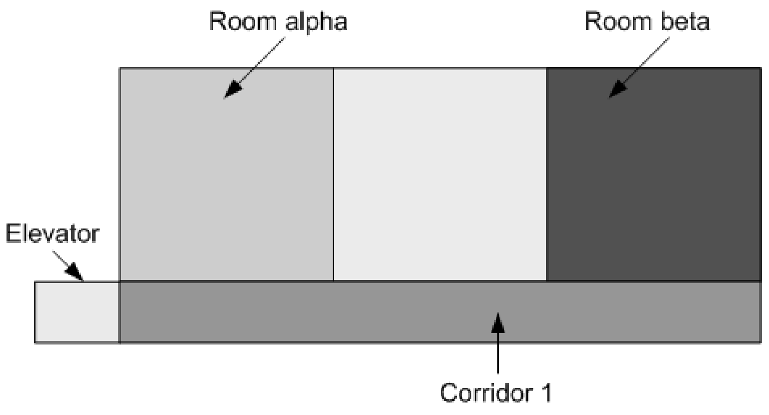


Fig. 1. 2nd floor of a building modeled with geometric figures

- From a design point of view, we show how to map these ideas into a concrete object-oriented framework which we describe in a step by step way.
- From an implementation point of view, we show how to reuse the relationships between domain objects as location information. For example, a relationship between a building and a room in a domain model could be considered as an inclusion relationship at the location level.

The rest of the paper is organized as follows: In Section 2, we present the basic characteristics of location models and briefly present the main idea of our approach. In Section 3 we describe our approach in more detail explaining: how we model locations and how previous models are adapted. In Section 4 we show how to plug models. Section 5 describes the way we can reuse some relationships between domain objects to express physical relationships. Section 6 explains some implementation details on how operations and paths are computed. Section 7 discusses some related work; in Section 8 we conclude and present some further work.

2. Basic concepts

2.1. *Location models*

The decision of which location model will be used in a location-sensitive application is one of the key aspects before the development stage. The purpose of a location model is to give a computational representation of real-world locations. The most common examples are the geometric and symbolic models [18] which define an easy way of modeling locations.

In the first case, locations are modeled as geometric figures on a Cartesian plane. Each figure is a composition of one or many of the three primitives that the model provides: point, polyline and polygon. Suppose that we are modeling the locations of the 2nd floor of a building (Figure 1). In this model, the location of each room is

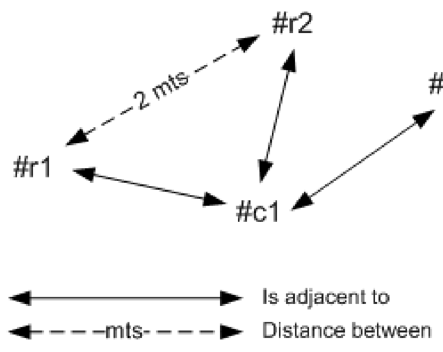


Fig. 2. 2^{nd} floor of a building modeled with symbols

modeled as a rectangle. We also model the locations of corridors since we need to find paths between different places that may cross them and the elevator because we can switch between the floors of the building when we use it. Notice that the fact that two rectangles touch on one side doesn't tell us that they are adjacent in the sense that one person can go from one to the other directly. So, for expressing adjacency in this context, we may chose between using a symbolic model or polluting the model by adding some semantic information to figures. This last approach is used in GIS applications by storing semantic information in dbf files that complement their shape files [24].

Symbolic models provide an alternative way for modeling configurable locations. We are able to define the semantic of the operations either by extension or by comprehension. A location is modeled as a symbol that may be linked with other symbols. Different links provide different relationships that define the semantic of the operations. For the 2^{nd} floor example, we model the locations of each room, the elevator and the corridor as symbols *#r1*, *#r2*, *#e*, *#c1*. Relationships are defined by different types of links providing information for computing the operations (Figure 2). The main drawback of symbolic models is that in general, the relationships between locations have to be manually configured and for huge models this is complicated to handle. Other approaches use symbols with well defined semantic so that operations are defined by comprehension over the set of symbols [19].

As Leonhardt shows in his thesis [18], each model has advantages and disadvantages. While geometric models are preferred when we want to compute distances and directions, symbolic ones offer a configurable way of expressing some relationships that don't clearly emerge in geometric models, e.g. adjacency. While these models are the basic ones for representing locations, there are also proposals to combine them in the form of pair [18], so that we can have the advantages of both. Although these approaches improve some characteristics of the base models, they fall short when we need to integrate them. Adding new relationships force a modification on previous models therefore producing tangled components.

2.2. *Composition and integration problems*

The composition and integration of locations models is not a simple task, especially if we take into account the incompatibilities that basic models have with each other. For instance, let's suppose that we have a particular location modeled as a symbol and we have another one modeled as a geometric figure, we may wonder how we compute the distance between this two locations.

This problem emerges from the interaction between different models. For example, assume that we want to integrate the location models of a building and the streets of a city on the salesman example (Section 1). For the sake of understanding we ignore the integration and composition of geometric models because it has been already studied in the GIS domain, and solved using transformations.

Let's suppose that we have an extended version of the 2^{nd} floor model where we add some objects like: floors, elevators, stairs segments and their relationships. Also, we suppose that the streets of the city are modeled using a symbolic model. Composing these models involves the creation of links between the symbols. In this case, the composition would be expressed by the addition of a new link that states the inclusion relationship between the building and the city. This can't be easily done with these models because they are expressed using two different instances of a symbolic model and we can only add links between symbols inside the same model. Symbols must be therefore migrated to a common model, and then the link could be created. This leads to a new problem: symbol's name conflicts should be solved after the migration process.

The integration between these models faces the same problem. We need to add a new relationship between the location of the entrance door and the street segment nearby the building. This can't be done if the locations are in different instances of a symbolic model, so if we want to connect these locations we need them to be in the same model. To sum up, composing and integrating symbolic models produce the grown of a huge monolithic model that contains all the symbolic locations.

The scenario would be somewhat different if the streets segments were modeled like geometric figures and the building was modeled with symbols. In this case, we would assume that we have some information of the streets such as which are the segments of the streets and how they are connected. In GIS systems, this information is usually stored on dbf files associated to the shape files where the geometric figures are stored. The composition and integration between these models will involve the creation of new relationships between the locations. For composing geometric and symbolic models we face the problem that the inclusion operation is defined differently: in the first one it is formally defined as an operation (for two specific figures we can not change the result of the operation), and in the second one is defined by connections between the links which may involve a look up on a table or collection. One solution for this problem is adding new tables to define relationships between geometric figures and symbols. So we may express the inclusion relationship between a figure and a symbol by adding a new entry on the table. On the other hand, in order to obtain integration some adjacency relationships have to be added. These kinds of relationships are defined in a similar way on both cases. In geometric models, big tables that express that two locations are adjacent are defined. These tables can be considered similar to a big set of relationships in the symbolic model. So, integration would involve the creation of a new table for expressing the adjacency relationship between the locations, which is a simple solution.

Both approaches lacks from being a homogeneous solution and try to fill a huge gap between geometric and symbolic models on how operations are defined. The solution of adding tables for expressing relationships doesn't scale if we add a new kind of location like a hybrid one [19, 18] which uses both types of locations. This will oblige us to rethink the solution each time we add new kinds of locations to fill the gap that location models have to express some kinds of relationships.

We next outline the basic ideas of our approach to solve the previously mentioned problems.

2.3. *Our approach in a nutshell*

As previously mentioned, location models are computational representation of locations. The decision of which model we chose for a particular application is not trivial, and in the case of ubiquitous applications which evolve permanently, things get harder since these models have to interoperate.

State of the art models provide several ways of modeling locations. We can model a location as a point because it is easy to compute Euclidian distances and/or directions or we can chose a symbol to compute paths because they clearly express the adjacency relationship with links. Although there are more models than basic ones [18, 25, 26, 19] they all use some common concepts.

The located object concept [27] is used to talk about the objects that have a location. Usually, located objects are grouped on maps that have a specific purpose, e.g. group all the located objects inside a building. State of the art models also use the location concept but they don't agree on how it is defined. For the purpose of this paper we assume that a location is used to express the relationships that a located object has with its physical environment. For example, if the location of the building is modeled as a point, it allows us to compute the distance, direction, and inclusion relationships with other objects with the same type of location.

All models provide the same conceptual approach for representing locations. They use the location and located object concepts posing the restriction that a located object can only have one location: point, symbol or hybrid (Figure 3). We can talk about a located objects layer containing all the located objects that we use to compute location operations. These objects reference an element in the location layer that represents the physical relationships that the object has with its environment.

These models are somehow strict because they force the object in the location layer to express the whole set of relationships with one element representing the

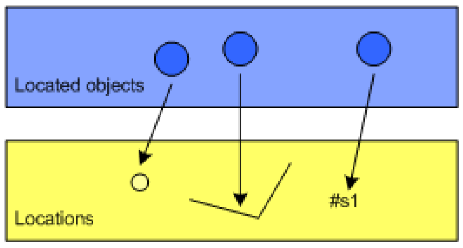


Fig. 3. Standard approach for modeling locations

location. The objects in that layer are points and symbols that as we previously saw fall short to express the whole set of relationships that a located object may have.

Instead of thinking about models so strictly, we can think about them as different ways of representing some relationships a location has. Some models are better to express some kind of relationships than others. Going back to the example, the location of the entrance door can be represented differently if we are modeling the map of the building geometrically or if we are showing the relationships between the door and the streets of the city (Figure 4). In this last case we would prefer to represent the location symbolically, so that the adjacency relationship between the door and the street segment would be expressed easily as a link.

Our approach is based on the fact that these models (e.g. geometric, symbolic, etc) only express a subset of the relationships that a location has with its environment.

Locations are not easy to extend when we need to express new relationships if we modeled them either as a geometric figure or as a symbol. For example, the entrance door or the building could be first represented by a symbol to express the adjacencies with the located objects inside the building, but when we integrate this with the streets model we want to compute distances to located objects such as streets which are on a different model.

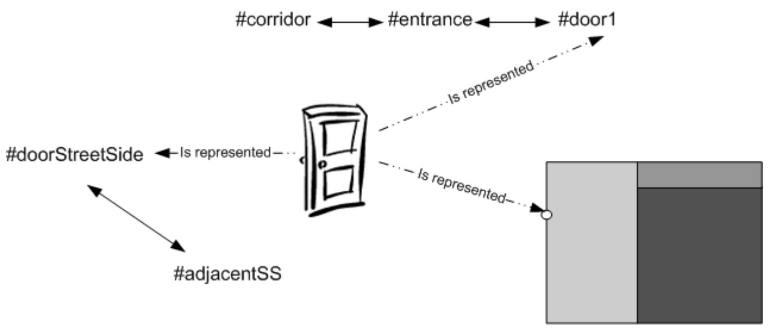


Fig. 4. The location of the entrance door using multiple representations

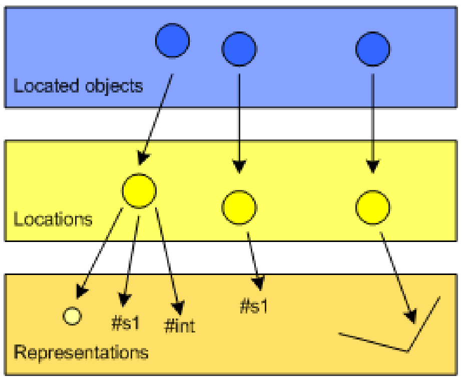


Fig. 5. An outline of our approach for modeling locations

Clearly, we need to use symbols and geometric figures at the same time. This pushes the idea of having an abstract location which can be multiple represented with points, symbols, etc. This idea introduces a new layer for abstracting the location pushing down the geometric and symbolic elements (Figure 5). From now on, we will call *representations* to those elements that can represent a location, allowing us to express some physical relationships. Examples of representations are points and symbols. With this approach, expressing new relationships is as simple as adding representations that can provide the information for computing them.

To sum up, the evolution of models pushes the introduction of new relationships between the objects. Relationships are expressed with points or symbols but new ways of expressing relationships may appear. Depending on our needs, the way we model the location of an object may vary. With our approach, a *representation* such as a point or a symbol denotes a way of expressing some relationships of a location. Abstracting the location allows us the late introduction of new relationships that it has with its environment. By adding new representations we are enriching the location with new relationships.

In conclusion, our approach is based on decoupling locations from how they are represented on a particular model. In this way, the entrance door of the building can be represented in multiple ways; allowing to express in the future new relationships that do not emerge from previous models. In the following section we show how we used an object-oriented approach to put our ideas to work.

3. Composable Location Models

3.1. *Abstracting Locations*

As previously mentioned, the core of our approach consists in adding a new layer between the located objects and the way we represent their locations. This strategic decision gives us the flexibility to define new ways of representing a location; for example if somebody has represented a location as a URL [25, 26], we can easily

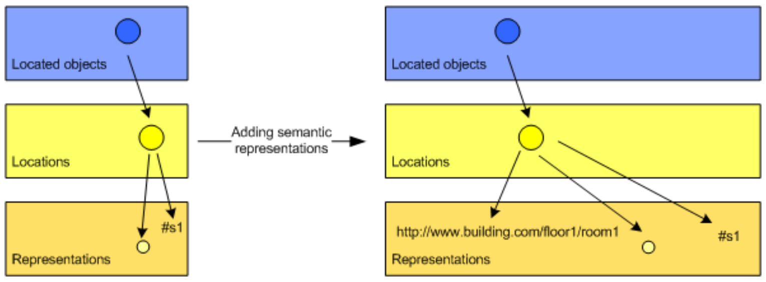


Fig. 6. Adding semantic representations to our approach

accommodate this new representation in our model (Figure 6). Also, we can dynamically express new relationships on an existent location. For example, the entrance door of the building was first modeled as a symbol. Then, a new requirement for computing distances with the streets of the city appeared. With our model, expressing this relationship is simple; we just need to add a geometric representation to its location.

To illustrate these ideas we use the application example described in the introduction (Section 1). The application is intended to help salesmen to find their ways to clients. For the sake of comprehension we assume that the model of the application has been described using UML. In Figure 7 we show the simplified model of the application comprising the main application classes according to the previous requirements.

In the model of Figure 7, we can add and remove buildings from a particular customer and define appointments with contacts of the companies on specific rooms. Suppose that we aim to extend this model to incorporate location information. Particularly, we intend to have the location of the customer's buildings, their rooms and our salesman's locations so that we can guide them on those places. Some new objects may also appear: corridors for computing paths inside the buildings; elevators, stairs and floors to physically describe the most important objects of the building. Notice that these objects are not in our base model, so we should add

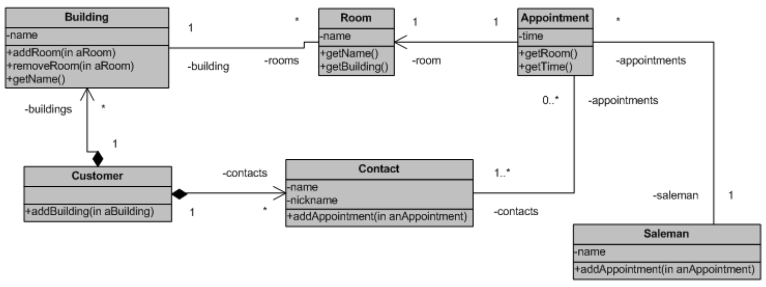


Fig. 7. Simplified model of the company system

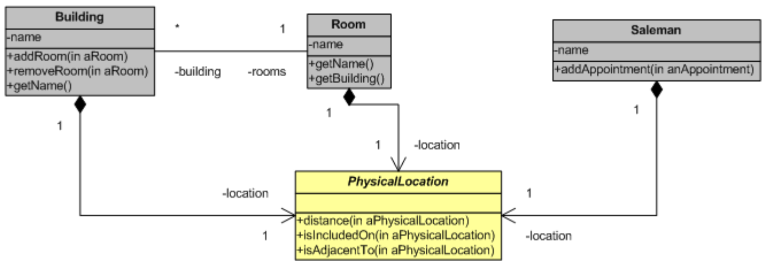


Fig. 8. Model polluted with location information

them in order to express these relationships.

In our domain model, there are classes like *Building* and *Room* to which we want to attach new behaviors realizing the new responsibility of knowing their location, (they are now located objects). We have several approaches for adding this responsibility. The simplest solution is to incorporate the location into these classes with an instance variable and proper accessors (as shown in Figure 8). This solution has some drawbacks: it couples the domain model with particular location information and it replicates code if we want to implement a message that computes a location operation. For instance, the distance message computes the physical distance between two located objects. Its signature is: *distance(anotherLocatedObject)* and its implementation is *return this.getLocation().distance(aLocatedObject.getLocation())* which would be replicated on each class. A better solution is to provide a generic wrapper [28] to enhance the behavior of the objects with their new location aspect. The *LocatedObject* class will be responsible of the behavior related to locations and to delegate to its subject all the sent messages that it doesn't understand. We can change an object's location by sending the *setLocation(aLocation)* message (Figure 9).

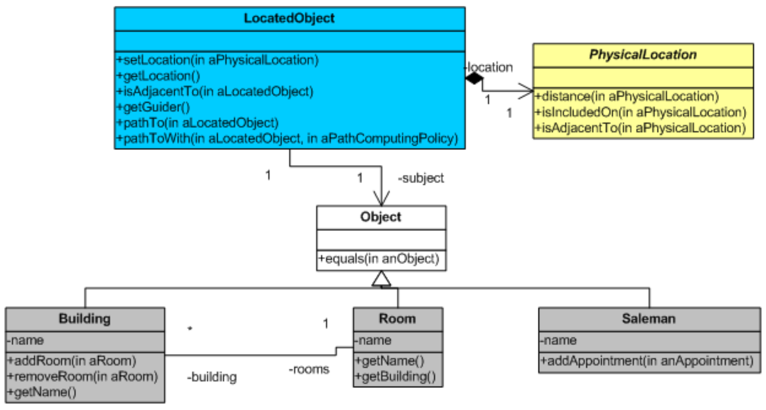

LocatedObject
+setLocation(in aPhysicalLocation)
+getLocation()
+isAdjacentTo(in aLocatedObject)
+getGuider()
+pathTo(in aLocatedObject)
+pathToWith(in aLocatedObject, in aPathComputingPolicy)
PhysicalLocation
+distance(in aPhysicalLocation)
+isIncludedOn(in aPhysicalLocation)
+isAdjacentTo(in aPhysicalLocation)
-location
-subject
Object
+equals(in anObject)
Building
-name
+addRoom(in aRoom)
+removeRoom(in aRoom)
+getName()
Room
-name
+getName()
+getBuilding()
Saleman
-name
+addAppointment(in anAppointment)
-building
-rooms

Fig. 9. Generic wrapper to capture the location aspect of an object

Located objects have a location in order to compute location operations. This concept is modeled with the class *PhysicalLocation*. We differentiate between two kinds of locations: an *UnknownPhysicalLocation* and a *ConcretePhysicalLocation*. The first one is used as a Null object [29] to express the lack of knowledge of the location of a located object. The second is used as a location that has at least one representation to compute their operations. This kind of location uses their representations to compute operations (Section 6).

For realizing representations like points and symbols we use a *Representation* base class and apply some well known design patterns [28]. This class provides a protocol that subclasses must implement in order to provide the necessary functionality to locations. It also defines coercing and compatible checking so that subclasses only need to implement the way in which they operate with each other. Some of these operations are implemented using the template method pattern [28]. We also define a *RepresentationSystem* base class with the purpose of grouping the representations that belong to the same set. This class works in the same way as a geometric plane or the set that contains the symbols, specifying which representations are compatible to operate. For example, on a symbolic representation system, two representations can operate if they belong to the same representation system. This matches the concept definition of when we can operate between two symbols (Section 2.2).

In an OO environment relationships between objects may express physical relationships. For example, the relationship between *Building* and *Room* classes in Figure 9 shows an inclusion relationship (inheritance). Also, the adjacency and inclusion relationships between corridors, rooms and buildings may be reflected at the OO level. To reuse these OO relationships as physical relationships with the lowest

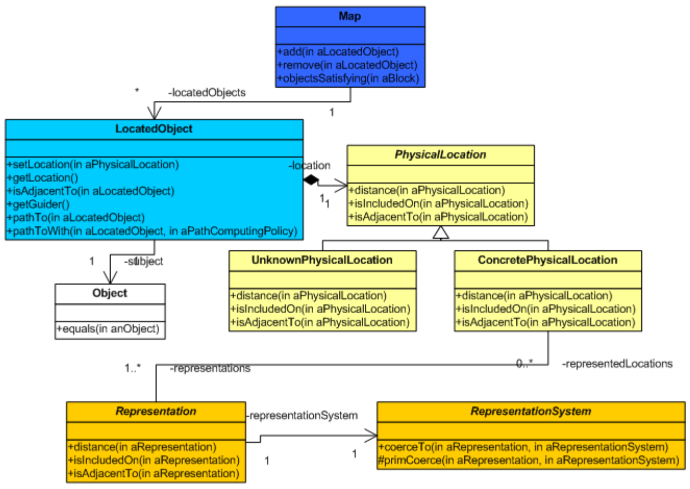

Map
+add(in aLocatedObject)
+remove(in aLocatedObject)
+objectsSatisfying(in aBlock)
*
-locatedObjects
1
LocatedObject
+setLocation(in aPhysicalLocation)
+getLocation()
+isAdjacentTo(in aLocatedObject)
+getGuider()
+pathTo(in aLocatedObject)
+pathToWith(in aLocatedObject, in aPathComputingPolicy)
-location
1
1
PhysicalLocation
+distance(in aPhysicalLocation)
+isIncludedOn(in aPhysicalLocation)
+isAdjacentTo(in aPhysicalLocation)
1
-subject
Object
+equals(in anObject)
UnknownPhysicalLocation
+distance(in aPhysicalLocation)
+isIncludedOn(in aPhysicalLocation)
+isAdjacentTo(in aPhysicalLocation)
ConcretePhysicalLocation
+distance(in aPhysicalLocation)
+isIncludedOn(in aPhysicalLocation)
+isAdjacentTo(in aPhysicalLocation)
1..*
-representations
0..*
-representedLocations
Representation
+distance(in aRepresentation)
+isIncludedOn(in aRepresentation)
+isAdjacentTo(in aRepresentation)
-representationSystem
1
1
RepresentationSystem
+coerceTo(in aRepresentation, in aRepresentationSystem)
#primCoerce(in aRepresentation, in aRepresentationSystem)

Fig. 10. Located object, location, representation and map concepts

cost we introduce the usage of Domain Representations (Section 5) as a simple and customized solution.

Finally, a *Map* class is used to group a set of related located objects. *Map* provides a set of useful behaviors to add, remove and search located objects on the map. A complete diagram of these classes and their relationships can be seen in Figure 10.

With this abstract view of the most important classes of the approach we can proceed with our exemplar domain model. As we previously mentioned, object instances from classes like *Building*, *Room* and *Saleman* will be wrapped with the *LocatedObject* instances. Our customers can now create maps that express the most important physical relationships between the objects of their buildings. In order to build their maps, they may use a tool to edit the map on a UI which generates the object model. This helps somebody without programming skills to create maps without dealing with the objects behind. An example of one of the maps generated by a tool could be seen on the simplified object diagram of Figure 11. Notice that some classes (*Floor*, *Corridor*, *Elevator*) where added to the model because they are necessary to compute paths between some located objects. For example, when we compute paths between rooms we would like to know which corridors we will have to go through. Notice that these objects are not at the same level of abstraction as the domain objects since they don't have any responsibility on the domain of the application. They belong to the "location" level because they were added just for the purpose of computing paths and giving visual feedback to the user.

3.2. *Basic representations*

Geometric figures and symbols offer a convenient way for representing locations, because their semantics are clear and they are easy to specify. Representations give us a way for expressing relationships between locations and as previously mentioned, they are defined abstractly in the *Representation* base class. We will illustrate our approach with these two common ways of representing locations, showing how they are modeled by extending the base classes of our conceptual framework.

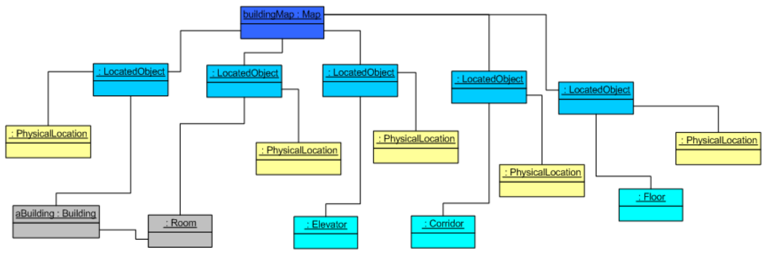


Fig. 11. A simplified object diagram of a building's map

3.2.1. *Dealing with symbols*

When using symbols, we can choose between defining their relationships by comprehension or by extension. The first option is generally coupled to a particular model, so this option is not taken into account in our model, despite it's easy to add one of them. For instance, we can define the symbolic location of the rooms and floors of the building with symbols such as: *#1*, *#101*, *#102*, and *#205*. The operations are defined by comprehension over the set of symbols:

- Inclusion: s1 includes s2 iff s1 length = 1 AND s2 length = 3 AND substring(s1,1) = substring(s2,1)
- Adjacency: s1 isAdjacentTo s2 iff abs(s1 asInteger - s2 asInteger) = 1

For instance, *#1* includes *#101* will answer true, but *#1* includes *#205* and *#101* includes *#221* will answer false. And for the adjacency *#101* isAdjacentTo *#102* will answer true, however *#1* isAdjacentTo *#205* and *#101* isAdjacentTo *#221* will answer false.

The second option is more flexible and besides it is easy to implement tools for creating symbols and manipulating their relationships. We model this type of representation with the *ExtensionalRepresentation* class. This class extends the *Representation* base class provided by our framework; as its name suggests, the relationships between symbols are defined by knowledge relations in the object model. This class provides a suitable protocol to configure the relationship between symbols providing methods for defining the inclusion, adjacency and distance relationships. We have also defined the representation system that is used to give a context where symbols are defined. The *ExtensionalRepresentationSystem* class is responsible for creating the instances of the *ExtensionalRepresentation* class avoiding name conflicts as it keeps track of the symbols that have been created. A class diagram of this type of representation is shown on Figure 12. An example of the use of these representations will be shown on section 3.3.

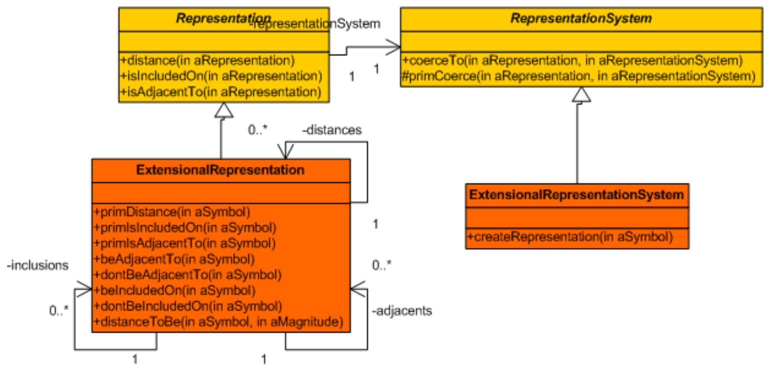

Representation
-representationSystem
+distance(in aRepresentation)
+isIncludedOn(in aRepresentation)
+isAdjacentTo(in aRepresentation)
1
1
RepresentationSystem
+coerceTo(in aRepresentation, in aRepresentationSystem)
#primCoerce(in aRepresentation, in aRepresentationSystem)
0..*
-distances
ExtensionalRepresentation
+primDistance(in aSymbol)
+primIsIncludedOn(in aSymbol)
+primIsAdjacentTo(in aSymbol)
+beAdjacentTo(in aSymbol)
+dontBeAdjacentTo(in aSymbol)
+beIncludedOn(in aSymbol)
+dontBeIncludedOn(in aSymbol)
+distanceToBe(in aSymbol, in aMagnitude)
-inclusions
0..*
1
1
0..*
-adjacents
1
ExtensionalRepresentationSystem
+createRepresentation(in aSymbol)

Fig. 12. Symbolic representations

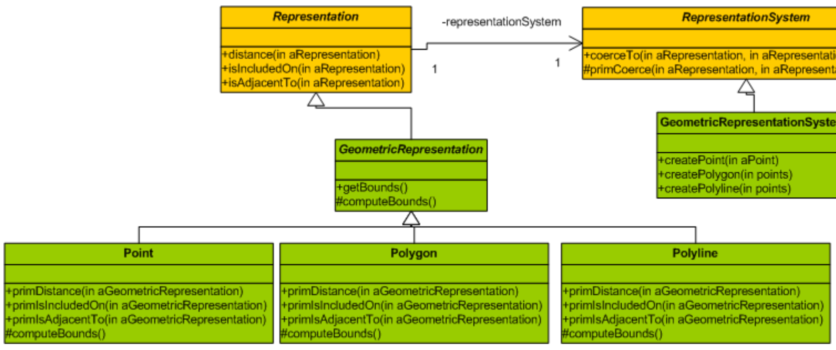

Representation
+distance(in aRepresentation)
+isIncludedOn(in aRepresentation)
+isAdjacentTo(in aRepresentation)
-representationSystem
1
1
RepresentationSystem
+coerceTo(in aRepresentation, in aRepresentationSystem)
#primCoerce(in aRepresentation, in aRepresentationSystem)
GeometricRepresentationSystem
+createPoint(in aPoint)
+createPolygon(in points)
+createPolyline(in points)
GeometricRepresentation
+getBounds()
#computeBounds()
Point
+primDistance(in aGeometricRepresentation)
+primIsIncludedOn(in aGeometricRepresentation)
+primIsAdjacentTo(in aGeometricRepresentation)
#computeBounds()
Polygon
+primDistance(in aGeometricRepresentation)
+primIsIncludedOn(in aGeometricRepresentation)
+primIsAdjacentTo(in aGeometricRepresentation)
#computeBounds()
Polyline
+primDistance(in aGeometricRepresentation)
+primIsIncludedOn(in aGeometricRepresentation)
+primIsAdjacentTo(in aGeometricRepresentation)
#computeBounds()

Fig. 13. Geometric representations

3.2.2. *Representing geometric figures*

Points, polygons and polylines are the other usual type of representation for locations. These geometrical objects have some common behaviors: they all have a bounding box and, if a geometric transformation is defined, they can be converted to an equivalent element on other plane. For these reasons we define a base class called *GeometricRepresentation* from which *Point*, *Polygon* and *Polyline* classes are derived. Each subclass is responsible of knowing how to compute its own bounding box and how to operate with other geometric figures. The *GeometricRepresentationSystem* is responsible for creating these basic representations (Figure 13).

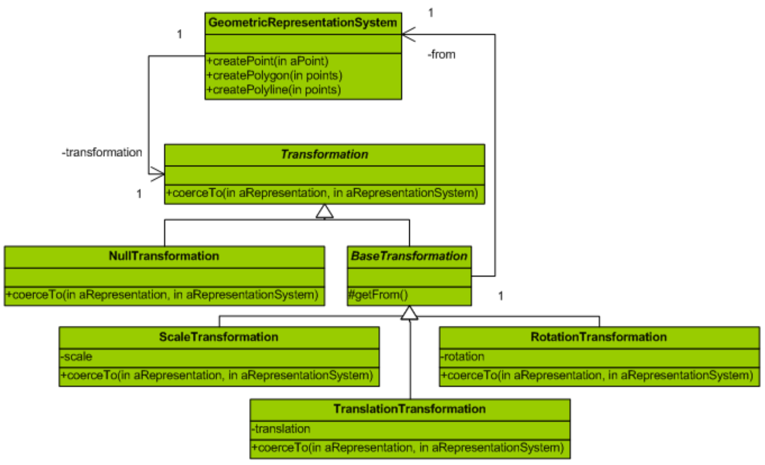

1
1
GeometricRepresentationSystem
+createPoint(in aPoint)
+createPolygon(in points)
+createPolyline(in points)
-from
-transformation
1
Transformation
+coerceTo(in aRepresentation, in aRepresentationSystem)
NullTransformation
+coerceTo(in aRepresentation, in aRepresentationSystem)
BaseTransformation
#getFrom()
1
ScaleTransformation
-scale
+coerceTo(in aRepresentation, in aRepresentationSystem)
RotationTransformation
-rotation
+coerceTo(in aRepresentation, in aRepresentationSystem)
TranslationTransformation
-translation
+coerceTo(in aRepresentation, in aRepresentationSystem)

Fig. 14. Geometric transformations

Transformations such as translation, rotation and scaling are taken into account in our conceptual framework. These transformations allow us to transform a geometric figure on one plane to an equivalent one on the other. They are useful when we want to model geometric figures relative to a different origin. Transformations are modeled on the transformation hierarchy. *ScaleTransformation*, *TranslationTransformation* and *RotationTransformation* are the classes that model these concepts (Figure 14). We provide a *NullTransformation* class for those geometric representation systems that are not relative to others i.e. a representation on one of this representation systems can't be converted to an equivalent on another.

3.3. *Using representations*

Geometric and symbolic representation systems are modeled with classes that extend the *Representation* and *RepresentationSystem* base classes. These types of representations are the one the tool uses to create the building map. As previously seen, we have modeled the floors, rooms and corridors of the buildings. In Figure 11 we presented a simplified version of the map in an object diagram describing all the objects without their representations. With the information discussed in the previous subsection, we can now complete the diagram (shown in Figure 15) based on the map of the 2^{nd} floor of the building (Figure 1). To keep the diagram small, we omit located objects, the map and the corresponding domain objects. We use a naming convention for location objects: e.g. buildingLocation refers to the *Location* object referenced by the *LocatedObject* object which wraps a *Building* object. The diagram shows the location of several objects such as the elevator, rooms and the corridor. Some locations such as the elevator and the rooms are represented with a symbol and a geometric figure. Others like the floor and the building are just represented with a symbol with inclusion relationships.

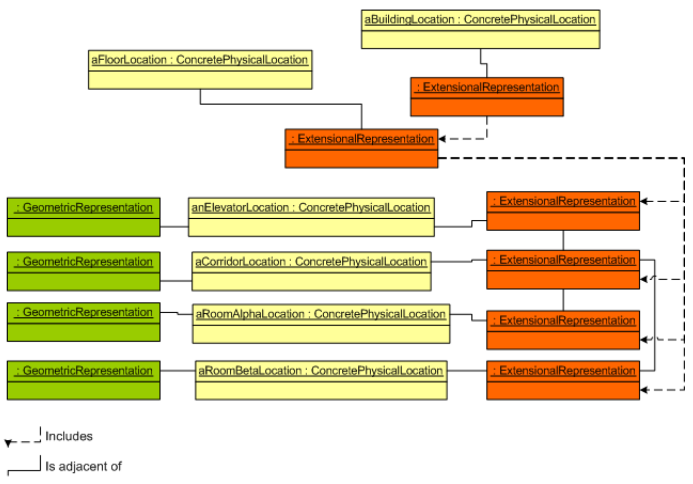

aBuildingLocation : ConcretePhysicalLocation
aFloorLocation : ConcretePhysicalLocation
: ExtensionalRepresentation
: ExtensionalRepresentation
: GeometricRepresentation
anElevatorLocation : ConcretePhysicalLocation
: ExtensionalRepresentation
: GeometricRepresentation
aCorridorLocation : ConcretePhysicalLocation
: ExtensionalRepresentation
: GeometricRepresentation
aRoomAlphaLocation : ConcretePhysicalLocation
: ExtensionalRepresentation
: GeometricRepresentation
aRoomBetaLocation : ConcretePhysicalLocation
: ExtensionalRepresentation
Includes
Is adjacent of

Fig. 15. Extended version of building's map with representations

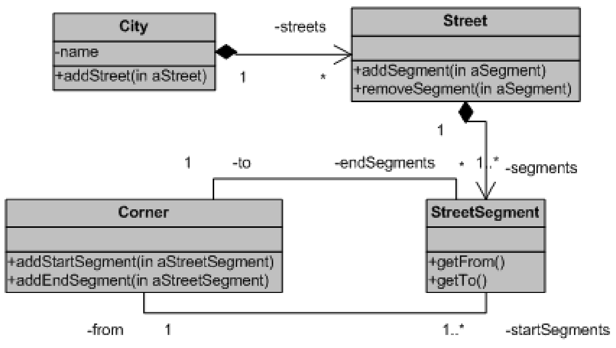


Fig. 16. Class diagram representing a simplified city

4. Towards pluggable location models

In our previous example we explained the need for integrating different maps; particularly we mentioned that our customer could provide us with a map of their buildings and we may want to integrate it with the streets' map. The integration of maps is important because we can build bigger and more complex maps by plugging little ones.

The streets' map contains the streets of the city and how they are connected. The map is useful for calculating directions. A simplified object-oriented model, containing only 4 classes: *City*, *Street*, *StreetSegment* and *Corner* are presented in Figure 16. Class *City* represents a container for the streets. *Street* knows all the segments that it contains. A *StreetSegment* corresponds to a piece of street between corners and a *Corner* is a point where the streets segments join.

The building is nearby the center of the city. It is crossed by two important streets and surrounded by a number of buildings. To model this area we have instantiated several objects to model the streets' map. A simplified version of the map is shown in Figure 17.

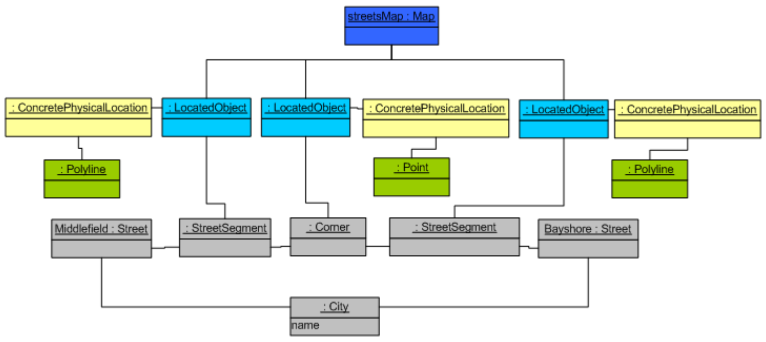


Fig. 17. Simplified version of the object diagram for the streets' map

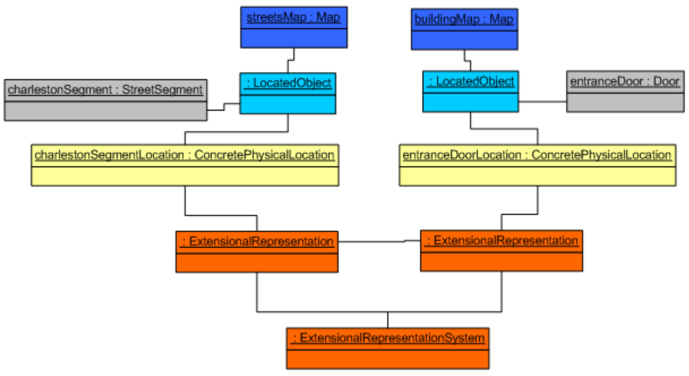

streetsMap : Map
buildingMap : Map
charlestonSegment : StreetSegment
: LocatedObject
: LocatedObject
entranceDoor : Door
charlestonSegmentLocation : ConcretePhysicalLocation
entranceDoorLocation : ConcretePhysicalLocation
: ExtensionalRepresentation
: ExtensionalRepresentation
: ExtensionalRepresentationSystem

Fig. 18. Integration of streets and building maps

We now show how to integrate and compose the maps. Both tasks are performed by adding new representations to the existing locations. In the first case, we need to express a new relationship between the door and the nearby street segment. The relationship should indicate the adjacency between the door and the nearby street segment. We express this relationship with the use of symbolic representations. We need to create two new representations and link them with the use of an adjacency link. Then, the locations of both objects need to be modified with the right symbols. For this task we need to instantiate a new object for representing the entrance door of the building, its corresponding located object and location instances. The objects and links created as a consequence of the integration of the maps are shown in Figure 18.

The composition between both maps is easier to achieve. We need to express that the city contains the building (and transitivity all the objects that the building contains). As we previously showed in Figure 17, the city was not considered as a located object on the streets map. We need to add a located object that wraps the *City* object with its location represented by a polygon. We chose to represent the location of the building as a point because it allows expressing the inclusion relationship between the city and the building. The locations of both objects are now modified by the incorporation of the new representations and the streets map is modified by the inclusion of the city located object (Figure 19).

Summarizing, our model allows extending the relationships of the existing located objects. Integrating or composing models is fairly simple: we need to define the representations needed to express the new relationships and then add them to the existing locations.

5. Domain representations

Domain and application models help to understand an application domain without delving into its details. For example, the streets domain model (Figure 16) could

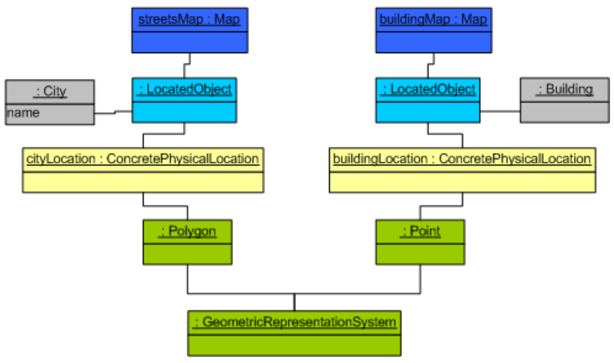

streetsMap : Map
buildingMap : Map
: City
name
: LocatedObject
: LocatedObject
: Building
cityLocation : ConcretePhysicalLocation
buildingLocation : ConcretePhysicalLocation
: Polygon
: Point
: GeometricRepresentationSystem

Fig. 19. Composition of streets and building maps

have been created without considering that the model will be used for location purposes in the future. However, sometimes, the model may expose some relationships that can be useful at the location level. For example, the relationship between the corners and the street segments can be used to express the adjacency relationship at the location level. The rule for this relationship is the following: if a street segment joins a corner then the street segment is adjacent to the corner and if a street segment starts from a corner then the corner is adjacent to the street segment. If we follow this rule then we can move along the streets segments of the city using it.

The adjacency relationship defined by the previous rule could be defined with symbolic representations. This kind of representations is easy to configure for this rule. However, if we don't pay special attention to the evolution of the model, we can obtain an inconsistency between the objects in the model with the links on the symbolic representation system. This approach is inappropriate for two reasons: we have to duplicate the effort each time we add streets to the model because we need to update the symbols, and it tends to generate inconsistencies regarding to the manipulation of the symbols.

Notice that the concept of representation also applies to this scenario because we can use domain objects and their relationships to represent physical relationships between locations. In this case, the classes *Corner* and *StreetSegment* and their relationships define an adjacency relationship at the location level. The idea behind domain representations is to reuse the relationships that may emerge from the domain model with the less possible effort, helping us not to duplicate the effort of maintaining both models: domain and location model. In our framework this requirement can be easily achieved as we aim to express new relationships from preexistent objects. For this, we define a new kind of representation called *DomainRepresentation*. Our framework contains two new subclasses of classes *Representation* and *RepresentationSystem* to implement this behavior. The *DomainRepresentation* class will have a subject which will be the domain object that it's adapting (Figure 20).

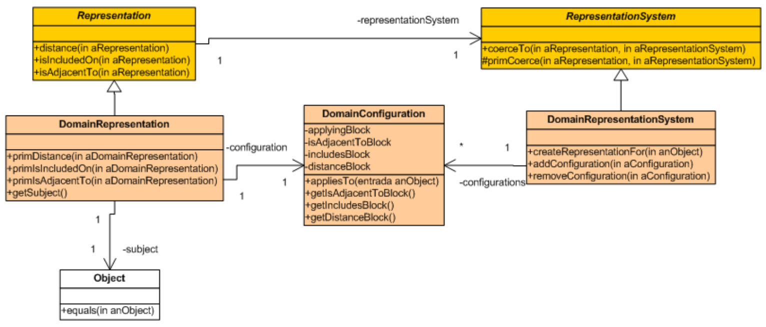

Representation
+distance(in aRepresentation)
+isIncludedOn(in aRepresentation)
+isAdjacentTo(in aRepresentation)
-representationSystem
1
1
RepresentationSystem
+coerceTo(in aRepresentation, in aRepresentationSystem)
#primCoerce(in aRepresentation, in aRepresentationSystem)
DomainRepresentation
+primDistance(in aDomainRepresentation)
+primIsIncludedOn(in aDomainRepresentation)
+primIsAdjacentTo(in aDomainRepresentation)
+getSubject()
-configuration
1
1
DomainConfiguration
-applyingBlock
-isAdjacentToBlock
-includesBlock
-distanceBlock
+appliesTo(entrada anObject)
+getIsAdjacentToBlock()
+getIncludesBlock()
+getDistanceBlock()
*
1
-configurations
DomainRepresentationSystem
+createRepresentationFor(in anObject)
+addConfiguration(in aConfiguration)
+removeConfiguration(in aConfiguration)
1
1
-subject
Object
+equals(in anObject)

Fig. 20. Domain representations class diagram

When a *DomainRepresentation* object receives a message like *isAdjacentTo* it adapts it to the right message at the domain level returning the value after its computation (Figure 21). The *DomainRepresentationSystem* acts like a factory for creating *DomainRepresentation* instances. This class should be configured with the appropriate behavior that a *DomainRepresentation* should adapt over the domain model.

Going back to the street segments and corners example, let's now define the rules using a Smalltalk notation. The first rule states that a street segment is adjacent to its incident corner. The configuration for this rule is:

```
domainRS := DomainRepresentationSystem new.
ruleConfiguration := DomainConfiguration
    appliesTo: [:object | object isStreetSegment ]
    isAdjacentTo: [:aSS :object | aSS getTo = object ]
    includes: [:aSS :object | false ]
    distance: [:aSS :object | Infinity positive ].

domainRS addConfiguration: ruleConfiguration.
```

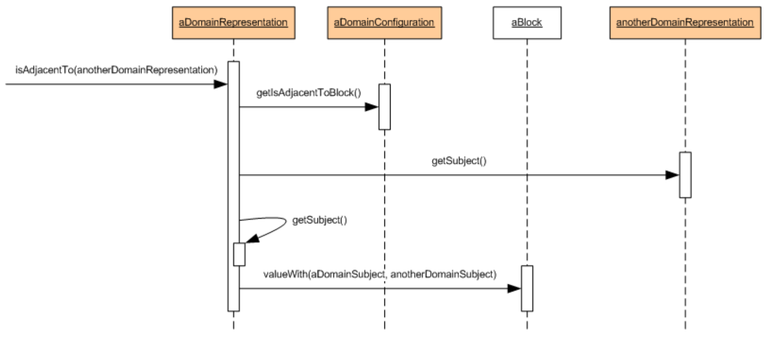

aDomainRepresentation
aDomainConfiguration
aBlock
anotherDomainRepresentation
isAdjacentTo(anotherDomainRepresentation)
getIsAdjacentToBlock()
getSubject()
getSubject()
valueWith(aDomainSubject, anotherDomainSubject)

Fig. 21. A sequence diagram of how a domain representation resolves the isAdjacentTo message

The second rule stats that a corner is adjacent to their street segments that have origin on it. We state this rule as:

```
ruleConfiguration := DomainConfiguration
    appliesTo: [:object | object isCorner ]
    isAdjacentTo: [:corner :object | corner startSegments
                                             includes: object]
    includes: [:corner :object | false ]
    distance: [:corner :object | Infinity positive ].

domainRS addConfiguration: ruleConfiguration.
```

This approach has a clear advantage: it simplifies maintenance because we don't need to duplicate the relationships that already exist in the domain model. Thus, when a relationship changes at the domain level, the representation will transparently reflect this because its operations are based on domain operations.

6. Putting models to work

6.1. *Introduction*

The idea of decoupling the location from how it is represented implies a new way of modeling locations. Locations can be represented in multiple ways, offering different implementations for expressing physical relationships. While at the representation level it is clear, for example, if a rectangle contains a point, we have so far ignored what happens at the location level. For example, two locations represented by a symbol and a geometric figure (e.g. the building and the corridor) will respond yes from a symbolic point of view to the question: do you include the corridor? But it will say no from a geometric point of view. At the location level we need to consolidate the result of the operation. For example, we may require that on of the values computed by the representations to be true to answer true at the location level. We are going to present different policies that are used in our framework to consolidate the results obtained from the representation level.

Adjacency and distance are important relationships between located objects, specially, when we want to compute paths between objects. In this section we will present an effective way for computing paths on huge models using heuristic algorithms. Our approach is based on using different heuristics while we move through the graph.

In the following subsections we show how our framework deals with operation computing, how we compute paths and as proof of concept we briefly describe an application that uses location information based on our framework.

6.2. *Operation resolution*

Let us recall the map of the building shown in Figure 15; it has several objects like rooms, corridors and the building itself. Suppose that we need to compute

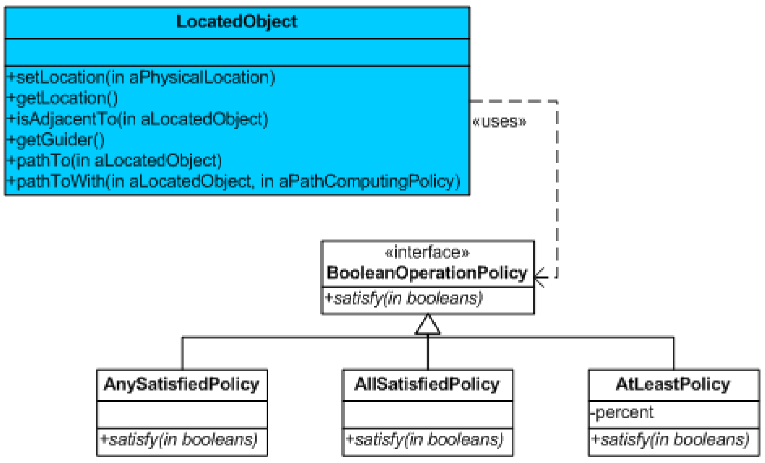


Fig. 22. Operation policies class diagram

the *isAdjacentTo* operation between room alpha and the corridor. The location of both objects has multiple representations: a symbol and a geometric figure in both cases. Computing the operation at the location level involves a specific processing at the representation level: perform the same operation between their compatible representations. In this case the symbols are compatible (they belong to the same representation system) and geometric figures too. The operation performed between symbols will answer “true”, because there is a link between them; however in the geometric case it will return “false” because we don’t have any information about adjacencies on a table. Therefore, the operations between the representations give a set of results: {true, false} that needs to be consolidated. The consolidation between these results is done with a policy that determines which is the answer of the operation. If the policy is not specified we assume an “any satisfied” policy or negation by failure for boolean operations such as inclusion and adjacency. This means that, if we cannot ensure that the operation is true, we answer false. In this case, one of the results is true so the operation will answer true. As an example, we show in Figure 22 the classes we have added and the new operation defined in the protocol of the *LocatedObject* class. A new message is added on the class so that we can set which is the policy we will use. The framework contains implementations that adjust to different situations. We have also implemented an *AllSatisfiedPolicy* and *AtLeastPolicy* which are used to ensure that all the values are true to answer true, and that at least a percent of the values are true to answer true respectively.

Notice that the approach we have followed is similar to what a chemist follows to take a measure: it repeats the same experience and consolidates their results according to some process. In the distance case for example, we can estimate a physical distance by taking many measures and averaging them. This is quite similar to representations. Each time that a pair of representations computes a distance, they give us a measure. Then, all measures can be averaged or filtered to obtain the final result.

6.3. *Path computing*

Path computing is a key aspect of applications that use locations. With the approach we have adopted, we can plug models by expressing the relationships that objects have between each other. This task is done by creating representations that express the new relationships among objects. Using this approach we can plug maps which were conceived independently. Moreover, the composition is somewhat oblivious to those maps. Maps may have many different objects so it's just a matter of software evolution to have millions of objects. Path computing in this scenario is somewhat complicated and algorithms such as Dijkstra [30] cannot be naively applied.

To overcome the problem of finding a path between an origin and a destination we introduced the use of heuristic algorithms on our framework. This kind of search algorithms has proven to be successful for huge graphs in the IA area [31] if the heuristic function is admissible, that is, if it never overestimates the costs of reaching the destination. In this area two of the most notable algorithms are Greedy and A* [31]. The Greedy algorithm tries to obtain the locally optimal choice with the hope of finding the global optimum. On the other hand, the A* algorithm is similar to Greedy but it has two important differences: it takes into account the distance already traveled from the origin to choose the next node and it uses the mathematical distance function as part of the heuristic function.

These two advantages promote the usage of a slightly modified version of the A* algorithm. This algorithm is based on a heuristic function h that is applied over every node, in our graph of located objects. Those objects that are on the frontier (that can be introduced in the visited nodes in the next state) are chosen based on *h*. The object o with less value of *h(o)* is the one introduced in the visited nodes set. This iteration is repeated until the node we want to reach is the one chosen or if there are no more objects in the frontier set.

The modification we introduced to the A* algorithm is based on the usual heuristics people use when wanting to travel from one place to another which is not close. We need some guidance to correctly reach the destination place. Suppose one of our salesmen wants to travel from a room in Madrid to another room in Paris using a car. We are inside the building and we need some guides to get outside Madrid in order to go to Paris. So, we ask the guard of the building how to go to Paris. He tells us that we are far away from Paris, so first we need to go outside the building; we do so. We get into the car and ask a police officer how to go to Paris. The police officer tells us to go out of Madrid first. So he explains us how to go move through the streets of the city. If we continue with this idea we end asking different people how to reach a specific located object. Each guider will help us to choose among the possibilities we have to go to a specific place.

In our model, each located object knows a *Guider* that is able to provide the

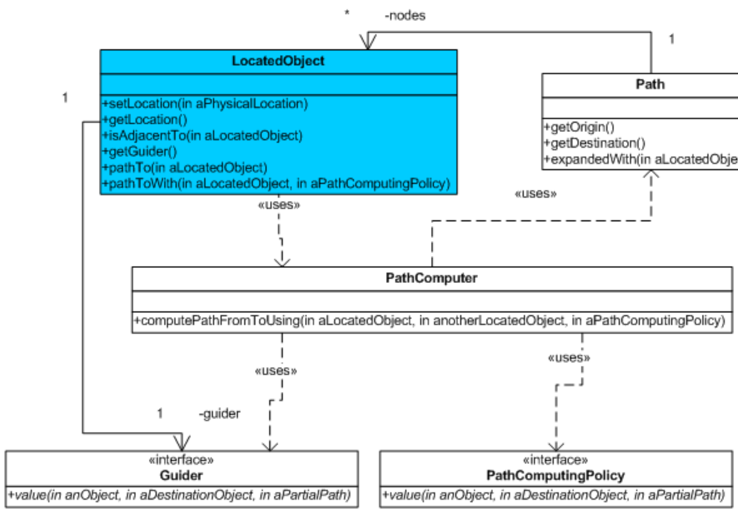

LocatedObject
+setLocation(in aPhysicalLocation)
+getLocation()
+isAdjacentTo(in aLocatedObject)
+getGuider()
+pathTo(in aLocatedObject)
+pathToWith(in aLocatedObject, in aPathComputingPolicy)
Path
+getOrigin()
+getDestination()
+expandedWith(in aLocatedObject)
PathComputer
+computePathFromToUsing(in aLocatedObject, in anotherLocatedObject, in aPathComputingPolicy)
«interface» Guider
+value(in anObject, in aDestinationObject, in aPartialPath)
«interface» PathComputingPolicy
+value(in anObject, in aDestinationObject, in aPartialPath)
«uses»
-nodes
-guider
*
1

Fig. 23. Path computing

heuristics when the algorithm is computed. The *Guider* is an interface that guiders must implement to answer the heuristic value. As an example, let's look at the building's guider. In this case, we choose to implement the guider very simply. This guider will have a set of located objects that he knows (Figure 23) and a subset of this set that we are going to consider exit objects. The guider will help by using his knowledge set: if the located object we want to reach is in this set, then it will give the best heuristic (which will guide to the destination) to the nearest located objects of the destination in the frontier set. If not, it will give the best heuristic to the nearest located object in the exit set. This guider is useful for indoor situations because, if the located objects are inside the building, then the guider will know them. If not, the guider will guide it to one of the exit located objects.

As previously mentioned we have chosen to travel by car. In path computing there are some aspects that may change the ways paths are computed. These aspects are subjective because it is the user who decides if he wants to walk, use the public metro, etc., at the moment of the path computing. For example, in the universe of located objects, we may find some cases where a street segment is adjacent to other objects that model walking paths. Why shouldn't we choose among these objects to be intermediate objects in a path? The answer in this situation is that we want to travel by car, not walking; thus we need some policy that users may configure in order to define their preferences for path computing. For this aim we introduce a path computing policy. This policy gives us another heuristic that is used together with the one the guider computes. Therefore, the *Guider* computes the heuristic using a *PathComputingPolicy*. Each time he computes a heuristic, he adds it with the one the *PathComputingPolicy* has computed.

Continuing with our example, let's suppose that we have areas which are considered "dangerous" and we want to skip them in our trip from Madrid to Paris. Also, we would like to use highways and avenues whenever is possible. To do so, we implement a *CompoundPathComputingPolicy* which is a composite of *PathComputingPolicy* that computes the heuristic by adding the heuristics of its children so that both: dangerous areas and highways can be considered at the same time. To skip dangerous areas we implement a *SkipAreasPathComputingPolicy* which basically contains a set of polygon areas, if a located object is included in one of these areas then the heuristic is Infinity otherwise is the physical distance between the object and the destination:

```
>>value: anO to: destinationO partialPath: path
^(self areas anySatisfy:[:area | area includes: anO])
ifTrue:[Infinity positive]
ifFalse:[anO distance: destinationO]
```

In the same way we implement a *WeightPathComputingPolicy* that weights the physical distance between the current located object (CLO) and the destination based on the class. For instance, if the CLO is a *Highway* then we multiply the distance by 0.9, if CLO is an *Avenue* then we leave the distance as usual, otherwise we multiply by 1.2:

```
wpcp := WeightPathComputingPolicy new.
wpcp addConfigurationFor: Highway weight:[:distance | distance * 0.9].
wpcp addConfigurationFor: Highway weight:[:distance | distance].
wpcp addOtherwiseConfiguration:[:distance | distance * 1.2].
```

The implementation is shown as follows:

```
>>value: anO to: destinationO partialPath: path
  |configuration|
  configuration := self configurations at: anO class.
  ifAbsent:[ ^self otherwiseBlock value: (anO distance: destinationO)].
  ^configuration value: (anO distance: destinationO).
```

Finally, the configuration of the *CompoundPathComputingPolicy* is shown next:

```
pathComputingPolicy := CompoundPathComputingPolicy new.
pathComputingPolicy addPolicy: wpcp.
pathComputingPolicy addPolicy: (SkipAreasPathComputingPolicy areas: areas)
```

Although we show a rather statically implementation of a *PathComputingPolicy* we can implement a more dynamically version that takes into account traffic reports to skip traffic jams.

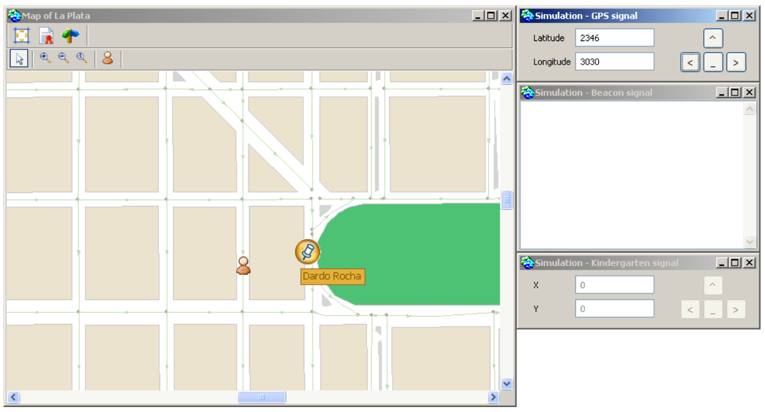


Fig. 24. Example look of the application

6.4. *Implementation and proof of concept*

We have implemented this framework based on the idea of decoupling locations from how they are represented using the Visualworks 7.4 NC Smalltalk environment [32]. We took this implementation decision because Smalltalk provides a good platform for testing pure oriented object solutions. Particularly, we have used blocks to implement domain representations. A more "conventional" object-oriented solution, such as using Java, would need some modifications in the corresponding class-hierarchy

On top of the basic implementation we have developed some applications as a proof of concept for this approach. We will detail the one which uses the same kind of problem we have used as an example in the previous sections. The application's purpose is to give assistance to the attendees of a conference. The conference is held on a city but spread on multiple buildings. We already have the streets maps of the city. Also, the conference committee has developed a map for each of the building where the conference takes place. They give details about the corridors, rooms and the schedule for the conference. In the application the attendee (user) can request to compute paths between rooms on different buildings. Also it can take a look at a resume of the talk he is assisting. The sensing aspect of the application, which gives the location of the attendee, is simulated with a set of panels (Figure 24). We assume that the mobile device has the capacity of sensing GPS and beacon signals.

With little effort, we connect the maps detailing which objects on the "frontier" of the buildings are connected with the street segments. As an example we show how we connect one of the buildings with the streets map. In this case we chose to determine a geometrically represented zone of the building to be the exit zone. This zone is next to the entrance door and is near to a street segment that passes by the building. So, from our point of view, we can consider them adjacent and we represent this relationship adding two new symbols to the locations of the objects in the same way we have done with the example (Figure 18). Also, the building is

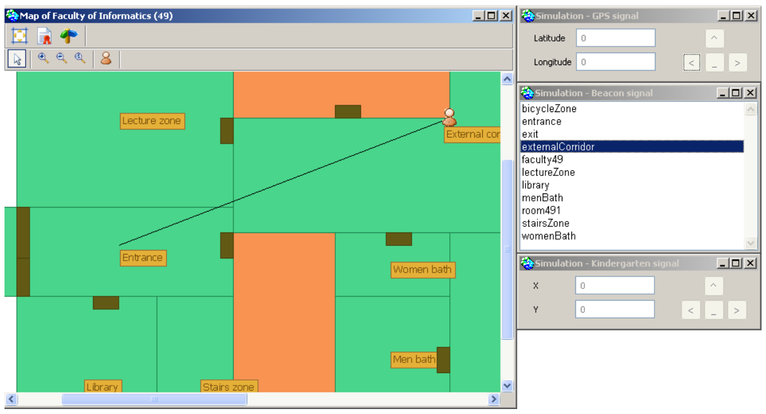


Fig. 25. Path computing from physically spread rooms

configured with the guider we have suggested in the path computing section. The guider knows as an exit point the exit zone of the building which gives the necessary information for path computing.

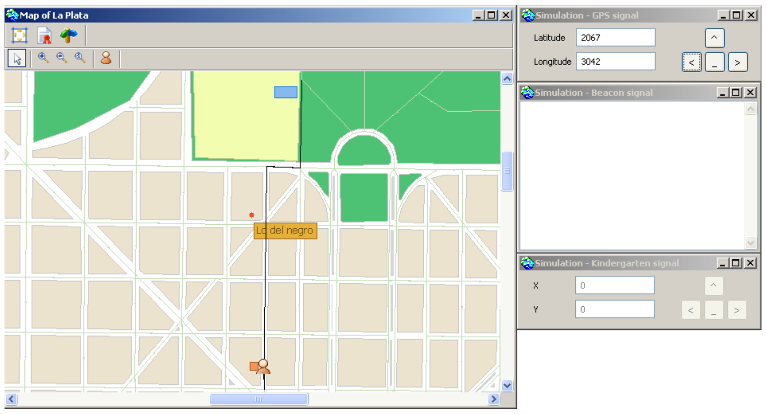
To compose the building location in the streets map we use the same simple strategy we used in the example. We chose to represent these locations with points that describe the position of the building in the streets map.

As the conference is physically spread into multiple buildings we need to give assistance to the attendees on how to move from one room to the other. Suppose one of the attendees is located inside of one of the buildings and he wants to attend to a talk which will take place in a room on another building. The application allows selecting one of the talks and requesting a path to the place where it will take place. The path is shown as a polyline that guides the attendee to his next talk (Figure 25). When the attendee gets out of the building, the application automatically notice the GPS signal and changes the location of the user to the right map keeping the computed path so that the user gets feedback of his location and the path he has to travel (Figure 26).

7. Related Work

Location modeling has attracted the attention of many areas in computer science as it use is a key concern for context aware, location aware and location base services. In each of these areas there have been many efforts to develop different models to easily handle new requirements. These models can be classified into two sets:

- Ad-hoc models which try to solve particular problems of the application that was under development
- Abstracted models that are built in the form of framework allowing developers to instantiate and extend them.



Simulation - GPS signal
Latitude 2067
Longitude 3042

Simulation - Kindergarten signal
X 0
Y 0

Fig. 26. View of the path to reach the destination room

We concentrate on the second set of models which have captured the conceptual ideas behind some of the solutions of ad-hoc implementations. The semi symbolic model [18] introduced by Leonardh was developed to get the advantages of geometric and symbolic models. Each location is modeled as a pair (geometric, symbol). The geometric information is used to compute physical distances between locations and the symbolic is used to compute relationships between locations such as inclusion and adjacency. This model was one of the firsts that introduced the idea of combining more than one model to fully model a location. Our research shows that it is better to decouple the location from how it is represented in a particular model, such as the geometric or the symbolic ones. The semi-symbolic model forces to do a modification if we want to represent new relationships between the locations. In our work this can be easily handled by adding a new representation that gives the semantic of the new relationship. Also, our work is not constrained to these types of models (geometric, symbolic). We can take advantage of reusing the domain model to express physical relationships between different located objects.

Semantic locations in the form of URLs [25, 26] are mapped from different information sources such as bar codes, GPS points and beacon signals to an URL. This mapping helps to provide a unique access to the same location. The application running at the URL could provide location services for the area in which the user is located, although the application doesn't know exactly the location of the user. This approach introduces the notion of mapping information to a single object. Each time we map a new piece of information to the same URL we are enhancing the previous information we have mapped. Although the model provides a mapping from sensed data to the URL, it doesn't provide and inverse mapping from the URL to the representations as the mapping will give many representations. This model becomes similar to our work as it deals with geometric and beacon signals as representations of the same location. Also, it introduces the mapping to a unique concept which allows enhancing the location each time we represent it on a different representation system. In our work, we allow multiple representations for the same location so each representation expresses some of the relationships between the environment and the location. We also allow a knowledge relationship between the location and the representations because the location computes its operations using the representations, so a location must know how it is represented.

The hybrid location model [19] is similar to the semi symbolic approach. This model merges a symbolic location that expresses a containment relationship with optionally geometric information. Each location (named ali) is represented by a path in the form of: *a/b/c* where the character "/" represents the containment relationship. A complete location is modeled like *ali://a/b/c#(0,1)*. The ali:// is the way we access this type of locations. Then *a/b/c* is the symbolic hierarchy and the information beyond character "#" is the geometric shape of the location. The model exposes a collection of functions that can be applied over alis such as: *distance(ali, ali)* and *contains(ali, ali)*. The model is similar to the semi symbolic approach of Leonarth as it tries to use the advantages of each existing model. The *ali://* and the way the location is represented, shows the intention of modeling the location with a URI. This fact helps the model to be unambiguous when it tries to express a location. The same idea of representing the location with multiple models is adopted in our work. In this way, we combine many of the benefits of working with geometric and symbolic locations. The key difference between the hybrid model and our work is that the hybrid model constraints the way location can be represented to symbolic and geometric ones. Our work focuses on multiple representing the same location but not restricted to geometric and symbolic ones. In section 5 we showed how to take advantage of the domain model to enhance the location concern of the application.

The realms and states model [33] formalizes the approach introduced by the semi-symbolic approach. A location is modeled as a state on a particular realm. A realm can be seen as a set where their elements are states. States are related with raw geometric information for positioning purposes. The symbolic part of the model is obtained by the relationships that the realms has. For example, we can define a containment relationship between a state X in the realm 1 and the realm 2 and thus a state in the realm 2 inherits the property of been contained on X. This model guarantees some properties as it is formally conceived but it is constrained to raw locations and the realm, state concept. We use another approach distinguishing that a location serves as a way of expressing the relationships that a particular place has with its environment. As the possible relationships are infinite, the model decouples the location from how it is represented on a particular model. This helps the model to be easily extended with other representations while the application is running. By adding new representations we enrich the physical semantics of the location.

To summarize, our model distinguishes a location as a first class object that could be represented in many ways. The way we represent a location can be (but it is not restricted to): geometric shapes, symbols and even domain objects. Decoupling these concepts helps the model to be dynamically enriched with other representations of the same location. A location that was firstly conceived as the point (12, 24) on a Cartesian plane is enriched when we add an adjacency relationship with a symbol. This approach helps people to establish different relationships with the use of representation systems. Plugging these representations to locations enriches the previous locations without modifying the located objects.

8. Concluding Remarks and Further Work

In this paper we have presented a novel approach for dealing with the integration and composition problems found when using different location models in ubiquitous applications. We showed that the approach has several advantages:

- It abstracts the location concept in such a way that multiple representations are possible for a particular location (using for example, geometric figures, symbols, urls, etc.).
- It is easy to extend the relationships which arise when integrating and composing different models, by adding new representations to existing locations.
- It allows accommodating other types of representations such us domain representations helping to reuse domain relationships.

In the paper we have discussed several further problems which are summarized below.

Having multiple representations for a single location poses the necessity of consolidating the results obtained when operating with them. For this purpose we proposed the usage of different policies and an architectural solution to deal with them modularly; these policies performed very well in the applications we developed.

Composing and integrating models implies that a huge number of objects might exhibit location properties; therefore the need for path computing algorithms that perform acceptably is a must. We presented a modified version of the A* algorithm that behaves correctly with the introduction of the guider concept.

We are working in two different fields in order to extend and improve the presented approach. We are analyzing the introduction of probabilistic information in each of the representations in order to deal with vague information. This addition must be taken into account by the computing policies to get a real benefit. As an example, we can have the location of a person represented by a point (GPS signal) and by an ellipse (GSM signal) with different probabilities depending on the signal quality. If we want to compute the operation *distance*, we could use a policy that only takes into account those representations with a probability higher than 0.5. This example illustrates the necessity of developing new policies that take into account the probability of representations.

We are also integrating our framework into a more general architecture for building ubiquitous applications in two domains: physical hypermedia [11, 12, 13] and location-based services [6, 7]. The challenge in this integration lies in the fact that both domains pose their own challenges to the way locations are managed. The separation between located object, location and representation provides a good grain of hooks to plug the components needed to deal with locations with other objects in the architecture.

References

[1] Robert Laurini and Derek Thompson. *Fundamentals of Spatial Information Systems.* Academic Press, 1992.

[2] Jing Dai and Chang-Tien Lu. Clam: concurrent location management for moving objects. In *GIS '07: Proceedings of the 15th annual ACM international symposium on Advances in geographic information systems*, pages 1–8, New York, NY, USA, 2007. ACM.

[3] Matthew Perry, Farshad Hakimpour, and Amit Sheth. Analyzing theme, space, and time: an ontology-based approach. In *GIS '06: Proceedings of the 14th annual ACM international symposium on Advances in geographic information systems*, pages 147–154, New York, NY, USA, 2006. ACM.

[4] Elzbieta Malinowski and Esteban Zimányi. Representing spatiality in a conceptual multidimensional model. In *GIS '04: Proceedings of the 12th annual ACM international workshop on Geographic information systems*, pages 12–22, New York, NY, USA, 2004. ACM.

[5] A. Belussi, B. Catania, and E. Bertino. A reference framework for integrating multiple representations of geographical maps. In *GIS '03: Proceedings of the 11th ACM international symposium on Advances in geographic information systems*, pages 33–40, New York, NY, USA, 2003. ACM.

[6] Roy Want, Andy Hopper, Veronica Falcao, and Jonathan Gibbons. The active badge location system. *ACM Trans. Inf. Syst.*, 10(1):91–102, 1992.

[7] Nissanka B. Priyantha, Anit Chakraborty, and Hari Balakrishnan. The cricket location-support system. In *MobiCom '00: Proceedings of the 6th annual international conference on Mobile computing and networking*, pages 32–43, New York, NY, USA, 2000. ACM Press.

[8] Andrew Capella Valerie Bennett. Location based services. wherever you are, wherever you go, get the information you want to know.

[9] Anind Kumar Dey. *Providing architectural support for building context-aware applications.* PhD thesis, Georgia Institute of Technology, 2000. Director-Gregory D. Abowd.

[10] Paul Dourish. What we talk about when we talk about context. *Personal and Ubiquitous Computing*, 8(1):19–30, 2004.

[11] Kaj Gronbaek, Jannie F. Kristensen, Peter Orbaek, and Mette Agger Eriksen. "physical hypermedia": organising collections of mixed physical and digital material. In *HYPERTEXT '03: Proceedings of the fourteenth ACM conference on Hypertext and hypermedia*, pages 10–19, New York, NY, USA, 2003. ACM Press.

[12] Frank Allan Hansen, Niels Olof Bouvin, Bent G. Christensen, Kaj Gronbaek, Torben Bach Pedersen, and Jevgenij Gagach. Integrating the web and the world: contextual trails on the move. In *HYPERTEXT '04: Proceedings of the fifteenth ACM conference on Hypertext and hypermedia*, pages 98–107, New York, NY, USA, 2004. ACM Press.

[13] Simon Harper, Carole A. Goble, and Stephen Pettitt. proximity: Walking the link. *J. Digit. Inf.*, 5(1), 2004.

[14] Mark Weiser and John Seely Brown. The coming age of calm technolgy. pages 75–85, 1997.
[15] Adam Greenfield. *Everyware: The Dawning Age of Ubiquitous Computing.* Peachpit Press, Berkeley, CA, USA, 2006.
[16] Scott Elrod, Richard Bruce, Rich Gold, David Goldberg, Frank Halasz, William Janssen, David Lee, Kim McCall, Elin Pedersen, Ken Pier, John Tang, and Brent Welch. Liveboard: a large interactive display supporting group meetings, presentations, and remote collaboration. In *CHI '92: Proceedings of the SIGCHI conference on Human factors in computing systems*, pages 599–607, New York, NY, USA, 1992. ACM.
[17] Christopher Kent Kantarjiev, Alan Demers, Ron Frederick, Robert T. Krivacic, and Mark Weiser. Experiences with x in a wireless environment. In *MLCS: Mobile & Location-Independent Computing Symposium on Mobile & Location-Independent Computing Symposium*, pages 11–11, Berkeley, CA, USA, 1993. USENIX Association.
[18] U. Leonhardt. Supporting location-awareness in open distributed systems, 1998.
[19] C. Jiang and P. Steenkiste. A hybrid location model with a computable location identifier for ubiquitous computing, 2002.
[20] Gregory D. Abowd. Software engineering issues for ubiquitous computing. In *ICSE '99: Proceedings of the 21st international conference on Software engineering*, pages 75–84, Los Alamitos, CA, USA, 1999. IEEE Computer Society Press.
[21] Svetlana Domnitcheva. Location modeling: State of the art and challenges. Workshop on Location Modeling for Ubiquitous Computing, 2001.
[22] B. Brumitt and S. Shafer. Topological world modeling using semantic spaces, 2001.
[23] G Rossi, Javier Bazzocco, Silvia Gordillo, and Robert Laurini. "Designing Evolvable Location Models for Ubiquitous Applications, 2003. In Proceedings of the 9th International Conference on Object-Oriented Information Systems (OOIS'03), Geneva, 2-5 September 2003. Springer Verlag LNCS 2817, pp 289-293.
[24] ESRI. Esri shapefile technical description.
[25] Salil Pradhan. Semantic location. *Personal Ubiquitous Comput.*, 4(4):213–216, 2000.
[26] Haibo Hu Dik-Lun. Semantic location modeling for location navigation in mobile environment.
[27] Bill Schilit and M. Theimer. Disseminating active map information to mobile hosts. *IEEE Network*, 8(5):22–32, 1994.
[28] Erich Gamma, Richard Helm, Ralph Johnson, and John Vlissides. *Design Patterns.* Addison-Wesley Professional, January 1995.
[29] B. Woolf. The null object pattern, 1996.
[30] Thomas H. Cormen, Charles E. Leiserson, Ronald L. Rivest, and Clifford. *Dijkstra's algorithm*, chapter 24.3, pages 595–601. MIT Press and McGraw-Hill, second edition, 2001.
[31] Korf. Artificial intelligence search algorithms. In *Algorithms and Theory of Computation Handbook, CRC Press, 1999.* 1999.
[32] Inc Cincom Systems. Visualworks. http://www.cincomsmalltalk.com/.
[33] Ajith K. Narayanan. Realms and states: a framework for location aware mobile computing. In *WMC '01: Proceedings of the 1st international workshop on Mobile commerce*, pages 48–54, New York, NY, USA, 2001. ACM Press.