Designing Evolvable Location Models for Ubiquitous Applications

Silvia Gordillo[1], Javier Bazzocco[1], Gustavo Rossi[1],
and Robert Laurini[2]

[1] Lifia. Facultad de Informatica
Universidad Nacional de La Plata,
Argentina
{gordillo,jb,gustavo}@lifia.info.unlp.edu.ar
[2] INSA-Lyon, France
Robert.Laurini@lisi.insa-lyon.fr

Abstract. In this paper we present an object-oriented approach for building location models in the context of ubiquitous applications. We first motivate our research by discussing which design problems we face while building this kind of applications; we stress those problems related with applications' evolution. We then present a set of simple design micro-architectures for representing locations and their interpretation. We finally discuss some further research work.

1 Introduction

In the last 5 years, we have experienced an increasingly interest in the development of ubiquitous applications, i.e. those applications that follow the anytime/anywhere/any-media paradigm and provide transparent access to information and other kind of services trough different (in general portable) devices. One of the most important features of these applications is their ability to gracefully adapt themselves to the user's context, e.g. his location, the device he is using (a laptop, palm computer, cell phone, etc), his preferences, etc. Research issues related with ubiquitous computing range from hardware (small memory devices, interface appliances) and communication networks (trustable connections, security, etc) to software and data management aspects such as new interface metaphors, data models for mobile applications, continuous queries, adaptive applications, information exchange between disparate applications, etc.

In this paper we address one of the interesting facets of these applications: their evolution. According to Abowd [1]: "Ubicomp applications evolve organically. Even though they begin with a motivating application, it is often not clear up front the best way for the application to serve its intended user community". As a consequence, design issues are critical for the application to evolve seamlessly when requirements change. In our research we are pursuing the identification of a set of design micro-architectures to build evolvable location models, i.e. those application components that represent the user location and which are used to customize the application's behavior accordingly.

D. Konstantas et al. (Eds.): OOIS 2003, LNCS 2817, pp. 289–293, 2003.

Suppose for example a simple application to help the user move through a city like Paris; while using his preferred device he can be informed about how to go to a place from where he is now, which hotels and restaurants he can find in the neighborhood, etc. In our first application's release we assume that we can obtain the user's location in terms of the address where he is and we use a cartography service such as [5] to inform him what he needs. Existing state-of-the-art technologies such as positioning devices and Internet cartography [7] make this scenario absolutely feasible. Being the application successful, we want to integrate it with a new component that helps the user guide through the Metro (or bus) network. Using a new set of positioning artifacts like beacons [6], we know in which station he is and we can tell him how to go where he wants. Notice that we now need to represent the location as the name of a Metro station or bus stop. Eventually, some stations (huge ones) will have their own information systems offering shops and bars and we might need to guide him through the station; once more, the location representation changes and the functionality needs to be extended. When he enters a Museum the problem has a new shift: if we are able to know the artwork the user is watching (another kind of "location"), we may want to explain him some facts about its author, the historical context, etc. There is no need to say that the application's structure might get rather complex and evolution and maintenance may become a nightmare when we add new location classes and contexts for these kinds of queries.

The structure of this paper is as follows: In Section 2 we discuss why we should carefully design the location model. Next we outline our solution by presenting an adaptive location model and carefully describing its most important features. We then summarize evolution issues related with locations. In section 3 we present some further work and concluding remarks

2 Designing a Flexible Location Model

The above scenario shows that we face a set of problems regarding the structure of classes related to the representation of locations; while rule-based approaches (see for example [4]) can help in expressing context-related expressions such as: If the user is in position X, execute action A, they do not suffice to solve other problems like those presented in the introduction. More precisely we have the following design problems:

1 Objects representing locations have different attributes according to the positional system we use. It may be not possible or reasonable to define new classes each time the application evolves.

2 The way in which we interpret the location's attributes varies with the context (for example *x, y* in a local Cartesian system or in a global positioning one).

3 For each new kind of location we might need new ways to calculate distances, trajectories; moreover, new services, previously unforeseen, may appear.

4 The "granularity" of locations might change, e.g. we want to see the station as a point in the Metro network or as a building with corridors, rooms, facilities, etc.

For the sake of conciseness we will only address points 1 to 3 above. We assume an object-oriented representation of the geographic objects as discussed in [3].

2.1 Using an Adaptive Object Model for Locations

To solve the first problem indicated above we use a generalization of the Type Object Pattern, named "Adaptive Object Model" in [8], replacing different location classes with a generic class *LocationType* whose instances are different types of locations as shown in Figure 1. Each Location type defines a set of property types, having a name and a type (class *PropertyType*). Instances of *Location* contain a set of properties (instances of class *Property*) each one referring to one property type. Using the "square" in Figure 1 we can manage the meta (or knowledge) level by creating new instances of the "type side" (at the right) and the concrete level by creating new instances of classes in the left.

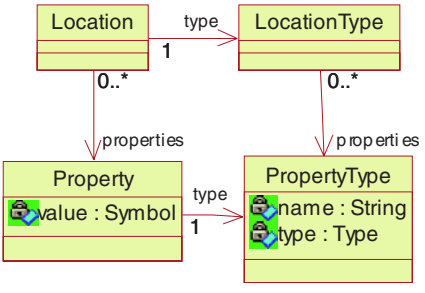

Fig. 1. Adaptive model for locations and their properties.

By this mean, adding new types of locations is not restricted by the code, compile & deploy process, which is known to be a very "static" solution. By using the preceding approach, the definition of a new kind of location can be easily made by arranging the required properties instances as needed (each one of them belonging to a particular type of property). The static definition of the structure imposed by the classes approach is changed in favour of the more dynamic alternative presented by the "square" solution presented above. This can be done primarily because the differences found in different types of locations resides in their structure rather than in their behaviour.

2.2 Decoupling Location from Its Context

It is clear from the discussion above that certain computations (distance, trajectories, etc.) depend on the interpretation of the location attributes, being them coordinates, street names, rooms in a museum, etc. We have generalized the idea of reference system defined in [3] which is used to decouple latitude, longitude pair from the corresponding (global) reference system. In this way we define the *LocationContext* class hierarchy shown in Figure 2; each class defines a new application context for locations providing specific behaviors according to the specific context. Usually, location contexts are singletons since we can see them as providing behaviors that do not depend on the particular location object. Some location classes may just act as adapters of existing applications (e.g. a museum information systems) to improve interoperability. Notice that class behaviors will be generally mapped to interface

options in the user's device. Modeling *LocationContext* classes as typed objects as in 2.1 is also possible, though we do not discuss it in this paper. Each location object collaborates with the corresponding context to be able to answer usual queries such as: how do I reach a place, how far am I from somewhere, etc.

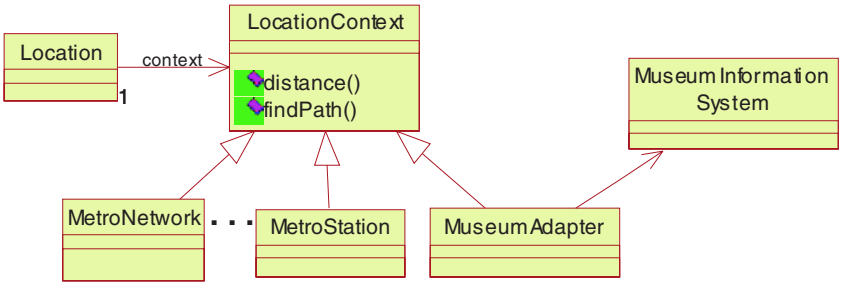

Fig. 2. The relationship between the location and its context

2.3 Adding New Functionality

One of the most complex aspects related with the evolution of location contexts is the fact that new unforeseen functionality may arise, e.g. while in a museum, operations to know more about an artwork; in the metro network operations to find the shortest path between two stations, etc. In our model we implement context-specific operations as Commands [2]. New operations are just implemented as new classes, which are integrated seamlessly in the model as shown in Figure 3.

It is neccesary to note that a solution based on new classes to add new functionality, as the Command Pattern states, is particularly useful since no changes to the existing system has to be done in order to use the new function. Here the "new class" approach is used since the different commands vary in their behaviour rather than in their structure, which is the opposite of the situation presented in 2.1.

While location contexts have a simple polymorphic interface (providing primitive operations for computing distance, trajectories, etc), an interface for commands is also provided. Thus, a location context is able to *execute (acommand)* which is an instance of one of the sub-classes of Command.

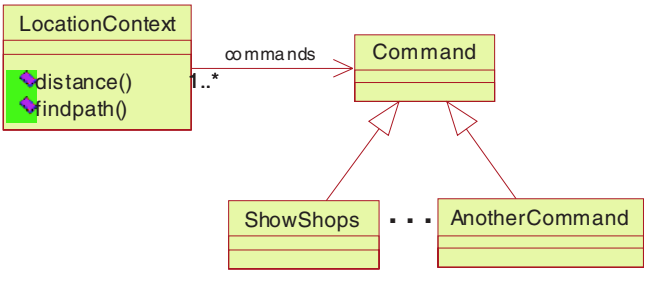

Fig. 3. Adding new functionality

Location contexts may refine specific locations; for example (as discussed above), we might want to see the station either as a node in the network or we might be "inside" the station. We express this possibility with the relationship "refines" between LocationContext and *Location* (not shown in the diagram) that allows us to easily provide the user with all behaviors associated to the set of possible nested contexts.

3 Concluding Remarks and Further Work

In this paper we have discussed the problem of dealing with evolvable location models; this problem is typical of ubiquitous information systems because of the way in which they grow, i.e. as new communication and positioning services appear new unforeseen functionality has to be added. We have shown that using a combination of an adaptive object model with varying location contexts, we can make the evolution seamless by eliminating the need to modify existing classes or code. Furthermore, decoupling new functionality from location classes by using commands, we can also cope with the addition of new behaviors. We are now studying the integration of location models into applications that deal with more general kind of geographic behaviors. In this kind of software, application objects (cities, stations, etc) are generally geo-referenced and described using pre-defined topologies such as points, lines, polygons, etc. We are studying the impact of combining location models with discrete and continuous geographic models.

References

1. Abowd, G.: Software Engineering Issues for Ubiquitous Computing. Proceedings of the International Conference on Software Engineering (ICSE 99), ACM Press (1999), 75–84
2. Gamma, E., Helm, R., Johnson, J., Vlissides, J. : Design Patterns. Elements of reusable object-oriented software, Addison Wesley 1995.
3. Gordillo, S., Balaguer, F., Mostaccio, C., Das Neves, F. Developing GIS Applications with Objects: A Design Pattern Approach. GeoInformatica. Kluwer Academic Publishers. Vol 3:1, pp. 7–32. 1999.
4. Kappel, G., Proll, B., Retschitzegger, W.: Customization of Ubiquitous Web Applications. A comparison of approaches. International Journal of Web Engineering and Technology, Inderscience Publishers, January 2003
5. Navigation Technologies Corporation, www.navtech.com
6. Pradham, S.: Semantic Location. Personal and Ubiquitous Computing. Springer Verlag 2002 (6) 213–216.
7. Virrantaus, K., Veijalainen, J., Markkula, J.: Developing GIS-Supported Location-Based Services. Proceedings of the Second International Conference on Web Information Systems Engineering (WISE'02).
8. Yoder, J., Razavi, R.: Metadata and Adaptive Object-Models, ECOOP 2000 Workshops, in www.adaptiveobjectmodel.com