Extending the Client Side Object Model in the Location API for J2ME^{TM}

A position paper for the 11th ECOOP MOS Workshop

David Parsons
Institute of Information and Mathematical Sciences
Massey University, Auckland, New Zealand
d.p.parsons@massey.ac.nz

Abstract

Location based service applications comprise a number of layers, distributed between clients and servers, either (or both) of which may be mobile. One of these layers is the Application Programming Interface (API) that enables an application to acquire information about the location of mobile devices and points or areas of interest. A number of location APIs have been developed that are intended for use by applications running on large terminals. However more recently we have seen the development of APIs that are designed to be hosted by small mobile clients. In this paper we explore the Location API for J2ME, a standard Java mobile client API that is intended to provide a generic interface to multiple positioning technologies. In particular we investigate the relationship between the client side object model and some resource constraint issues that impact on the practical usage of this API as part of a larger location-based service infrastructure. We also consider how the current object model might evolve to provide a more comprehensive framework for location based services.

1. Location APIs

Location based service applications are supported by APIs that give access both to simple objects that encapsulate raw geolocation data and other more complex objects that may represent points or areas of interest, mobile components, context information or routing advice. The various underlying positioning technologies that support such applications can be categorised as either device aware, where a mobile communications infrastructure is able to ascertain the location of devices within it, or location aware, where a device is able to ascertain its own location without assistance from the infrastructure [Bu04]. Most published location APIs [Er03, Or04, Re04] are based on the non-generic Mobile Location Protocol [LI04], which works on the assumption that the locating system is a device aware cellular network (providing, for example, information about cell shape and area) and that location information is only available to clients by being pulled from a server-based API. In contrast, in a location aware system where a client device incorporates positioning technology (e.g. GPS), the location API can be installed on the mobile device, which can then manage its own location related data. An API implemented on the client can take responsibility for its own geocoding (deriving a location from a landmark) and reverse geocoding (deriving a landmark from a position) by maintaining a local database of point of interest (POI) data. This information may then be used for local processing (such as calculating the azimuth or distance from one position to another) or for managing ad-hoc networks (e.g. Bluetooth networks) that do not require the support of a control network.

Generic location APIs, which can utilise both device aware and location aware systems, encourage new and longer-lived location aware applications and, since they are able to transparently combine location information from multiple sources, promote the adoption and integration of new location sensing technologies [CS03]. Supporting multiple positioning technologies enables applications to make selections between alternative available systems, depending on quality, suitability or priority [ME02, RL03]. The move towards generic location APIs that can be hosted on the mobile client raises a number of important questions. First, there are considerations relating to the resources required to support such APIs. Second, there is a question as to how useful a partial offloading of a location object model to the client may be. Location based services can be person oriented (e.g. 'finder' type applications such as [Ro04]) or device oriented (e.g. 'tracker' applications, as described by [Ta03]) and require specialised object models to represent these domains. Location applications also require the support of special purpose databases, which may include combinations of digitised maps, yellow pages, user profiles and dynamic data such as current weather conditions, traffic information and moving object trajectories [Tr04]. Again, complex object models are required to represent this information in an object-oriented application. Since it is unlikely that many of these models can realistically be fully hosted on the client, we must ask which subsets, if any, can be usefully based or replicated on a mobile device. In the following discussion we explore some aspects of the object model that is used by the generic client-hosted Location API for J2ME [Lo04], identify some resource constraint and object modelling issues, and suggest some ideas for the further development of this API.

2. The Location API for J2ME

The Java Location API for J2ME became available at the specification and reference implementation level in late 2003. Nokia published a reference implementation that can be used for simulation purposes in 2004 [No04] and device implementations have followed, including the Motorola iDEN i860 platform [Mo05]. The API defines a generic interface for location data that is intended to be deployed on small mobile devices and work with most positioning methods. The object model covers two aspects. First, there are a number of classes that support the gathering of location data (LocationProvider, Location, Coordinates, Orientation, etc.). This aspect of a client side API can be seen as particularly useful in the sense that as long as the device is location aware, hosting this part of the object model on the client avoids unnecessary calls to a server for location data. With a Java enabled client, these objects can be processed locally, for example to calculate distances and azimuths between coordinates or to monitor quality of service parameters. The second aspect of the object model is a set of classes and interfaces that support local storage of POI data, in the form of a database object known as the LandmarkStore, and Landmark objects that may be stored within it. The main role of this database is to enable the geocoding and reverse geocoding of landmark POIs.

2.1 Resource issues

Given that the Location API for J2ME has an object model that goes beyond simple positioning data to managing a landmark database, there are implications for the potential resource demands of applications using this API. The LandmarkStore has no maximum size (or indeed maximum number of separate stores) defined by the specification but will be limited in practice by physical memory size and probably by vendor implementations (for example the Motorola implementation limits the store size to 256 landmarks and allows only one store). Given these constraints, the LandmarkStore would have to be used as a dynamic cache in most location based applications, storing only the currently relevant landmarks. To avoid application exceptions due to insufficient available storage, we would have to adopt a suitable strategy for downloading new Landmark objects and discarding old objects according to the current location and perhaps some kind of least recently used (LRU) algorithm. However there is no specification within the location API that suggests this kind of necessary caching behaviour, without which attempting to add a Landmark to a full store will throw an I/O exception.

In addition to the storage of Landmarks that are specified within the API, there will be many other types of semi-persistent data that will be required in a client-hosted location based application. For example, since most location based service applications rely to some extent on utilising map data, a typical resource demand in a location based system is the requirement to support image files. Since most applications will require access to map data much larger than could reasonably be stored on a small mobile device, maps have to be divided into cells that can be efficiently tiled on the client display [Je04]. Once map components have been downloaded from the server it is helpful to retain them in some form of image cache, preferably related to the current content of the landmark store. Currently the location API provides no support for this kind of image cache, but if caching support were added to the LandmarkStore then a similar form of memory management could be applied to other application components and these could be linked together via listeners. It may be appropriate to consider a generic caching interface that could be utilised with different types of object in location based applications. Since a mobile device data store might be regarded as similar to an in-memory cache that replicates part of a larger database of objects, transparent persistence mechanisms such as those used with Java Data Objects [JDO03] might provide us with some ideas for managing local object stores. Methods to selectively evict or refresh objects in the cache using a range of parameters would be very useful in the LandmarkStore or other associated stores. Figure 1 shows how a proposed CachingStore interface might be integrated into the existing API and allow extensions for new stores

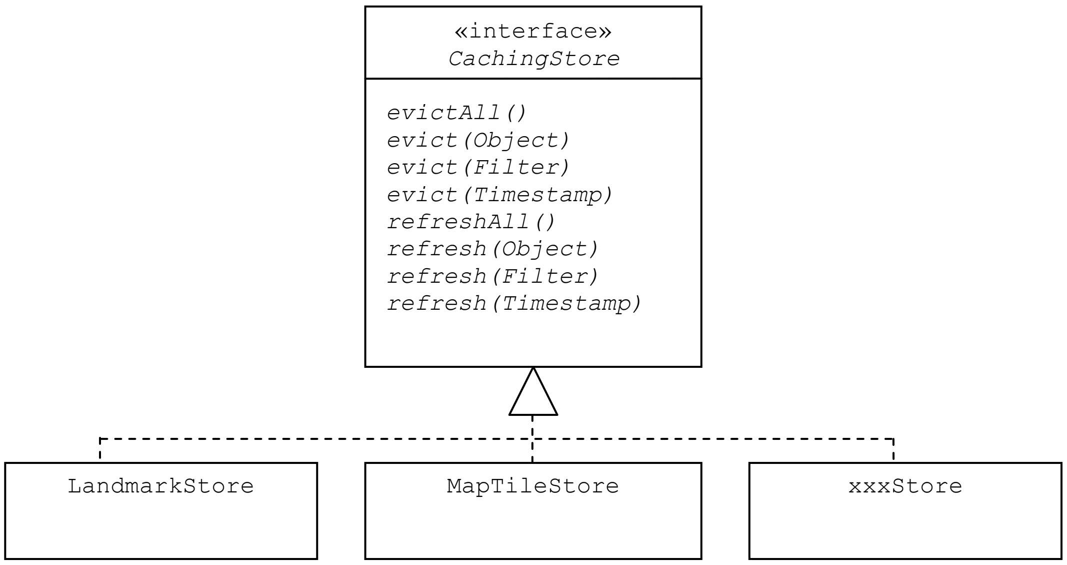

Figure 1: Integrating a CachingStore interface into the API

2.2 Extending the object model

The current J2ME location API provides us with an object model that only pushes the boundaries a little in terms of supporting rich location based applications on the client. An application that directly uses the object model provided by the location API will be primarily a reverse geocoding service, i.e. given a three dimensional location specified by latitude, longitude and altitude it will return information about that location, such as its street address. Although there are methods to calculate distance and azimuths between coordinates, these coordinates are tightly coupled in the API with Locations (stored as Landmarks.) Since the API essentially only enables us to interact with Location objects and Landmarks, useful applications will need to add additional layers of functionality to the client. A location based service framework that could support, for example, route finding or friend finding applications, would require the representation of mobile objects, their behaviours and their contexts. It would need to incorporate methods for tracking, finding and querying such objects and encapsulate an explicit query language. It is clear that the current location API does not provide an adequate framework for any but the simplest of location based services, and that even then the developer has to code defensively to cater for the limited sophistication of the LandmarkStore. Since the existing object model would be only a small part of a realistic application, it seems appropriate to consider whether it is worth developing the API further, or alternatively to acknowledge that extending the client side object model to anything beyond raw positioning data is unproductive and that more complex elements of the object model should remain on the server.

3. Conclusions

In this paper we have explored some issues within the Location API for J2ME specification relating both to resource management and object modelling for location based services. In terms of resource management we are concerned that there is no intrinsic link between the API implementation and any automatic cache management or middleware layer that could transparently balance the resource demands on the client by intelligently freeing memory and/or interacting with the server. This indicates a need to design location-based applications to use local database resources carefully, since unrestrained use of the landmark store, for example, will soon lead to the inability to store new landmarks. Because the store does not automatically evict stale data, it is not, unmodified, a suitable cache for dynamically changing data. This caching behaviour could usefully be added either to the landmark store implementation or, more realistically if we are using a third party implementation of the location API, to a wrapper layer. Useful further work could be done in building data management relationships between the location API and middleware, for example by using a distributed Java Data

Objects architecture similar to that described by [PKC05], whereby the persistence layer could manage the transfer of data between the local cache database and the server.

In addition to these concerns about implementations of the current object model specification, we have also addressed some questions about the overall semantics of a client side location API that attempts to expand the object model beyond the core components of location and orientation data that may be locally determined. The inclusion of a LandmarkStore is just one aspect of object models for location based services, but there are many more common objects that could be integrated into location based service frameworks. It is not suggested that all of these should be integrated into a core API, but it may be appropriate to consider a number of extension packages that could provide a standard set of Java APIs for the more common types of location based services. Alternatively orthogonal frameworks could be developed on top of the current Java API both to mitigate its weaknesses and extend its functionality.

In summary we believe that the current location API for J2ME does not yet provide an adequate framework for the management of data intensive location based services. The current specification may evolve to include such features as cache management within the landmark store, but in the meantime it may be necessary to add a utility wrapper layer above the current API to provide these and other services to ensure robust and efficient applications. Further, we suggest that the viability of moving location based service functions onto the client is dependent on developing a richer reusable framework that can support features such as mobile object representation without placing excessive resource demands on the mobile client.

References

[Bu04] Butz, A. 2004. Between location awareness and aware locations: where to put the intelligence, *Applied Artificial Intelligence, Special Issue on AI in Mobile Systems*. 18(6).

[CS03] CSTB. 2003. *IT Roadmap to a Geospatial Future*. Washington D.C.: National Academies Press.

[Er03] Ericsson, *Mobile Positioning Protocol Specification Version 5.0*, Ericsson, 2003, http://www.ericsson.com/mobilityworld/developerszonedown/downloads/docs/mobile_positioning/mpp50_spec.pdf, last accessed 20th December, 2004

[JDO03] Java Data Objects Expert Group. 2003. *Java^{TM} Data Objects JSR12 Version1.0.1*, Sun Microsystems, http://jcp.org/aboutJava/communityprocess/final/jsr012/index2.html

[Je04] Jensen, C. 2004. Database Aspects of Location-Based Services, in *Location-Based Services*, J. Schiller and A. Voisard, Editors. Morgan Kaufmann: San Francisco. p. 115-148.

[LI04] LIF. 2002. Mobile Location Protocol Specification Version 3.0.0, Location Interoperability Forum, http://www.openmobilealliance.org/tech/affiliates/lif/lifindex.html, last accessed December 17th, 2004

[Lo04] Loytana, K. 2003. JSR-000179 *Location API for J2ME^{TM} (Final Release*), Java Community Process, http://jcp.org/aboutJava/communityprocess/final/jsr179/index.html, last accessed December 17th, 2004

[ME02] Myllymaki, J. and S. Edlund. 2002. Location aggregation from multiple sources. *Third International Conference on Mobile Data Management*. IEEE. p.131-138

[Mo05] Motorola. 2004. *iDEN i860 SDK - Ver 1.1 for J2ME(™) Technology*. Motorola, http://idenphones.motorola.com/idenDeveloper/developer/developer_home.jsp, last accessed 9th February, 2005

[No04] Nokia. 2004. *RI Binary For JSR-179 Location API For J2ME^{TM}*. Nokia.

[Or04] Orange, *Orange UK Location API*. 2004.

[PKC05] Parsons, D., I. Kirsh and M. Cranshaw. 2005. Java Technology for the Wireless Industry – Toys or Tools? *Java Developers Journal*, 10(1).

[Re04] Redknee, *Synaxis-2200™: ELS Release 2.0 Client Interface Specification Document*, Redknee Inc., 2002, http://www.sourceo2.com/O2_Developers/Tools/Location_API.htm, last accessed 17th December, 2004

[RL03] Ranchordas, J. and A. Lenaghan. 2003. A Flexible Framework for using Positioning Technologies in Location-Based Services. *EUROCON 2003 - Computer as a Tool*. Ljubljana, Slovenia: IEEE. p.95-98

[Ro04] Roth, J. 2004. Data Collection, in *Location-Based Services*, J. Schiller and A. Voisard, Editors. 2004, Morgan Kaufmann: San Francisco. p. 175-205.

[Ta03] Taylor, S. 2003. Developing automatic vehicle location systems. *Computing & Control Engineering Journal*, 14(1): p.20-25.

[Tr04] Trajcevski, G., O. Wolfson, K. Hinrichs, and S. Chamberlain. 2004. Managing Uncertainty in Moving Objects Databases. *ACM Transactions on Database Systems*. 29(3).