On Building a Reflective Middleware Service for Location-Awareness*

Uzair Ahmad[1], Uzma Nasir[2], Mahrin Iqbal[2], Young Koo Lee[1], Sung Young Lee[1,] Inook Hwang[1]
Computer Engineering Dept. Kyung Hee University, 449-701 Suwon, Republic of Korea[1] NIIT, Pakistan[2]
{uzair, yklee, sylee}@oslab.khu.ac.kr,{54mahrin, 54uzma}@niit.edu.pk

Abstract: Location based services are becoming essential feature of context-awareness in ubiquitous computing. Reflective distribute component programming model is proposed to systematically provide location to Location Based Services (LBS). We integrate distributed component technology and reflection to develop localization capability as middleware service. Concept of Meta Object Protocols (Reflection) is used in different way than traditional reflective mechanisms. It deals with the state of the component that is not inside the component rather resides outside of it, namely extrinsic. This component model provides the basis for middleware architecture to support location providing service at design, implementation and run time. We describe the methodology we used to build location-awareness as middleware service based upon our reflective component model.

1: Introduction.

Location based services need to tailor themselves based on the context triggers such as system's location [2]. Problem of location-awareness, particularly in building, has produced very extensive research under the names of location determination, localization, positioning so and so forth. It is how ever observed [3] that current middleware support lack the kind of support needed for develo[1]pment of wide range of location driven autonomous mobile applications belonging to different domains. We believe that middleware capable of supporting the development of location driven applications will play a key role in the success of location based applications development, the same way traditional middleware did for wired distributed systems
In this paper we present middleware service architecture as proof of our concept of "*Extrinsic Reflection*". This service is complementary part of our previous work on Location based Autonomic Adaptation of mobile devices. [19]

2. Extrinsic Reflection.

Reflection is a well established discipline to achieve intelligent flexibility in Object oriented databases [23] and Object-oriented programming [21, 22]. The protocols designed to achieve reflection are often referred to as the *MetaObject Protocols or MOPs*. So far reflection namely MOPs have been limited to inspecting and adapting the properties intrinsic to an object [5]. However we have extended reflection to capturion, an extrinsic property of an object. Using our MOPs a mobile object can inspect its location itself within the vicinity. We call this reflection "*Extrinsic Reflection*" and the protocols to achieve this reflection "*Extrinsic Reflection MOPs (E-R MOPs)*". The MOPs for Extrinsic Reflection encourages a clean separation between the basic functionality of location determination of the application object from its representations and controls that lie in the Meta level objects. We give the responsibility of location estimation of a base level object, residing on a mobile device, to its Meta level objects. The goal of extrinsic reflection is to allow a base level location-driven object to reflect on its own location and eventually use this location information to alter its behaviors. For dynamic adaptation of the behaviors we developed other part of the middleware described in [19].

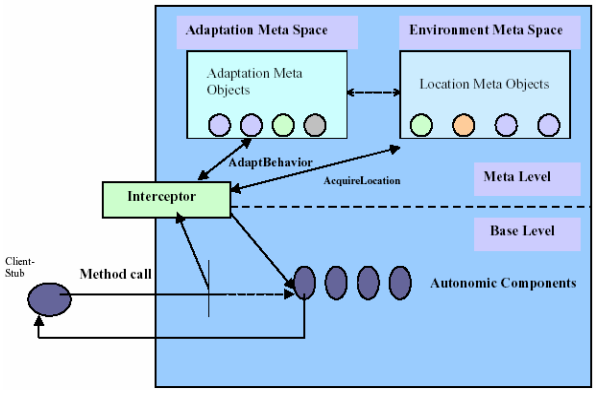

Fig 1: Extrinsic Reflection Approach for location Aware component development framework

3. Location via Reflection.

Location-driven applications require a middleware that provides development as well as run time support across different environments. We provide this through our reflective framework in a principled manner by providing a Meta Object Protocol (MOP) for "*location monitoring, acquision and updation*" on one hand and behavior adaptation at the other end [19] It facilitates the development of location aware components by providing

* The research was supported by the Driving Force Project for the Next Generation of Gyeonggi Provincial Government in Republic of Korea.

standard vocabulary in the form of APIs that encapsulate in them the description of location awareness.
This shields the programmer from the way underlying location determination technology is working. It enables a server component to control its extrinsic property as calling a reflection routine on itself in a seamless manner.

4.1 Location-Aware Service Development Phases

We provide a standard programming model for development of location aware server side components. Under this model an application developer has to follow the development steps in figure 2, for location aware components development .For each of the phase, we provide standard APIs that the programmer can use and develop the specific applications

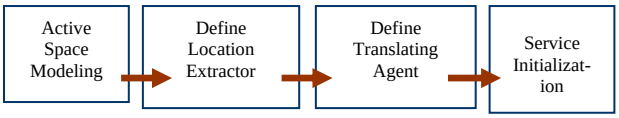

Fig 2: Location Aware Application development Phases

Active Space Modeling: Every application that needs location based services, require a location model that is an electronic representation of the actual world within which the device has to run and its location tracked. We call it an "active space". We call the parts of the active space that are important for an application as "Points of Interest"*(PoI)*. We provide standard vocabulary to model PoIs making up the active space. This vocabulary is called active space modeling language (ASML). Using ASML POI could be modeled in terms of absolute coordinates relative to some reference point. Such a model is called the physical model [16]. An active space can also be modeled in topological manner, which caters the containment, inclusion and proximity aspects [17] of each *PoI*. These *PoIs* are also expressed in human understandable form that we call the semantic form. In some cases the active space needs to be modeled in terms of more complex geometric shapes like polygons, rectangles ect. [18] ASML provides vocabulary for all these types of models. Since location model is the representation of the actual world, any change in the reality needs to be reflected in the model. Thus extension or alterations in the model is also needed. We also enable the application programmer to define as well as alter or extend the active space programmatically using extension APIs.

Location Extractor: For every instance of a mobile application one Location Extractor object is provided that constantly interacts with the underlying sensor driver software and hardware through the sensing APIs. Using these APIs a Location Extractor can configure it self with the underlying sensor and extract raw position data. The APIs encapsulate all the complexities of interaction with the sensors.

Translating Agent: The translating agent is responsible for translation of the data provided by the Location Extractor object into a higher-level form as mentioned in the Active Space Model. Translation APIs are provided by the middleware to define translating agents.
By keeping the implementation of all of these components open, a generic programming environment is provided that is usable in a variety of places. This leads to a cross environment solution

Service Initialization: It includes initializing the Meta space, which includes pooling the appropriated Meta objects needed for location service

4.2: Core Components of Run Time Location Servicing

The core components that make run time location reflection possible for server components are described below
Autonomic Component: These are the components that are client's representative at the server. They need to be location aware but are completely isolated from location related complexities. All autonomic components need to extend a standard autonomic Component class that keeps the application developer away from handling many complexities. It consist of special method(s) called extrinsic method which is used to reflect on extrinsic property. The property it self lies at the Meta level where there is a Meta class with exactly the same extrinsic method

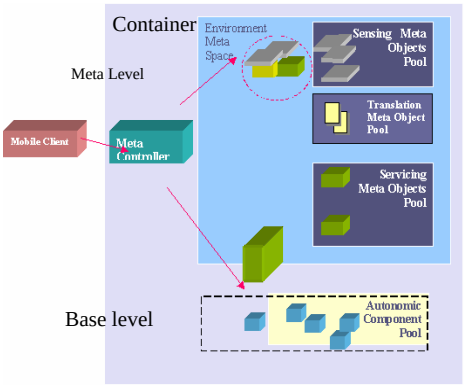

Fig 3: Core Components of Run time location Servicing

Meta Controller:
It acts as a mediator between the Meta space and the autonomic component at the base level that wants to reflect on its external property. It manages the run time aspects of seamless and active location servicing

including call interception, monitoring and intimation. Besides it also provides network transparency.

Anatomy of Environment Meta-Space.

The environment Meta space holds all the Meta objects that are required to seamlessly serve an autonomic component with its extrinsic property like location in our case it includes Meta objects for sensing, Meta objects for translation and run time servicing objects. Each Meta object knows the Meta space it is a part of and thus interaction among various Meta objects is done via the Meta space. Based upon the different functionalities that these Meta objects perform, we give a layered view of our environment Meta space. It consist of the sensing layer with meta objects responsible for sensing raw position, the translation layer with meta objects that do raw to appropriate level translation and servicing layer that consist of extrinsic service objects like location service object in our case .The location service object serves the location seamlessly to the client

Meta Objects at Sensing Layer: At the sensing layer there are dedicated sensing Meta objects, one for each registered client. These objects are responsible for constantly interacting with the underlying sensor and extracting the client's movement data in the active space for as long as the client remains registered .In our approach, the sensing layer is intelligent enough to keep a track of the state of the clients. If the client is inactive and thus the movements are not detected, then for such clients, the sensing Meta objects do not invoke the translation Meta object at the translation layer. This is because, for an inactive client, translation and then the behavior switching is not needed at the same place again and again. By doing so translation only occurs when there is a noticeable change in the position of a mobile object.

Meta Objects at Translation Layer: The Meta objects at this layer are called translating agents. These objects are responsible for taking the raw data extracted from sensing Meta object and converting into a form understandable by the mobile applications. For all the instances of once particular application, same translation objects are needed. In such case, there is no need for dedicated Meta objects at this layer. The sensing Meta objects invoke the translation Meta objects automatically when change in position is detected. The numbers of objects at the translating layer expand or shrink depending upon the frequency of requests from the sensing layer. Thus the layer adapts it self according to the need .By doing so memory can be saved in case frequency of request is low and time can be saved when frequency is high by expanding it self to accommodate increasing translation requests

Meta Objects at Servicing Layer: At this layer lie the Meta objects that are responsible for extrinsic servicing. It includes Meta location service objects for location servicing, one for each autonomic component. The extrinsic property, like location, of a server component is obtained from Meta objects at this layer. These Meta objects consist of both the extrinsic property and the extrinsic method corresponding to the autonomic component

Pooling in Environment Meta-Space: To make the process of location servicing real time and robust we have adopted the approach of pooling of the Meta objects mentioned above inside the Meta space. Depending upon the application need the size of the objects pool is mentioned. There are separate pools of all the Meta objects containing their ready-made instances to serve the future application instances. When an autonomic component wants to get location aware these Meta objects are fetched from the respective pools

Container: Lastly we would mention the Middleware container that is a place holder for all the components that make up the location Service It encapsulates both the base objects as well as the Meta spaces .The Meta Controller instance is also created by the container for each client. It acts as an execution environment for run time location servicing .It is responsible for initialization of both Meta and Base level components.

5. Extrinsic Reflection MOP (ER-MOP)

To enable server side components to reflect on their external properties that lie at the Meta level we have coined a Meta object protocol called "Extrinsic Reflection MOP abbreviated as E-R MOP.

For each autonomic application instance that wishes to reflect on its location, a Meta controller is created. The Meta controller interacts with the Meta space; loads the appropriate Meta objects; activate the Meta objects, which start working for the specific client. Finally it makes autonomic component aware of its Meta level service object.

After configuring the Meta level for the base autonomic component, the Meta controller constantly calls extrinsic method of the base autonomic component. Since, the Meta Location service has been configured for the autonomic component, the call goes go the extrinsic method at the Meta service which is being updated with the extrinsic property value (Location here) from the Meta objects lying at the sensing and translation layer. Meta Service is always provided with the updated extrinsic property since comparison is done as we explained before by the sensing Meta objects .So the updated extrinsic property is returned to the Meta controller. The Meta controller then notify base autonomic component of its extrinsic property. The

Meta controller keeps calling extrinsic method of the base component and getting updates. In this way a server side autonomic component is able to find a property that is out side it self.

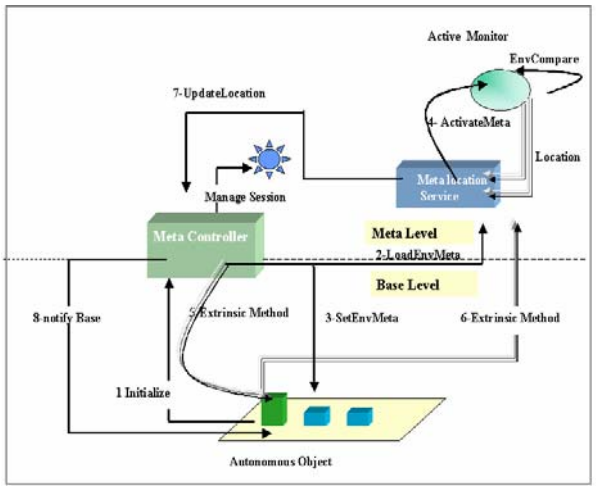

Fig 4: E-R MOP for location Awareness of a server component

6: Discussion and conclusion

We observe that using Extrinsic MOP for location services in reflective middle-ware, a server component can reflect on its location property. The process works actively and based upon these changing location behaviors adaptation can be done very efficiently.

An alternate approach to this could have been where the client would get its raw location it self through some sensing mechanism like blue tooth for example and send this raw information as a parameter inside the method call to the middleware. Comparison of this approach to the extrinsic MOPs reveals that in case the location sensing is embedded within the client, the client code will become more complex .In our case client gets its location, and is still independent of how sensing is done. Secondly in case the location is sent as a parameter in the method call, the interpretation of raw location has to be done at the time of the call and once that is done then the appropriate behavior against that location needs to be instantiated. This can take a lot of time Extrinsic MOP on the other hand provides active location information irrespective of whether method is called or not. Thus using Extrinsic MOPs, the location is achieved in an efficient manner and thus adaptation process also becomes fast.

By dividing the system into a simple base level, a location encapsulating meta level and Meta Controller that mediates between the two levels, any location based application developer can get location services in a principled manner without worrying about defining the detailed underlying mechanism. By simply calling an extrinsic method location can be obtained. This makes the applications development quicker and easier. Additionally pooling, session management, autonomous sensing and adaptive translation layer gives added performance and efficient resource management.

In this paper we discussed our idea of *"Extrinsic-Reflection"* for designing a reflective middleware service for location-Awareness. *Extrinsic*-Reflection enables an object to inspect its extrinsic properties, location in our case. To provide efficient run time support, the container uses object pooling both at base level and meta-level. The future work also involves provision of group location services where mobile clients could dynamically form groups and be informed of group member's locations.

8. References

1. L. Capra et al "*Exploiting Reflection in Mobile Computing Middleware*". In ACM SIGMOBILE Mobile Computing and Communications Review.
2. G. Kiczales, J. des Rivires, and D.G. Bobrow. The Art of the Metaobject Protocol. MIT Press, 1991
3. A.K. Dey, M. Futakawa, D. Salber, and G.D. Abowd. The Conference Assistant: Combining Context-Awareness with Wearable Computing. In *Proc. of the 3*rd *International Symposium on Wearable Computers (ISWC '99)*, pages 21–28, San Franfisco, California, October 1999. IEEE Computer Society Press.
4. Licia Capraa Gordon S. Blair, Cecilia Mascolo,Wolfgang Emmerich. Paul Grace;Exploiting Reflection in mobile computing middleware Mobile Computing and Communications Review, Volume 1, Number 2
5. Mahrin et al, Reflective Middleware for Location-Aware Application Adaptation, UWSI 2005
6. P Maes Concepts and Experiments in Computational Reflection OOPSLA 87, Sigplan Notices Vol 22 No pp 147-155
7. S Matsuoka, T Watanabe,Y. Ichisugi, and A Yonezawa, Object Oriented Concurrent Reflective Architectures 1992
8. B.Smith. Reflection and Semantics in a Procedural Language MIT TR 272 1982