Open Standard Query Interface for Geospatial Databases in OSA / Parlay

Lebogang Kenneth Masenya

A project report submitted to the Faculty of Engineering and the Built Environment, University of the Witwatersrand, Johannesburg, South Africa, in partial fulfilment of the requirements for the degree of Master of Science in Engineering.

Johannesburg, October 2001

Declaration

I declare that this project report is my own, unaided work, except where otherwise acknowledged. It is being submitted for the degree of Master of Science in Engineering in the University of the Witwatersrand, Johannesburg, South Africa. It has not been submitted before for any degree or examination in any other university.

Signed this ___ day of __________ 20___

____

Lebogang Kenneth Masenya.

Abstract

Telecommunication networks have evolved from voice only single service networks to multimedia networks providing bearer services such as voice, data and video transportation. Moreover, these networks, collectively called Next Generation Networks (NGNs), enable rapid creation, deployment and management of advanced services in an efficient manner. However, the initial business model of telcos was to internally develop and provide these advanced services to customers. In this monopolized environment, service development is driven by technological availability rather than customer demands. Furthermore, vendor specific network elements prohibit the development of re-useable service components, which in turn increases the time-to-market of services. Deregulation and advances in Distributed Computing Systems (DCSs) are driving towards open networks and rapid service delivery. Third party Application Service Providers (ASPs) are envisioned to develop and supply the services, with the telco providing bearer services. The use of softswitch architectures such as Open Service Access (OSA) / Parlay (OSA / Parlay) in an open NGN environment abstract services from core network elements through its Application Programming Interface (API). Services are thus decoupled from vendor and protocol specific network equipment and can be provided across a plethora of network architectures. One major advantage of NGN is the ability to provide bearer service in a mobile environment. Location Based Services (LBSs) are envisaged to be an important class of services provided in the NGN environment. For an LBS service to be complete, a geospatial database is necessary to provide location information. This report documents the design and implementation of a Geospatial Data Access Service Capability Feature (GDASCF) as an extension to the OSA / Parlay gateway. The GDASCF encapsulates necessary APIs that offer uniform access to query geospatial databases. One key component of the design is the realization of the Adapter layer which adapts function calls to an appropriate Database Management System (DBMS). The introduction of the GDASCF and Adapter layer provides a solution which results in flexible and rapid service creation.

Acknowledgements

The following research was conducted under the endorsement of the Center for Telecommunications Access and Services (CeTAS) and the Information Engineering Research Programme (IERP) at the University of the Witwatersrand, Johannesburg, South Africa. CeTAS is funded by Telkom SA Limited, Siemens Telecommunications and the Department of Trade and Industry's THRIP programme. I would also like to give thanks to the National Research Foundation (NRF). Their financial support is much appreciated.

I would like to extend thanks to my supervisors, Prof. Hu Hanrahan, Prof. Barry Dwolatzky and Prof. Rex van Olst for their kind supervision and valuable input towards this research. In addition, I would like to thank Nkosinathi Khoza, Sefako Tholo, Ted Mwakabaga and my other colleagues and friends for their valuable support during the research. Finally, I would like to thank my parents (Dikgale and Suzan Masenya), brothers (Brian and Keletso Masenya), sisters (Lerato and Kwena Masenya) and my nephew (Paballo Masenya) for their love, support and patience throughout my tertiary studies. Without them, I would have lost site of my goals and aspirations.

Contents

Declaration **i**

Abstract **ii**

Acknowledgements **iii**

Contents **iv**

List of Figures **viii**

1 Introduction **1**

1.1 Background 1

1.2 Emerging Business Case 2

1.3 Characteristics of Applications and their Execution Environment . . 4

1.3.1 The “My Nearest” type of services 4

1.4 Requirements for Content Providers 6

1.4.1 Handling Different Content 7

1.4.2 Handling Different Devices 8

1.4.3 Deployment of Mobile Applications 9

1.4.4 Working with Different Providers 9

1.5 Location Based Services with OSA / Parlay 9

1.6 Overview of Research Report 10

2 OSA / Parlay Overview 12

2.1 Background on OSA / Parlay 12

2.2 OSA / Parlay Architecture 13

2.2.1 Logical Architecture 13

2.2.2 Physical Architecture 14

2.2.3 Role of the Parlay Framework 15

2.2.4 Security Issues 17

2.2.5 Scalability Issues 18

2.3 Support for Location Based Services 18

3 Design of the OSA / Parlay Geospatial Data Access SCF 20

3.1 Proposed Architecture 20

3.2 Design Requirements and Constraints 22

3.3 Design Methodology 23

3.3.1 Use Case Modelling 23

3.3.2 Use Cases 24

3.3.3 Sequence Diagrams 30

4 Implementation of The Geospatial Data Access SCF 38

4.1 Implementation Approach 38

4.2 Deployment of Service and Gateway Components 38

4.3 Example services with and without the GDASCF 39

4.3.1 First Approach : Implementation without using the GDASCF Interface 40

4.3.2 Second Approach : Implementation using the GDASCF Interface 41

4.3.3 Comparison of both approaches 44

4.4 Chapter Summary 46

5 Critical Design Review 47

5.1 Introduction 47

5.2 Coupling 47

5.3 Cohesion 48

5.4 Completeness and Primitiveness 50

6 Conclusion 51

6.1 Discussion 51

6.2 Conclusions 52

6.3 Recommendations for Further Work 53

References 55

A Robustness Diagrams 59

A.1 Establish User Position 59

A.2 Proximity To A Region 62

B Interface Class Definitions 65

B.1 Interface Class for IpProximity 65

B.1.1 Method periodicProximityReportRequest() 66

B.1.2 Method triggeredProximityReportRequest() 67

B.1.3 Method periodicProximityReportingStop() 68

B.1.4 Method triggeredProximityReportingStop() 69

B.2 Interface Class for IpAppProximity 69

B.2.1 Method proximityReportRes() 70

B.2.2 Method periodicProximityReportRes() 71

B.2.3 Method triggeredProximityReportRes() 71

B.2.4 Method proximityReportErr() 72

B.2.5 Method periodicProximityReportErr() 72

B.2.6 Method TriggeredProximityReportErr() 73

C IDL specifications for the Geospatial Data Access Service Capability Feature 74

C.1 CommonDataDefinitions.IDL 74

C.2 IpProximity.IDL 76

C.3 IpAppProximity.IDL 77

List of Figures

1.1 Single Service Networks 2

1.2 Converged Networks 3

2.1 Parlay Logical Architecture 13

2.2 Service Enabler Sub-Layer 15

2.3 Service Registration And Discovery 16

2.4 Trust And Security Management Protocol 17

3.1 Proposed Architecture 21

3.2 Use Case Diagram 23

3.3 Establish User Position - Interactive Request 31

3.4 Establish User Position - Periodic Request 32

3.5 Establish User Position - Triggered Request 33

3.6 Proximity to a Region - Interactive Request 35

3.7 Proximity to a Region - Periodic Request 36

3.8 Proximity to a Region - Triggered Request 37

4.1 Deployment of Components 39

4.2 Deployment of a GIS in a corporate 40

4.3 Sequence diagram to access corporate database using RPC 41

4.4 Sequence of events for “Find Nearest ATM” Service 43

5.1 Vertical and Horizontal Coupling 48

5.2 Class Diagram 49

A.1 Establish User Position - Interactive Request 59

A.2 Establish User Position - Periodic Request 60

A.3 Establish User Position - Triggered Request 61

A.4 Proximity - Interactive Request 62

A.5 Proximity - Periodic Request 63

A.6 Proximity - Triggered Request 64

B.1 Interface Class IpProximity 65

B.2 Interface Class IpAppProximity 70

chapterAcronyms

Chapter 1

Introduction

1.1 Background

It is usually the case that different telecommunications technologies progress independently from one another. This independence in progress is because technologies are designed to resolve problems that are unique to their problem space and time. It is for that reason that the current telecommunications networks are separate. Adapter patterns as described in [1] and [2] can be used to build gateways between different networks' protocols. For example, media gateways exist between Internet Protocol (IP) Networks and the Public Switched Telephone Networks (PSTNs). However, services are usually implemented on one particular network infrastructure and not readily transferable to another. Thus, in most cases, services are not re-usable and need to be re-designed to be deployed in another network infrastructure [3]. The use of adapters only complicates service development and results in non reusable software. Furthermore, since services developed in this manner are strongly coupled to their network infrastructure, developers need to have intimate knowledge of the inner workings of the network protocols. Clearly this will prohibit new role players and third party organizations from entering the telecommunications environment, further increasing the monopoly hold of existing operators [4][5][6]. This type of network environment can be classified as single service network as illustrated in Figure 1.1.

Telecommunications networks have evolved from voice only transportation networks to multimedia mobile networks capable of carrying voice, data and video

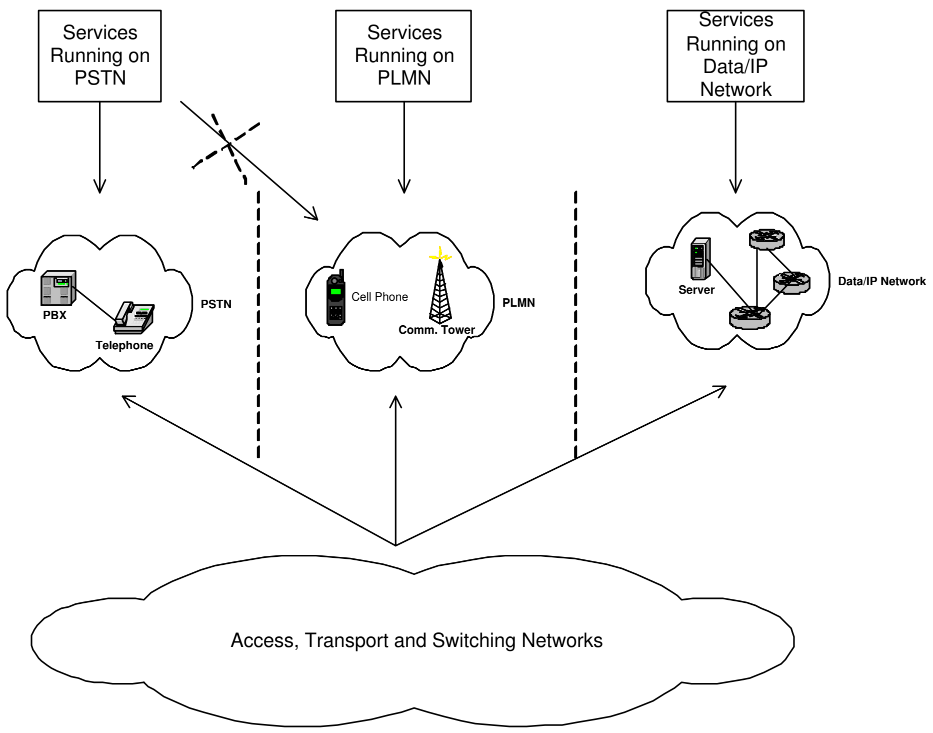


Figure 1.1: Single Service Networks

in a mobile environment such as Universal Mobile Telephony System (UMTS) [7]. Services in such an environment are deployed in an open and scalable manner using Application Programming Interfaces (API) such as OSA / Parlay [8] and Java API for Intergraded Networks (JAIN) [9] as shown in figure 1.2.

1.2 Emerging Business Case

Current telecommunications environments influence the way services are provided to customers. This factor suggests that there is a greater drive towards telecommunications monopolies [5]. This approach may appear to be lucrative for the telecommunications provider because it acts as both the network operator and service provider, but customers are placed at a disadvantage. This results from services being driven by available technologies rather than customer demands [3]. The main reason for this drive is that the telecommunications infrastructure is mostly based on vendor specific network equipment with proprietary interfaces which is usually limited in capability [3]. Design patterns for developing, deploying and introducing [10] services in a common manner across multiple network platforms do not exist.

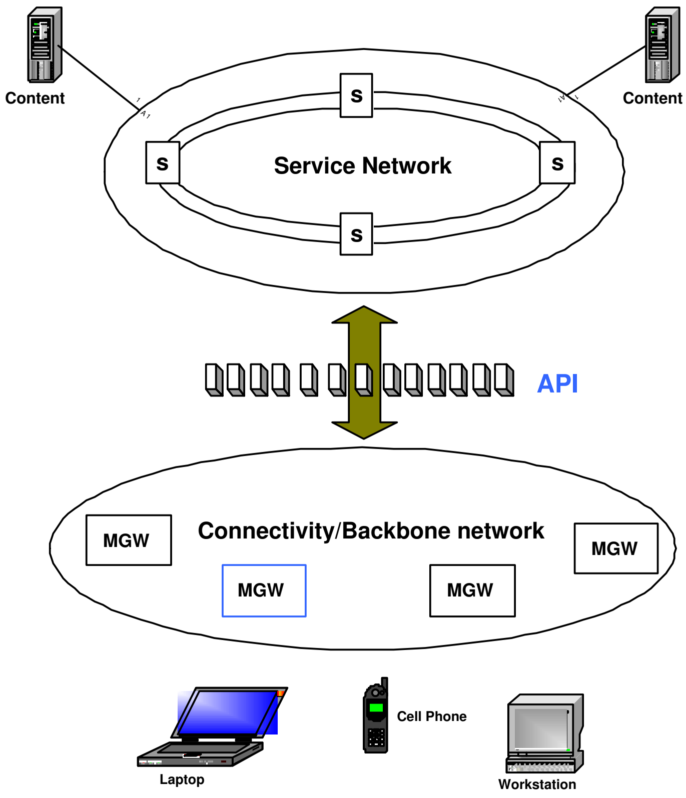


Figure 1.2: Converged Networks

Several approaches are employed to remedy the situation outlined above. Firstly, as telecommunications technology progresses, it is realized that software will play an important role. New software architectures are being developed which will affect the way services are deployed. These architectures provide an advantage in that an abstraction layer is introduced between the vendor specific network elements in the physical layer and the services in the application layer. The necessity of this approach is reflected in the fact that services are decoupled from the physical layer and thus can be deployed across a plethora of network infrastructures. Specifically, future telecommunications services are expected to be distributed in nature [11], running in a Distributed Processing Environment (DPE) [12].

Furthermore, the telecommunications environment is undergoing changes as deregulation takes place. New policies are put in place to ensure that existing monopolies disappear while new role players emerge [5]. In this regard a new type of market is created. This market, which is open [13] and global in nature, addresses the communication needs of a Global Information Society.

The entrance of new role players in the telecommunications environment will also affect the way services are deployed to customers. While customers initially were bound to utilizing services at the discretion of the provider, service development will now be driven by specific customer demands.

The problems associated with implementing a GIS system are its cost, literacy amongst users, lack of awareness of spatial data suppliers and data standards and lack of coordination between users. It has been estimated [14] that in implementing a GIS, data costs represent 80% of the total system cost. These factors will severely limit small budget 3^{rd} party vendors and utilities from implementing GIS systems or applications based on such systems such as Location Based Services. To reduce these costs, we propose an infrastructure where 3^{rd} party Application Service Providers (ASPs) can access geospatial databases maintained and developed by separate vendors. This approach removes the need for 3^{rd} parties to incur the costs of developing and maintaining such databases thus reducing the implementation costs of a GIS system. ASPs can therefore concentrate on developing innovative applications while they gain access to already existing and maintained quality spatial data at a lower cost.

1.3 Characteristics of Applications and their Execution Environment

The business case outlined in Chapter 1.2, provides an indication of the characteristics that services and the environment hosting the services must posses. To illustrate this factor, Consider the following generic example.

1.3.1 The “My Nearest” type of services

Users can access this type of service to locate their nearest restaurants, bank ATMs etc. The “My Nearest” services depend largely on the GIS data stored in different GIS databases. This information can include spatial information such as the geodetic coordinates, as well as non spatial information such as the street addresses of places. Users can use their mobile units to access the service. The information contained in the query could be the spatial coordinates of the mobile unit at service access time. Such coordinates can be in any form ranging from more accurate GPS coordinate to less accurate mobile cell location. Moreover a Universal Location Framework [15] can be used to aggregate location information from various sources. The service must then query the GIS database for the requested location nearest to the mobile units coordinates. The implementation of the "My Nearest" service is discussed in detail in Chapter 4.3.

To satisfy the business case as outlined above, the services and their environment must posses the following characteristics.

1. Applications must adopt a standard calling convention. A generic set of Application Programming Interfaces (APIs) must exist to ensure that GIS information residing in different databases, is queried in a standard, scalable and consistent manner. ASPs can then use the same set of APIs to implement different services.
2. Controlled access must be permitted for APSs accessing the GIS databases.
3. ASPs must be able to subscribe to different GIS database providers. Different GIS databases can be used during service run-time. This factor is necessary since ASPs can make a choice of using a different database based on cost, type and quality of information they require etc..
4. The environment hosting the service must allow for more standardized interfaces to be added without breaking client applications using old interfaces.
5. Applications must be decoupled from the GIS database. To ensure their transparency with the GIS database. This will ensure that the service is independent from the database structure and implementation.
6. Services must be deployed in a Rapid Application Development (RAD) approach for reduced time-to-market.

OSA / Parlay gateway provides an infrastructure that implements APIs in a generic and scalable manner. Applications can call these APIs to query for a multitude of network information. More specifically, in "My Nearest" type of services, applications will query the network for the location of the mobile terminating unit. The

OSA / Parlay gateway also provides a framework that controls access to the Service Capability Features (SCFs) and allows more interfaces to be introduced in a scalable manner.

Since different GIS database providers will implement their databases differently, transparency of accessing the GIS information is essential. This will ensure that application developers do not concern themselves with different structures and schemas from a multitude of different providers further enhancing scalability. The purpose of the GDASCF component in the Parlay gateway will ensure this type of transparency.

1.4 Requirements for Content Providers

Mobile devices are continually evolving at an explosive rate. This presents opportunities for development of new and innovative applications and services. To capitalize on these opportunities, the physical attributes of mobile devices must be considered. These attributes include form factor, browsers, input/output limitations as well as the type of transactions that are likely to be conducted on the device. These attributes expose limitations of the existing mobile devices, while at the same time enabling the development of applications that are not possible on normal desktop devices. This can be seen in leveraging Location Based Services (LBSs) for mobile applications. LBSs need to be location-aware as well as being able to aggregate technology and content from a multitude of sources. The content is usually stored as objects within a database. Reference [15] describes the Universal Location Framework technology that is used in aggregating multiple location technologies.

The evolution of mobile devices has been influenced by the evolution of the telecommunications networks over the years. The deployment of second-generation network technologies such as Global System for Mobile communication (GSM) and General Packet Radio Service (GPRS) resulted in the first generation of mobile devices that were only focused on phone calls. Access to data was possible only in an offline environment. Incremental improvement on mobile devices enabled access to mobile wireless data applications. The large usage of these applications was limited by the small display and difficult data input mechanisms. Portable Digital Assistants (PDAs) addressed these issues as they gained access to wireless networks.

They could host a multitude of wireless applications, but the lack of bandwidth and poor coverage resulted in these applications working in both the offline and online environment. As a result applications were not able to compute their current location.

The emergence of smart phones addresses most of the limitations that mobile devices have. These include local storage, advanced displays, better input mechanisms, faster and enhanced mobile data networks and low cost positioning devices. These devices are ideal for implementing mobile applications. Moreover, since the GDASCF will be designed in accordance with the OSA / Parlay specifications, it will inherently posses the benefits of security, scalability and flexibility.

1.4.1 Handling Different Content

Third Generation Networks are required to provide mobility with a wide range of services and data content, independent of the type of the user equipment. In the Universal Mobile Telephony System (UMTS) network, the user equipment is not restricted in functionality [7]. Applications that reside on the device must be able to use content that is contained on the Web, any database or file system. This fact implies that applications must be provided with the correct level of abstraction for this to be possible. With this approach, code reusability can be exploited and an advantage in development time and cost can be gained.

In an attempt to achieve this level of abstraction, applications running on mobile devices can be provided with runtime libraries to connect to databases with the use of technologies such as Open Database Connectivity (ODBC) and Java Database Connectivity (JDBC). However, more value can be added by providing context aware applications. Geospatial databases are necessary for such context aware applications. However, there are no existing mechanisms to support uniform and generic access to these types of databases.

1.4.2 Handling Different Devices

Different devices vary in the way they handle different content. The spectrum ranges from content that requires advanced display capabilities such as high resolution streaming video, to high quality audio capabilities such as voice recognition, while other require large, fast storage capabilities. However, by providing the right level of abstraction, the application logic can be unaware of these features yet transparently gain benefit them.

A data model should be separated from the presentation model. In a mobile environment, context aware information can be provided by geospatial databases, however the model of this data should be abstract enough to be handled by any device. It is up to the application to display the information in an application specific manner. This approach closely relates to the Model View Controller (MVC) [16] design pattern where the model and the view are independent of each other while the controller defines the way they interact. Existing data access technologies and standards depend on the application designer to implement the MVC pattern. Moreover, different geospatial content providers will structure their databases in different ways requiring the application designer to know the database schema beforehand. Clearly, changes in the schema will break the application from accessing the database. Standards such as ODBC and JDBC provide APIs to access the databases directly. However, since in a general case databases will be provided by different providers, with some providing more accurate or updated location information at different costs, the application designer will have to know all this information, which is dynamic in nature, at design time. An efficient solution is to provide a layer of abstraction on top of the existing database access API technologies. This solution will further enhance scalability since the database structure will be transparent from the applications. By decoupling application logic from database structure, vendor independence is also encouraged. ASPs can thus be free to change between different content providers who in turn can implement any database access technology while keeping it transparent from the applications.

1.4.3 Deployment of Mobile Applications

Since the NGN standard does not restrict the functionality of user equipment in any way, mobile devices in any form naturally benefit from location awareness and can be supported by any mobile technology. The location of user equipment can be based on automatic positioning such as Global Positioning System (GPS) or a hybrid of mobile technologies as described in the Universal Location Framework (ULF) [15]. In cases where an application cannot use automatic positioning systems, it can use manual positioning that has been defaulted to it to perform its task. For instance, in LBS services that need to perform location queries, a query can be based on a location other than the mobile's current position by using the defaulted position.

1.4.4 Working with Different Providers

Vendor independence is becoming important in today's Open Services Market (OSM) [3]. In general, a provider of a specific service will not contain all the necessary resources, technology and content to offer all services internally within an organization. Architectures to deploy a service or a range of services may involve several role players all sharing resources to provide a solution. A framework to integrate internally developed solutions with externally hosted content is thus necessary. This framework must enable Application Service Providers (ASP) to switch seamlessly between different service and content providers in a prioritized and automatic manner.

1.5 Location Based Services with OSA / Parlay

The sponsors of the OpenGIS Open Location Services (OLS) initiative state [17]: ”Spatial connectivity is a primary, universal construct for business planning and modelling, service development and deployment, network provisioning and operation and customer satisfaction. The location based services are of universal industry service significance and depend upon the availability of relevant spatial information infrastructure in forms useful for small devices.”

One class of applications that is supported by the geospatial databases is Location Based Services (LBSs). A Location Based Service can be defined as a service that has the ability to determine and transmit the location of a mobile device in a mobile network. Moreover, these applications make use of location sensing technologies such as (but not limited to) Global System for Mobile Communications (GSM) and General Packet Radio Service (GPRS). Different network protocols introduce difficulties in the way LBS applications are developed. An architecture that allows applications to be developed in a generic, reusable and maintainable way is thus necessary.

OSA / Parlay introduces a layer of abstraction between the application layer and the core network. This OSA / Parlay gateway is made available to the application layer by use of generic APIs. The OSA / Parlay API are extensive including, Mobility Management, Call Control, User Interaction, User location / User status, Terminal capabilities, Data session control, Generic Messaging, Connectivity Management, Account Management, Content based Charging. Mobile applications can use these APIs to obtain dynamic location information about other mobile devices. The OSA / Parlay gateway however does not have a component to provide a generic query API for geospatial databases.

The purpose of this research is thus to define the geospatial data access interface for the OSA / Parlay gateway. Since it is impossible to capture all the requirements for LBS services at design time, the geospatial interface is defined in a manner that makes it easily extensible. To achieve this objective, the ICONIX design process was adopted. As discussed in chapter 5, new interfaces can be defined and added with ease to the existing interfaces.

1.6 Overview of Research Report

This dissertation discusses the design of OSA / Parlay based architecture that address the limitation is discussed above. The report is structured as follows:

Chapter 2 introduces and discusses the OSA / Parlay architecture and highlights the important literature of this subject that contributed significantly to this research. The nature of Geospatial Information Systems (GISs) and Location Based Services

(LBSs) and their applications are outlined in detail within the necessary context that will enable the design of the OSA / Parlay Geospatial Data Access SCF (GDASCF).

Chapter 3, presents the proposed architecture for the design of the OSA / Parlay GDASCF . The software engineering methodology used is then described. A detailed layout of the design of the GDASCF is presented in detail using the ICONIX design process.

Chapter 4 presents the implementation of the GDASCF. Example applications of benchmark services are provided to better clarify its usage.

Chapter 5 discusses the critical design review from the software engineering perspective. Software engineering matrices are outlined. They are then described on how they apply to the design of the GDASCF.

Chapter 6, in this chapter conclusions and limitations of the design are discussed and recommendations for future research are proposed.

Chapter 2

OSA / Parlay Overview

2.1 Background on OSA / Parlay

The Parlay group was formed in 1998 as an industry forum with five initial companies to create the first version. This initiation was sparked by the explosive growth of the Internet and wireless technologies. As advantages of open APIs became apparent, more companies joined the Parlay group to produce the second version known as Parlay 2. At this time the first version of the Parlay API was submitted to the European Telecommunications Standards Institute (ETSI) and the International Telecommunication Union - Telecommunications standardization Sector (ITU-T) for international standardization. In partnership with the 3rd Generation Partnership Programme (3GPP), the Parlay initiated work on a similar API to provide access to applications built on top of the Universal Mobile Telecommunications System (UMTS) network. At this time three different standards were being developed to serve the same purpose. To avoid such a situation, the three groups were combined into one working group called the Open Service Access (OSA). With this arrangement more APIs were defined and fedback into Parlay. An agreement with ETSI and Parlay was reached to make joint work formally possible. Currently the definition of the APIs is done in the context of OSA / Parlay and the entire participants are collectively called the Joint Working Group (JWG). The JWG has also collaborated with the Java APIs for Intergraded Network (JAIN) community [18] to develop Java versions of the OSA / Parlay APIs. Furthermore, the JWG is divided into smaller groups responsible for developing and testing specific new APIs.

2.2 OSA / Parlay Architecture

2.2.1 Logical Architecture

The logical architecture of the OSA / Parlay consists of the Applications and Application Servers residing in the application layer, Service Capability Servers (SCS) and the OSA / Parlay Framework in the gateway, as well as the core network elements. The logical architecture is shown in figure 2.1.

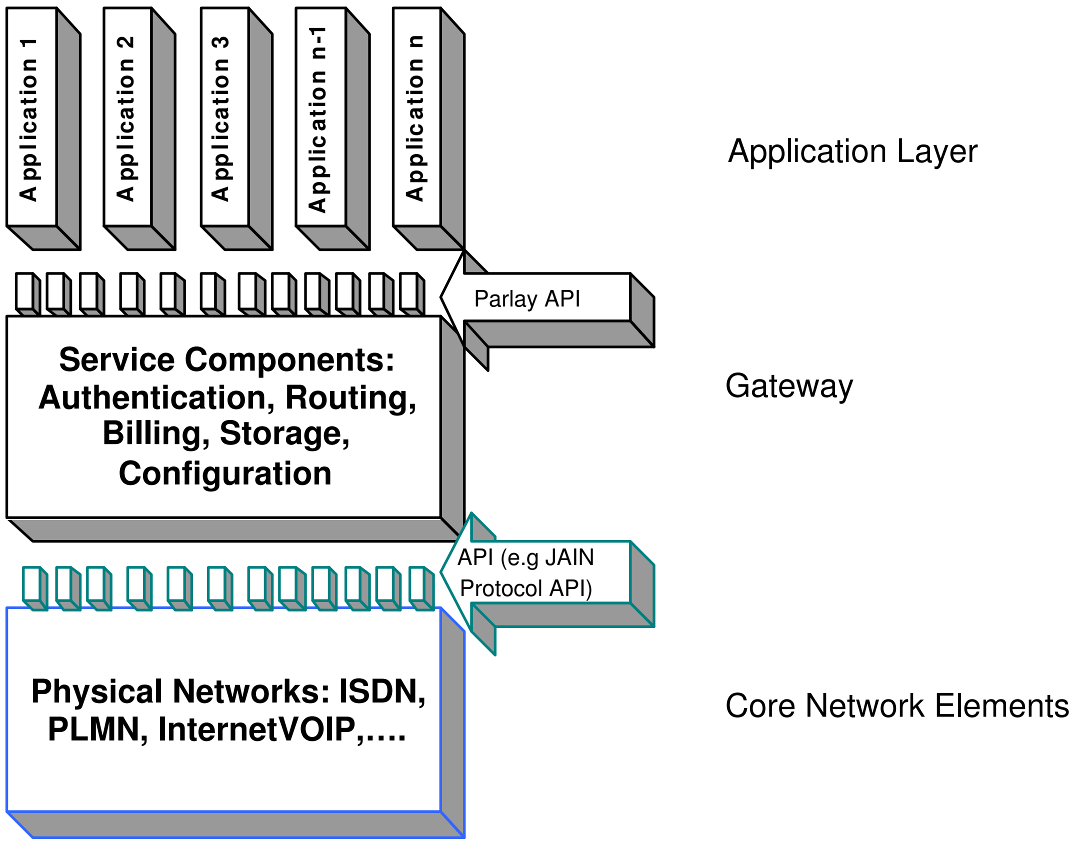

Application 1
Application 2
Application 3
Application n-1
Application n
Application Layer
Parlay API
Service Components: Authentication, Routing, Billing, Storage, Configuration
Gateway
API (e.g JAIN Protocol API)
Physical Networks: ISDN, PLMN, InternetVOIP,....
Core Network Elements

Figure 2.1: Parlay Logical Architecture

The applications reside in the application servers in the application layer and access the capabilities defined in OSA / Parlay through the OSA / Parlay APIs. The capabilities known as Service Capability Features (SCF) are encapsulated in SCSs which implement the server side API. It is the responsibility of the applications in the application layer to implement callback interfaces which the SCFs use to return results. The communication environment is realized by using standard middleware architectures such as Common Object Request Broker Architecture (CORBA) [19].

Through this communication infrastructure, the SCS serves as proxy that interacts with core network elements [20]. Note that the SCS is a proxy in the sense of being a common point of contact and not because that the working logic resides in it and not elsewhere. It is also important to note that since the SCSs are logical entities, SCFs need not be implemented in the same box. Typically, different vendors will offer their own SCFs that will reside in different business domains as the other SCFs. There may also be cases where different SCFs from different vendors are offered which accomplish the same task. With this arrangement, an Application Service Provider (ASP) in the application layer will choose an SCF to use during service run-time. Moreover, a particular vendor may choose to implement different SCFs placed in disparate locations but all belonging to the same SCS.

2.2.2 Physical Architecture

The applications will interact with the SCSs via the APIs. Eventually it is the network entities that implement the required functionalities. SCSs are thus interfaces that abstract the core network entities from the applications. This abstraction is necessary since network elements such as switches, Interactive Voice Response systems (IVRs) and Generic Packet Radio Service (GPRS) Support Nodes (SGSNs) can be supplied by different vendors and thus specific operation logic be different.

As was described earlier, the specification for SCSs does not place a limitation on how they can be deployed in a network. There are thus various ways in which this objective can be satisfied. One approach is to deploy an SCS directly on a network element such as an HLR [20]. With this configuration the SCS is a component part of the network element [20]. The advantage of this configuration is that it results in a reduction of network traffic between the SCS and the network element. However, it also requires that the SCS be tightly coupled to the network element. Thus the software for the SCS may not be reusable across a plethora of network entities belonging to disparate vendors.

The second option is to deploy the SCS as a separate node [20]. The application will thus access the SCS via the API as usual while the SCS communicates with the network elements via a Customized Applications for Mobile Enhanced Logic (CAMEL) protocol [21]. This configuration is particularly useful where it is undesirable to deploy SCS directly on core network nodes such as a Mobile Switching

Centre (MSC) [20]. If this is the case, then the SCSs should be deployed to every MSC in the network to offer ubiquitous services. This solution is not easily scalable for larger networks with many MSCs. A Service Enabler Sublayer (SES) [20] in the network can be distinguished, as shown in figure 2.2

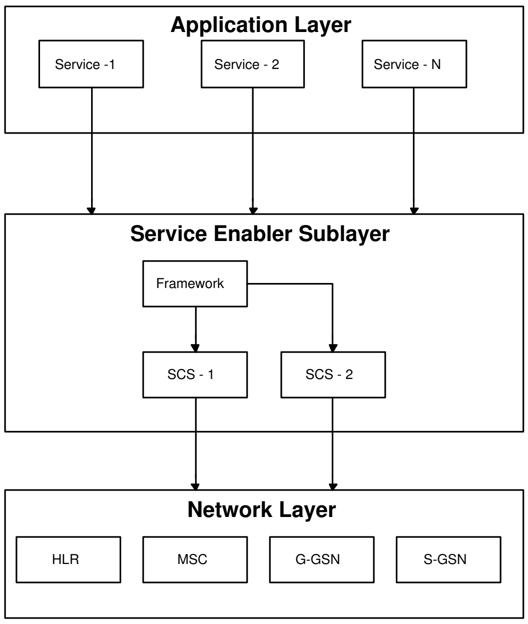


Figure 2.2: Service Enabler Sub-Layer

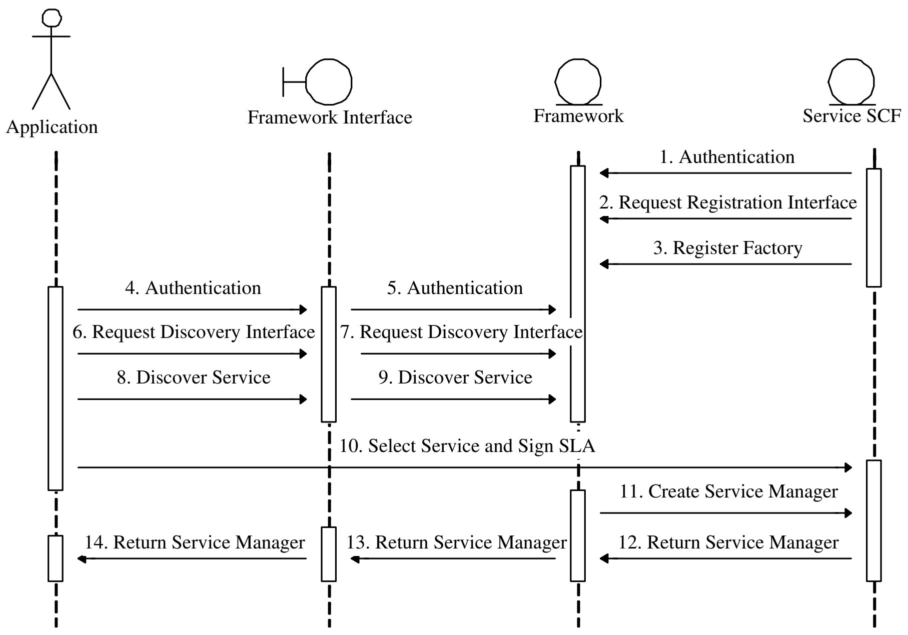
The SES layer will contain the SCSs and the Gateway Framework. The API that implements the functionalities of the gateway can be offered in one physical node or the SCSs can be distributed in different nodes while the OSA / Parlay Gateway node contains the framework and a few other SCSs.

2.2.3 Role of the Parlay Framework

The idea behind the Parlay Framework is to allow the telecommunications network operator to control access to the Parlay Gateway. Moreover, the Framework allows for scalability and introduces an element of competition between vendors. To encourage competitiveness, the framework allows for the registration of SCSs from different vendors. SCSs implementing non-standard APIs can also be permitted by the operator. The Framework will thus be a single point of access under the control of the network operator that applications can use to access the Parlay API. This approach ensure the ease of implementing and enforcing different security policies by the network operator. The Framework implements the following group of APIs

- Trust and Security Management (TSM) [22] for authentication of domains
- Registration of new SCSs to the framework
- Service Life Cycle Manager for the object instantiation of API implementation.
- Service Discovery to allow an application to find existing interfaces in the network.
- Integrity Management for issues such as load balancing and fault management.
- Event Notification to allow applications to register for specific events such as the registration of new SCSs.
- Contract Management to manage the contracts such as Service Level Agreements (SLA) between domains.

To elaborate on the operation of the above mentioned API types, a diagram in figure 2.3 shows a Message Sequence Diagram detailing the typical actions and API calls necessary to register an SCS with the Framework, application authentication by the Framework and the usage of the SCS by the application.



Application
Framework Interface
Framework
Service SCF
1. Authentication
2. Request Registration Interface
3. Register Factory
4. Authentication
5. Authentication
6. Request Discovery Interface
7. Request Discovery Interface
8. Discover Service
9. Discover Service
10. Select Service and Sign SLA
11. Create Service Manager
14. Return Service Manager
13. Return Service Manager
12. Return Service Manager

Figure 2.3: Service Registration And Discovery

2.2.4 Security Issues

As was stated above, having a single point of access and control allows for the network operator to easily enforce security policies as well as centralized management. It must be noted that since SCSs are logical entities, they need not reside in the same physical location. Applications will thus need to obtain a reference to the appropriate SCS at service run-time. This reference is similar to the CORBA Object Reference. For this to be possible, the application has to be authenticated first. This type of authentication is realized at the application layer and can be accomplished by using Trust and Security Management (TSM) Protocol as shown in figure 2.4.

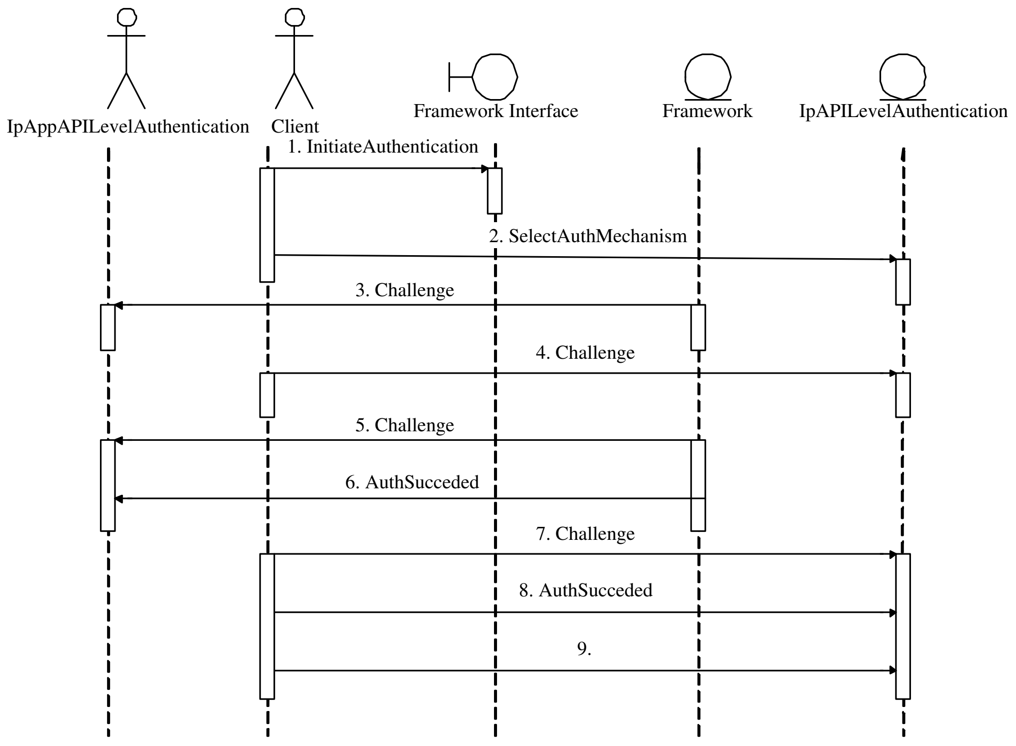


Figure 2.4: Trust And Security Management Protocol

Apart from authenticating at the application layer, it is necessary to ensure a secure communication channel as well as controlled access to capabilities [20]. One approach to ensure a secure communication channel is to establish dedicated links between the network operator domain and the Application Service Provider (ASP). Both parties can then ensure that only OSA / Parlay communications are allowed on the link. Communications protocol used on the dedicated link can then be restricted in accordance to the agreement between the parties. For instance, CORBA can be used as a communication platform through a firewall. Since CORBA uses object references to identify and locate run-time instance of object, Secure Socket Layer (SSL) [23] protocol can then be used to prevent the stealing of object references [20]. Security can also be ensured by controlling access using Service Level Agreements (SLAs)

2.2.5 Scalability Issues

It was mentioned earlier that the OSA / Parlay Framework provides applications with a single point of access to the SCSs. This allows scalability on various dimensions.

Firstly new SCSs can be added and registered with the Framework. Some of these SCSs may belong to different vendors all supporting the same protocol in communicating with the Framework. Some of these SCSs may be performing the same task. In that case, applications can negotiate SLAs and Quality of Service (QoS) parameters that can be used to choose the SCS to be used during service run-time.

As applications using the Gateway increase in number, the number of queries and transactions performed will also increase. The Framework provides the ability for load management. Applications can be directed to other SCSs performing the same task but running on different processors. This ability removes the disadvantage of a limited number of applications that can use the Gateway at a time. Since the OSA / Parlay Gateway is not an actual transport network entity, this type of scalability is limited by the network number and capabilities of network entities. This can be achieved without the applications being aware that different sessions run on different processors. Load management can further be achieved on a session basis by using middleware such as CORBA.

2.3 Support for Location Based Services

The previous sections have outlined the architecture of the OSA / Parlay standard. Within the architecture, considerations were taken for scalability and security. The emergence of 3G networks has highlighted the need for mobility. It is now necessary to provide subscribers of the mobile network with value added services as those enjoyed by Internet static subscribers. Moreover, mobile subscribers can leverage more services such as Location Based Services (LBS).

By their nature LBSs require location information. Location information exists in two forms, namely dynamic and static information. Dynamic location information includes data such as the mobile's location at every instant as the mobile device moves in the network. This dynamic information can be the cell in which the mobile exists at that instant in the GSM network or it ca be geodetic coordinates such as in a GPS environment.

There is also static location information that exists in databases which contains information about certain locations. For instance, a database may contain location information about certain restaurants or train services available in a particular area. A LBS service can thus be written to provide subscribers with a subset of this information, depending on the area they are in at that time. Moreover the static database must store spatial information so that queries can be tailored for specific locations.

The OSA / Parlay APIs are extensive, covering a large range of functionalities including Mobility, Location, Presence and Availability management, Call Control User Interaction, Messaging, Content Based Charging and Policy Management. These interfaces are valuable in providing dynamic location information that is needed by LBS services. However the OSA / Parlay standard does not define any interface for the access of static location information contained in geospatial databases. *It is thus the purpose of this report to motivate the extension of the OSA / Parlay specification to include support for LBS services by including the Geospatial Data Access SCF (GDASCF).* The GDASCF will reside in the OSA / Parlay gateway and will provide access to different types of data sources via the Adapter layer as explained in chapter 3.

Chapter 3

Design of the OSA / Parlay Geospatial Data Access SCF

3.1 Proposed Architecture

In creating applications, we would expect a set of reusable components for Location Based Services (e.g. Display a Map) to emerge, together with application specific logic. Based on that factor, the application layer is divided into service components and applications as shown in figure 3.1.

The service components are reusable, while the applications encapsulate the application specific logic. Service components can be accessed by applications for a particular service and will query the Geospatial Data Access SCF (GDASCF). They also contain callback interfaces as a way of returning results to the applications.

As opposed to other components of the OSA / Parlay gateway, which are used to control communications resources directly, by its very nature, the Data Access SCF is a data access API. The adapter layer above has a role similar to the adapters in the Java API for the Integrated Networks (JAIN) [9] technology, adopting the function calls to different operating environments. This approach closely matches the function of the adapter design pattern [1][2]. Rather than forcing the Data Access SCF to implement remote methods to access the databases individually, which creates excessive class bloating and diminishes performance and readability, concrete adapters are inherited and implemented by their providers. The Data Access SCF is

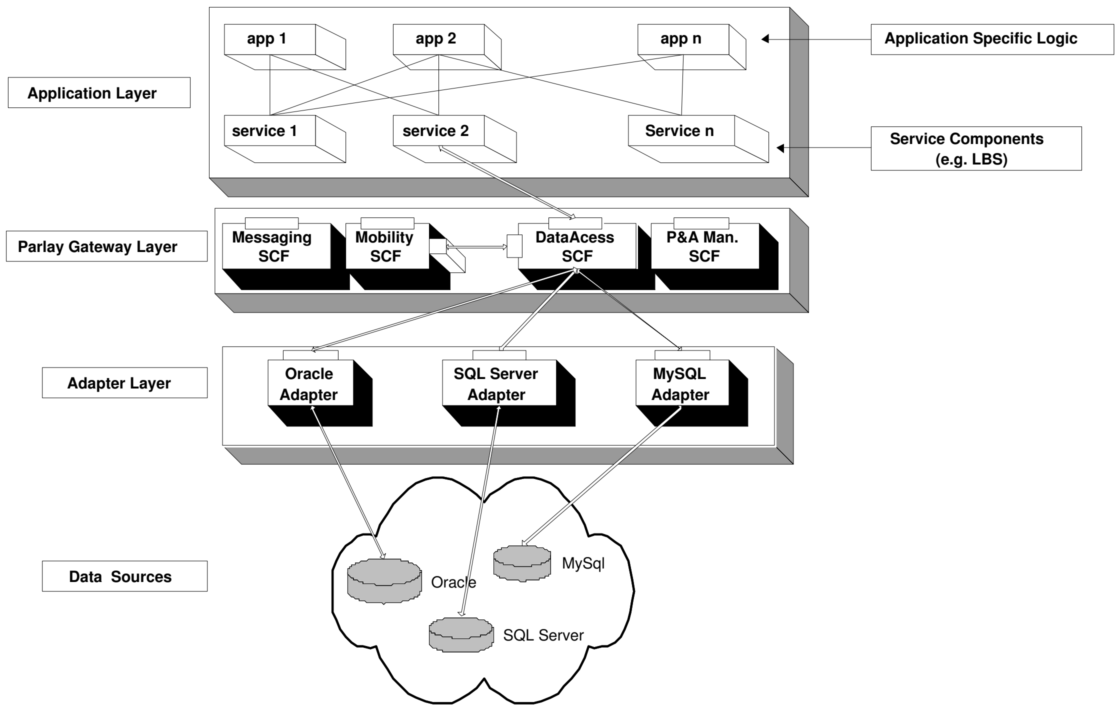


Figure 3.1: Proposed Architecture

unchanged as the adapter always supports the same interfaces.

Each adapter must be created for each Database Management System (DBMS). Adapter providers must inherit from the abstract adapter class to create their application specific concrete adapter class. This will ensure that all concrete adapters support the same interfaces while the implementation varies according to the application. This approach also adds a level of flexibility so that adapter implementors can add more non-standard interfaces over and above the standard interfaces. This approach closely resembles the Extension Interface [1] Design Pattern which allows multiple interfaces to be exported by a component, to prevent bloating of interfaces and breaking of client code when developers extend or modify the functionality of the component.

Note that different DBMSs may possess different functionalities or they may execute certain functionalities differently by adopting the above architecture, that information is abstracted from the service or application using the functionality. For instance, the Oracle DBMS has the Locator functionality, which is used extensively in LBS applications. A similar functionality may exist in the other DBMSs. However the GIS component of the OSA / Parlay will only expose the same set of APIs to the applications and services requiring the use of these functionalities. It is then up to the adapters in the adapter layer to convert to the correct syntax during service-runtime. Furthermore the API in the adapter layer between the gateway and the adapter can be standardized. This will ensure that the implementers of the Data Access SCF develops against the same set of APIs for the adapters, regardless of existing DBMSs further enhancing software reuse.

3.2 Design Requirements and Constraints

The GDASCF is a component of the Parlay gateway. Application will use the the GDASCF to query for static geospatial data contained in various databases from different providers in a generic manner. The GDASCF must also be able to communicate with the Mobility Management Service Capability Feature (SCF) use-cases already supplied by the Parlay gateway.

To support context aware computing [24], the Mobility Management [25] Service Capability Feature supports three types of requests.

1. Interactive Requests - To support interactive requests from the applications.
2. Periodic Requests - To support periodic requests from applications. A period is set at which the SCF must return results to the application.
3. Triggered Requests - To support triggered requests, an event is set by the application. The SCF will return results back to the application when the event is triggered.

This places an important constraint on the design of the GDASCF. Applications must be able to invoke these types of requests at any point in time to support context aware computing.

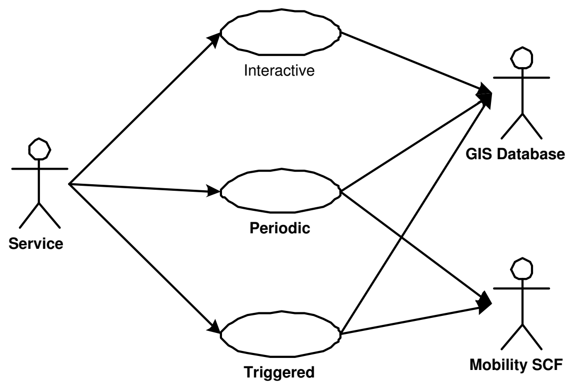


Figure 3.2: Use Case Diagram

3.3 Design Methodology

To identify the methods of the GDASCF interface, the ICONIX [26] design process was followed. This process is based on the Unified Modelling Language (UML) [27] and inherits from Rational Unified Process (RUP) [27] and Reference Model for Open Distributed Processing (RM-ODP) [28]. The process is also use-case driven in accordance with the OSA / Parlay standard.

3.3.1 Use Case Modelling

The use-case diagram in figure 3.2 shows the actors of the Data Access Component and their use-cases. The actors are Service, Mobility SCF and the GIS Database.

The actor GIS Database is taken as a combination of the actual database storage medium, Database Management System (DBMS) and Database Adapter. As explained above, the use-cases are divided into three parts. The actor Service can invoke any type of use-case at any time. The GIS Database is a passive object that the Data Access SCF uses to store and retrieve geospatial data. The Mobility SCF is used by the periodic and triggered use-cases to deliver the required position information. The use-cases represent the functionality called ‘Establish User Position’ and ‘Proximity to a Region’. Every service must register with the OSA / Parlay Framework before it can use any SCF. This registration, which includes authentication and authorization, is not shown but is discussed in detail in [20].

For the purposes of this report, callback interfaces implemented at the application layer will have names starting with the prefix “IPApp”. SCFs in the gateway will be named starting with prefix “IP”. The Communications Session Manager (CSM) manages the communication sessions established by service components at the application layer. The IPAppInterface serves as the initial contact for services.

3.3.2 Use Cases

Establish User Position - Interactive Request

1. PRE-CONDITIONS

   The application contacts the IPAppInterface, which invokes the CSM. The CSM then instantiates the IPAppUserPosition and the IPUserPosition.

2. MAIN-FLOW

   The service sends a message to the IPAppUserPosition. This message contains the user’s device longitude and latitude and device ID. It also contains the name of the Spatial Reference System (SRS) that it uses. The map displayed on the user’s device will have the SRS. The IPAppUserPosition in turn passes a message to the IPUserPosition containing the same parameters. The IPUserPosition determines which database supports Position information with that required SRS. The IPUserPosition then connects to that database’s driver. It passes a message to the driver. The driver then formulates an SQL query specific to the DBMS and sends it to the database. The DBMS then returns a result set to the driver. The driver passes this result to the IPUserPosition. This information is finally relayed to the IPAppUserPosition back to the service.

3. ALTERNATE-FLOWS

   - The IPUserPosition does not find a database containing the Position information. So it throws an exception back to the IPAppUserPosition. The IPAppUserPosition then throws an exception back to the service.
   - The IPUserPosition finds a database supporting the Position information but the required information is not contained inn the database. The DBMS returns an error back to the driver. The driver will then throw

an exception back to the IPUserPosition, which is finally relayed to the service.

Establish User Position - Periodic Request

1. PRE-CONDITIONS

   The application contacts the IPAppInterface, which invokes the CSM. The CSM then instantiates the IPAppUserPosition and IPUserPosition and the IpUserLocationCamel.

2. MAIN-FLOW

   The service sends a message to the IPAppUserPosition, requiring a Periodic Position request. This message contains the user's device longitude and latitude and device ID. It also contains the name of the Spatial Reference System (SRS) that it uses. The IPUserPosition sends a message to IpUserLocationCamel for periodic location request. After a certain time interval, the IpUserLocationCamel reports this to the IPUserPosition until periodic reporting is stopped. The IPUserPosition determines which database supports Position information with that required SRS. The IPUserPosition then connects to that database's driver. It then passes a message to the driver. The driver then formulates an SQL query specific to the DBMS and sends it to the database. The DBMS then returns a result set to the driver. The driver passes this result to the IPUserPosition. This information is then relayed to the IPAppUserPosition back to the service.

3. ALTERNATE-FLOWS

   - The IPUserPosition does not find a database containing the location information. It then throws an exception back to the IPAppUserPosition. The IPAppUserPosition then throws an exception back to the service.
   - The IPUserPosition finds a database supporting the location information but the required information is not contained in the database. The DBMS returns an error back to the driver. The driver will then throw an exception back to the IPUserPosition, which is finally relayed to the service.

- The IPAppUserPosition sends a Triggered Location Stop Message to the IPUserPosition. The IPUserPosition will then send a TriggeredLication-ReportingStop message to the IPUserLocationCamel and a Database connection is closed.

Establish User Position - Triggered Request

1. PRE-CONDITIONS

   The application contacts the IPAppInterface, which invokes the CSM. The CSM then instantiates the IPAppUserPosition and IPUserPosition and the IpUserLocationCamel.

2. MAIN-FLOW

   The service sends a message to the IPAppUserPosition, requiring a triggered Position request. This message contains the user's device longitude and latitude and device ID. It also contains the name of the Spatial Reference System (SRS) that it uses. The IPUserPosition sends a message to IpUserLocationCamel for triggered location request. When the user's location changes, the IpUserLocationCamel reports this to the IPUserPosition until triggered reporting is stopped. The IPUserPosition determines which database supports Position information with that required SRS. The IPUserPosition then connects to that database's driver. It passes a message to the driver. The driver then formulates an SQL query specific to the DBMS and sends it to the database. The DBMS returns a result set to the driver. The driver passes this result to the IPUserPosition. This information is finally relayed via the IPAppUserPosition back to the service.

3. ALTERNATE-FLOWS

   - The IPUserPosition does not find a database containing the location information. It then throws an exception back to the IPAppUserPosition. finally, the IPAppUserPosition throws an exception back to the service.
   - The IPUserPosition finds a database supporting the location information but the required information is not contained in the database. The DBMS returns an error back to the driver. The driver will then throw an exception back to the IPUserPosition, which is finally relayed to the service.

- The IPAppUserPosition sends a Triggered Location Stop Message to the IPUserPosition. The IPUserPosition will send a TriggeredLicationReportingStop message to the IPUserLocationCamel and a Database connection is closed.

Proximity to a Region - Interactive Request

1. PRE-CONDITIONS

   The service has been contacted by the application to request a proximity service. The service contacts the CSM and the IpAppProximity and IpProximity are instantiated.

2. MAIN-FLOW

   The IpAppProximity sends a message to the IpProximity for interactive proximity query. The IpProximity then determines which driver supports proximity queries from the registry. The IpProximity then connects to the driver. The driver then formulates a query statement and sends it to the corresponding DBMS. The DBMS then returns the result set to the driver, which passes it back to the IpProximity. The information is finally relayed back to the IpAppProximity.

3. ALTERNATE-FLOWS

   - The IpProximity does not find a database containing the proximity information. It then throws an exception back to the IpAppProximity. The IpAppProximity then throws an exception back to the service.
   - The IpProximity finds a database supporting the Position information but the required information is not contained in the database. The DBMS returns an error back to the driver. The driver will then throw an exception back to the IpProximity, which is finally relayed to the service.

Proximity to a Region - Periodic Request

1. PRE-CONDITIONS

   The service has been contacted by the application to request a proximity service. The service contacts the CSM and the IpAppProximity and IpProximity are instantiated.

2. MAIN-FLOW

   The IpAppProximity sends a message to the IpProximity for periodic proximity query. The IpProximity sends a message to IpUserLocationCamel for periodic location request. After a certain time interval, the IpUserLocationCamel reports the location of the mobile user to the IpProximity until periodic reporting is stopped. The IpProximity then determines which driver supports proximity queries from the registry. The IpProximity connects to the driver. The driver formulates a query statement and sends it to the corresponding DBMS. The DBMS then returns the result set to the driver, which passes it back to the IpProximity. The information is finally relayed back to the IpAppProximity.

3. ALTERNATE-FLOWS

   - The IpProximity does not find a database containing the proximity information. It then throws an exception back to the IpAppProximity. The IpAppProximity finally throws an exception back to the service.
   - The IpProximity finds a database supporting the Position information but the required information is not contained inn the database. The DBMS returns an error back to the driver. The driver will then throw

an exception back to the IpProximity, which is finally relayed to the service.

Proximity to a Region - Triggered Request

1. PRE-CONDITIONS

   The service has been contacted by the application to request a proximity service. The service contacts the CSM and the IpAppProximity and IpProximity are instantiated.

2. MAIN-FLOW

   The IpAppProximity sends a message to the IpProximity for Triggered proximity query. The IpProximity sends a message to IpUserLocationCamel for Triggered location request. When the user's location changes, the IpUserLocationCamel reports this to the IPUserPosition until triggered reporting is stopped. The IpProximity then determines which driver supports proximity queries from the registry. The IpProximity connects to the driver. The driver formulates a query statement and sends it to the corresponding DBMS. The DBMS then returns the result set to the driver, which passes it back to the IpProximity. The information is finally relayed back to the IpAppProximity.

3. ALTERNATE-FLOWS

   - The IpProximity does not find a database containing the proximity information. It then throws an exception back to the IpAppProximity. The IpAppProximity then throws an exception back to the service.
   - The IpProximity finds a database supporting the Position information but the required information is not contained inn the database. The DBMS returns an error back to the driver. The driver will then throw an exception back to the IpProximity, which is finally relayed to the service.

Note that application designers are free to model the application layer to their liking. It is for that reason that the activation of a application is modelled as a passed message and not a function call. This opens different ways in which designers can implement their message passing protocols between the application and the interface object. For instance, the interface can be implemented as Graphical User interface (GUI) as in a Web service or cell-phone application or it can be implemented as another API with the service calling its interfaces. In both cases, the way messages are passed to it is influenced by the type of the interface. The Model View Controller (MVC) model can be used to address this issue.

3.3.3 Sequence Diagrams

The sequence diagrams figure 3.3 to 3.5 outline the flow of the use-cases, including the objects that collaborate to achieve these functionalities. As can be seen from the diagrams, two new objects were discovered. The 'Driver' helps to convert the calls from the gateway to specific Structured Query Language (SQL) commands of the implemented database. The 'Registry' is necessary to store information concerning every driver that registers with the Data Access SCF.

Sequence Diagrams for Establish User Position

The service will send a message to the appropriate *IpApp* interface. An appropriate *PosionReportReq()* method will thus be invoked on the *Ip* interface. The interactive use case will thus query the Registry for the database supporting the required information. The periodic and triggered cases will first send a message to the mobility SCF for periodic or triggered updates. The Registry will then be queried for database information. The *Ip* interface will then connect to the driver and a database will be queried. The result set from the database will be relayed back to the service as shown in figures 3.3 to 3.5.

The application must first register for events at periodic time intervals. The Period is passed as a parameter by the application. The Mobility SCF will then return results via a callback interface to the GDASCF at those specified intervals. The results are finally relayed back to the application.

The applications will first register for events with the GDASCF. For instance in a GSM network, an event can be set for when a mobile changes cells. This will cause the Mobility SCF to call a callback method of the GDASCF notifying it of the event and its parameters.

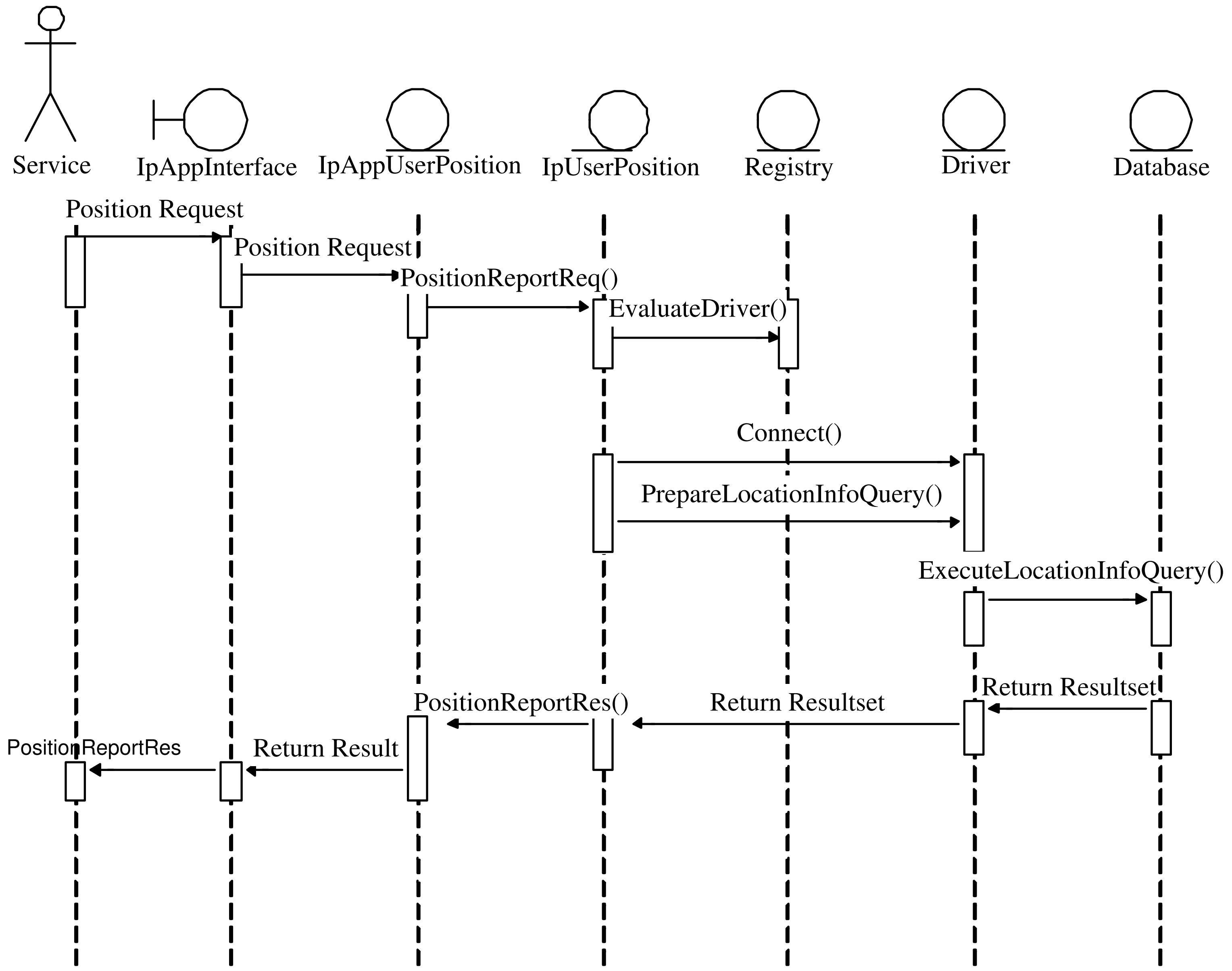


Figure 3.3: Establish User Position - Interactive Request

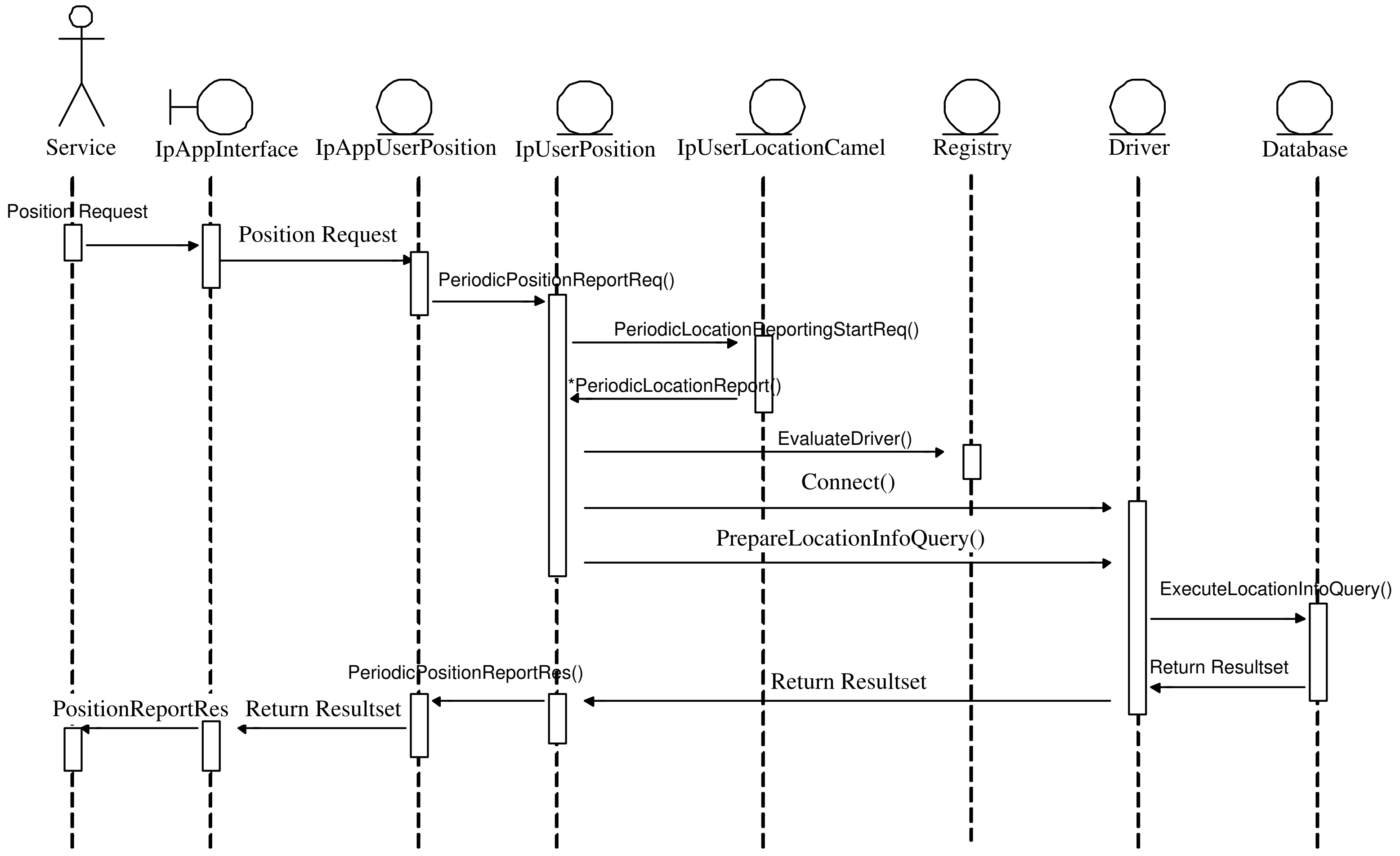


Figure 3.4: Establish User Position - Periodic Request

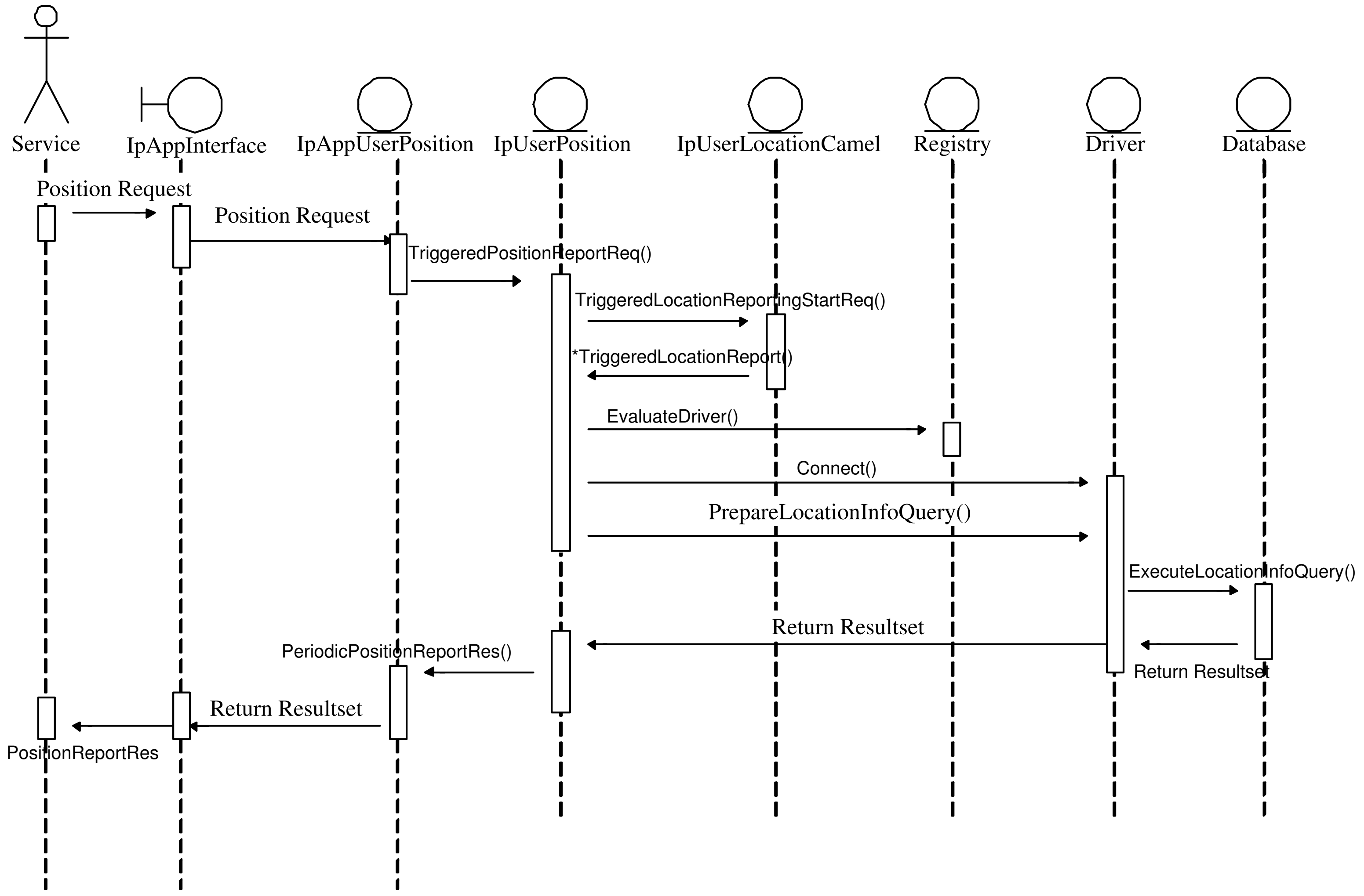


Figure 3.5: Establish User Position - Triggered Request

It must be noted that the difference between the three use-cases in figures 3.3 to 3.5 is the inclusion of the *IpUserLocationCamel* Interface from the Mobility Management SCF [29] in the Periodic and Triggered cases. This is necessary since geographic information is required by the GDASCF in this instances.

Sequence Diagrams for Proximity to a Region

The service will send a message to the appropriate *IpApp* interface. An appropriate *PosionReportReq()* method will thus be invoked on the Ip interface. The interactive use case will thus query the Registry for the database supporting the required information. The periodic and triggered cases will first send a message to the mobility SCF for periodic or triggered updates. The Registry will be queried for database information. The *Ip* interface will connect to the driver and a database will then be queried. The result set from the database will be relayed back to the service as shown in figures 3.6 to 3.5 below.

Similar to the 'Establish User Position', The service will send a message to the appropriate *IpApp* interface. An appropriate *ProximityReportReq()* method will thus be invoked on the *Ip* interface. For the Periodic and Triggered cases, the *IpApp* interface will first send a message to the Mobility SCF notifying it to return periodic and triggered updates respectively. The procedure for querying the database and returning results is the same as above.

By their very nature, sequence diagrams show the time flow of information and method calls. However, they are derived from dynamic view of ICONIX that does not show time dependance but is very useful in determining collaborating candidate objects and their methods. This view is called Robustness modelling [26] and the corresponding diagrams for the above use-cases are shown in Appendix A.

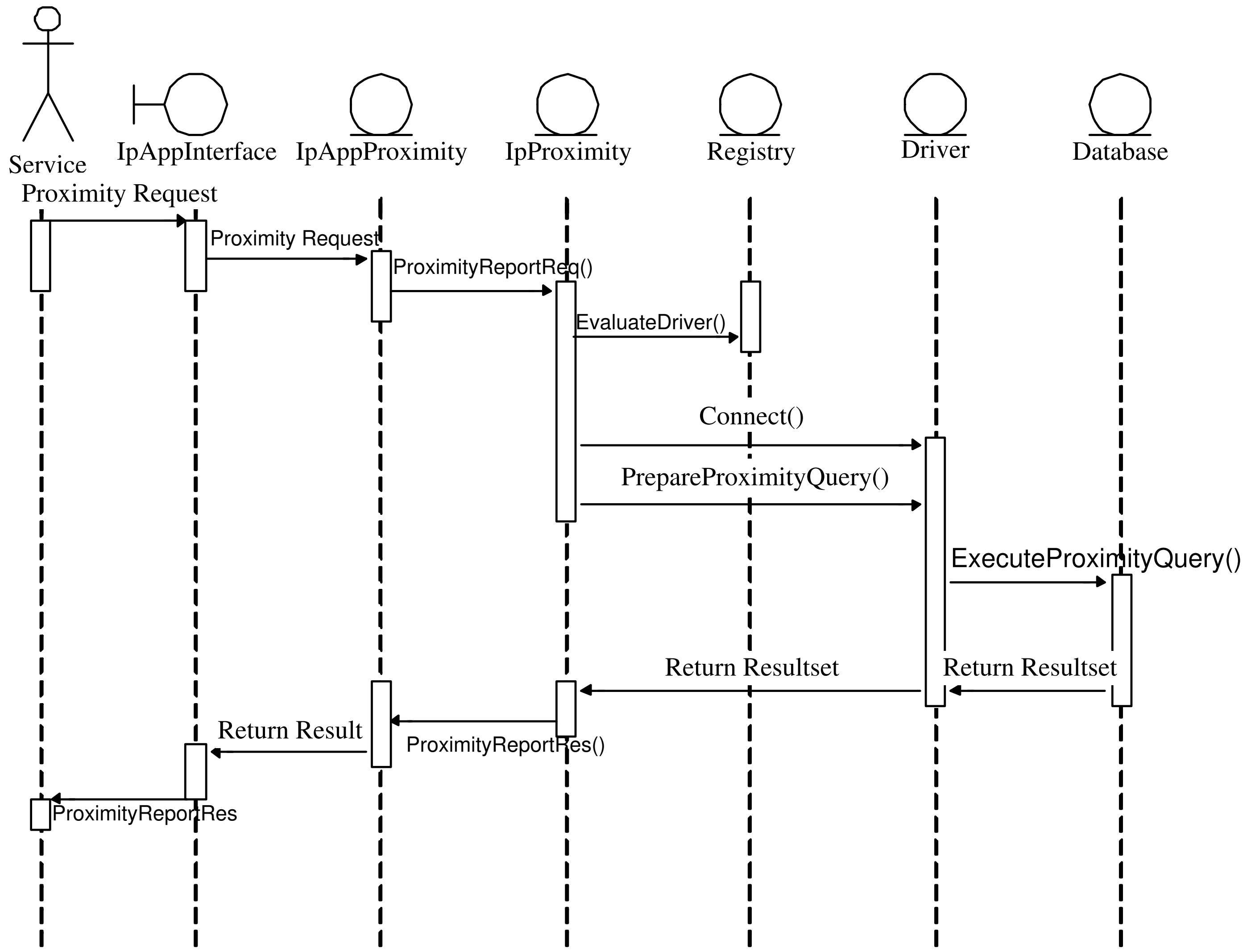


Figure 3.6: Proximity to a Region - Interactive Request

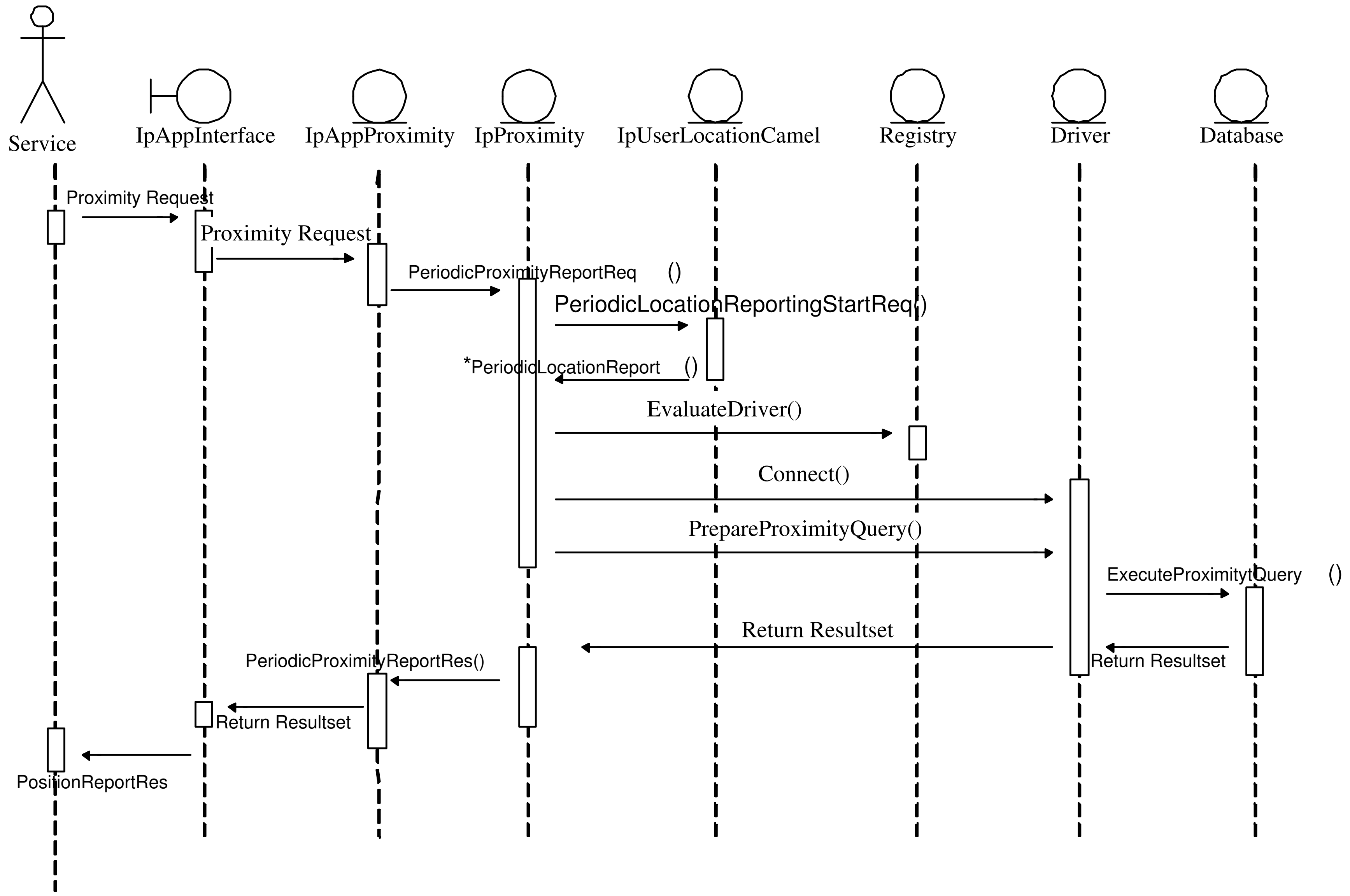


Figure 3.7: Proximity to a Region - Periodic Request

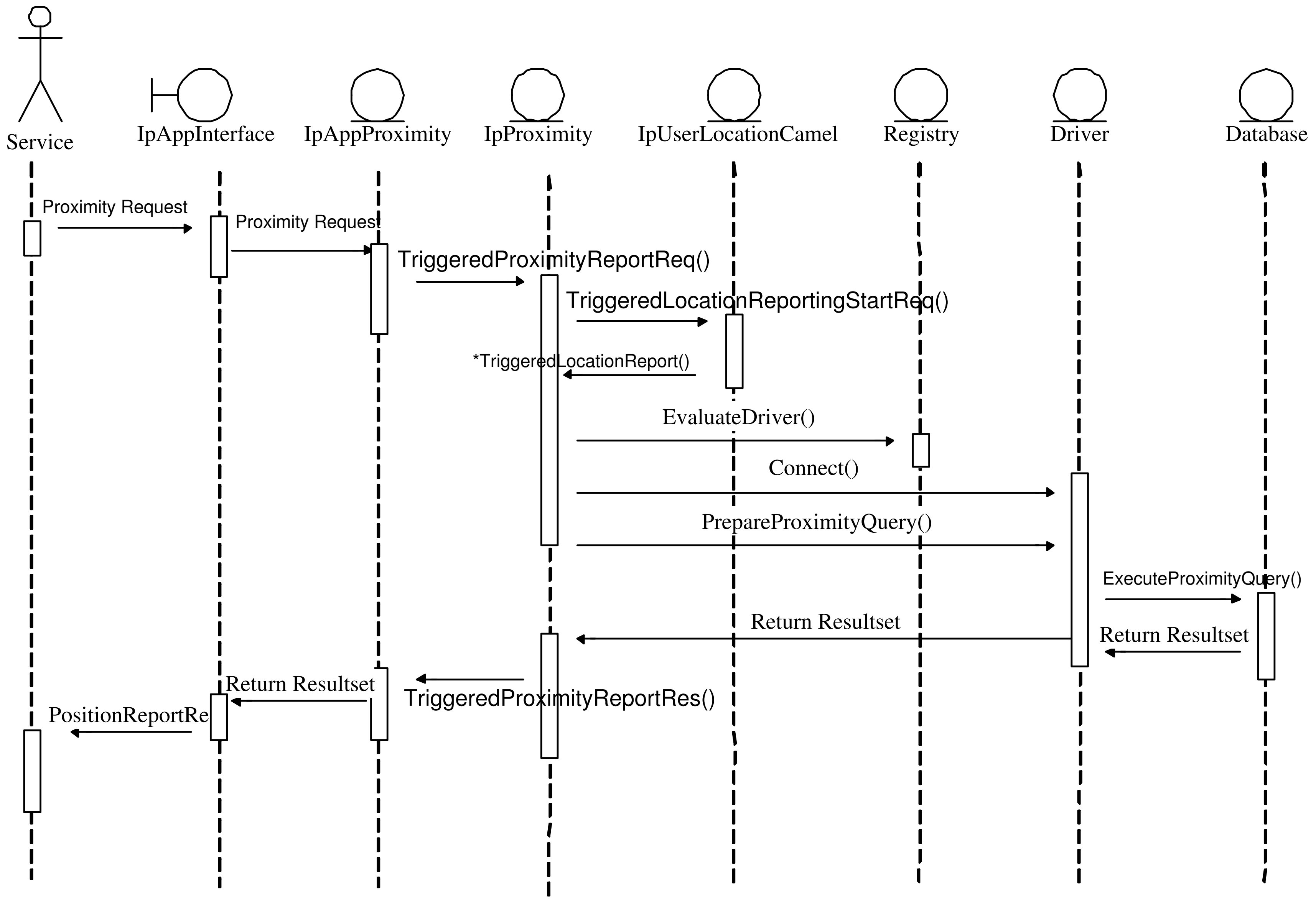


Figure 3.8: Proximity to a Region - Triggered Request

Chapter 4

Implementation of The Geospatial Data Access SCF

4.1 Implementation Approach

The ICONIX design process suggests that a design of a system be started from modelling interactions with the outside world in the form of use-cases. Chapter 3 outlines the use-cases for services that will be running on top of the Gateway and are necessary to model the internal functionality of the GDASCF. Note that these services are not restricted from using other SCFs of the Gateway.

To implement the services outlined in chapter 3, all layers of the Proposed Architecture shown in figure 3.1 had to be simulated. The Mobility Management SCF was simulated by writing a simple method that returns arbitrary location information to the GDASCF.

4.2 Deployment of Service and Gateway Components

Deployment and development of the services was achieved by using *The Ace ORB* (TAO) which is a CORBA version 2.0 compliant with the Object Management Group (OMG) Object Request Broker (ORB). All implementation is performed using C++ on a Linux Redhat 9 platform. Figure 4.1 below outlines the logical view of component deployment.

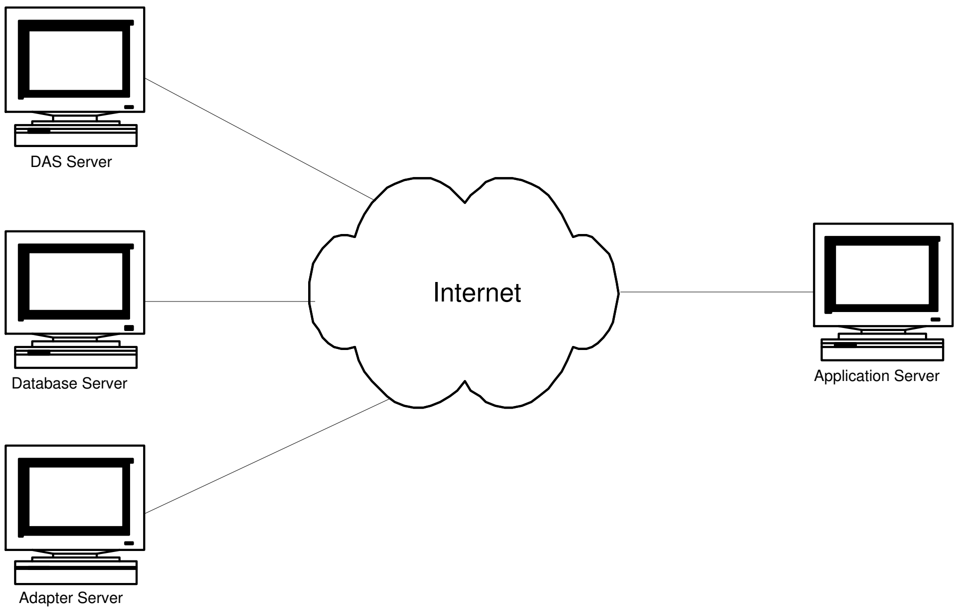


Figure 4.1: Deployment of Components

For generality, different servers run components at different layers of the Proposed Architecture shown in figure 3.1 and explained in section 3.1. However, note that all the interfaces can be run on the same server. Figure 4.1 suggests that the interfaces may exist in different business domains in the general case.

Every interface that runs inside a server has its unique address known as the Interoperable Object Reference (IOR). The semantics of the IOR are similar to that of the C++ pointer except that they go beyond pointing at objects in the same memory space. For simplicity, all interfaces know each other's IORs. These IORs are stored in different files with the same names as the name of the interfaces. The Interface Definition Language (IDL) files for the interfaces are shown in Appendix B.

4.3 Example services with and without the GDASCF

This section describes two approaches of implementing a *"My Nearest"* service. The first approach looks at how this type of service is implemented without using the OSA / Parlay gateway. In the second approach, this service is implemented by using the GDASCF as part of the OSA / Parlay gateway. Advantages of implementing the service using the GDASCF are then extracted.

4.3.1 First Approach : Implementation without using the GDASCF Interface

The most common way of implementing a GIS is to deploy the GIS server within an organization or a corporate Local Area Network (LAN). The server includes the GIS Database as well as services that can be invoked by users. Each organization defines rules of how to access the database and services in a specific manner. Figure 4.2 shows how a GIS can be deployed in an organization.

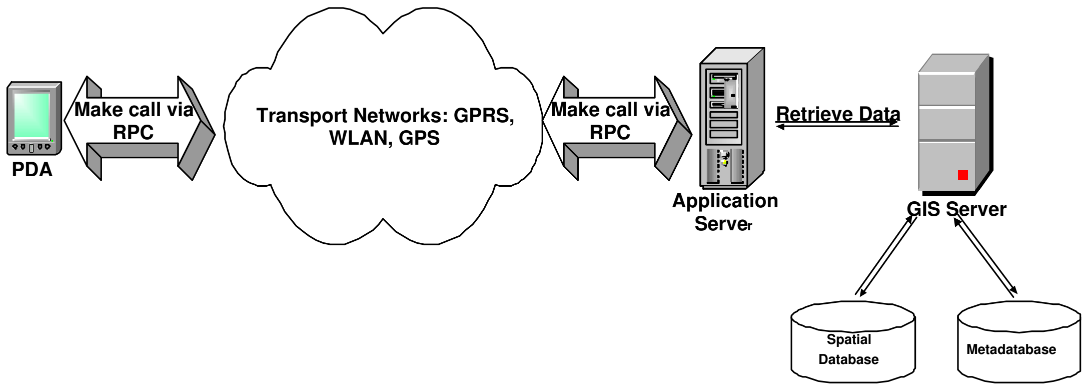


Figure 4.2: Deployment of a GIS in a corporate

Users are usually employees of the organization with controlled access to the GIS. Furthermore, users have to understand interaction rules defined by the organization.

The client application can access the service in the GIS using Remote Procedure Calls (RPCs). This is essential since RPC abstracts the client and the service from different network infrastructures.

A sequence diagram in figure 4.3 shows how a client accesses a corporate database using RPC.

1. Firstly, the *RPCClient* makes a call *ExecuteQuery()*, which is actually a stub interface to the actual method implemented by *RPCService*.

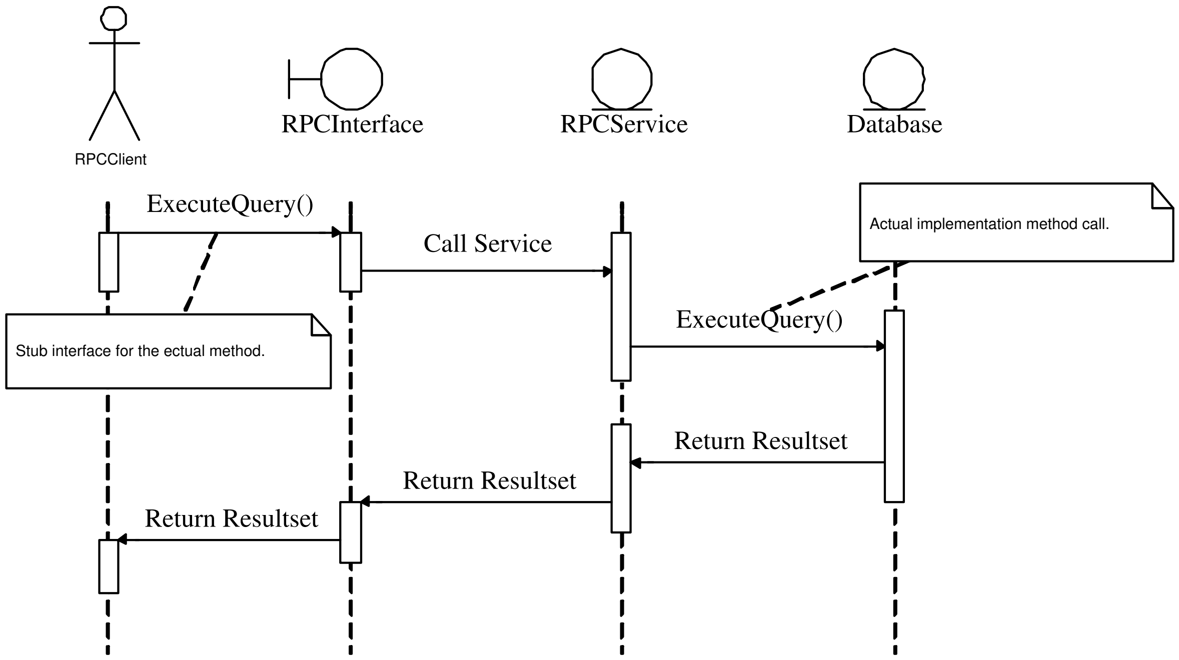


Figure 4.3: Sequence diagram to access corporate database using RPC

2. The *RPCInterface* implements the skeleton interfaces, which interprets component invocation requests and calls the corresponding methods in the *RPCService*, with the parameter values supplied by the RPCClient.

3. The *RPCService* implements the *ExecuteQuery()* method which queries the database and returns the result set back to *RPCService*. The result is then returned to *RPCClient*.

4.3.2 Second Approach : Implementation using the GDASCF Interface

In this section, we will demonstrate how a “My Nearest” type of a location based service can be implemented using the GDASCF. The service implemented is called *Find Nearest ATM* and is accessed via the Short Message Service (SMS). The flow of events are as follows:

1. The user sends an SMS message to the SMS gateway. The message contains the keywords “ATM *name-of-bank*” where *name-of-bank* can be any name of a bank.

2. The SMS is intercepted by an SMS gateway. The Kannel [30] is used as as an SMS gateway in this service.
3. The gateway extracts the keywords from the SMS message and invokes the corresponding application, passing the keywords as arguments.
4. The IPAppProximity is instantiated.
5. The IpAppProximity sends a message to the IpProximity for interactive an proximity query.
6. The IpProximity determines which driver supports proximity queries from the registry.
7. The IpProximity connects to the driver.
8. The driver formulates a query statement and sends it to the corresponding DBMS.
9. The DBMS returns the result set to the driver, which passes it back to the IpProximity.
10. The information is relayed back to the IpAppProximity.
11. The gateway finally sends an SMS back to the user containing relevant information.

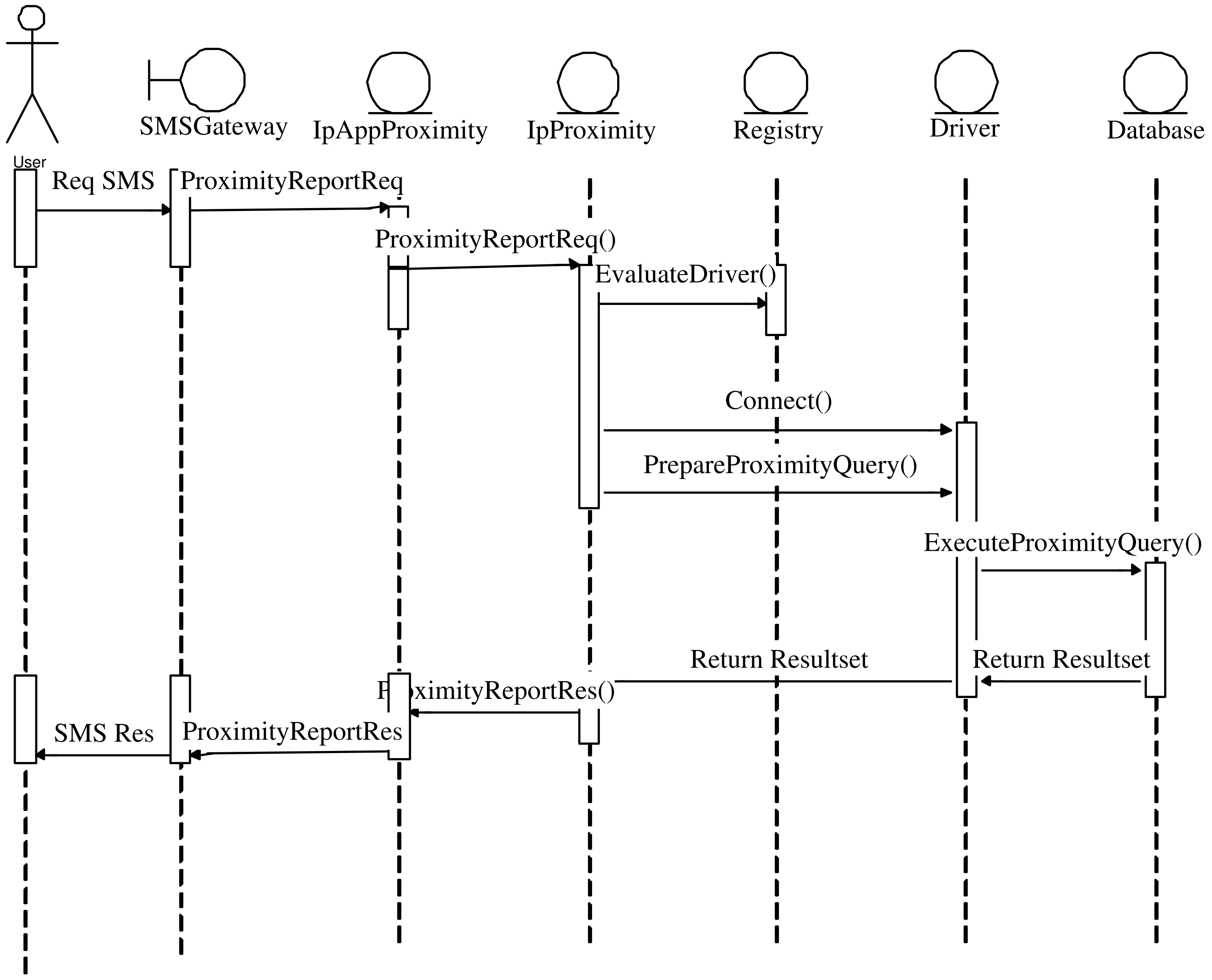


Figure 4.4: Sequence of events for "Find Nearest ATM" Service

4.3.3 Comparison of both approaches

In the first approach, the methods of the *RPCService* are written to be specific for a particular implementation. Thus the implementation of this service must be aware of the structure of the database at design time since it will be querying it directly. Should the database implementation change, the *RPCService* will have to be rewritten and recompiled. The abstraction provided by the GDASCF in the second approach, does not require the application accessing the service to know the structure of the database. Thus changes to the database structure are kept transparent from the application. Moreover, applications can query a different database at run-time with no recompilation required. This factor has a considerable impact on the development time and time-to-market of applications.

Since methods of the *RPCService* are not standardized, a protocol for client / server communication must be specified during design time. This can be done achieved by generating client stubs and server skeletons. If the server implementation changes, new stubs and skeletons must be generated and installed in both the client and the server. Clearly if there are many clients accessing the service, this will impact negatively on deployment time. Standardization of method calls in the GDASCF ensures that the client knows its input and output parameters all the time. Moreover, the GDASCF is contained in the OSA / Parlay gateway, which is maintained by a separate entity. The client application is thus free to choose between different GDASCFs at run-time.

In the first approach, since the GIS and the services are maintained by one organization, the interface of the service is limited to the scheme that the organization chooses. As shown in figure 4.3, the interface is limited to communicating using only RPC. However, more interfaces can be implemented, but they have to access the service via RPCInterface. In the second approach, the access to the service uses the SMS Gateway interface as shown in figure 4.4. More interfaces can be implemented that are independent from the SMS Gateway interface.

Use of the OSA / Parlay gateway with GDASCF motivates the fact that the GIS and services that run on it, do not have to be maintained by the same organization. This fact is important since 3^{rd} party organizations are motivated to develop different services while gaining controlled access to the GIS.

The table 4.1 summarizes the differences between the two approaches as outlined above.

First Approach	**Second Approach**
Database structure dependant.	Database structure and implementation independent.
No standard calling convention i.e. Different server implementations can break the client	Standardized OSA/Parlay APIs. New APIs can be added in a controlled manner without breaking the client.
Client applications need to be recompiled for different service implementations.	Client Applications can choose between different SCFs at run-time without recompilation.
Implementing new interface objects is not scalable enough i.e. It depends on the initial interface.	More interfaces can be implemented for flexibility and scalability.
Applications are highly coupled to the service. This restricts 3^{rd} party organizations from accessing the service.	3^{rd} party organizations can gain controlled access to the service. This has the effect of increasing revenue for the service provider, while reducing costs associated with implementing the service infrastructure, such as a GIS, for 3^{rd} party organizations.
Increased time-to-market of applications and services.	Reduced time-to-market of applications and services.

Table 4.1: Comparison of the two approches

4.4 Chapter Summary

In the previous chapter we discussed the design of services that run on the GDASCF and as well as to model its internal behavior. In this chapter we discussed the platform on which the services are implemented. The purpose of this chapter was to show how different components at different layers of the proposed infrastructure can be deployed. Two example where provided to show the difference between services that use GDASCF and those that do not. Advantages gained by using GDASCF are then and outlined.

Note that in figure 4.4, the calls to *IpProximity* could be changed by calling the *TriggeredProximityReportReq()* and *PeriodicProximityReportReq()* without changing method calls to other interfaces. The only difference is that the *IpProximity* interface would then make calls to the *Mobility Management* interface for for triggered and periodic location reports cases. However the implementation logic to call the methods of the *Mobility Management* interface lies within the gateway and is hidden from the application or service developer. We conclude that the GDASCF provides generic interfaces across a plethora of services.

Chapter 5

Critical Design Review

5.1 Introduction

Best practices in software engineering require that software be evaluated against a set of pre-defined matrices. This will ensure that the software is designed to be highly reusable and easy to maintain. Amongst the matrices that will be discussed in this chapter are *Coupling*, *Cohesion* and *Completeness* and *Primitiveness*. Detailed explanations of these matrices can be found in [31]. The purpose of this chapter is to show how these software matrices apply to the design of the GDASCF.

5.2 Coupling

Coupling is *a measure of strength of association established by a connection from one module to another* [31]. The intent is to produce a design with weak coupling between classes, as this reduces complexity and results in a system that is easier to change or correct.

However, there is a trade-off between coupling and inheritance. While weak coupling is desirable, inheritance is necessary to help us exploit the commonality among abstractions and thus be able to reuse classes. Achieving this desired effect of inheritance results in strong coupling between superclasses and subclasses.

To balance the tradeoff between coupling and inheritance, we divide coupling into two forms which are

- Vertical coupling describes the coupling between superclasses and subclasses.
- Horizontal coupling describes coupling between classes on the same level of the hierarchy in a class diagram.

This two forms of coupling can be best illustrated by use of a diagram. Figure 5.1 shows inheritance results with strong vertical coupling.

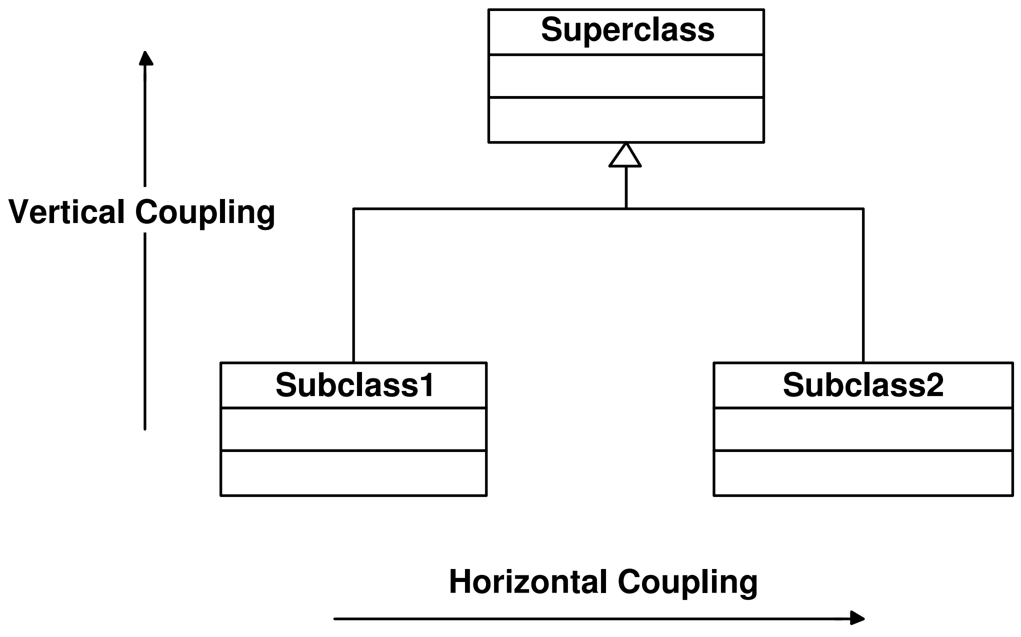


Figure 5.1: Vertical and Horizontal Coupling

As shown in the class diagram of figure 5.2, vertical coupling is increased by employing inheritance. This is necessary to make sure that interfaces can be easily changed and reused. Horizontal coupling is reduced by employing as little association between the classes across the hierarchy as possible.

5.3 Cohesion

Cohesion is defined *as the degree of connectivity among the elements of single module, class or object* [31]. The degrees of cohesion varies across the spectrum from the least desirable coincidental cohesion to the most desirable functional cohesion. Coincidental cohesion occurs when entirely unrelated abstractions are grouped together in a class or module. Functional cohesion is achieved by grouping elements with identical functionality to provide well-bounded behavior.

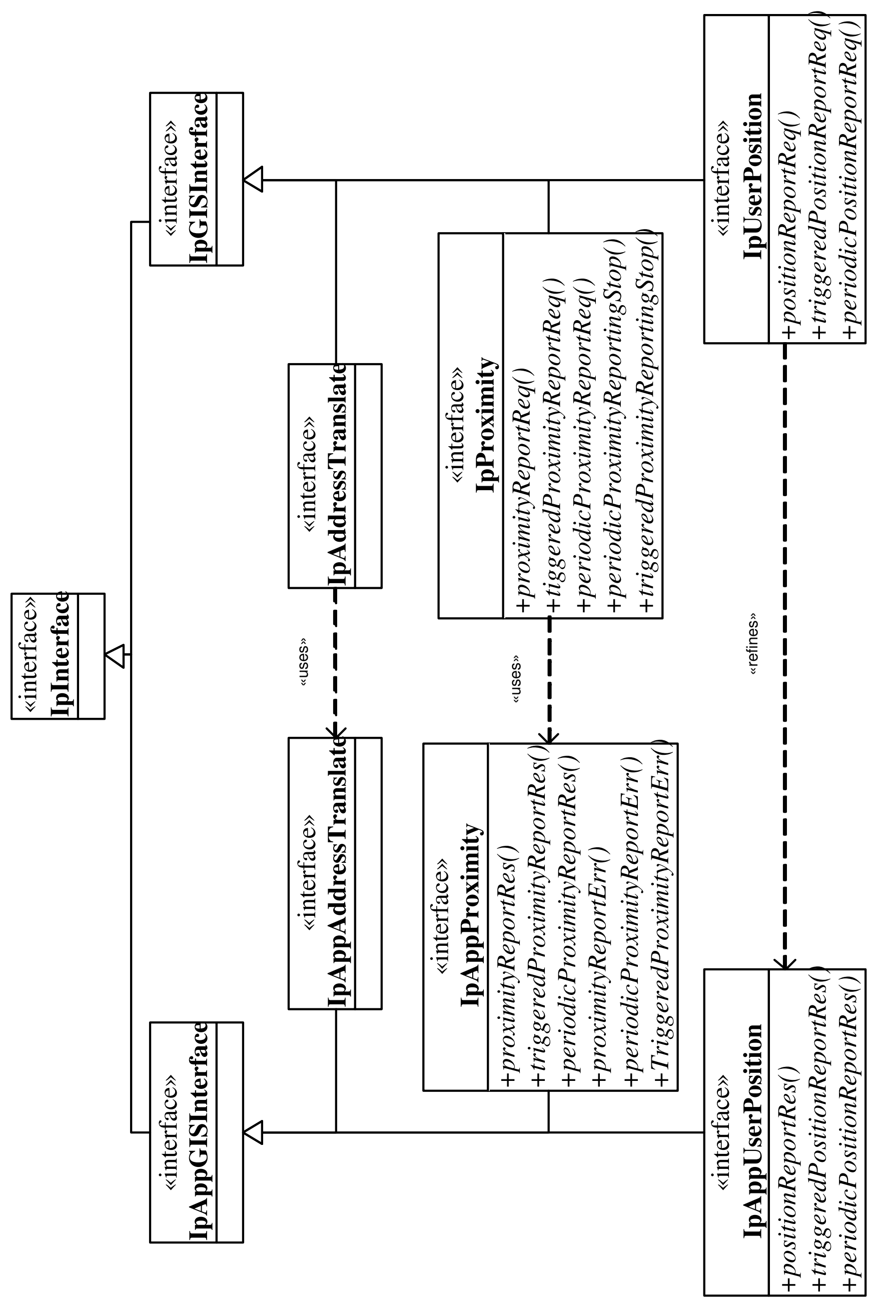


Figure 5.2: Class Diagram

The class diagram in figure 5.2 shows two sets of interface classes. These classes are grouped as *IpApp* and *Ip* interface classes. The *IpApp* classes are the application layer callback interface classes to be implemented by the service designer. The *Ip* classes are generic gateway SCF interface classes.

Functional cohesion is thus achieved by employing a UML *Using* relationship between corresponding classes. Thus the *IpProximity* interface uses the *IpAppProximity* interface, the *IpUserPosition* interface uses the *IpAppUserPosition* interface and the *IpAddressTranslate* uses the *IpAppAddressTranslate* interface.

5.4 Completeness and Primitiveness

Completeness refers to *an interface class that captures all the meaningful characteristics of the abstractions* [31]. Clearly the concept of completeness is subjective and providing all the characteristics will complicate the use of the interfaces. However, inheritance was used to implement primitive interfaces [31]. This means that existing interfaces can be extended by means of inheritance. Newer interfaces can also be defined by inheriting from the *IpInterface* interface.

Chapter 6

Conclusion

This report set out to describe the design of the Data Access SCF to be used in the Parlay/OSA gateway. The purpose of the research was not to identify all the interfaces and characteristics of their abstractions, but rather to define an infrastructure that can be easily extended to include more functionality from a set that is initially provided.

6.1 Discussion

The report outlines two business cases that necessitate the need for an architecture that allows rapid service creation. Firstly, a business case was presented, describing the architecture for the elimination of monopolies in the telecommunications environment. This way, service creation can be driven by customer demands. Secondly, for developing countries, the cost of implementing a GIS can be very high for third party ASPs. An approach was thus suggested to reduce such costs.

To satisfy the above business cases, this report proposed an architecture for Geospatial Data Access based on Parlay / OSA. The report presented the design and implementation of the Data Access SCF residing in the Parlay / OSA Gateway. To accomplish these tasks, the ICONIX design process was employed. This process is a light weight methodology based on the Unified Software Development Process [27]. The process is use-case driven, architecture centric, iterative and incremental in nature and uses the Unified Modelling Language (UML).

The Data Access architecture consists of layers with the purpose of providing necessary abstraction to develop reusable and generic applications. The application layer implements application specific logic for a particular service. Method calls are made from the application layer to the gateway layer through the Data Access SCF interfaces. The Data Access SCF offers uniform access to GIS databases residing in a lower layer. To ensure that different DBMSs can be supported, an adapter layer was introduced to adopt different function calls to different operating environments. The Data Access SCF can also make internal calls to the Mobility SCF to query for location information.

6.2 Conclusions

Chapter 1 outlined existing problems with the current telecommunications architectures. These problems present difficulties in the way applications are developed in that environment. The GDASCF was designed to address the solutions to these problems. A number of conclusions can be made about the GDASCF.

While the Parlay/OSA gateway provides a level of abstraction to the applications from the actual network transport entities, the GDASCF provides several layers of abstraction to the application layer from different geospatial databases. The result of such an approach is evident from the fact that application developers need not concentrate on the finer details of setting up a complete GIS infrastructure. Thus application vendors will be able to concentrate on developing innovative applications and services at a reduced risk and cost.

The GDASCF architecture also has benefits to large organizations that maintain GIS systems. Firstly, by removing the application development to third party organizations, they can then concentrate their time and effort on building generic and robust GIS infrastructures. Moreover, by so doing, they will be opening their infrastructure to a vast number of third party application developers. This has the advantage that more innovative applications will be developed to use such infrastructures resulting in more revenue for the business.

The design chapter outlined the key features of the Data Access SCF. The report made the important conclusion that the design process followed results in a reusable and maintainable software design. This conclusion is evident from the discussion in the critical design review chapter. Each interface class of the Data Access SCF implemented three types of method calls. These types of methods are Interactive, Periodic and Triggered calls, in accordance with the Parlay / OSA standard of the Mobility Interface. The reuse provided by Data Access SCF allows for rapid creation and delivery of reusable and maintainable services.

To show the applicability of the GDASCF, a typical service that can be implemented was chosen. The “My Nearest” type of service represents an important class of services that users with mobile units can access. Users can query for nearest restaurants, bank ATMs etc. Chapter 4 discusses in detail how the service can be implemented using the OSA / Parlay gateway with the GDASCF encapsulated. The same type of service is then implemented using the normal Remote Procedure Call (RPC) approach. Comparing the two, it can be seen that the implementations with the GDASCF provides more advantages as discussed in table 4.1.

6.3 Recommendations for Further Work

The design of the GDASCF has highlighted the need for the adapter interface. This is necessary since different gateway vendors will be forced to develop the GDASCF against the same set of APIs everytime. While the GDASCF must be considered part of the OSA / Parlay gateway, it must be noted that the Adapter layer resides outside the gateway. However, standardizing the Adapter interfaces will ensure that database vendors need only provide adapters supporting the same set of interface.

Since the adapter layer may reside in a separate business domain, different adapters will be provided by different vendors. Similar to the OSA / Parlay gateway, there will be cases when the adapters performing the same task are deployed. Access to the adapters will thus need to be controlled. The most important extension is thus to extend the Adapter layer to include interfaces for the Registry, which will be used to register adapters and their capabilities similar to the OSA / Parlay Framework. The Registry interface is shown explicitly in the Sequence Diagrams in chapter 3. It is also recommended that the proposed data access architecture be adopted by the OSA / Parlay standard and the Data Access SCF be included as an integral part of the OSA / Parlay gateway.

The design of the GDASCF concentrated mostly on the access of applications to geospatial data. For the GIS infrastructure to be open, there needs to be a access control, accounting and subscription interfaces. It is therefore recommended that further research be done on the design of such interfaces to reside in the GDASCF.

References

[1] D. Schmidt, M. Stal, H. Rohnert, and F. Buschman, *Pattern-Oriented Software Architecture, Patterns for Concurrent and Networked Objects*, vol. 2. Chichester, England: John Wiley, September 2000. ISBN: 0-471-60695-2.

[2] E. Gamma and R. Helm and R. Johnson and J. Vlissides, *Design Patterns: Elements of Reusable Object-Oriented Software*. Indiana, U.S.A.: Addison Wesley Professional, October 1994. ISBN: 0201633612.

[3] T. Magendanz and R. Popescu-Zeletin. International Thomson Computer Press.

[4] A. Gillwald and S. Kane Edited by C. Armstrong, "South African Telecommunications Sector Performance Review," *University of the Witwatersrand, LINK Research Agenda, 2002/3 Policy Research Paper No 5, http://www.researchictafrica.net/images/upload/sector-perfomance-rev.pdf, Last Accessed 15 August 2005*, August 2003.

[5] L. Correa, "Natural or Unnatural Monopolies In UK Telecommunications ?," *University of London, Economics Working Paper No. 501, http://www.econ.qmul.ac.uk/papers/doc/wp501.pdf, Last Accessed 11 August 2005*. ISSN 1473-0278.

[6] C. S. Allen and A. L. Fletcher, "From Alte Tante to European Macher: German Telecommunications in the Global Economy," *A Publication of the American Institute for Contemporary German Studies, http://www.aicgs.org/Publications/PDF/allenfletcher.pdf, Last Accessed 15 August 2005*, March 1999.

[7] R. Kruger and H. Mellein. Rohde and Schwartz, Munchen, Germany.

[8] R. R. Bhat and R. Gupta, “JAIN Protocol APIs,” *IEEE Communications Magazine*, vol. 38, no. 1, pp. 100–107, January 2000.

[9] J. de Keijzer, D. Tait, and R. Goedman, “JAIN: A New Approach to Services in Communication Networks,” *IEEE Communications Magazine*, vol. 38, no. 1, pp. 94–99, January 2000.

[10] L. Rising, *Design Patterns In Communications Software.* New York, USA: Press Syndicate of the University of Cambridge, 2001. ISBN-10: 0521790409.

[11] S. S. Yau and F. Karim, “A Lightweight Middleware Protocol for Ad Hoc Distributed Object Computing in Ubiquitous Computing Environments,” in *Proceedings on the 6th IEEE International Symposium on Object-Oriented Real-Time Distributed Computing (ISORC 2003)*, pp. 172–179, May 14-16 2000.

[12] S. Vinoski, “CORBA: Integrating Diverse Applications Within Distributed Heterogeneous Environments,” *IEEE Communications Magazine*, vol. 35, no. 2, February 1997.

[13] C. Egelhaaf, P. FitzPatrick, P. Loosemore, O. Risnes, and F. Stoinski, “Middleware for telecommunications,” in *European Institute for Research and Strategic studies in Telecommunications*, (http://www.eurescom.de/ pub/deliverables/documents/P900-series/P910/P910_brochure.pdf, Last Accessed 01 August 2005), EURESCOM Project P910, Technology Assessment of Middleware for Telecommunications, February 2001.

[14] B. Dwolatzky and R. van Olst, “A Mobile Geographic Data Capturing Device for Use By Unskilled Operators,” (http://synergy.intergraph.com/catalogfiles/documents/1627.pdf, Last Accessed 18 August 2005), University of the Witwatersrand, Johannesburg, South Africa, 2003.

[15] W. Lara, Senior Software Engineer, “Universal Location Framework: A New Wireless Building Block,” *IntelDeveloperUPDATEMagazine, http://www.intel.com/technology/magazine/communications/wi02031.pdf, Last Accessed 27 July 2005*, February.

[16] L. J. Pinson and R. S. Wiener, *An Introduction to Object-Oriented Programming and Smalltalk*. Colorado, Springs: Addison-Wesley, 1988. ISBN: 0-201-19127-X.

[17] K. Kwan and W. Shi, "A Study of Dynamic Database in Mobile GIS," in *Proceedings of the ISPRS Technical Commission IV Symposium: Symposium on Geospatial theory, Processing and Applications*, (Ottawa, Canada), July 09-12 2002.

[18] S. Beddus, G. Bruce, and S. Davis, "Opening Up Networks with JAIN Parlay," *IEEE Communications Magazine*, vol. 38, no. 4, pp. 136–143, April 2000.

[19] M. Henning and S. Vinoski, *Advanced CORBA programming with C++*. Addison-Wesley, 1999.

[20] A. Moerdijk and L. Klosterman, "Opening the networks with Parlay/OSA: Standards and Aspects Behind the APIs," *IEEE Communications Magazine*, vol. 17, pp. 58–64, May/June 2003.

[21] 3rd Generation Partnership Project, "Customised applications for mobile network enhanced logic service description," *Stage 1 3G TS 22.078 version 3.4.1 Release 1999, http://webapp.etsi.org/exchangefolder/ts_122078v030401p.pdf Last Accessed 11 August 2005*, 2000-03.

[22] R. Corin, G. D. Caprio, S. Etalle, S. Gnesi, G. Lenzini, and C. Moiso, "Security Analysis of Parlay/OSA Framework," in *7th IFIP WG 6.1 International Conference on Formal Methods for Open Object-Based Distributed Systems (FMOODS)*, (Athens, Greece), pp. 131–146, Springer-Verlag, June 2005.

[23] A. O. Freier, P. Karlton, and P. C. Kocher, "The SSL Protocol Version 3.0," (http://wp.netscape.com/eng/ssl3/draft302.txt, Lasts Accessed 11 August 2005), November 18.

[24] M. R. Ebling, G. D. H. Hunt, and H. Lei, "Issues for Context Services for Pervasive Computing," in *Proceedings Workshop on Middleware for Mobile Computing, IFIP/ACM Middleware 2001*, (Heidelberg, Germany), November 2001. http://www.cs.arizona.edu/mmc/13%0Ebling.pdf, Last Accessed 18 August 2005.

[25] 3rd Generation Partnership Project, "Open Services Access, Application Programming Interface, Mobility SCF," *3GPP TS 29.198-6 version 4.3.0 Release 4 ETSI TS 129 198-6 V4.3.0 (2001-12),*

http://webapp.etsi.org/exchangefolder/ts_12919806v040300p.pdf Last Accessed 11 August 2005, 2002.

[26] D. Rosenberg, M. Collins-Cope, and M. Stephens, *Agile Developmet with ICONIX Process: People, Process, and Pragmatism*. Apress L.P, January 1999. ISBN: 0201571692.

[27] I. Jacobson, G. Booch, and J. Rumbaugh, *The Unified Software Development Process*. Addison-Wesley, January 1999. ISBN: 0201571692.

[28] K. Raymonds, "Reference Model of Open Distributed Processing RM-ODP:Introduction," in *International Conference On Open Distributed Computing 1995 (ICODP'95)*, (http://www.dstc.edu.au/Research/Projects/ODP/papers/icodp95.slides.ps, Last Accessed 01 August 2005), February 20 1995.

[29] The Parlay Group., *"Open Services Access, Application Programming Interface, Mobility SCF"*., etsi es 202 915-6 v1.1.1 (2003-01), ed.

[30] L. Wirzenius, "Kannel architecture and design - revision: 1.20," Master's thesis, Wapit Ltd, http://www.kannel.org/kannel-arch-snapshot/arch.html#EN16, Last Accessed 01 August 2005. Work-in-progress for Masters Thesis.

[31] G. Booch, *Object Oriented Design with Applications*. Benjamin/Cummings. ISBN: 0805353402.

Appendix A

Robustness Diagrams

A.1 Establish User Position

The following robustness diagrams correspond to Sequence diagrams in chapter 3.3.3

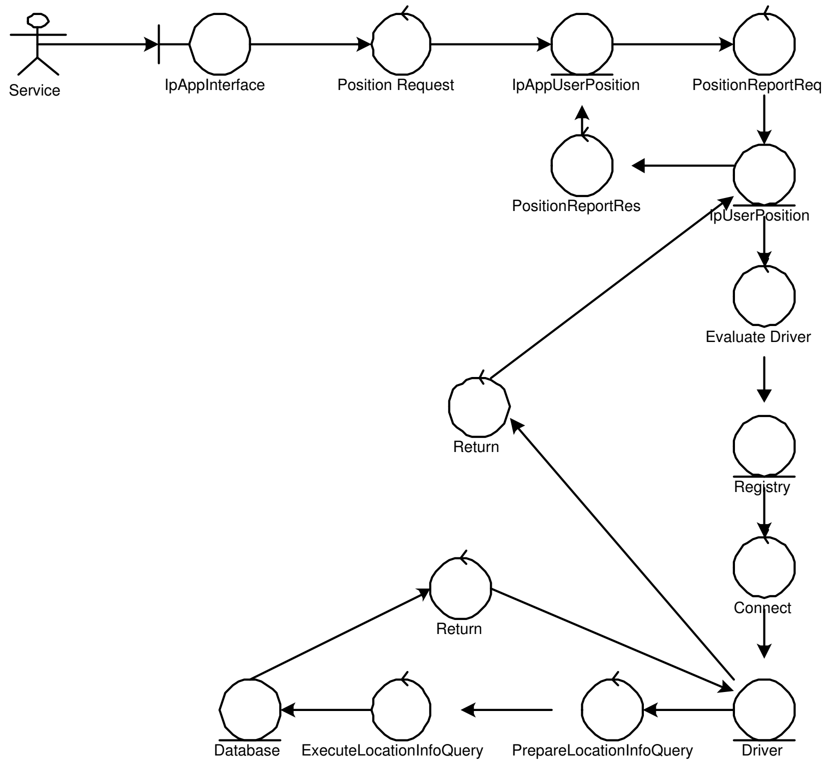


Figure A.1: Establish User Position - Interactive Request

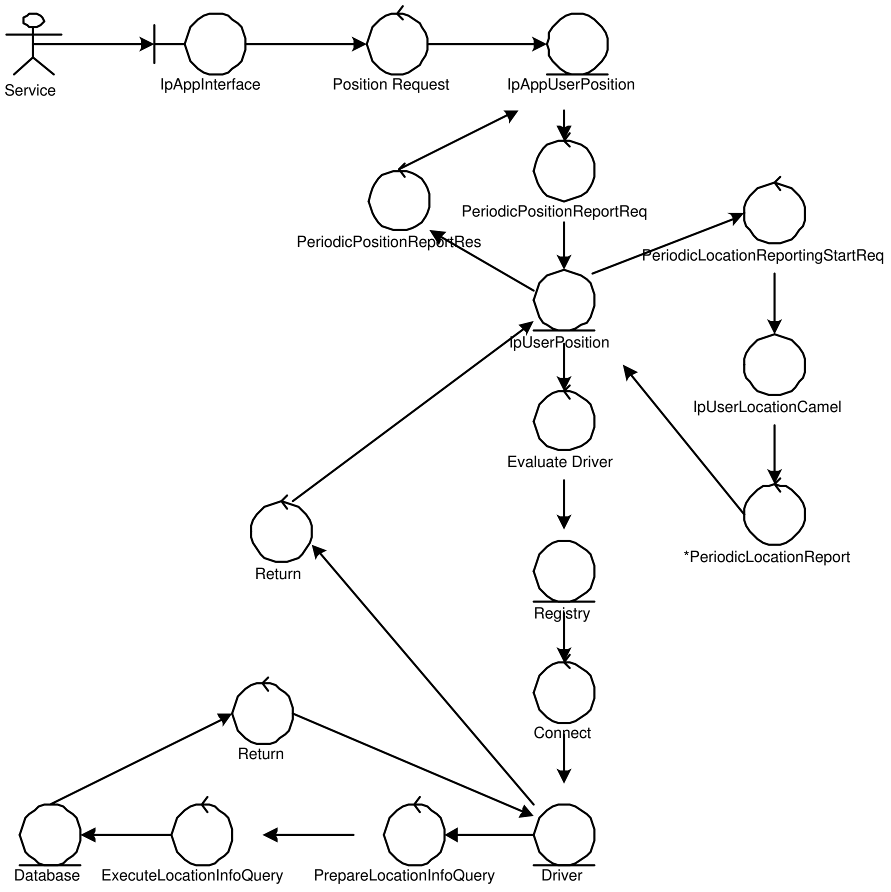


Figure A.2: Establish User Position - Periodic Request

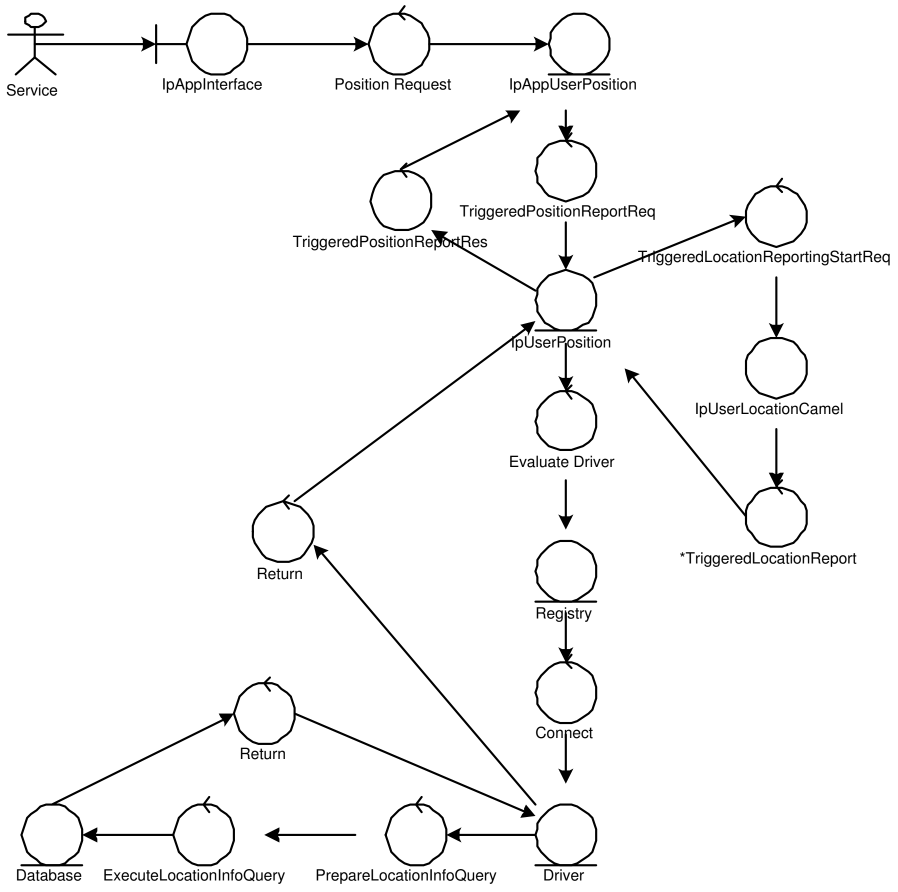


Figure A.3: Establish User Position - Triggered Request

A.2 Proximity To A Region

The following robustness diagrams correspond to Sequence diagrams in chapter 3.3.3.

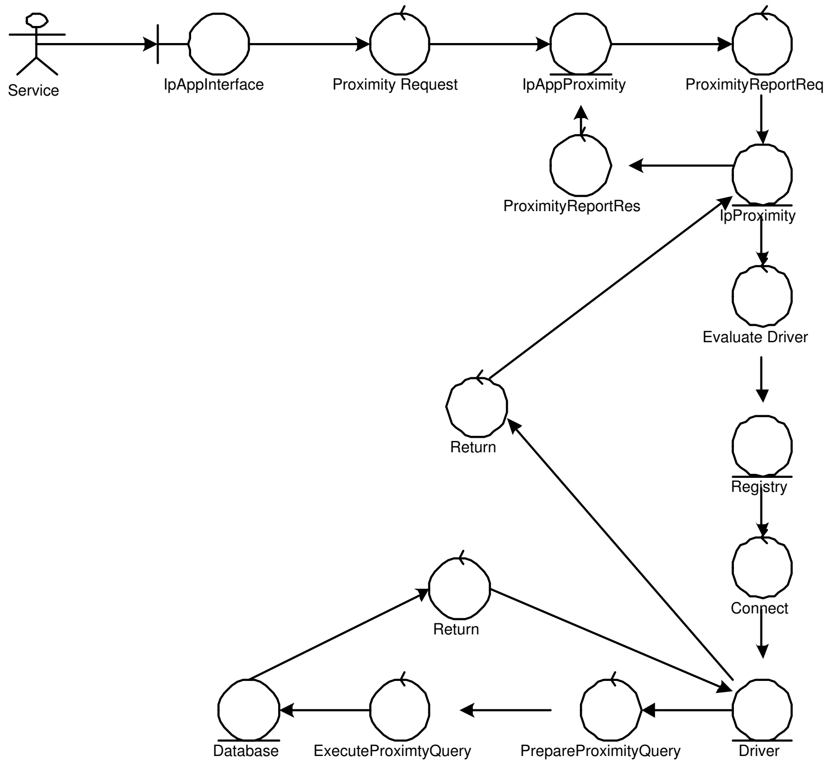


Figure A.4: Proximity - Interactive Request

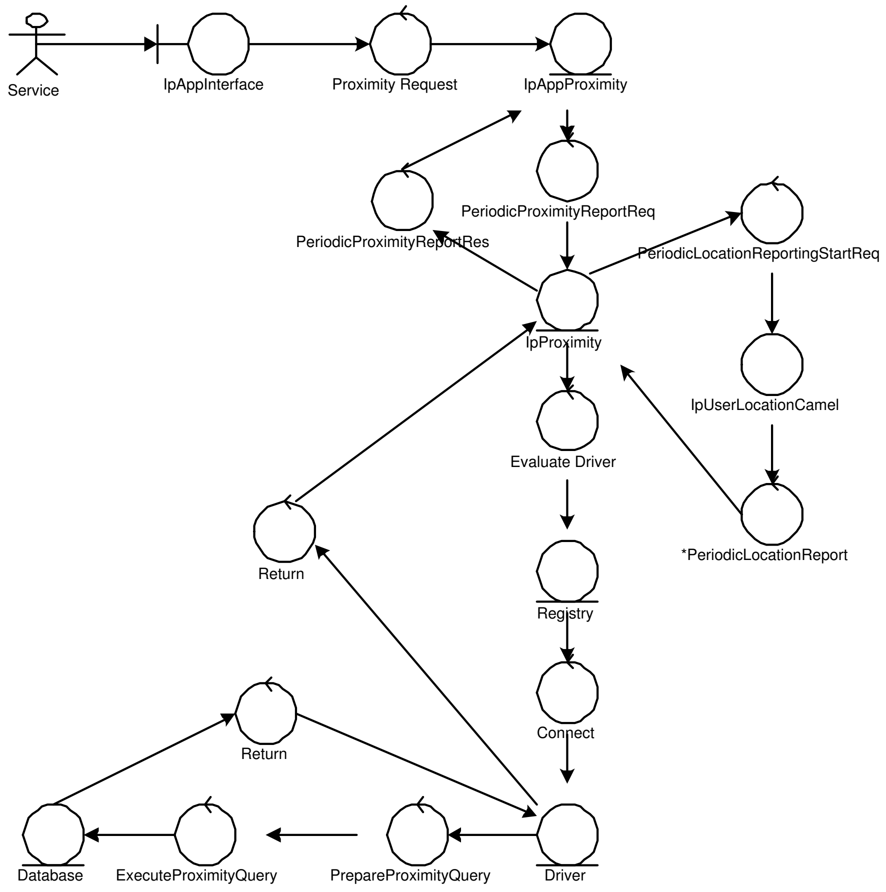


Figure A.5: Proximity - Periodic Request

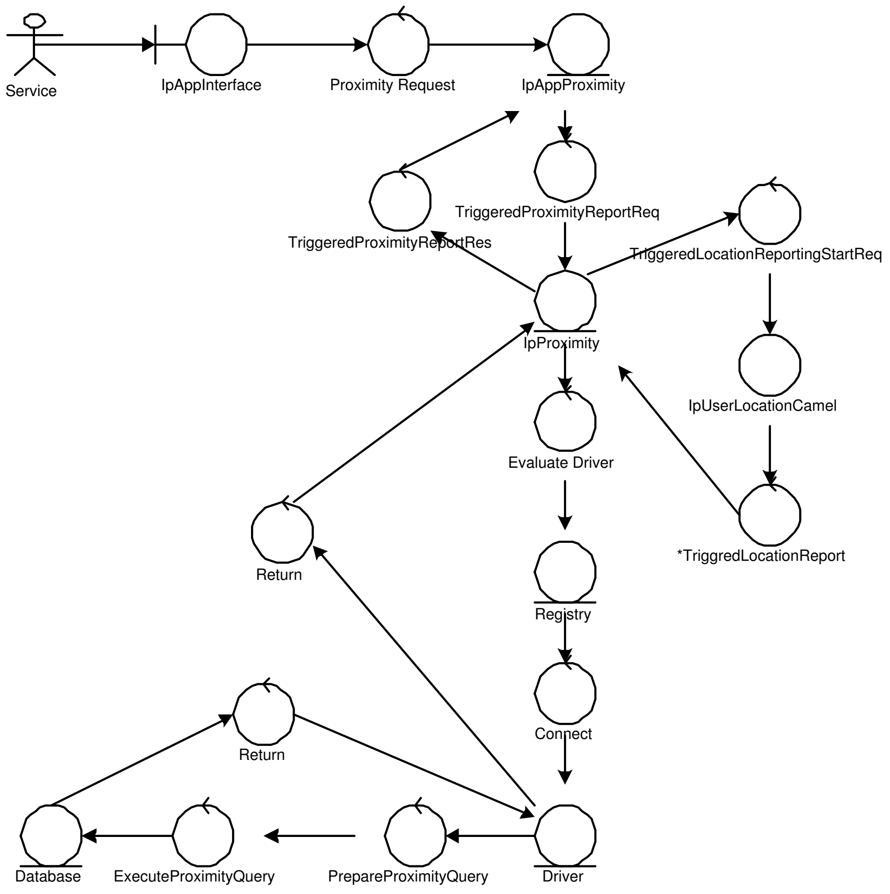


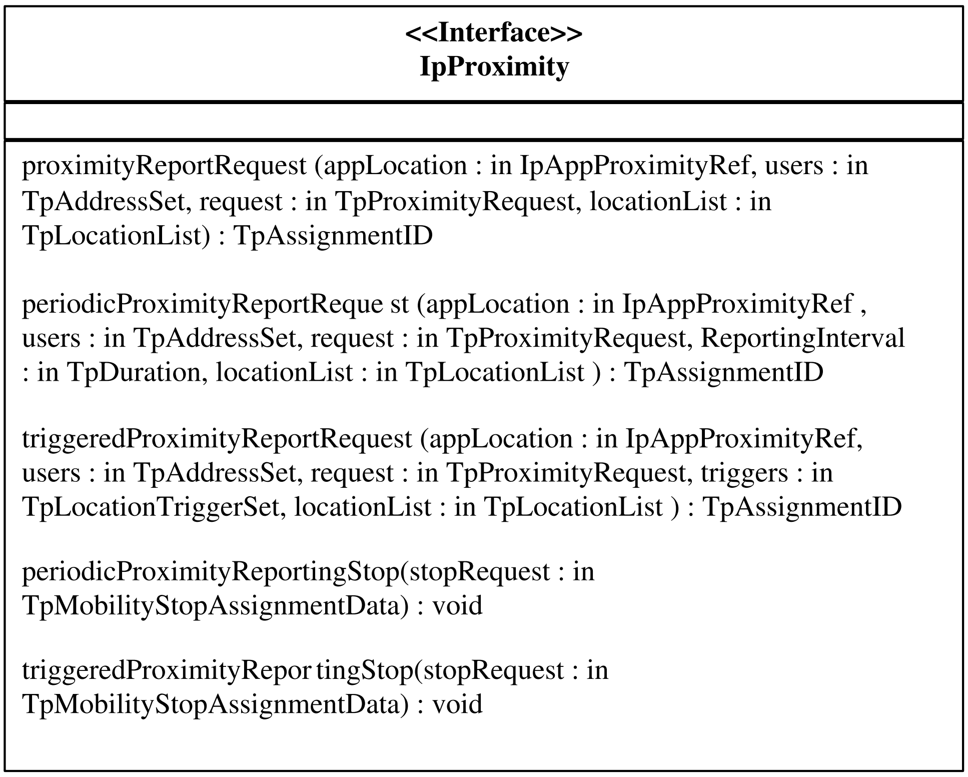
Figure A.6: Proximity - Triggered Request

Appendix B

Interface Class Definitions

B.1 Interface Class for IpProximity

The *IpProximity* Interface class inherits from *IpGISInterface* which in turn inherits from *IpService*. Note that the OSA / Parlay standard requires that each interface inherits from *IpService*.



<<Interface>>
IpProximity

proximityReportRequest (appLocation : in IpAppProximityRef, users : in TpAddressSet, request : in TpProximityRequest, locationList : in TpLocationList) : TpAssignmentID

periodicProximityReportReque st (appLocation : in IpAppProximityRef , users : in TpAddressSet, request : in TpProximityRequest, ReportingInterval : in TpDuration, locationList : in TpLocationList) : TpAssignmentID

triggeredProximityReportRequest (appLocation : in IpAppProximityRef, users : in TpAddressSet, request : in TpProximityRequest, triggers : in TpLocationTriggerSet, locationList : in TpLocationList) : TpAssignmentID

periodicProximityReportingStop(stopRequest : in TpMobilityStopAssignmentData) : void

triggeredProximityRepor tingStop(stopRequest : in TpMobilityStopAssignmentData) : void

Figure B.1: Interface Class IpProximity

Method proximityReportRequest()

Parameters

appLocation : in IpAppProximityRef

Specifies the application interface for callbacks from the Proximity service.

users : in TpAddressSet

Specifies the user(s) for which the location shall be reported.

request : in TpProximityRequest

Specifies among others the requested location type, accuracy time, and priority

locationList : in TpLocationList

Specifies the list of locations for which proximity must be reported. This parameter is a list of strings specifying the names of locations that reporting should be done.

Returns

TpAssignmentID

Raises

TpCommonExceptions, P_APPLICATION_NOT_ACTIVATED, P_REQUEST_ACCURACY_CANNOT_BE_DELIVERED, P_REQUEST_RESPONSE_TIME_CANNOT_BE_ DELIVERED, P_INFORMATION_NOT_AVAILABLE, P_INVALID_INTERFACE_TYPE, P_CANNOT_CONNECT_TO_DATABASE, P_LOCATION_LIST_IS_EMPTY.

B.1.1 Method periodicProximityReportRequest()

Parameters

appLocation : in IpAppProximityRef

Specifies the application interface for callbacks from the Proximity service.

users : in TpAddressSet

Specifies the user(s) for which the location shall be reported.

request : in TpProximityRequest

Specifies among others the requested location type, accuracy time and priority

ReportingInterval : in TpDuration

Specifies the requested interval in seconds between the reports

locationList : in TpLocationList

Specifies the list of locations for which proximity must be reported

Returns

TpAssignmentID

Raises

TpCommonExceptions, P_INVALID_REPORTING_INTERVAL, P_APPLICATION_NOT_ACTIVATED, P_REQUEST_ACCURACY_CANNOT_BE_DELIVERED, P_REQUEST_RESPONSE_TIME_CANNOT_BE_ DELIVERED, P_INFORMATION_NOT_AVAILABLE, P_INVALID_INTERFACE_TYPE, P_CANNOT_CONNECT_TO_DATABASE, P_LOCATION_LIST_IS_EMPTY.

B.1.2 Method triggeredProximityReportRequest()

Parameters

appLocation : in IpAppProximityRef

Specifies the application interface for callbacks from the Proximity service.

users : in TpAddressSet

Specifies the user(s) for which the location shall be reported.

request : in TpProximityRequest

Specifies among others the requested location type, accuracy time and priority

triggers : in TpLocationTriggerSet

Specifies the trigger conditions

locationList : in TpLocationList

Specifies the list of locations for which proximity must be reported

Returns

TpAssignmentID

Raises

TpCommonExceptions, P_TRIGGER_CONDITIONS_NOT_SUBSCRIBED, P_INVALID_REPORTING_INTERVAL, P_APPLICATION_NOT_ACTIVATED, P_REQUEST_ACCURACY_CANNOT_BE_DELIVERED, P_REQUEST_RESPONSE_TIME_CANNOT_BE_DELIVERED, P_INFORMATION_NOT_AVAILABLE, P_INVALID_INTERFACE_TYPE, P_CANNOT_CONNECT_TO_DATABASE, P_LOCATION_LIST_IS_EMPTY.

B.1.3 Method periodicProximityReportingStop()

Parameters

stopRequest : in TpMobilityStopAssignmentData

Specifies how the assignment shall be stopped, i.e. if whole or just parts of the assignment should be stopped.

Raises

TpCommonExceptions, P_INVALID_ASSIGNMENT_ID

B.1.4 Method triggeredProximityReportingStop()

Parameters

stopRequest : in TpMobilityStopAssignmentData

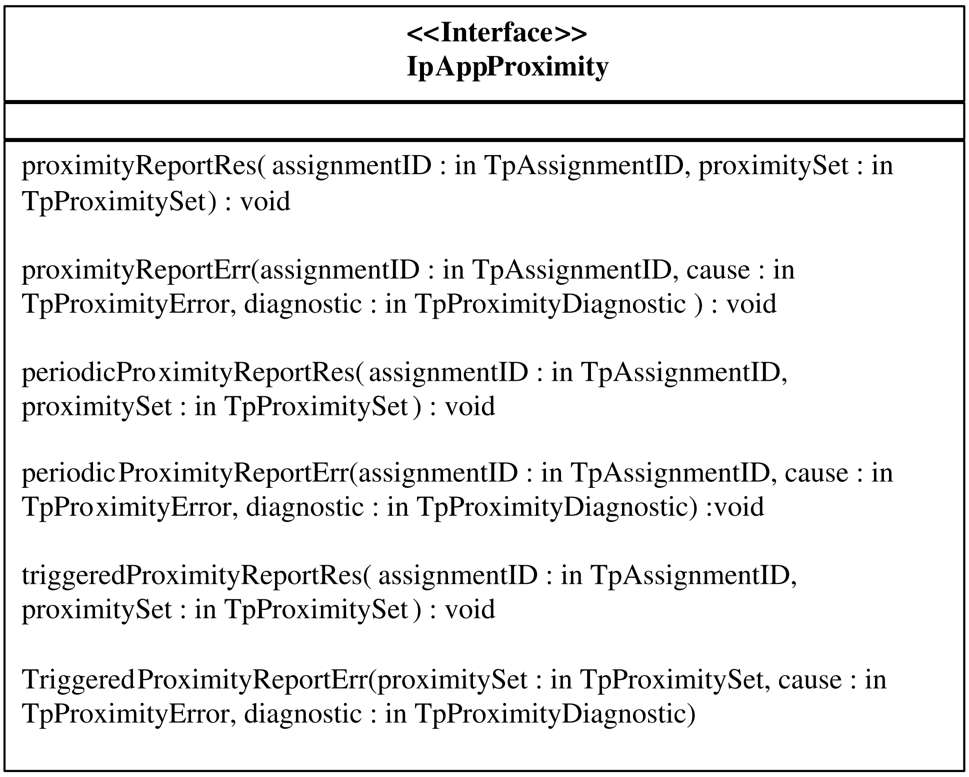
Specifies how the assignment shall be stopped, i.e. if whole or just parts of the assignment should be stopped.

Raises

TpCommonExceptions, P_INVALID_ASSIGNMENT_ID

B.2 Interface Class for IpAppProximity

The *IpAppProximity* Interface class inherits from *IpAppGISInterface* which in turn inherits from *IpService*. Note that the standard of Parlay requires that each interface inherits from *IpService*.



<<Interface>>
IpAppProximity

proximityReportRes(assignmentID : in TpAssignmentID, proximitySet : in TpProximitySet) : void

proximityReportErr(assignmentID : in TpAssignmentID, cause : in TpProximityError, diagnostic : in TpProximityDiagnostic) : void

periodicProximityReportRes(assignmentID : in TpAssignmentID, proximitySet : in TpProximitySet) : void

periodicProximityReportErr(assignmentID : in TpAssignmentID, cause : in TpProximityError, diagnostic : in TpProximityDiagnostic) :void

triggeredProximityReportRes(assignmentID : in TpAssignmentID, proximitySet : in TpProximitySet) : void

TriggeredProximityReportErr(proximitySet : in TpProximitySet, cause : in TpProximityError, diagnostic : in TpProximityDiagnostic)

Figure B.2: Interface Class IpAppProximity

B.2.1 Method proximityReportRes()

Parameters

assignmentID : in TpAssignmentID

Specifies the assignment ID of the proximity report request.

proximitySet : in TpProximitySet

Specifies the location(s) of one or several location. This parameter is similar to TpLocationList. As a matter of fact, it can be defined as

Typedef TpLocationList TpProximitySet

Returns

Void

B.2.2 Method periodicProximityReportRes()

Parameters

assignmentID : in TpAssignmentID

Specifies the assignment ID of the failed location report request.

proximitySet : in TpProximitySet

Specifies the location(s) of one or several location. This parameter is similar to TpLocationList. As a matter of fact, it can be defined as

Typedef TpLocationList TpProximitySet

Returns

Void

B.2.3 Method triggeredProximityReportRes()

Parameters

assignmentID : in TpAssignmentID

Specifies the assignment ID of the failed location report request.

proximitySet : in TpProximitySet

Specifies the location(s) of one or several location. This parameter is similar to TpLocationList. As a matter of fact, it can be defined as

Typedef TpLocationList TpProximitySet

Returns

Void

B.2.4 Method proximityReportErr()

Parameters

assignmentID : in TpAssignmentID

Specifies the assignment ID of the failed location report request.

cause : in TpProximityError

Specifies the error that led to the failure.

diagnostic : in TpProximityDiagnostic

Specifies additional information about the error that led to the failure.

Returns

Void

B.2.5 Method periodicProximityReportErr()

Parameters

assignmentID : in TpAssignmentID

Specifies the assignment ID of the failed location report request.

cause : in TpProximityError

Specifies the error that led to the failure.

diagnostic : in TpProximityDiagnostic

Specifies additional information about the error that led to the failure.

Returns

Void

B.2.6 Method TriggeredProximityReportErr()

Parameters

assignmentID : in TpAssignmentID

Specifies the assignment ID of the failed location report request.

cause : in TpProximityError

Specifies the error that led to the failure.

diagnostic : in TpProximityDiagnostic

Specifies additional information about the error that led to the failure.

Returns

Void

Appendix C

IDL specifications for the Geospatial Data Access Service Capability Feature

C.1 CommonDataDefinitions.IDL

```
/**************************************************************
 * The C++ code MAY NOT compile if the IDL definition contains
 *variables that are aliased multiple times. E.G. the TpInt32
 *definition below is aliased multiple times. If this is a
 *problem, this multiple aliases must be removed from the IDL
 *file.
 *************************************************************/

typedef long TpInt32;
typedef TpInt32 TpAssignmentID;

typedef long IpAppProximityCamelRef;
typedef string TpString;
```

```
/*******************************************************
 *The following enumerated types are as defined in the
 *OSA / Parlay Common Data Definitions specification.
 *******************************************************/

enum TpAddressPlan {
    P_ADDRESS_PLAN_NOT_PRESENT,
    P_ADDRESS_PLAN_UNDEFINED,
    P_ADDRESS_PLAN_IP,
    P_ADDRESS_PLAN_MULTICAST,
    P_ADDRESS_PLAN_UNICAST,
    P_ADDRESS_PLAN_E164,
    P_ADDRESS_PLAN_AESA,
    P_ADDRESS_PLAN_URL,
    P_ADDRESS_PLAN_NSAP,
    P_ADDRESS_PLAN_SMTP,
    P_ADDRESS_PLAN_MSMAIL,
    P_ADDRESS_PLAN_X400,
    P_ADDRESS_PLAN_SIP,
    P_ADDRESS_PLAN_ANY,
    P_ADDRESS_PLAN_NATIONAL
};

enum TpAddressPresentation {
    P_ADDRESS_PRESENTATION_UNEFINED,
    P_ADDRESS_PRESENTATION_ALLOWED,
    P_ADDRESS_PRESENTATION_RESTRICTED,
    P_ADDRESS_PRESENTATION_ADDRESS_NOT_AVAILABLE
};

enum TpAddressScreening {
    P_ADDRESS_SCREENING_UNDEFINED,
    P_ADDRESS_SCREENING_USER_VERIFIED_PASSED,
    P_ADDRESS_SCREENING_USER_NOT_VERIFIED,
    P_ADDRESS_SCREENING_USER_VERIFIED_FAILED,
    P_ADDRESS_SCREENING_NETWORK
};

/*************************************************************
 *The struct below according to the OSA / Parlay Specification
 *Common Data Definitions, page 9.
 ************************************************************/
struct TpAddress
{
    TpAddressPlan       plan;
    TpString        AddrString;
    TpString        Name;
    TpAddressPresentation   Presentation;
    TpAddressScreening  Screening;
    TpString        SubAddressString;
} ;//TpAddress;

struct TpAddressSet
{
    TpInt32 Number;
    TpAddress Set[100];
} ;//TpAddressSet;
```

C.2 IpProximity.IDL

```
 interface IpProximity {
    TpAssignmentID proximityReportReq
           (in IpAppProximityRef appLocation,
            in TpAddressSet users,
            in TpProximityRequest request,
            in TpLocationList locationList );
```

```
     TpAssignmentID triggeredProximityReportReq
           (in IpAppProximityRef appLocation,
            in TpAddressSet users,
            in TpProximityRequest request,
            in TpLocationTriggerSet triggers,
            in TpLocationList  locationList );


    TpAssignmentID periodicProximityReportReq
           (in IpAppProximityRef  appLocation,
            in TpAddressSet users,
            in TpProximityRequest request,
            in TpDurationReportingInterval,
            in TpLocationList  locationList ) ;

    void triggeredProximityReportStop
           (in TpMobilityStopAssignmentData stopRequest) ;
};
```

C.3 IpAppProximity.IDL

```
interface IpProximity {
    void proximityReportRes(
             in TpAssignmentID assignmentID,
             in TpProximitySet proximitySet);

    void proximityReportErr(
             in TpAssignmentID assignmentID,
             in TpProximityError cause,
             in TpProximityDiagnostic diagnostic);
    void periodicProximityReportRes(
            in TpAssignmentID assignmentID,
            in TpProximitySet proximitySet);


    void periodicProximityReportErr(
            in TpAssignmentID assignmentID,
            in TpProximityError cause,
            in TpProximityDiagnostic diagnostic);


    void triggeredProximityReportRes(
            in TpAssignmentID assignmentID,
            in TpProximitySet proximitySet);


    void triggeredProximityReportErr(
            in TpProximitySet proximitySet,
            in TpProximityError cause,
            in TpProximityDiagnostic diagnostic);
};
```