THE DEVELOPMENT OF A STRUCTURED APPROACH TO SERVICE PROVISIONING IN A PARLAY ENVIRONMENT

Barry Fricke

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

Johannesburg, 2007

DECLARATION

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

_____

Barry Fricke

___ day of _____ _____

ABSTRACT

The environment in which services are provisioned in existing networks has a number of shortcomings. Neither the service domain nor the services therein have a standardised structure. Signalling between terminals and services uses network protocols that are inappropriately oriented towards bearer management. The control of bearer connections, and the view of call states, is maintained in the network layer, making bearer management difficult and limited.

A service-centric service provisioning environment is proposed, which advocates a structured service domain, and a structured approach to service development and provisioning. A direct communication path between terminals and services at the application layer, that utilises high-level, service-oriented protocols, is proposed. Control of the call / session layer and the bearer network, and view of connection states is relocated to the application layer, facilitating bearer manipulation by services located in the service domain.

It is shown that the capabilities and features of services provisioned in the proposed service provisioning environment are of a greater range, more advanced and more complex. It is also shown that the proposed service provisioning environment brings about potential efficiency gains for the initiation of 2-party calls, and significant efficiency gains for the initiation of multiparty calls.

ACKNOWLEDGEMENTS

This work was performed under the Centre for Telecoms Access and Services at the University of the Witwatersrand. The centre is funded by Telkom SA Limited, Siemens Telecoms and the Department of Trade and Industry's THRIP programme. This financial support is much appreciated.

I would like to thank my supervisor, Prof. H.E. Hanrahan for his guidance and support throughout the duration of the research project. Special thanks go to the Telkom Center of Excellence (CoE) program for providing me with this exciting opportunity.

CONTENTS

LIST OF FIGURES

LIST OF TABLES

NOMENCLATURE

Acronym	Definition
API	Application Programming Interface
AS	Application Signalling
BCS	Basic Connectivity Service
CC	Call Control
CCF	Call Control Function
CSCF	Call Session Control Function
CSN	Circuit Switched Network
DTMF	Dual Tone Multi Frequency
GSM	Generic Service Module
HSS	Home Subscriber Server
I-CSCF	Interrogating - CSCF
IMS	IP Multimedia Subsystem
ID	Interaction Diagram
IN	Intelligent Network
INAP	Intelligent Network Application Protocol
ISDN	Integrated Services Digital Nework
IT	Information Technology
LAN	Local Area Network
MMS	Multimedia Message Service
MPCC	Multi Party Call Control
MPLS	Multi Protocol Label Switching
MSC	Message Sequence Chart
NS	Network Signalling
OSA	Open Services Architecture
P-CSCF	Proxy - CSCF
PSN	Packet Switched Network
PSTN	Public Switched Telephone Network
QoS	Quality of Service
R&A	Reuse and Abstraction
RBB	Reusable Building-Block
RC	Resource Control
S-CSCF	Serving - CSCF
SCF	Service Capability Feature
SIP	Session Initiation Protocol
SMS	Short Message Service
VPN	Virtual Private Network
WAN	Wide Area Network

Chapter 1

INTRODUCTION

1.1. Overview of communications networks

A communications network is an organisation of stations capable of interconnection [1]. Many different types of communications networks exist, each interfacing with its users in a different way, and providing a different service to its users. The following are examples of different types of communications networks [2]:

- Fixed-line telecoms networks
    - PSTN (Public Switched Telephone Network)
    - ISDN (Integrated Services Digital Network)
- Packet switched networks
    - IMS (IP Multimedia Subsystem) networks
    - MPLS (Multiprotocol Label Switching) networks
- Mobile telecoms networks
    - Cellular (2G, 2.5G, 3G) networks
    - Radio networks
- Enterprise networks
    - LAN (Local Area Network), WAN (Wide Area Network)
    - VPN (Virtual Private Network)
- Internet

Since each of these networks has different objectives, their architectures differ significantly. However, all communications networks have a few, common, fundamental elements. All communications networks have a transport mechanism, terminals which interface with the network's users, services which the network offers to its users, and network management systems. In their essence, communications are more similar than they are different.

This research project is primarily centred on telecoms networks. However, due to the stated similarities between communications networks of all kinds, many of the results presented in this report apply equally to networks other than telecoms networks.

This fact is especially true due to the primary focus of the project: the service domain. As will be shown later in this chapter, softswitch technologies and the Open Service Access (OSA) principle allow the separation of application services from the underlying network. Therefore, many of the developments made to the service domain in this project are independent of the underlying network which it serves.

1.2. Telecoms networks

The generic description of communications networks are specialised to the case of telecoms networks in the following ways:

- Telecoms network end users are people, who interface with the network using a small, portable terminal (a telephone with a growing set of features).
- Telecoms network users predominantly communicate with other network users in real time, using a combination of voice and video. Additionally, network users are able to communicate using data, e.g. Short Message Service (SMS), Multimedia Message Service (MMS) etc.

The telecoms network has undergone significant changes since its inception, and continues to evolve. These changes have been necessitated by the continual advancement of the services the network aims to support, and this project report aims to further the network's metamorphosis to this same end.

1.2.1. Telecoms services

The historical approach used to provision services for fixed-line telecoms networks is to have the services distributed throughout the network domain in various network elements.

This approach is currently being phased out in telecoms networks due to the following problems:

- Service implementation and alteration is a significant logistical exercise since the services are duplicated in various elements throughout the network, making service deployment and modification necessary in a number of different locations.
- Services are not easily transportable to different networks since they are highly dependent on the network technology and are thus inextricably tied to the underlying network.
- Since services are so dependent on the technology of their underlying network, service programmers are required to have a detailed understanding of both application development and telecoms networks operations and protocols.

1.2.2. The development of the service domain

The increasing complexity of services rendered the ad hoc approach of implementing services in a distributed manner throughout the network domain prohibitively difficult, as well as clumsy. Additionally, the requirement that service developers have a good understanding of telecoms networks, and their myriad protocols, precluded many application developers from telecoms service development. A new approach to service implementation was thus required.

In the approach presently being pursued by network operators, which developed from the recognition of these problems, services are physically removed from their multiple locations throughout the network domain, and relocated to a logically and physically distinct element. This element became known as the ‘service domain’, and is thought of as occupying a completely different domain to the network, as shown in figure 1.1.

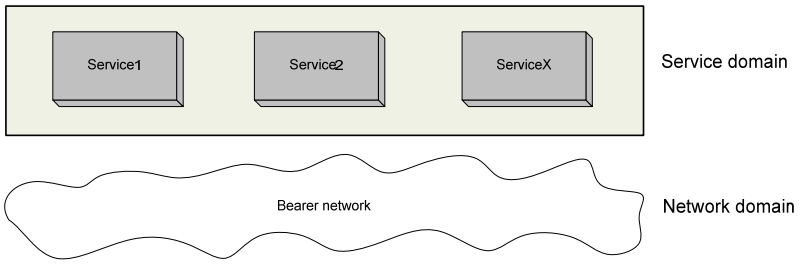


Figure 1-1: The service and network domains

The service domain contains the services in a centralised location; these services were distributed throughout the network domain before the advent of the service domain. Note that, despite the relocation of services from the network domain to the service domain, the control and management of the fundamental and original service of providing end-to-end connections between users is still maintained in the network domain in current network architectures. (Specifically, the service is located in a layer of the network domain referred to as the "call / session layer", which is responsible for the control of media connections and streams in the bearer network. The call / session layer is more fully defined in chapter 3.) The service of providing end-to-end connections between users is referred to as the Basic Connectivity Service (BCS) in this report.

The introduction of the service domain to the telecoms network, and the relocation of the services to the service domain, requires an adjustment to the way services are invoked. When services were distributed throughout the network domain, terminals signalled to the call / session layer of the network for the required service. With the relocation of services to the service domain, this approach is no longer adequate.

Telecoms networks allow the new approach of having services located in the service domain through an ad hoc adjustment: the call / session layer of the network forwards the service signalling to the new location of the services- the service domain. The result of this ad hoc modification is that even though the services have been removed from the network domain, terminals still signal to the services via the network domain, due the historical network architecture.

Having the services located in a single location solved some of the problems outlined previously. However, the introduction of the service domain alone does not solve the problem of services being dependent on the technology of the underlying network, and thus the requirement that service developers have knowledge of the network's operation and protocols.

It is through the introduction of two principles that these problems are solved. Firstly, the softswitch principle separates the call / session control from the media transport mechanism. Secondly, the OSA principle separates application services from call / session control. The combination of these two principles provides an abstraction layer between the service domain and the network domain in the form of an Application

Programming Interface (API). This API provides services a generic and standard set of methods with which they can control various network elements, freeing them from the requirement of having knowledge of the specific protocols used by the network elements.

The differentiation of the service domain and the network domain in figure 1.1 is now extended to show the introduction of the softswitch and OSA API, shown in figure 1.2.

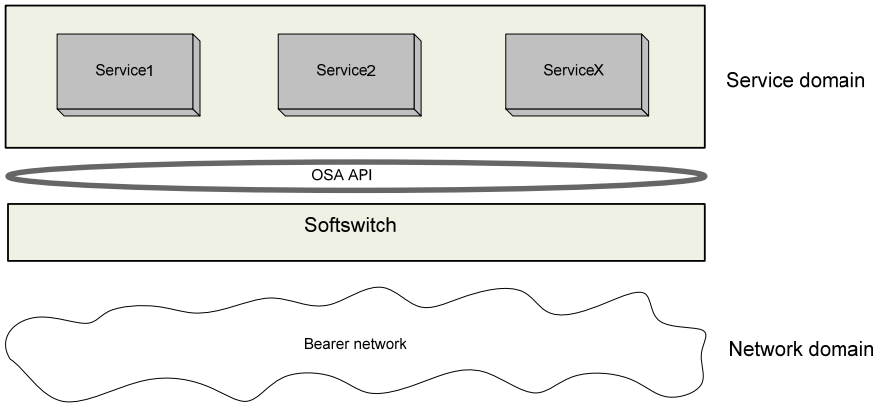


Figure 1-2: The service domain, using the softswitch and OSA principles

When the service domain is coupled with the softswitch and OSA approaches, the services are abstracted from the network domain by the API. The abstraction layer introduced by the API thus allows services to be independent of their underlying networks, and allows a far broader range of programmers to develop telecoms services.

It is the softswitch software and hardware that convert the generic methods offered to the service domain by the API to the specific signals and messages required by the protocols of the underlying network.

Various APIs have been developed, which follow the generic OSA principle, namely JAIN and OSA / Parlay. This project assumes the use of the OSA / Parlay APIs; however, many of the developments proposed and conclusions drawn are independent on the APIs which the service domain utilises and apply equally to other standards that follow the OSA principle.

1.2.3. Overview of OSA / Parlay

The objective of The Parlay Group it to intimately link Information Technology (IT) applications with the capabilities of the telecoms world, by specifying and promoting APIs that are secure, easy to use, rich in functionality, and based on open standards [3].

The OSA / Parlay specifications define an architecture that enables service application developers to make use of network functionality through an open standardised interface, i.e. the OSA / Parlay APIs. The network functionality is described as Service Capability Features (SCFs). See the Parlay standards [4] for more information concerning SCFs.

Parlay integrates telecom network capabilities with IT applications via a secure, measured, and billable interface. Parlay's open APIs release developers from having to write code for specific networks and environments, reducing risks and costs, and allowing for innovative new services to be delivered via the telco network-operator channel [3].

The Parlay APIs allow third-party applications to be hosted within a telecom operator's own network, and allow applications running on external application servers to offer their services to the operator's subscriber base via a secure gateway [3].

Figure 1.2 showed the generic service provisioning approach using the softswitch and OSA principles. In figure 1.3 this generic approach is specialised to show the Parlay architecture.

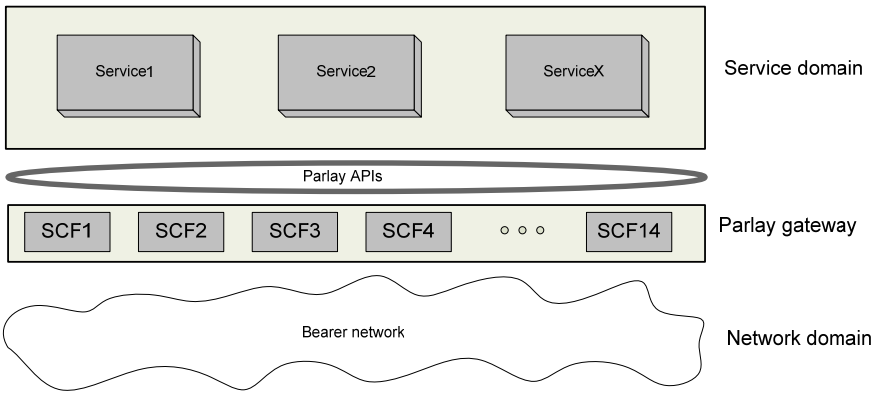


Figure 1-3: The Parlay architecture

The SCFs, within the Parlay gateway, convert the network independent methods offered by the Parlay APIs to the specific protocols and messages supported in the underlying network. Parlay presently specifies 14 SCFs, each of which provides a different set of API methods pertaining to different sets of network capabilities. The most frequently used SCFs include the Multi-Party Call Control SCF [5], which provides methods allowing applications to establish and manipulate multi-party connections, and the Generic Messaging SCF [6], which allows applications to send and receive voice and electronic messages.

The benefits of the Parlay APIs are numerous. Parlay APIs [3]:

- Are open and technology-independent, allowing the widest range of market players to develop and offer advanced telecom services.
- Eliminate the need for programmers to learn onerous telecom protocols, lowering costs and raising the programming abstraction level to the point where telecom capabilities become just "normal" IT APIs.
- Make it possible for external application servers to interact with telecom network capabilities.
- Bring the highly successful Internet development model to the telecom domain with the full participation of telco operators.
- Reduce business risk for all parties involved via the open API model.
- Enable the new business model: "Network Operator As Retailer of Services."
- Allow the creation of applications that function across multiple networks.
- Support 2G, 2.5G, and 3G networks with the same APIs, providing a future-proof evolution path for network services.
- Make it possible to introduce a wide variety of services and applications quickly, with faster development cycles and greater sensitivity to the needs of specific markets.

More information on the OSA / Parlay standard, and the full OSA / Parlay specifications, can be found at the Parlay website at http://www.parlay.org.

1.2.4. Limitations of the existing approach to service provisioning

Two aspects of the existing approach to service provisioning can be analysed: the approach to the telecoms environment in which the service domain operates, and the approach used for implementation of the service domain (using Parlay).

Limitations of the existing approach to the telecoms environment in which the service domain operates

Service signalling refers to the signalling that is used between services (provisioned as applications) and terminals to invoke and control the execution of services. The existing approach to service signalling is shown in figure 1.4. In figure 1.4, the invocation of services in the service domain can be achieved in either of two ways: services can be invoked by third party applications, e.g. a web service (5), or by the underlying network (1-4). The case where services are invoked by third party applications is not relevant to the analysis at present, and attention is focused on the invocation of services by the underlying network

In the existing approach, services are invoked by the underlying network as follows. Using network layer call / session signalling, a terminal signals to the call / session layer in the network domain (1). Having met a trigger condition (2), an event is sent to the Parlay gateway (3). The Parlay gateway then causes a notification to be sent to the service domain (4), and the appropriate service is invoked. (Call / session signalling is the signalling that is used in the call / session layer in the network domain, and is concerned with setting up media, bearer streams etc.)

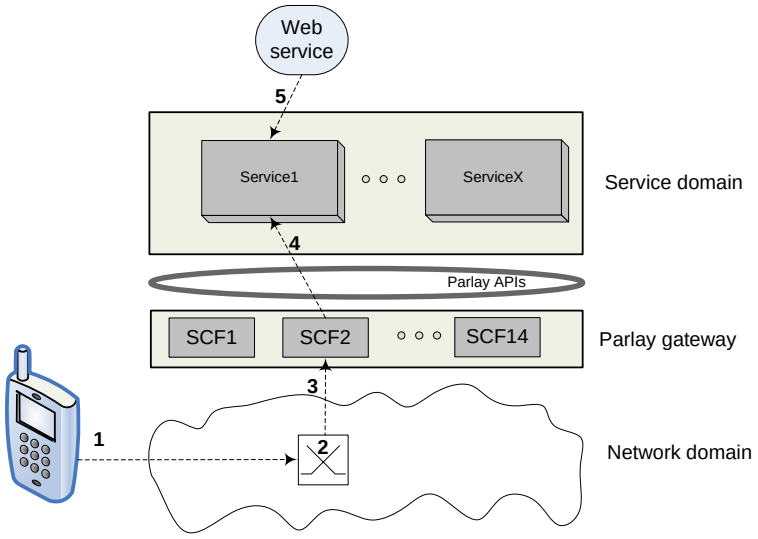


Figure 1-4: The existing approach to service signalling and invocation

This is the only way that intelligence in the terminal can interact with intelligence in the application domain, since OSA / Parlay assumes that all user to network signalling is intended for call / session signalling. Requiring that terminals signal to the application domain using protocols that are oriented towards call / session signalling places a limitation on the signalling complexity that can be achieved, since all signalling from terminals to the application domain relies on triggering mechanisms in the network.

The existing approach is adequate if services are simple, and the terminal is oblivious to the execution of the service (i.e. no further service-terminal interaction is required, other than requiring the user to respond to simple announcements, etc.). The existing approach using call / session signalling is inadequate for advanced services, which require complex interaction with the terminal.

The second point regarding the limitations of the existing approach to the telecoms environment in which the service domain operates concerns the locus of call / session control and management. The Parlay standards imply that the primary view and control of the basic telecoms service of providing end-to-end user connectivity (the BCS) is still located in the network domain (in the call / session layer). Terminals that wish to establish end-to-end connections signal to the network domain (specifically, the call / session layer) to establish the connection, and management of the connection is maintained in the network domain (in the call / session layer). A service-centric approach would allow for the control of the BCS in a centralised location, alongside other services

provided to customers, enabling easy handling and manipulation of connections by other services.

Limitations of the existing approach used for implementation of the service domain (using Parlay)

The obvious strength of Parlay is its well defined set of APIs. Thus, the way that Parlay applications (telecoms services) should interface with the underlying network (via the Parlay gateway) has been well defined in the Parlay specifications. However, besides the specification of the use of callback interfaces, the Parlay specifications do not define any structure for either the applications themselves, or for the service domain (which hosts the Parlay applications) as a whole. For example, no explicit provision has been made for the implementation of a software reuse methodology.

A second area which is as yet undefined is how applications should deal with various sources of invocation. Whereas, in the existing approach, 1st party services are invoked through the receipt of a notification from the Parlay gateway, 3rd party services are invoked through the receipt of an invocation message originating from elsewhere, e.g. a web service (signal 5 in figure 1.4). How services should deal with invocation from various sources, each using different protocols, is not covered in the Parlay specification. The potential for various types of sources to invoke telecoms services imposes a burden on each and every telecoms service, since each has to be equipped with the ability to process invocation requests of various forms, unless a standardised approach is developed.

1.3. Formal problem statement

The type and complexity of services that can be supported in existing networks are restricted by the following limitations of the service provisioning environment:

- The service domain is poorly developed;
- The telecoms environment in which the service domain operates is inefficient, from the perspective of service provisioning.

In the absence of a standardised service domain architecture, network operators need to either develop proprietary service domain architectures, or the operators' service domains remain devoid of structure altogether. This results in service developers needing to employ non-standardised and ad hoc approaches to service development and deployment.

Left unattended, the second part of the problem increases the effort required to develop services, and places a cap on the variety and complexity of services that can be successfully provisioned.

Each of the problem parts has a number of sub-problems:

- The service domain is poorly developed:
  - The service domain has no defined architecture;
  - The services have no defined structure;
  - No software reuse methodology has been defined;
  - No guideline exists concerning how services should handle invocations from multiple sources.
- The telecoms environment in which the service domain operates is sub-optimal:
  - Service signalling is required to call / session signalling protocols, which are designed for setting up connections, and are inappropriate for advanced service-oriented interactions;
  - The primary view and control of the BCS is maintained in the call / session layer of the network domain, and has no centralised point of monitoring and management.

By addressing these problems, this research project attempts to circumvent the limitations in existing service provisioning environments, and in so doing, redefines the environment in which services operate.

Looking ahead

Chapter 2 identifies broad characteristics of an ideal service provisioning environment, which attempts to overcome the problems and limitations identified above.

Chapter 2

THE IDEAL SERVICE PROVISIONING ENVIRONMENT

The OSA approach to the provisioning of services (e.g. using Parlay) offers networks significant advantages, as described in section 1.2.3. However, existing networks that subscribe to this service provisioning philosophy still have numerous shortcomings and limitations, some of which were identified in section 1.2.4. The ideal service provisioning environment would also employ the OSA approach; however, this environment requires aspects of the existing approach to the service domain to undergo significant modification.

The characteristics of the ideal service provisioning environment presented here reflect the goal of creating an environment conducive to the development, provisioning, invocation and execution of services.

2.1. Distinguishing features of the ideal service provisioning environment

All of the characteristics of the ideal service provisioning environment presented next have services as their focal point: their development, provisioning, invocation and execution. The characteristics and features can be divided into two distinct categories:

- Features that deal with the environment in which the service domain operates;
- Features concerning the service domain.

The service domain determines the environment in which services are implemented and executed, and thus has an obvious impact on service provisioning. The environment in which the service domain operates (i.e. the telco network as a whole) also has an impact on service provisioning since the environment dictates how services are invoked, how

they conduct their resulting actions, and thus the complexity of services that can be supported.

2.1.1. Ideal features of the telco environment

The telco network as a whole determines the environment in which services, and the service domain, operate. The network dictates both the possibilities and the constraints of services. For example, the telco network determines how terminals, and other network elements, communicate with services. That is, the network determines the nature and complexity of service signalling. This, in turn, dictates the variety and complexity of the services that can be provisioned.

Service signalling

Section 1.2.4 described how requiring service signalling to be diverted through the call / session layer of the network domain, and thus use network layer call / session signalling (which is oriented towards the setup and management of media streams and not service interaction) limits the capabilities of services that can be provisioned. Also, network layer signalling is, by its very nature, dependent on the network it serves. The use of network layer signalling thus places a requirement on terminals that they are able to send and receive communication using various network layer protocols.

The ideal service provisioning environment would use a service signalling protocol that is more powerful and appropriate to service signalling, and thus allow terminals and service logic to communicate with each other directly, without dependence on the network or specific network layer protocols.

Location of BCS primary control and view

In the existing approach, the primary control and view of the BCS is maintained in the (call / session layer of the) network domain. For reasons which are made apparent in section 2.2, the ideal service provisioning environment provides the control, view and management of the BCS in the service domain, along with the various other services offered to the network users.

2.1.2. Ideal features of the service domain

Since services reside in the service domain, the service domain determines how services should be structured and operate, and has a significant impact on service provisioning.

Service domain structure

Section 1.2.4 identified that in the existing approach, neither the service domain nor the services have any defined structure. The Parlay standards specify the use of callback interfaces, but provide no further guidance. The ideal service provisioning environment would remedy this problem, by employing a structured service domain, which contains appropriately structured services. Specifically, a software reuse methodology should be advocated.

The left side of figure 2.1 shows the existing approach to the service domain, which is devoid of any structure. The right side shows a well-defined service domain, advocating software reuse and a layered, hierarchical approach, conducive to the provisioning of advanced services.

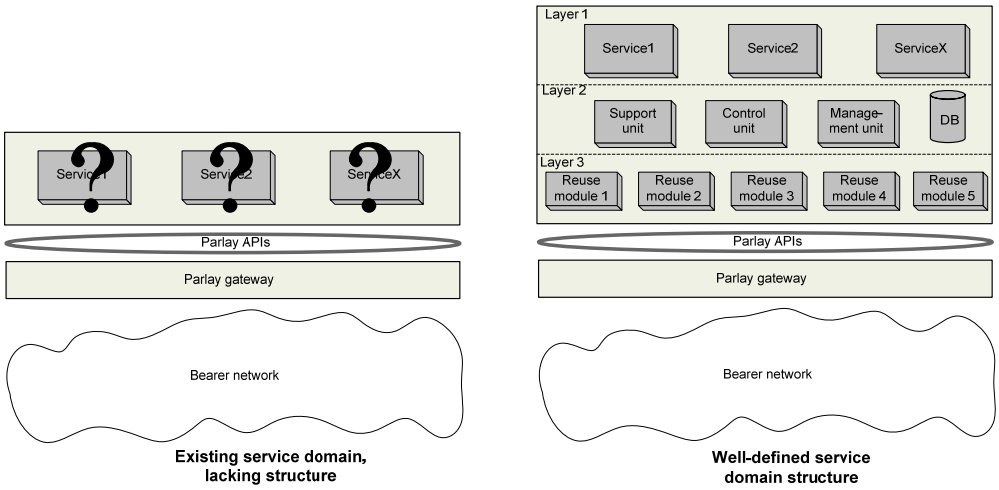


Figure 2-1: The ideal service provisioning environment advocates a structured service domain

Invocation mechanism

Section 1.2.4 stated that, in the absence of a standardised approach, each service would have to deal with application layer and triggered invocations, originating from various sources (e.g. terminals, web services), with proprietary approaches. The requirement that each and every service implement its own approach in such an ad hoc manner is inefficient, and burdensome on application developers.

The ideal service provisioning environment should implement a *generic invocation interface*. The generic invocation interface would serve two purposes:

- It would introduce a standardised approach for services to deal with application layer and triggered invocations, from various sources (e.g. terminals, web services). This standardised approach would hide the protocols used to achieve service signalling (e.g. Q.931, SIP) from the services.
- It would abstract the services from low-level technology dependent protocols, and allow services to communicate with the invocation interface using appropriately oriented high-level messages.

Figure 2.2 presents the existing approach used for service invocation (on the left), and the approach to service invocation using a generic invocation interface (on the right). In both sides of the diagram, service invocations from 3 different sources are shown: from the network (a), from the terminal (b), and from a web service (c).

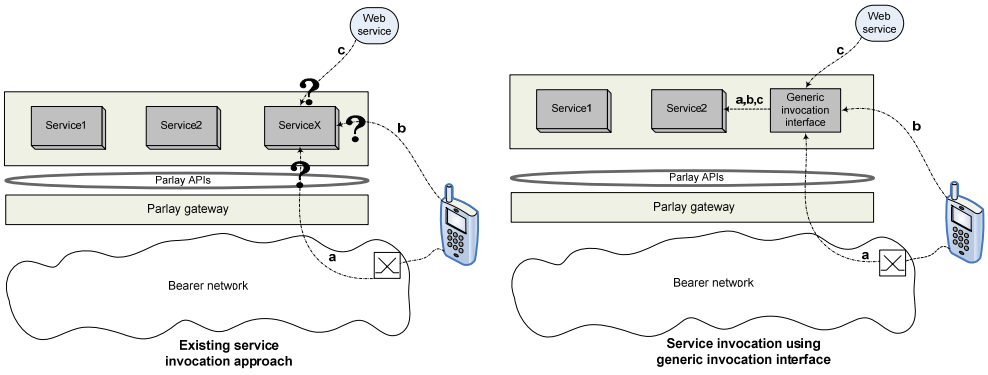


Figure 2-2: The use of a generic invocation interface to mediate service invocations

In the existing approach (on the left), there is no guidance as to how services should handle invocations originating from different technological domains. Services are

required to communicate with sources external to the service domain using various types of signalling protocols.

The generic invocation interface, used in the ideal service provisioning approach (on the right) relieves services from having to deal with varied signalling protocols from outside the service domain. The generic invocation interface adapts the diverse protocols used to signal to the service domain into methods of a single protocol that be easily assimilated by the service. In this approach, services only have to deal with a single source of signalling, i.e. the invocation interface.

2.1.3. Service logic and the intelligent terminal

The ideal service provisioning environment advocates the use of powerful and advanced service signalling oriented towards service communication. The signalling capabilities of traditional, “dumb terminals” (e.g. touch-tone or DTMF terminals) would be prohibitively rudimentary, and, as such, more advanced “intelligent terminals” are required by the ideal service provisioning environment.

Certain basic requirements of the intelligent terminal, as required by the ideal service provisioning environment, are described next. These assumptions concerning the intelligent terminal formed the premise for the research in this research project.

The intelligent terminal is assumed to have the following characteristics:

- The intelligent terminal should have an advanced user interface, and user-interaction ability. The intelligent terminal is thought of as being capable of presenting the user with various options in a menu-type format. The user should be able to select among the various options presented in a simple way, using a mouse / pointer, keypad etc.
- The intelligent terminal should have the ability to process the user’s request, and take subsequent actions based on the results of the processing.

The requirements of the intelligent terminal would be satisfied be any modern digital terminal equipment, e.g. cellular telephone, PDA etc.

Whereas dumb terminals preclude any logic from being located within the terminal, and require that all service logic be contained within the service domain, intelligent terminals introduce the potential of distributing the service logic between the service domain and the intelligent terminal. Thus, the decision of whether to distribute some service domain logic to the intelligent terminals arose in the project.

A major thrust of the ideal service provisioning environment proposed here is the move toward centralised service logic: all services should be located in the service domain. Centralised logic permits maximal software reuse and efficiency, among other things. Thus, the ideal service provisioning environment would minimise the service logic contained in the intelligent terminal, and therefore maintain the centralised location of the service logic, in the service domain. The only logic that would be contained in the terminal is the logic that drives the user interface, and permits advanced user interaction.

The role of the intelligent terminal is limited to the following 2 functions:

- Allow the user to select a service and associated parameters using an advanced menu-system.
- Process the service selection into an invocation request, and send the invocation request to the service domain.

2.2. The ideal service provisioning environment advantage

With the changes proposed in the previous section, the service provisioning environment would benefit in numerous ways.

A properly developed service domain architecture, coupled with a well-defined approach to application structure and an advocated software reuse methodology would afford software developers greater ease in developing telecoms services, and allow services increased simplicity and efficiency.

A generic invocation interface, would relieve services from having to implement proprietary approaches to deal with varied signalling protocols from outside the service domain, and abstracts the service from the underlying network. This decreases the demands required of the service, decreases the demands placed on the service developer,

and increases the number of application developers with the skill and ability to develop telecom services.

If terminals could signal directly to the service domain, and not require the use of the network domain, an appropriately designed, high-level signalling protocol could be used in lieu of call / session oriented signalling used in the network domain, and thus allow more advanced services to be implemented. Using a service oriented protocol that is not dependent on the underlying network also relieves terminals of having to communicate using network-specific protocols, making terminals network independent.

Finally, if the BCS is invoked, controlled and managed in a centralised location (the service domain) in the application layer, it would allow other services increased ease in connection control and manipulation, increasing the complexity of services that can be supported.

A network implementing the ideal service provisioning environment would be able to introduce more advanced services, in a simpler and quicker way, reaping economic benefits.

Looking ahead

The remaining chapters describe the proposed service provisioning environment, using the theoretical foundation developed in chapters 1 and 2. Chapter 3 describes the principal characteristics of the proposed service provisioning environment in rigour and detail, which integrate the concepts introduced in this chapter into a practicable solution.

Chapter 3

PRINCIPAL CONCEPTS OF THE PROPOSED SERVICE PROVISIONING ENVIRONMENT

This chapter presents the principal concepts of the proposed service provisioning environment. Since many aspects throughout a telecoms network influence the service provisioning environment, including (but not limited to) the service domain, concepts concerning the service domain and the rest of the telecoms network (which defines the operational environment of the service domain) are introduced. These concepts form the theoretical underpinnings of the proposed service provisioning environment upon which subsequent chapters build.

3.1. Important definitions

The description of the principal concepts of the proposed approach to service provisioning presented in this chapter requires the definition and clarification of various terms. For certain concepts, traditional terminology does suffice for their accurate description, and is used wherever possible. However, since the report presents a new service provisioning environment, and because of the theoretical nature of the work, traditional terminology is often inadequate to properly describe the concepts proposed. In this case, the definitions presented in this section are more appropriate for their description.

3.1.1. General definitions

Throughout this research report, reference is made to the *proposed service provisioning environment* and the *proposed service domain architecture*. Referring to figure 3.1, the service domain is seen to be a single element in the context of an multi-element environment. The environment as a whole, which is designed for the provisioning of

services, is called the *service provisioning environment*. All elements in figure 3.1 constitute the service provisioning environment. The service domain is only a single element in the service provisioning environment. This report proposes changes to the service provisioning environment, as a whole, as well as to the architecture of the service domain, resulting in the *proposed service provisioning environment* and the *proposed service domain architecture*, respectively.

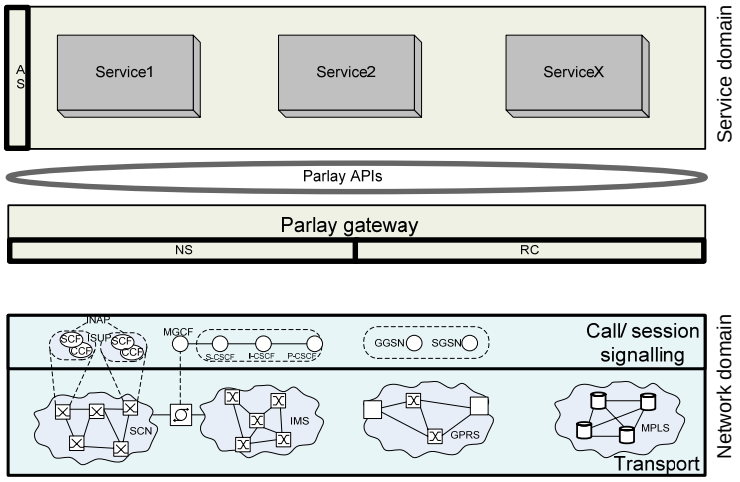


Figure 3-1: The introduction of service adaptors and multiple networks

Figure 3.1 differs from previously presented diagrams in two primary ways. Firstly, where the network domain was represented by a generic cloud, it is now shown in more detail. Secondly, 3 signalling adaptors have been added (the rectangles in bold): labelled NS, RC (in a horizontal layer, at the bottom of the Parlay gateway) and AS (on the left of the service domain, in an upright position).

In the network domain, two layers are shown: the call / session signalling layer, and the transport layer. The call / session signalling layer contains those elements which manage the use of the resources contained in the transport layer. The transport layer is responsible for the provisioning of bearer connections and the transport of data, as controlled by the call / session signalling layer elements. The transport layer also transports the signalling messages, using a suitable distribution mechanism, which allows the service domain, gateway and call / session signalling elements to communicate.

Also, in the network domain, both circuit-switched networks and packet-switched networks are represented. Specifically, the network domain includes (from the left) a PSTN, IMS network, GPRS network and an MPLS network.

The 3 adaptors shown in bold in figure 3.1 are central to service signalling, and are briefly discussed. At the bottom of the Parlay gateway are the Resource Control (RC) and Network Signalling (NS) adaptors. Both of these adaptors provide an interface between the service domain and the underlying network (through the Parlay gateway). These adaptors are not standardised, but their implementation, in some form, is required by the Parlay gateway for successful operation. On the left of the service domain is the Application layer Signalling (AS) adaptor.

The NS adaptor adapts signals directed from the network to the service domain. The RC adaptor adapts signals directed from the service domain to the network. The resources referred to in RC are those contained within the network, which affect low-level control over bearer connections.

For example, if a service wishes to influence how a bearer connection is handled, it signals to the resources in the network through the RC adaptor. For example, the RC adaptor will be used if a service wishes to add a media stream, play an announcement etc. On the network side, the RC adaptor primarily deals with the INAP (Intelligent Network Application Protocol) and SIP (Session Initiation Protocol) protocols.

The RC and NS adaptors are the only adaptors required in the existing service provisioning environment. The AS adaptor is introduced in the proposed service signalling provisioning environment, and provides an interface between the service domain and the terminals. This interface is required for the introduction of application layer signalling (which is described in section 3.1.2), which is used extensively in the proposed service provisioning environment.

The objectives of the NS and AS adaptors are functionally similar: both allow elements residing outside of the service domain to communicate with the services in the service domain. Whereas the NS adaptor allows network elements to communicate with the service domain (via the Parlay gateway) the AS adaptor allows terminals to communicate with the service domain.

Although the objectives of the NS and AS adaptors are similar, they have markedly different operations. The NS adaptor receives low-level network layer protocols, and adapts these network protocols into high-level event notifications to be sent from the gateway to the service domain. The NS adaptor is therefore dependent on the protocols used in the network.

The AS adaptor is network independent, since high-level service-oriented signals are used on both its service domain and network sides. The AS adaptor provides more of an interfacing role, as it is not required to adapt signals from one type to another, and thus performs a less complex task than the NS adaptor.

3.1.2. Definitions for signals

With the introduction of application layer signalling in the proposed service provisioning environment, the need arises to properly define all of the signals which can be found. Figure 3.2 presents a signalling framework, which is independent of any particular service or network. This neutral service signalling framework allows the various types of signals to be defined in their operational context.

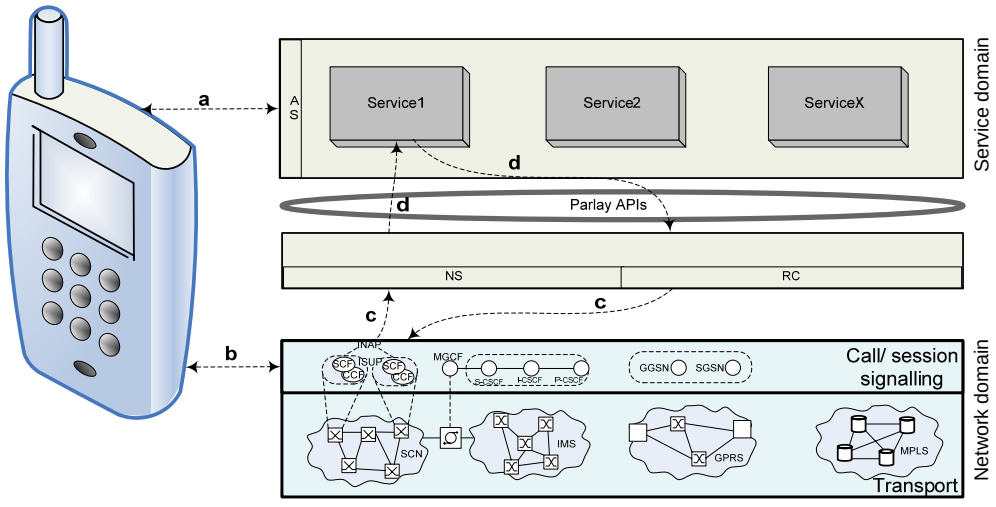


Figure 3-2: Service signalling framework

Four classes of signals are identified:

a. Application layer signalling
b. Call / session signalling
c. Breakout signalling
d. Parlay signalling

The terminal is shown in an elongated fashion to illustrate its functionality in the application layer, resource control layer and the network layer. Both call / session signalling and application layer signalling are shown as signals operating in a single, horizontal layer. Call / session signalling exists only in the call / session layer or network layer, and application layer signalling exists only in the application layer.

Breakout signalling and Parlay signalling involve vertical communication, and allow signalling between layers: breakout signalling allows signalling between the network and the Parlay gateway, and Parlay signalling encompasses all signalling between the service domain and the Parlay gateway.

Practical considerations of service signalling

The network underlying the service domain (the transport network) has two functions:

- Transport user streams (its bearer function)
- Transport signalling

Many pictures in this report depict the service domain to be ‘above’ the bearer network. This is to illustrate that the service domain is largely independent of the bearer function of the transport network. However, both network and application layer signals are shown to travel to (and from) the service domain, and these signals obviously require a distribution mechanism to achieve this. While the service domain is independent of the transport network’s bearer function, it is dependent on the transport network to transport signalling to and from the service domain.

Figure 3.3 shows that, although the service domain is independent of the bearer network (the bearer function of the transport network), the transport network (and its signalling transport function) actually envelopes the whole service provisioning environment.

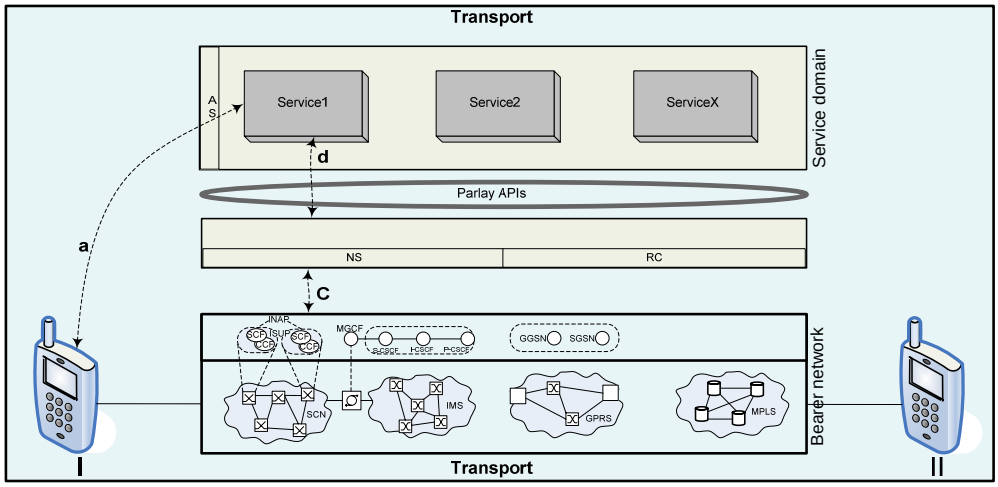


Figure 3-3: A network's signalling transport and bearer transport functions

When application layer signalling is used for service signalling (a), the signalling transport function of the transport network is still used to locate and contact either the AS adaptor on the service domain (for signalling to the service domain) or the terminal (for signalling from the service domain). Similarly, when breakout (c) and Parlay signalling (d) are used for service signalling, the transport network is used to find and contact the NS adaptor on the Parlay gateway (for signalling to the service domain) or the appropriate network element (for signalling from the service domain).

The critical difference between network layer signalling and application layer signalling is the node that the signalling messages are addressed to. For the purposes of this project, it is imagined that a service plane envelopes the network, service domain, and the terminals, and manages the implementation of the communication between all elements. This idea is conveyed in figure 3.4.

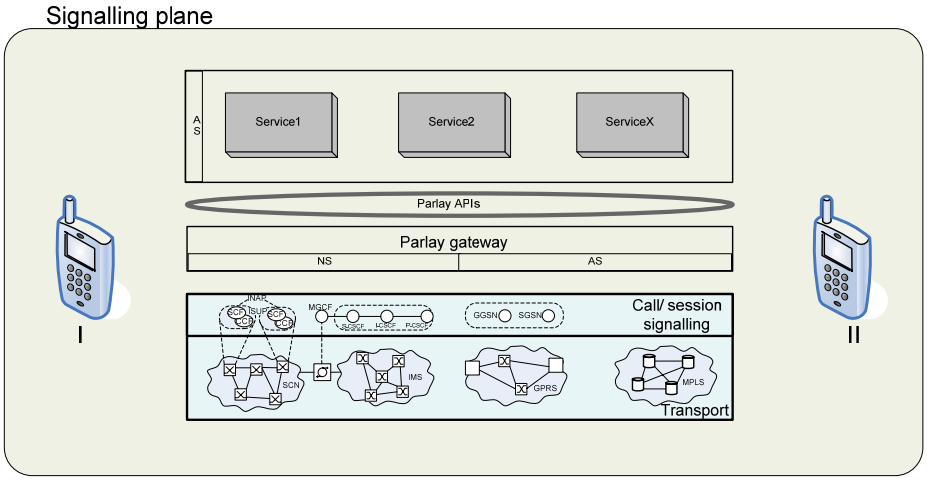


Figure 3-4: The signalling plane

3.1.3. Definitions for call, service and invocation categories

A telecoms concept or construct can be divided into multiple categories, based on fundamental characteristics, to aid in its convenient description and classification. For example, calls can be described as being 1st or 3rd party initiated, a service can be invoked using call / session signalling or application layer signalling, etc. Clarification of the categorisation of certain concepts is made in this section to ensure that the reader understands its implications when used in the report.

Categories of call initiation

Figure 3.5 depicts two scenarios of initiating a call or a session.

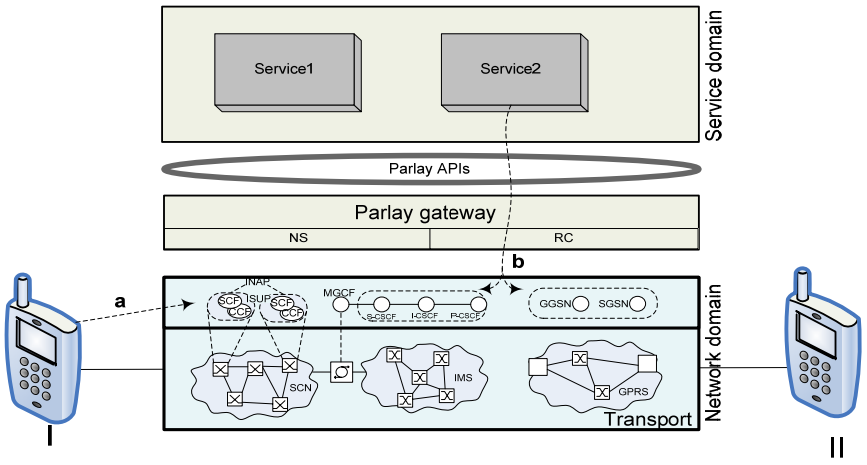


Figure 3-5: Categories of call initiation

A call can be initiated in one of two ways:

- 1st party call initiation. Shown by the call / session signal *a*, a terminal initiates the call. This is the more common case.
- 3rd party call initiation. Shown by the call / session signal *b*, a service initiates the call.

In both cases, a call is established between terminals I and II. This naming designation is the same as that used traditionally, and is appropriate for this report.

Categories of services

Figure 3.6 depicts the two types of services.

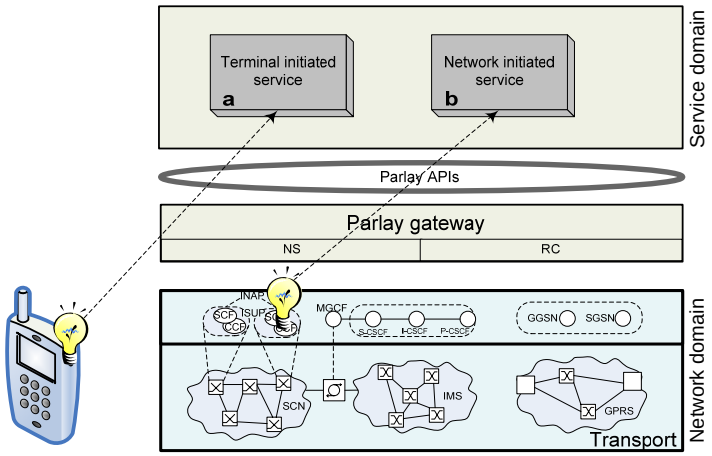


Figure 3-6: Categories of services

All services can be categorised into two categories, determined by the origin of their invocation:

- Terminal initiated services. The service invocation originates in the terminal. An example of this variety of service is the Call Hold service, where the service is explicitly invoked by the terminal.
- Network initiated services. The service invocation originates in the network. In this case, the terminal is oblivious to the invocation and execution of the service. These types of services are often triggered by a particular network event, e.g. call not answered.

Categories of service invocation

Figure 3.7 shows the two ways in which service invocation can be achieved.

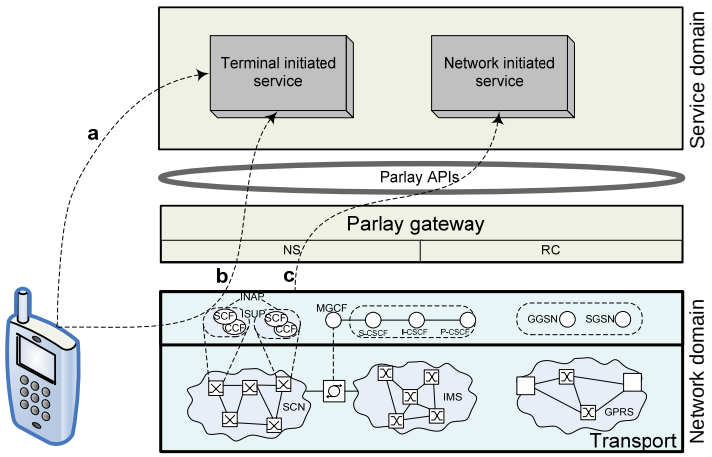


Figure 3-7: Categories of service invocation

There are two ways in which services can be invoked:

- Application layer invocation. This is shown by signal *a*. Application layer invocation is achieved using application layer signalling. With application layer invocation, services receive the invocation request directly from the terminal. This approach is not available in existing networks.
- Triggered invocation. This is shown by signals *b* and *c*. Triggered invocation is achieved using breakout and Parlay signalling. With triggered invocation, services receive the invocation request from the network, as a result of an event being triggered in the network. In the absence of application layer signalling, all services are invoked using triggered invocation.

In existing networks (in the absence of application layer signalling), services in the service domain can only be invoked in one way: using triggered invocation. With the introduction of application layer signalling in the proposed service provisioning environment, services can be invoked by the underlying network, with triggered invocation, or directly by a terminal, with application layer invocation (using application layer signalling). The type of invocation that is used to invoke a service depends on the service-type (network initiated service or terminal initiated service), as is explained in the next section.

3.2. The proposed approach to service signalling

Service signalling is the general name given to any signalling that is required by a service for its operation. Service signalling is used in two distinct ways:

- Service operation. During service operation, two different phases have been identified:
    - Service invocation. The initial invocation of a service requires service signalling, originating from either the network of a terminal;
    - Service execution. Service signalling may be used during the execution of a service to obtain additional information.
- Service management. The setup and configuration of certain services requires the use of service signalling.

The two phases of service operation are separated due to the different demands they place on service signalling: signalling for service invocation typically involves only communication directed towards a service, whereas signalling used during service execution often requires the use of additional signalling directed from a service to a terminal.

Service signalling includes communication between:

- The network and the service;
- The terminal and the service.

The proposed approach to service signalling deals with communication between the service and the terminal.

The path used by the service signalling between the terminal and the service determines the signalling protocol that is required to be used, which, in turn, determines the complexity of the communication and the capabilities of any potential service.

The existing approach to service signalling between the service and a terminal requires signals to use call / session signalling protocols that are not oriented towards service signalling. The proposed approach to service signalling provides a direct communication path between the service domain and the terminals, thereby offering a more powerful and appropriately designed service signalling protocol.

3.2.1. Service signalling for service invocation

Service invocation requires elements outside the service domain to initiate a service invocation request, and signal the request to the service domain. As was explained in section 3.1.3, if the underlying network originated the invocation request, the service is called; if a terminal originated the invocation request, the service is called a terminal initiated service. The analysis of the signalling and invocation for terminal initiated and network initiated services is presented separately.

Signalling and invocation for network initiated services

Both the existing and the proposed approaches to service signalling handle the invocation of network initiated services in the same way: using triggered invocation. This approach is shown in figure 3.8. Note that the labelling of the signals (*a*, *b*, *c* or *d*) follows the convention defined in section 3.1.2.

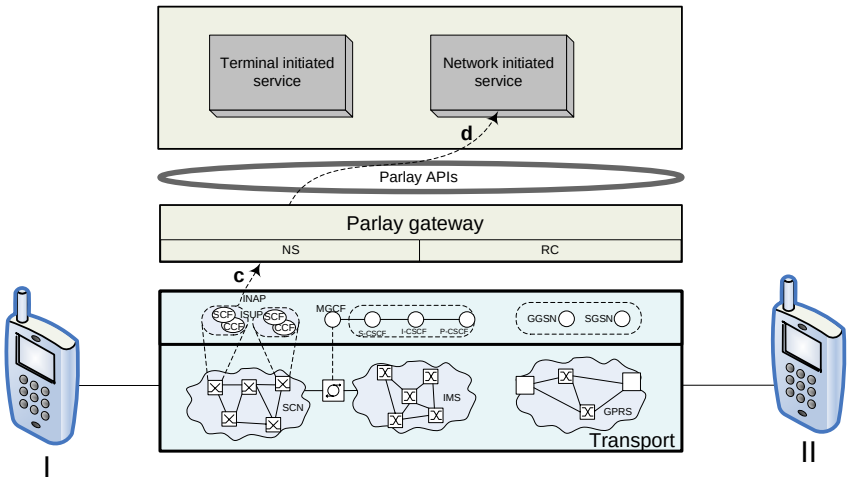


Figure 3-8: Triggered invocation for network initiated services

Network initiated services are invoked using triggered invocation as follows: Due to the triggering of a network event, a breakout signal is sent from the call / session layer of the network domain to the Parlay gateway (c). The Parlay gateway then sends a notification to the service domain using Parlay signalling (d).

The invocation of network initiated services does not require any interaction with the terminal, and it is appropriate that the invocation request travel directly from the network to the service domain, using triggered invocation. This approach is efficient, and is used in both the existing and proposed service provisioning environments.

Signalling and invocation for terminal initiated services

The proposed approach to handling the signalling and invocation of terminal initiated services is significantly different from the existing approach. In the existing approach, terminal initiated services require the use of triggered invocation, whereas the proposed approach uses application layer invocation to invoke terminal initiated services.

Since, in the past, the only service provided by a telecoms network was basic bearer connectivity, and since this service was located in the call / session layer, all service signalling was sent to and from the call / session layer. The signalling protocols used in the call / session layer are thus tailored for the requirements of the BCS, and are oriented towards call / session signalling. However, with the proliferation of services in the modern network, the use call / session signalling to achieve service signalling is restrictive.

Also, with the relocation of services to the service domain, signals should no longer be directed to the call / session layer for processing. Due to the origins of the telecoms network, all service signalling is still sent via the call / session layer, even if it is intended for the ultimate receipt by the service domain.

In the existing approach to service signalling, all signalling between a terminal and the service domain is forced to travel through the call / session signalling layer. That is, signals from the terminal are sent to the call / session layer by default, and the call / session layer redirects these signals to the service domain (using breakout and Parlay signalling). Thus, independent of whether the service is a terminal initiated or network initiated service, triggered invocation using call / session layer protocols has to be used.

Figure 3.9 illustrates the existing approach to the invocation of terminal initiated services with triggered invocation, using the Call Hold service as an example. Call Hold is

classified as a terminal initiated service, since all instructions for the operation of the service originate in the terminal.

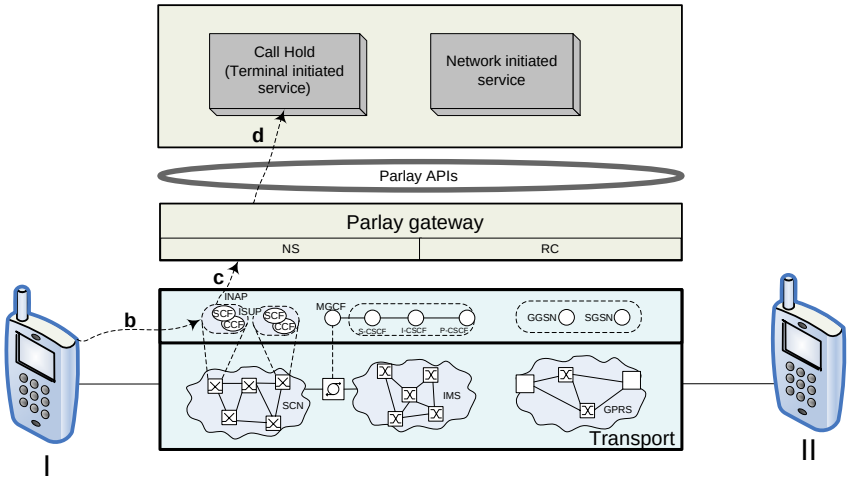


Figure 3-9: The existing approach: Triggered invocation for terminal initiated services

Assume that an end-to-end bearer connection is set up between two terminals. Now, assume terminal I wishes to place the call on hold (using the Call Hold service, which is provisioned as an application). In the existing approach, the following events would occur:

- Terminal I uses call / session signalling to request, from the call / session layer, that the call be placed on hold (b). For example, an ISDN terminal would use a Facility message.
- A trigger is set-off in the call / session layer, and call / session layer forwards this request to the Parlay gateway (using breakout signalling, c), and then to the service domain (using Parlay signalling, d), which processes the request. This process constitutes triggered invocation.

Since both terminal initiated and network initiated services are invoked using triggered invocation in the existing approach, the AS adaptor is not required; only the NS and RC adaptors are required.

This approach is rooted in the past, when service(s) resided in the network domain, could be completely processed by the network and were connection oriented. In this case, call / session oriented protocols were appropriate. However, services are now significantly

more complex than the original connection oriented services, and exert greater demands on the capabilities of service signalling.

With the relocation of services to the application layer in the service domain, a potential solution to this problem arises. Service signalling for terminal initiated services no longer has to traverse the call / session layer and use call / session oriented signalling protocols. A direct communication path between the terminals and the service domain, which bypasses the call / session layer, can be used. Such a signalling path does not have the same constraints and limitations as call / session layer signalling does, and allows the introduction of an appropriate signalling protocol, oriented towards service signalling and not call / session signalling. A signalling protocol that is oriented towards service signalling in the application layer enhances the range of services that can be implemented, and the complexity of these services.

Thus, signalling between services and terminals for terminal initiated services should travel directly between the service domain and the terminals, and not need to be redirected by the call / session layer. That is, terminals and services should communicate using application layer signalling. Application layer signalling is used in the proposed service provisioning environment. Figure 3.10 shows the invocation of terminal initiated services with application layer invocation.

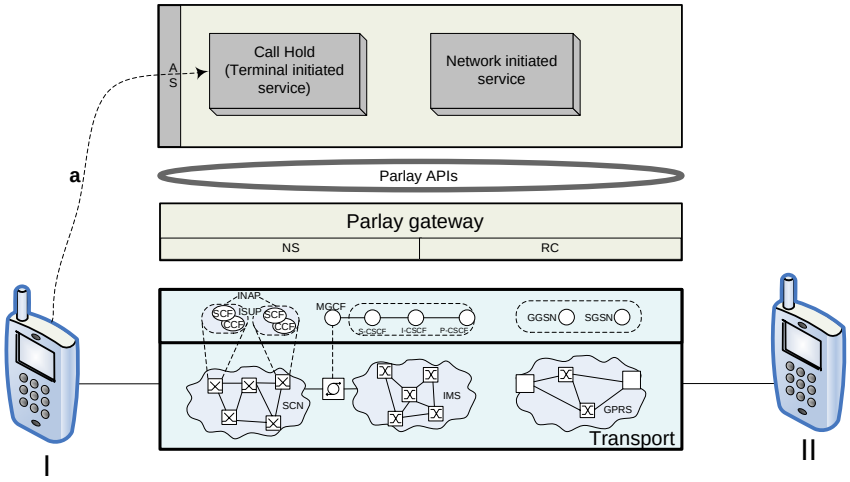


Figure 3-10: The proposed approach: Application layer invocation to invoke terminal initiated services

In the proposed approach, communication between terminals and services takes place using application layer signalling, and doesn't require processing by the network domain. Application layer signalling enables the use of a service oriented signalling protocol, which overcomes the limitations of the call / session oriented signalling protocol used in the network.

Now, the terminal invokes the Call Hold service directly, with application layer invocation (using application layer signalling (a)), as opposed to using triggered invocation (using call / session signalling, breakout and Parlay signalling). Since application layer invocation, using application layer signalling, uses a direct communication channel between the service domain and the terminal, the AS adaptor is used for the invocation of terminal initiated services.

Note that, in the proposed approach to service provisioning, both an AS and an NS adaptor are required. The conventional NS adaptor allows triggered invocation for network initiated services, and the TS adaptor allows application layer invocation for terminal initiated services.

To summarise: The use of triggered invocation for network initiated services is suitable, and used in both the existing and proposed service provisioning approaches. However, the existing and proposed approaches differ in how they handle service invocation for terminal initiated services. Whereas the existing approach to service provisioning requires terminal initiated services to be invoked with triggered invocation (using call / session signalling), the proposed approach allows terminal initiated services to be invoked with application layer invocation.

3.2.2. Service signalling for service execution and management

In addition to service invocation, service signalling can be used during service execution and for service management. Often, terminal initiated services require additional information from the terminal for the successful execution of the service. Also, both terminal initiated and network initiated services may require certain parameters to be configured, or user data to be setup, using communication between the terminal and the service domain.

Whereas service invocation requires signalling in one direction only (from the invocation initiator (i.e. the terminal or network) to the service domain), service signalling used during service execution and service management often requires communication in both directions.

Just as for the invocation of terminal initiated services, service signalling that is required during the execution of terminal initiated services in the existing approach is required to travel through the call / session layer, and use inappropriately oriented call / session signalling protocols. Similarly, in the existing approach, communication between terminals and the service domain for service management is required to use low-level network layer protocols that are designed for the control of bearer streams.

In the proposed approach, application layer signalling is used for all communication between terminals and the service domain (in either direction), and provides a signalling protocol designed for the signalling requirements of service execution and service management.

3.2.3. A comparison of service signalling approaches

The value of application layer signalling is most apparent when its characteristics are compared to those of other signalling approaches. In the first part if this section, an explicit comparison between application layer signalling and call / session signalling is made. Then, the proposed approach to service provisioning, using application layer signalling, is compared to other commercial approaches to service signalling, such as the 3G approach and the web services approach.

Application layer signalling compared to call / session signalling

In the existing approach to service signalling, signals between terminals and the service domain are required to traverse the call / session layer of the network domain, and use network layer protocols. Network layer protocols are oriented towards call / session signalling, and the control and maintenance of bearer streams. For service signalling to be achieved using the call / session oriented signalling protocols, an ad hoc approach is needed that adapts the service signalling information to the network layer protocols. The ad hoc manipulation of the network layer protocols to achieve service signalling limits the

amount, nature and complexity of information that can be transferred. By its very nature, network layer call / session signalling is thus network dependent.

For example, consider the case of a PSTN network. Call / session signalling is achieved using simple DTMF (Dual Tone Multi Frequency) tones, and end-exchanges of PSTN networks are often only capable of processing such signals. Thus, to achieve service signalling in this scenario, the terminal must signal to the service domain using DTMF tones, allowing only very simple signals can be transmitted between the terminal and the service in the service domain. The service is therefore solely responsible for controlling all complex interactions between it and the terminal, and the complexity of services that can be implemented is further limited.

In contrast, when application layer signalling is used, service signalling is not required to adapt and manipulate network layer signalling. Neither the type nor complexity of the service signalling is now constrained. Whereas low-level protocols such as Q.931, DTMF etc. were used for service signalling in the existing approach, high-level, abstracted communication can be employed with application layer signalling, using web services etc. For example, a method call such as sendMessage(MMS, number, body, media) could be made by a terminal directly on the service domain. Application layer signalling therefore allows far more complex services to be implemented.

Application layer signalling is not network dependent. This allows the terminals that utilise the network to be network independent. In contrast, terminals that operate in existing networks are required to use network layer call / session signalling, and are thus network dependent. The network independent method calls that can be made between the terminals and the service domain constitute a well-defined, standardised interface, or API. This API is called the “Application layer API set”. The Application layer API set defines the complexity of the application layer signalling, and the range and type of services that can be implemented. The Application layer API set is discussed further in chapter 4.

The proposed approach to service signalling compared to other commercial approaches

The proposed approach to service signalling and invocation is now compared to the 3G approach and the web service approach to service signalling and invocation. The precise

nature of the proposed service signalling approach can be more easily seen when it is compared to other service provisioning environments.

Figure 3.11 illustrates the approach used to achieve service signalling and invocation in the 3G environment.

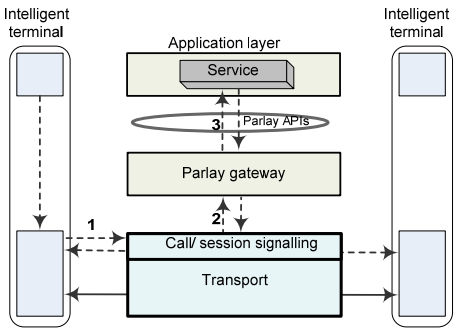


Figure 3-11: 3G approach to service signalling and invocation

In the 3G environment, signalling to the service domain is sent via the underlying network; the signal 'breaks out' of the network to reach the service domain (signals 1,2 and 3) if a trigger condition is satisfied. In this approach, the application layer in the terminal is kept isolated from the application layer in the service domain.

In contrast to the approach used in the current 3G network, the proposed service provisioning environment advocates the use of application layer signalling. The use of application layer signalling in the proposed service provisioning environment is shown in figure 3.12.

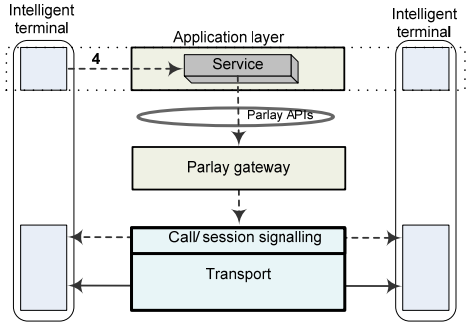


Figure 3-12: Proposed approach to service signalling

Whereas the application layer in the service domain and the application layer in the terminal were previously isolated, application layer signalling joins them and makes the application layer span the entire service provisioning environment in a continuum.

In the web services approach to service signalling and invocation, method calls and service invocation requests are thought to have originated from above the service domain (from a logical perspective). This approach can be seen on the left side of figure 3.13.

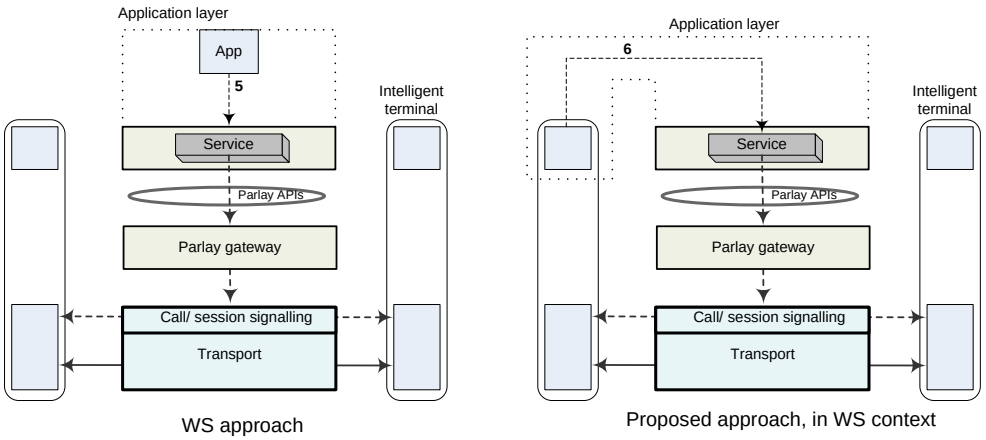


Figure 3-13: Web services approach to service signalling

On the right side of the figure, the proposed approach to service signalling and invocation is shown in the context of the web services approach. Note that the only difference between the two sides of the figure is that the invoking application has been logically relocated from above the service domain to the side of the service domain (inside the terminal). The right side of figure 3.13 is the same as the proposed approach to service signalling shown in figure 3.12.

Application layer signalling in the proposed approach is therefore logically similar to the signalling approach adopted by web services.

3.3. The proposed approach to BCS management

Almost all services that are offered to the users of a telecoms network require the use of bearer connections in their execution. Since the BCS is responsible for establishing and maintaining all bearer connections, it plays a major role in the service provisioning

environment. The ease with which the BCS controls, manipulates and manages bearer connections, and the ease with which services interact with the BCS, determine the range and complexity of services that can be implemented.

3.3.1. Relocation of the primary view and control of the BCS

The Basic Connectivity Service is defined as the service offered by a telecoms network to its users of providing end-to-end bearer connectivity. In this report, the distinction is made between the *control* of the BCS, and to the actual service itself: although the infrastructure required to provision the BCS is inextricably tied to the network, the control of the BCS is thought to be a separate functional entity. This distinction, and other similar considerations, is further discussed in section 3.3.4.

BCS control and management in the call / session layer

The principal objective of telecoms networks, traditionally, is to allow users of the network to communicate, usually in real time. Before the advent of value-added and supplementary services, this was indeed the sole goal of telecoms networks. The focus of the design of the network is to handle this very requirement: end-to-end bearer connectivity, and the BCS is thus the most fundamental of all services.

Since bearer connectivity was the only major focus of legacy telecoms networks, the logic and intelligence required to achieve this end, i.e. the BCS, was deeply entrenched and integrated into the call / session layer of the network. Various elements in the network maintained the view of the connection, and were responsible for the control of the service.

With the recent proliferation of services in telecoms, most of which supplement the BCS, services have been relocated to the service domain. However, in existing telecoms networks, the primary view and control of the BCS still maintained in the call / session layer of the network, despite the control of other services being maintained in the service domain.

Figure 3.14 depicts the existing approach to the management and control of the BCS.

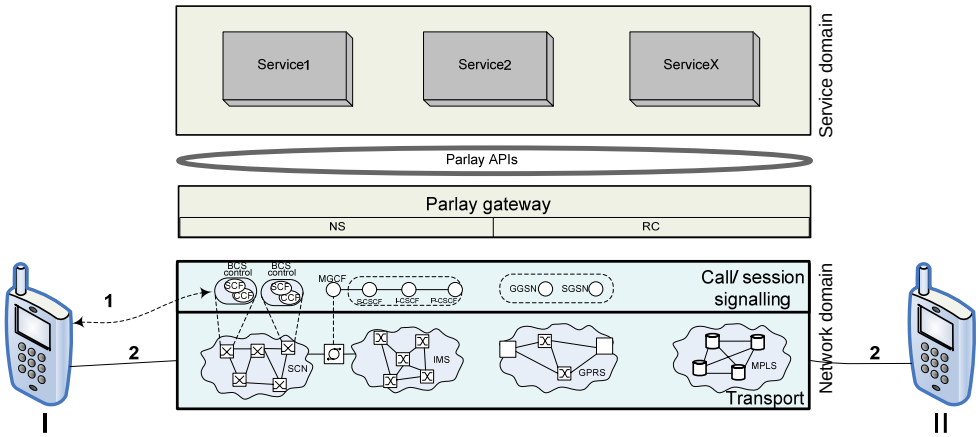


Figure 3-14: The existing approach to BCS control and management

In the existing approach, all processing, control and management of the bearer connection is maintained in the call / session layer of the network domain.

An example of how the BCS is invoked, and operates, illustrates the existing approach. When a basic bearer connection is required, the terminal signals to the BCS controller in the call / session signalling layer of the network that it requires an end-to-end, bearer connection to be established. This signalling is represented by the line numbered '1', between the terminal and the call / session signalling layer, in the network. The BCS controller in the call / session layer then executes, and sets up the end-to-end connection in the bearer network (2). All subsequent manipulation that is required of the basic connection will also be performed from the call / session signalling layer by the BCS.

The existing approach to services differentiates between the BCS and all other additional services: the view and control of the BCS is located in the network, and the control of all other services is located in the service domain.

Relocation of BCS control and management to the application layer

With the provisioning of myriad services as applications in the service domain, there is increasing need to maintain the primary view and control of the BCS in the application layer, alongside these services. Relocation of the control of the BCS to the service domain allows control of the BCS (and associated resources) to be maintained by objects in the application layer, as opposed to being maintained by objects in the call / session

layer in the network domain. With the relocation of the control of the BCS to the application layer, the primary view of the BCS is also then maintained in the application layer.

The relocation of the primary view and control of the BCS from the call / session layer in the network to the service domain is shown in figure 3.15.

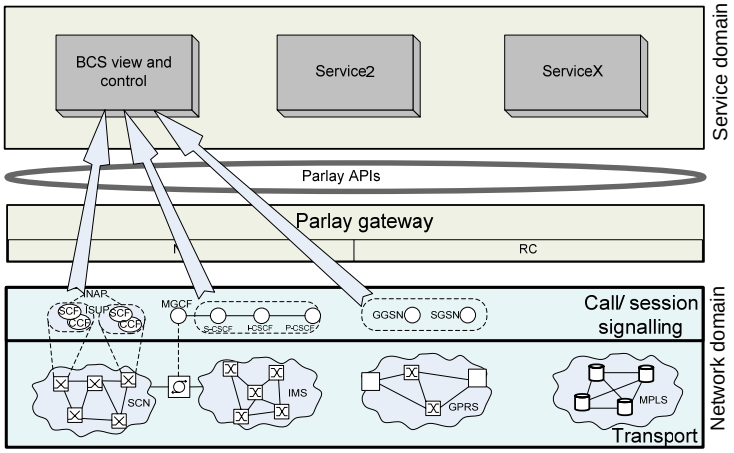


Figure 3-15: The relocation of BCS control and management

Having the primary view and control of the BCS in the service domain, and not distributed across numerous network elements, allows a more holistic, centralised approach to connectivity management, and a more homogeneous approach to service management. Services can communicate with other objects in the application layer to monitor and manipulate bearer connections, rather than communicating with multiple objects in the network domain. This approach allows the bearer network to be seen as more of a simple underlying transport mechanism.

With the primary view and control of the BCS in the service domain, services can control bearer connectivity more easily and effectively, minimising service development times and enhancing service complexity. The proposed approach to service provisioning advocates the relocation of the primary view and control of the BCS to the service domain.

3.3.2. Different approaches to BCS invocation

Since the initiation of the invocation request for a new call or connection always originates in a terminal, the BCS is classified as a terminal initiated service. As discussed in section 3.2.1, the approach used to invoke terminal initiated services is different in the existing and proposed approaches to service signalling, due to the introduction of application layer signalling in the proposed approach. Figure 3.16 shows the two potential approaches to the invocation of the relocated BCS controller.

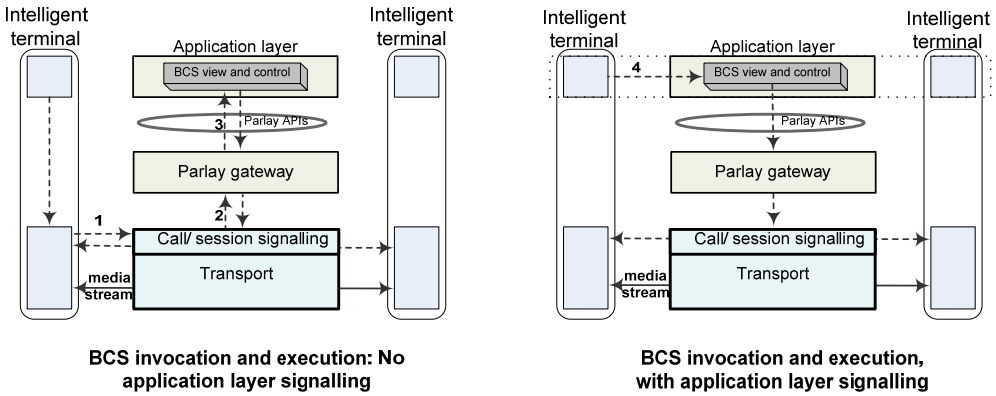


Figure 3-16: BCS invocation using triggered and application layer invocation

On the left, triggered invocation is used to achieve BCS invocation. On the right, application layer invocation is used to achieve BCS invocation.

Without application layer signalling, the invocation signal is required to travel through the call / session layer, requiring the use of call / session, breakout and Parlay signalling (signals 1, 2 and 3). In contrast, with application layer signalling, only signal 4 is required to achieve BCS invocation.

Using application layer signalling to control bearer connectivity enables simpler setup and manipulation of connections by terminals. For example, using application layer signalling, a conference call could be initiated using a single, high-level method call such as ConferenceCall(parties[], mediaType). Without application layer signalling, multiple call / session layer signals would have to be manipulated and adapted to achieve the same result.

In addition to the invocation of the BCS controller, figure 3.16 also depicts the subsequent actions taken by the BCS controller to establish the call. In both cases (on the left and the right), the actions are identical. That is, independent of whether application layer invocation (using application layer signalling) is used to invoke the BCS controller in the service domain, 3rd party call initiation is used. In contrast, 1st party call initiation is used exclusively in the existing approach, since the primary view and control of the BCS is located in the network. (Refer to section 3.1.3 for the difference between 1st and 3rd party call initiation.)

3.3.3. Integrating the proposed approaches to BCS control and service signalling

A final example is now presented, which highlights the two primary conceptual differences between the existing and proposed service provisioning environments discussed in this chapter thus far: application layer signalling, and the relocation of the BCS controller. The example, depicted in figure 3.17, makes use of both terminal initiated and network initiated services, using application layer and triggered invocation, respectively.

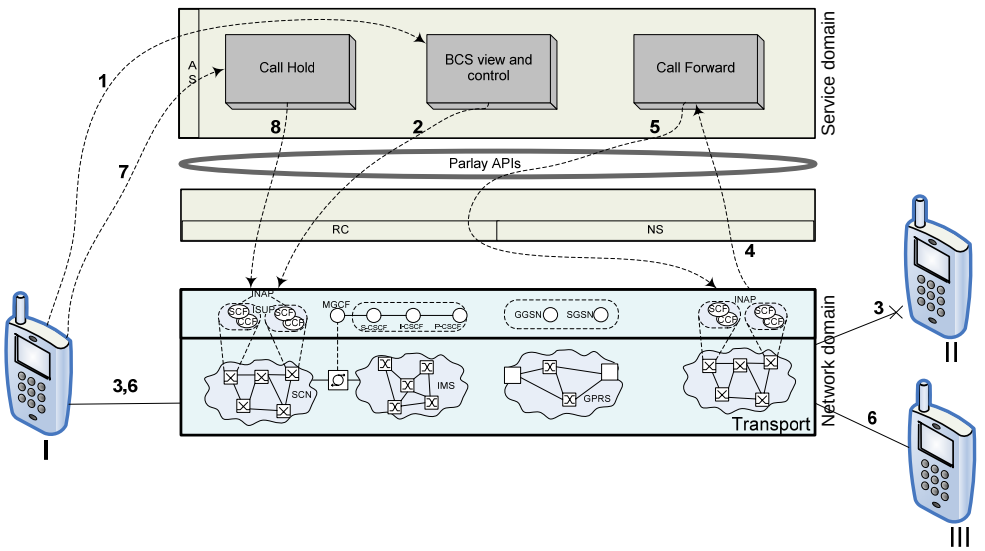


Figure 3-17: An example illustrating BCS control relocation and application layer signalling

Firstly, observe that the primary view and control of the BCS is located in the service domain, alongside the other services. Secondly, all communication between the service domain and the terminal is shown to use application layer signalling. Note that, although terminals I, II and III are shown to be served by the same gateway and application server, they would most likely be served by physically separate units in practice, possibly located on entirely different networks.

Before all of the signals in the example are discussed in detail, a qualitative overview is provided. The broad progression of the example is as follows:

- Party I successfully establishes a connection with party II.

Variant #1: Terminating side uses Call Forwarding

- Party I tries to establish a connection with party II, but party II uses Call Forwarding.
- The call is redirected, and a connection is established between party I and party III.

Variant #2: Originating side uses Call Hold

- During the connection between parties I and III, party I places party III on hold.

All of the signals are now discussed in detail.

1. Terminal I signals to the BCS controller (in the service domain) to set up a new call. The BCS controller is a terminal initiated service, and is appropriately invoked with application layer invocation, using application layer signalling.
2. The BCS controller instructs the Parlay gateway to set the end-to-end bearer connection up. The call is initiated using 3rd party call initiation.
3. An end-to-end bearer connection is cut through between terminals I and II.

Variant #1: Terminating side uses Call Forwarding

3. An end-to-end bearer connection is attempted to be cut through between terminals I and II. However, at the exchange serving terminal II, a network event is detected.
4. Intelligence in the terminating exchange, in the call / session layer of the network domain, processes the trigger, and notifies the Parlay gateway of the network event. The Parlay gateway then signals to the Call Forward service (in the service domain) to execute. The Call Forward service is a network initiated service, and is appropriately invoked using triggered invocation.

5. The Call Forward service performs the necessary interactions with the Parlay gateway to get the terminating exchange at terminal II to terminate the call on terminal III.
6. An end-to-end, bearer connection is cut through between terminals I and III.

Variant #2: Originating side uses Call Hold

7. Terminal I signals to the Call Hold service that it wishes to place the active call on hold. The Call Hold service is a terminal initiated service, and is appropriately invoked with application layer invocation, using application layer signalling.
8. The Call Hold service instructs the Parlay gateway to perform the appropriate call manipulations.

3.3.4. Practical considerations of BCS control relocation

The Basic Connectivity Service is defined as the service, offered by a telecoms network to its users, of providing end-to-end bearer connectivity. The BCS is implemented by all of the elements in a particular network that are responsible for the correct operation of media streams, bearer connections etc., and includes both hardware and software elements.

The relocation of the primary view and control of the BCS to the service domain is made possible by viewing the *control* of the BCS as being a functionally separate entity from the elements that physically implement the service. The meaning and implication of the relocation of the primary view and control of the BCS from the network layer to the application layer is clarified in this section.

In existing networks which don't employ Parlay (or a similar OSA approach) for service implementation, the control of the BCS is maintained exclusively within the network. With the introduction of Parlay to existing networks, some measure of control of the BCS is available at the application layer: the call and call leg objects in the Parlay SCFs do have their own state machines, and together contain a representation of the call and media, and thus a view of the connection. However, the main function of these objects is to receive method calls from the application and to initiate network actions. Also, the API only gives limited ability to interrogate the SCFs. Therefore, the connection view contained by the Parlay SCFs is neither complete nor particularly useful, and the control

of the BCS provided in the Parlay gateway is inadequate. The relocation of the control of the BCS to the application layer in the proposed service provisioning is both complete and useful.

The relocation of the BCS to the service domain, depicted in figure 3.15, is shown for all of the networks represented in the network domain, including packet-switched and circuit-switched networks. Specific practical considerations and implications of the relocation of the primary view and control of the BCS to the application layer for circuit-switched and packet-switched networks follow.

Relocation of BCS control in circuit-switched networks

Representing circuit-switched networks, consider the PSTN. The primary element of the BCS is the Call Controller (CC), which is represented by the Call Control Function (CCF) and physically resides in monolithic switches throughout the network. The function of the CCF is twofold:

- Keep the call state
- Routing / directing signalling

The CCF is the locus of primary control of bearer connections, and maintains a view of call states. The relocation of the primary view and control of the BCS to the application layer does not imply the relocation of the all of the CCF's functions. The role of the CCF in the directing of signalling to its destination is essential to the correct functioning of the network, and relocation of the control of the BCS does not concern this function. However, the second of the CCF's two tasks, viz. keeping the call state, is subject to relocation in the proposed approach. After relocation of the primary view and control of the BCS, the CCF still physically resides in the monolithic switches of the network domain (in the call / session layer), but its function of controlling the call and maintaining the call view is secondary to the view and control maintained in the application layer. The application layer is seen as the home for the primary view, and the highest level of control.

In the proposed approach just as in the existing approach, terminals still signal to the primary controller of the BCS. In the proposed environment, however, the primary controller resides in the application layer, and not in the network domain.

Relocation of BCS control in packet-switched networks

In packet-switched networks, the term ‘relocation’ is applied slightly differently to the case of relocation in circuit-switched networks. Representing packet-switched networks, consider an IMS network. The primary BCS element in IMS networks is the CSCF (Call Session Control Function). The function of the CSCF is primarily to concentrate on directing signalling to its destination, and, unlike the CCF in circuit-switched networks, the CSCF does not keep the call state. Thus, a primary view of the call state still needs to be introduced.

Whereas the primary control and view of the BCS is logically relocated in circuit-switched networks, there is no such logical relocation in IMS networks. Instead, implementation of the primary control and view of the BCS in the application layer is an additional feature that is missing in IMS networks.

3.4. The proposed service domain architecture

Two fundamental concepts of the proposed service provisioning environment have been presented in this chapter thus far, namely application layer signalling and the relocated control and view of the BCS. The third and final fundamental concept is presented here, and, unlike the previous two concepts, concerns the service domain exclusively.

3.4.1. An overview of the proposed service domain architecture

The proposed service domain architecture concerns the structuring of the service domain; the service domain architecture provides an architectural framework in which services are provisioned, invoked and executed. The architecture provides the context and setting for services residing in the service domain. In this section, an overview of the proposed service domain architecture is provided. The full specification of the architecture is presented in chapter 4.

A new service domain architecture is proposed in this report for two reasons:

- At present, no service domain architecture has been defined. The service domain, and the services therein, lack formal structure.

- The proposed approaches to service signalling (using application layer signalling), and BCS control in the application layer increase the capabilities and complexity of services that telecoms networks can potentially support. A new service domain architecture is needed to exploit these changes.

The existing service domain architecture

The Parlay standards provide extensive specification on how Parlay applications (telecoms services) should use the Parlay APIs. However, beyond the specification of callback interfaces, the standards do not go so far as to describe how such applications are typically structured, how software reuse can be achieved, or how the service domain as a whole should be structured. The service domain architecture implied by the Parlay standards is depicted in figure 3.18.

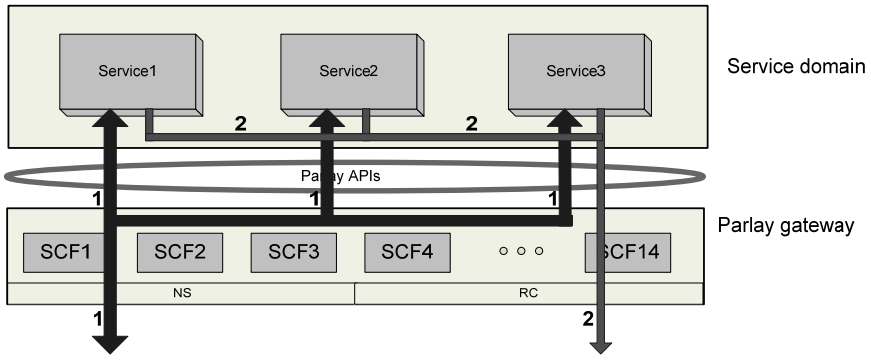


Figure 3-18: The service architecture implied by the Parlay standards

Shown in the diagram are the two types of signalling of relevance in the service domain architecture implied by the Parlay standards. These signals are explicitly identified and shown to facilitate a comparison between the existing architecture and the proposed architecture (which is introduced shortly).

The thicker arrows, collectively labelled ‘1’, represent the signals used for the invocation of services. The thinner arrows, labelled ‘2’, represent the signals sent from the service domain to the call / session layer of the network domain to control network resources (e.g. play announcements).

The service domain architecture implied by the Parlay standards is a very simple structure. Several observations, based on figure 3.18, follow:

- Since application layer signalling is not available, all services receive their invocation requests from the underlying network (signal 1), regardless of whether the invocation request originated in the network or by the terminals. Therefore, both terminal initiated and network initiated services are invoked using triggered invocation.
- Services affect their control by acting on the Parlay gateway directly (signal 2), using the Parlay API. Thus, services (and their developers) are exposed to the full complexity of the Parlay API.
- The service domain has a basic single-tier structure, and no methodology advocating software reuse has been adopted. No standardised approach is advocated for inter-component / inter-application communication in the service domain, and all services are required to be completely self-sufficient.

The proposed service domain architecture

The proposed service domain architecture attempts to solve the problems identified above. The proposed service architecture, shown in figure 3.19, is simplified to highlight its important features. The complete architecture, with a comprehensive description of its functionality, is provided in chapter 4.

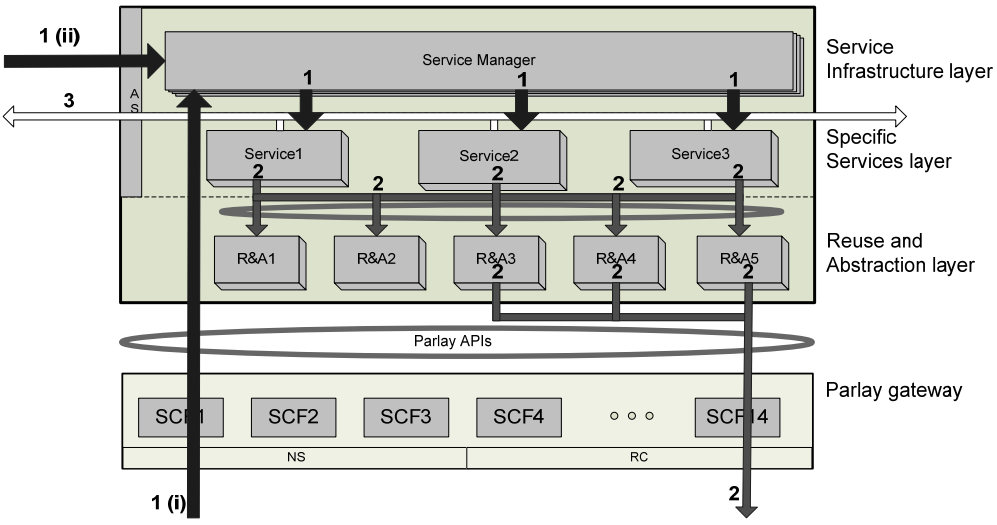


Figure 3-19: The proposed (simplified) service domain architecture

The proposed service domain architecture has a layered structure. Three layers are identified: a Service Infrastructure layer, Specific Services layer, and a Reuse and Abstraction layer.

The Service Infrastructure layer provides exactly what its name implies: infrastructure. It provides a framework for a more holistic approach to service provisioning: services don't simply respond to network invocations, and act on the network, in isolation of each other. The Service Manager ensures that services are invoked and managed by in a collaborative fashion, and with a single, common point of control and management.

The Specific Services layer contains the telecoms services, which are provisioned as applications. The Service Infrastructure layer and the Reuse and Abstraction layer both exist to serve the Specific Services layer, by either easing the task of service development or increasing the range of services that can be supported. The Specific Services layer is therefore the focal point of the service domain architecture.

Generic, reusable software components are found in the Reuse and Abstraction layer. This layer allows for software reuse, and provides a layer of abstraction between the services and the Parlay gateway, simplifying the task of service provisioning. The layered structure is further elucidated in the next chapter.

The specific signals in figure 3.19 are now discussed, and compared with those in the existing service domain structure, shown in figure 3.18.

Signal 1: Similar to the case in figure 3.18, the signals numbered 1 represent service invocations. These signals differ to the signals "1" in figure 3.18 in two ways.

- The service invocation signals can be received directly from where they were originated; from either the call / session layer (using triggered invocation, shown by "1 (i)"), or from the terminals (using application layer invocation, shown by "1 (ii)"). With the introduction of application layer signalling, terminal initiated services are appropriately invoked: using application layer invocation. In allowing service invocation using both application layer and triggered invocation, the proposed service domain architecture includes both the NS and AS adaptors.

- Services are no longer directly invoked directly by an invocation from outside the service domain: all invocations are managed by the Service Manager, which holistically controls the invocation of services.

Signal 2: In both figures 3.18 and 3.19, the signals labelled “2” represent the signals used to carry out the actions of the services to ultimately affect resource control in the underlying network. In the proposed service domain architecture, a layer of Reuse and Abstraction (R&A) modules are shown to separate the services from the Parlay gateway.

Whereas the existing approach requires that services interact directly with the Parlay gateway, the proposed approach separates the services and the Parlay gateway with an abstraction layer. By using R&A modules, services need not have any knowledge of Parlay APIs, or telecoms at all, and only need to interface with other service domain applications. For services to affect the changes they desire on the underlying network, they call high-level methods on the R&A modules, and it is the R&A modules that are responsible for interacting with the Parlay gateway to carry out the requests of the services.

The high-level methods made available to services by the R&As encapsulate sequences of commonly used Parlay messages. The R&As have well-defined northbound interfaces, offering services a standardised set of high-level APIs.

In figure 3.19, the service action signals (2) are shown to pass through the R&A layer, and then to the Parlay gateway. Note that certain R&A modules do not interface with the Parlay gateway at all; these modules simply offer services commonly required software building-blocks, and thus ease the task of service development by providing software reuse facilities thereby relieving developers of having to re-implement commonly required functions.

The R&As allow services to be highly abstracted, and allows their development to be far simpler, and within the skills of a greater number of software programmers. This approach removes unnecessary, redundant code, from services. Services no longer have to fully implement all of the functions they require; instead, they can utilise the R&As as desired. Also, since the R&As relieve the developer and the service from having deal with the Parlay APIs, unnecessary detail is hidden from the service implementation.

Signal 3: This signal is unique to the proposed service domain architecture. The proper execution of a service will sometimes require that the services signal back to the terminals, if more information is required for its successful execution. In the existing approach, the signalling from the services back to the terminals is required to be diverted through the call / session layer (along the same path service invocation was achieved in the first place, using signal 1), and use call / session signalling. In the proposed approach, services are able to signal directly to terminals, using application layer signalling (signal 3). Note that signal 3 is shown to extend through and out of the service domain to the right: this indicates the possibility for a service to communicate with another party (e.g. to invite party B to join a conference, at the application layer).

3.4.2. Integrating the proposed approaches to BCS control, service signalling and service domain architecture

The service example in section 3.3.3 highlighted two of the primary conceptual differences between the existing and proposed service provisioning environments: application layer signalling, and the relocation of the BCS controller. This example is now amended to show the additional effects of the proposed service domain architecture. Figure 3.18 depicts the consolidated example.

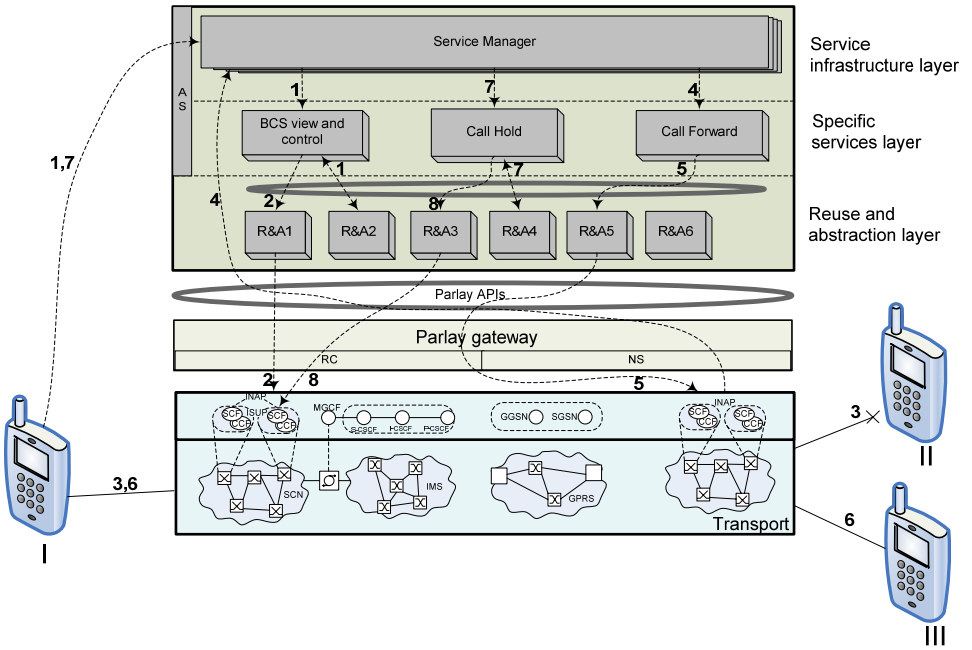


Figure 3-20: A consolidated example highlighting all major concepts

All of the signals are now discussed in detail. The main differences between this example and the example in section 3.3.3, brought about by the inclusion of the proposed service domain architecture, are highlighted.

1. **Terminal I signals** (using application layer signalling) **to the Service Manager, and not to the BCS controller directly**, to set up a new call.
The Service Manager then forwards this signal to the BCS controller. The BCS controller executes, **making use R&A2 as a building-block.**

2. **The BCS controller instructs an R&A, and not the Parlay gateway directly**, to set the end-to-end bearer connection up. R&A1 carries out this instruction by performing the necessary interactions on the Parlay gateway. The call is initiated using 3rd party call initiation.

3. An end-to-end bearer connection is cut through between terminals I and II.

Variant #1: Terminating side uses Call Forwarding

3. An end-to-end bearer connection is attempted to be cut through between terminals I and II. However, at the side of terminal II, a network event is detected.

4. **Intelligence in the terminating exchange** (in the call / session layer of the network domain) notifies the Parlay gateway of the network event, and the Parlay gateway signals **to the Service Manager, and not to the Call Forward service directly**, to process the network event. The Service Manager forwards this signal to the Call Forward service. Note that the Parlay gateways and the application servers serving terminals I and II are most probably physically distinct entities.

5. **The Call Forward service instructs an R&A, and not the Parlay gateway directly**, to complete the call. The R&A module then carries out this instruction, performing the necessary interactions with the Parlay gateway to get the terminating exchange at terminal II to terminate the call on terminal III.

6. An end-to-end, bearer connection is cut through between terminals I and III.

Variant #2: Originating side uses Call Hold

7. **Terminal I signals to the Service Manager, and not to the Call Hold service directly**, that it wishes to place the active call on hold. The Service Manager then forwards this request to the Call Hold service. The Call Hold service executes, **making use of R&A7 as a building-block.**

8. **The Call Hold service instructs an R&A, and not the Parlay gateway directly**, to perform the necessary call manipulations. The R&A module then carries out this instruction, performing the necessary interactions with the Parlay gateway.

3.5. A comparison of the existing and proposed service provisioning environments

The proposed service provisioning environment is fundamentally different from the existing service provisioning environment in the ways described in this chapter. In addition to these fundamental differences, many subtler differences result and permeate throughout the service provisioning environment as a consequence. Table 3.1 shows a comparison of the characteristic features of the existing and proposed service provisioning environments.

Table 3.1: A comparison of the existing and proposed service provisioning environments

	Existing service domain architecture	Proposed service domain architecture
BCS view and control		
Location	Bearer network	Service domain
Locus of control	Distributed	Centralised
Nature of control	Fragmented, isolated	Integrative, holistic
Terminal-service comms		
Path	Indirect	Direct
1st party service	3rd party invoked	1st party invoked
Protocol	Low-level, call / session oriented	High-level, service oriented
Service domain architecture		
Component hierarchy	Not layered	Layered
Reusability	Not specified	Supported
Service management	None	Centralised, holistic
Service (application) characteristics		
Source of invocation	Network	Service Manager
Destination of action	Network (Parlay gateway)	Generic service modules
Technological abstraction	Medium	High
Software reuse	None	Extensive

Looking ahead

Section 3.4 provided an overview of the proposed service domain architecture, and a brief description of its layers and components. Chapter 4 provides a comprehensive description of the proposed service domain architecture, and describes its various features in detail.

Chapter 4

A STATIC VIEW OF THE PROPOSED SERVICE DOMAIN ARCHITECTURE

In section 3.4.1 a simplified service domain architecture was presented, which highlighted the major architectural features of the design. This chapter presents a complete description of the service domain architecture, and describes the function of every component. The view provided in this chapter is a *static* one; the physical and logical structure of the architecture is discussed without considering its dynamic behaviour. After the presentation of certain critical implementation issues (chapter 5), a dynamic view of the architecture is provided in chapter 6, illustrating its active operation and the dynamic interaction among components.

4.1. Architectural overview

Figure 4.1 shows the proposed service domain architecture, within its operational environment, which includes the Parlay gateway, a terminal and a transport network.

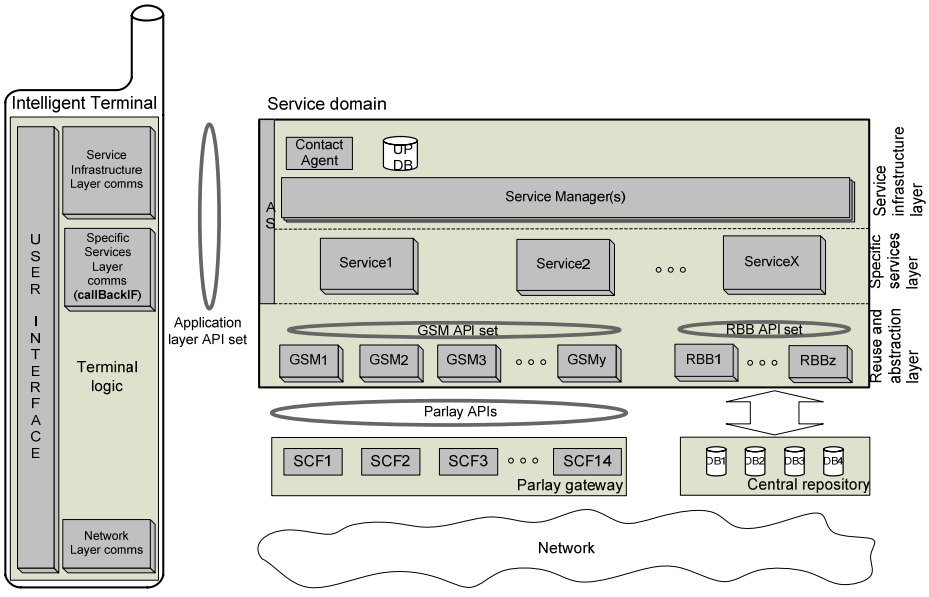


Figure 4-1: Basic architectural features of the proposed service domain architecture

In the top right of the diagram is the service domain, employing the proposed service domain architecture. The service domain is the focal point of the service provisioning environment, and the subject of this chapter. On the left of the diagram is a (large) intelligent terminal, with its large size emphasising that its functionality spans all three layers. Within each horizontal layer, elements in the terminal are able to communicate with corresponding elements in the service domain or network. This idea will become clearer in chapter 6, when the dynamic view of the architecture is presented.

The overview of the layered structure of the service domain provided in section 3.4.1 is now expanded. As previously stated, the service domain is divided into three horizontal layers, in which the various components of the service domain architecture are found. The top layer, named the *Service Infrastructure layer*, provides the structural framework under which the components of the other two layers operate. The Service Infrastructure layer is the architecture's nerve centre, and provides the architecture with management and control capabilities. The Service Infrastructure layer is central to the dynamic operation and behaviour of the architecture, as will be shown in chapter 6. The contents of the Service Infrastructure layer are provided by the operator of the service domain architecture.

The second layer from the top, named the *Specific Services layer*, contains the various telecoms services (provisioned as applications). It is ultimately for this layer that the other two layers, and the service domain architecture as a whole, exist. The contents of the Specific Services layer are provided by service developers.

The lowest layer is named the *Reuse and Abstraction layer*. In section 3.4.1, the Reuse and Abstraction layer was shown to be populated by Reuse and Abstraction (R&A) modules. The stated functions of the R&A modules was to provide an abstraction layer between the services and the Parlay gateway, and to simplify service development by implementing commonly required functions as reusable building-blocks. In figure 4.1, the generic R&As have been differentiated into two sets of more specific modules. The Service Infrastructure layer is now shown to contain Generic Service Modules (GSMs) and Reusable Building-Blocks (RBBs), both of which are direct aids to the services.

The objective of the GSMs is to provide an abstraction layer between the Parlay gateway and the Specific Services layer. The GSMs are responsible for all the interfacing with the

Parlay gateway, using the Parlay APIs, and relieve the services from this task. The RBBs allow the services in the Specific Services layer to implement only those functions which are unique to that specific service, and 'outsource' all common functions to the RBBs. The contents of the Reuse and Abstraction layer are provided by the operator of the service domain architecture.

In the following sections, the various components of the service domain architecture are discussed.

4.2. Services

The various services are shown in the Specific Services layer. These services represent the focal point of the service domain architecture, since it is the services that justify the existence and the need for the architecture in the first place. The service domain architecture is the means to the end of providing the most expedient environment possible for these services; for their provisioning, invocation and execution.

The objectives of the services are the same as those described in the Parlay standards; the difference is how the services' objectives are achieved, which is determined by the service domain architecture. Examples of typical services include the "Conference Call", "Abbreviated dialling", and "Call Hold" services.

4.3. Aids to services

There are two primary aids to services shown in figure 4.1, both of which are located in the Reuse and Abstraction layer: RBBs and GSMs. The objective of both of these components is to simplify the development and implementation of services, through reuse and abstraction.

4.3.1. Reusable Building-Blocks (RBBs)

In the present Parlay standards, no provision is made for software reuse. Thus, even if two services are similar in nature and function, each service has to implement all of its required objects and methods for itself. This approach is unnecessary, and inefficient.

Both the RBBs and the GSMs assist in software reuse, each through different mechanisms: RBBs are free-standing building-blocks that offer services commonly required functions, and GSMs simplify interaction between the services and the Parlay gateway by encapsulating sequences of commonly used Parlay messages into high-level methods.

RBBs are small building-blocks of commonly required functions and logic that are found to be frequently required by services. Having commonly used software components located in the RBBs allows numerous services to utilise these pre-packaged building-blocks, relieving each service from having to re-implement standard functionality. Instead of the services having to implement all of the methods they require, they are able to make calls to methods already implemented in the RBBs. This renders the services simpler, and through reuse, makes the services more efficient.

RBBs can provide telecoms specific services, such as number translation and service management support, or non-telecom services, such as database look-up functions, allowing services to outsource all of their database interfacing and manipulation tasks to an RBB. Other RBB examples are presented in section 5.3. Note that RBBs are by no means central or integral to the proposed service provisioning environment or its core operation. Their presence is proposed and discussed to illustrate the extensibility of the proposed service provisioning environment, and to illustrate where such modules could be logically thought to reside, if required.

All RBBs provide a standardised, well-defined northbound interface. The set of all the methods offered by all of the RBBs comprise the RBB API set, discussed in section 4.4.2.

4.3.2. Generic Service Modules (GSMs)

GSMs serve two purposes. The first purpose of GSMs is to provide an abstraction layer between services and the Parlay gateway, relieving services of having to use the Parlay APIs. The second, much like RBBs, is to enable software reuse by providing services with reusable functions and logic.

The present Parlay standards describe services which interface with the Parlay gateway directly, utilising the Parlay APIs. The number of SCFs and the richness of methods of each SCF make the Parlay API very powerful. [8] For example, if a service needs to check the status of a terminal, it has to perform all of the signal interactions with the appropriate SCF in the Parlay gateway itself to achieve this. This requirement places a burden on the services, as they will have to implement many detailed and complex Parlay message sequences to accomplish simple tasks, and requires that service developers have the necessary skills and knowledge to use the Parlay APIs effectively. GSMs solve this problem.

GSMs enable software reuse by encapsulating commonly required Parlay message patterns into pre-packaged modules. This functional encapsulation and abstraction is possible due to the recognition of recurring patterns of messages to and from the Parlay gateway needed to achieve simple, generic operations. The use of GSMs allows abstraction on two levels [8]:

- Identity abstraction. This is an abstraction brought about by hiding the identity of the Parlay computational objects.
- Logical abstraction. This abstraction is brought about by grouping Parlay methods, to provide a new simpler method.

Using GSMs, services are not required to communicate with the Parlay gateway directly. Instead, high-level, abstracted method calls can be made by the services on the GSMs, and these modules execute the required interactions with the Parlay gateway, using the Parlay APIs. A single GSM method often results in the exchange of multiple messages between the GSM and the Parlay gateway, which the service would have to implement otherwise. By relieving the service developer from requiring detailed knowledge of the Parlay gateway, the skills required of service developers are reduced, which increases the size of the set of developers technically able to develop services for telecoms networks.

For example, a “Conference Call” GSM would offer services commonly required functions used to establish and control a multimedia conference, and would predominantly interface with the Parlay MPCC SCF. Other GSM examples are presented in section 5.2.

All GSMs provide a standardised, well-defined northbound interface. The set of all the methods offered by all of the GSMs comprise the GSM API set, discussed in section 4.4.2.

4.4. API interfaces

Figure 4.1 identifies four API sets:

- The Application layer API set
- The GSM API set
- The RBB API set
- The Parlay APIs

Whereas the GSM and RBB API sets offer interfaces internal to the service domain, the Application layer API set and the Parlay APIs provide interfaces that operate between the service domain and other objects external to the service domain. The Application layer API set, GSM API set and the RBB API set are unique to the proposed service provisioning environment.

This section provides a qualitative overview of the four API sets identified in figure 4.1. Chapter 5 is used exclusively to further describe the API sets, focusing on aspects of a technical nature.

4.4.1. Application layer API set

The Application layer API set defines and standardises all communication that uses the application layer signalling channel.

In section 2.1.3, it was stated that proposed service provisioning environment would have all service logic located centrally, in the service domain, and no service logic would reside in the terminals. The terminals are only required to perform user interaction, and, based on the user's service request, construct and execute the appropriate method call corresponding to the user's service selection. The execution of the service occurs entirely in the service domain, remote of the terminal.

Locating all service logic centrally, in the service domain, determines the nature of the communication application layer signalling is required to support. While application layer signalling is required for the interaction between the terminal and service logic (e.g. for service invocation), it is not used for the execution of the service logic itself, since this takes place in the service domain exclusively. The type of communication application layer signalling supports has consequences for the nature of the Application layer API set.

The signals in the Application layer API set can be divided into two categories:

- Signals sent from terminals to the service domain. Terminals signal to:
    - The Service Manager for service invocation
    - Specific services either for service management, or to provide additional information during service execution.
- Signals sent from the service domain to terminals. Specific services signal to the terminals if additional information is required during service execution.

The Application layer API set consists of all methods offered by the terminals to the service domain, and all methods offered by objects on the service domain to the terminals. The standardisation of the application layer signalling channel through the Application layer API set has the following consequences:

- Terminals are independent of proprietary and unique implementations of services in the service domain, since the interface provided by the service domain is standardised. The Application layer API set dictates to terminals how to construct all method calls to the service domain by fully specifying the arguments, parameters, data types etc. that should be used.
- Services are independent of proprietary and unique implementations of specific terminal logic, since the interface provided by the terminal is standardised. The Application layer API set dictates to relevant objects in the service domain how to construct all method calls to terminals by fully specifying the arguments, parameters, data types etc. that should be used.
- Terminals, services and the service domain are made network-independent, since the Application layer API set is independent of the technological characteristics and protocols used in the underlying network.

Since the Application layer API set fully defines all communication that is permitted on application layer signalling channel, it plays a significant role in determining the

capabilities of the services that can be successfully provisioned, and the success of the service provisioning environment as a whole.

The Application layer API set is further described in section 5.1.

4.4.2. GSM and RBB API sets

Every GSM or RBB has numerous functions that it offers to services, where each function has an associated method call. The set of all method calls offered by the GSMs and RBBs to services is thus large. If proprietary interfaces were used by each GSM or RBB, service developers would be required to have knowledge of the implementation of every GSM or RBB that could potentially be utilised, which would prohibit effective and efficient use of the full range of GSMs and RBBs. The GSM and RBB API sets solve this problem.

The interfaces offered by all GSMs and RBBs are well-defined and standardised. The set of methods offered by all of the GSMs constitute the GSM API set, and the set of methods offered by all of the RBBs constitute the RBB API set.

The GSM and RBB API sets are further described in section 5.2 and 5.3.

4.4.3. The Parlay APIs

The Parlay APIs provide the service domain with a set of well-defined, standardised methods that can be used to affect control on the underlying network, and were described in chapter 2. The Parlay APIs are standardised by The Parlay Group.

The use of the Parlay APIs is a premise of this research project, although the proposed service provisioning environment is applicable to most OSA approaches to service provisioning, including the Parlay approach.

4.5. Service infrastructure components

4.5.1. Service Manager

The Service Manager is a defining feature of the proposed service domain architecture, and is the most influential of all the components on its dynamic character.

The Service Manager has a number of functions. The Service Manager:

- Handles service invocation
    - Prepares for the invocation of all services to which the user subscribes.
    - Receives all service invocation requests, for both terminal initiated and network initiated services.
    - Processes the service invocation request.
    - Invokes the appropriate service.
- Shields the services from the protocols used in the invocation signal.
- Provides a single point of holistic service control and management.

Each of these functions, and how the Service Manager implements them, is explained in the paragraphs that follow.

Service Manager instantiation

A new Service Manager is instantiated for a user at the beginning of every service session, and lasts for only the duration of the service session. A service session is defined as the period in which a terminal actively engages the resources of the network. For example, a conference call, or an electronic message transfer, requires the use of a service session. The Service Manager is initially instantiated, and ultimately destroyed, by the Contact Agent, described in section 4.5.2.

Each and every user has a user profile, stored in the *User Profile database*, which contains the details of which services the user is subscribed to. When the Service Manager is instantiated, it is loaded with the profile of the user the Service Manager is

serving. The Service Manager that serves a user is unique to that particular user, and distinct from all other Service Managers.

Trigger setup

After instantiation, the Service Manager prepares to receive invocation requests for any of the services to which the user is subscribed, based on the user's subscription profile: this is referred to as setting up the service 'triggers'. Only services for which triggers have been set can be invoked by the Service Manager. The services a user subscribes to could include both terminal initiated and network initiated services. Trigger setup for terminal initiated and network initiated services differs.

The arming of triggers for network initiated services (where the invocation request originates in the network) is the same as that used in the present Parlay standards, using event notifications. All that is required of the Service Manager to setup the service triggers for network initiated services is to instruct the Parlay gateway to setup the necessary event notifications.

The arming of triggers for terminal initiated services (where the invocation request uses application layer signalling) requires the development of a new approach, since application layer signalling is unique to the proposed service provisioning environment. application layer service invocations do not pass through the Parlay gateway, so the well developed network notification mechanism employed by Parlay can not be used. A similar mechanism is required for the invocation of terminal initiated services, which responds to application layer signalling.

The Service Manager itself provides this mechanism. The Service Manager makes the decision of whether the user should be allowed to have the requested service invoked, based on the user's subscription profile. Thus, the triggers for terminal initiated services are automatically setup when the Service Manager is instantiated with the user's subscription profile.

Service invocation

All service invocation requests are sent to the Service Manager, and never directly to the service. It is the function of the Service Manager is to process the service invocation request, and invoke the appropriate service. The tasks that the Service Manager performs in processing the request depend on whether the invocation was a application layer invocation request for a terminal initiated service, or a triggered invocation request for a network initiated service.

For terminal initiated services, the terminal explicitly requests the specific service that should be invoked. Based on the user's selection, the terminal constructs the appropriate service invocation request, and requests that the Service Manager invoke that particular service specified by the request. If, upon receiving a application layer invocation request for a terminal initiated service, the Service Manager determines that the user does not subscribe to the service, it will not honour the request, and the service will not be invoked. If it is determined that the service is subscribed to, the Service Manager will then invoke the requested service.

For network initiated services, it is the Service Manager determines the appropriate service to invoke. The event notification sent to the Service Manager from the Parlay gateway specifies only the network event that occurred. The Service Manager has to determine which service should be invoked, based on the type of network event (and other information contained in the notification). For network initiated services, the Service Manager does not have to check whether the service requested by an invocation is actually subscribed to. A notification from the Parlay gateway of a network event automatically implies that the service is subscribed to, since the Service Manager purposely established that network notification upon instantiation.

Thus, the processing the Service Manager performs between receiving an invocation signal and invoking the appropriate service is different for terminal initiated and network initiated services.

- For application layer service invocation requests (from a terminal), the Service Manager needs to determine whether the service is subscribed to, but does not need to determine the appropriate service to invoke.

- For triggered service invocation requests (from the network), the Service Manager does not need to determine whether the service is subscribed to, but does not need to determine the appropriate service to invoke.

Since services are abstracted from their original invocation request by the Service Manager, both terminal initiated and network initiated services utilise the service domain architecture in exactly the same manner. Thus, the invocation and execution of all services is identical.

Service shielding, control and management

Service invocation requests can originate from a number of different technological domains, e.g. the call / session layer of the network domain, web services etc., and invocation requests from each of these domains use a different signalling protocol. Since all service invocation requests are handled by the Service Manager, services are shielded from the technical characteristics of the invoking method, which simplifies the development and implementation of services. The Service Manager provides an adapting function; invocations from different domains, using different protocols, are converted to a common and standard format, which is used to invoke the appropriate service.

In the existing approach, services operate completely independently of each other, since they each respond to service invocations from independent sources. This approach is ad hoc, with no central point of service control and management, and allows the undesirable possibility of service interaction. By invoking all services using the Service Manager, all services are controlled and managed by a centralised point of control.

4.5.2. Contact Agent

The Contact Agent plays a very simple, yet essential, role at the beginning and the end of every service session. The Contact Agent instantiates the user's Service Manager at the beginning of every service session, and destroys is at the end of every session.

Initiation of a service session

Before a service session can start, a Service Manager, incorporating the user's subscription profile, needs to be instantiated. This is performed by the Contact Agent. Consider 'user X'. The Contact Agent instantiates the Service Manager for a user X in either of two scenarios, depending on whether user X elects to use a service, or is invited to use a service by another party:

- User X wishes to use a service, and user X's terminal contacts the Contact Agent, indicating that a service session is required. For example, user X might start a video conference (and subsequently invite other parties to join the conference).
- The network notifies the Contact Agent that user X has been invited to use a service by another party, indicating that a service session is required. For example, another party might attempt to establish a video conference with user X.

Each and every user that requires the use of the network's resources requires the setup of their own service session and Service Manager. For example, every user in a multiparty call has a distinct service session and Service Manager.

Upon receiving notification that a service session is required (from either the terminal or the network), the Contact Agent retrieves the user's profile from the User Profile database, and instantiates the user's Service Manager. The Contact Agent then returns the reference of the Service Manager to the user's terminal, independent of whether the user initiated the service session, or was invited to join a service session. The service session for the user is now deemed to have begun.

Prior to the initiation of a service session, the only object in the service domain a terminal has (or can obtain) a reference to is the Contact Agent, and can thus only signal to the Contact Agent. During a service session, a terminal only signals to the Service Manager. The Contact Agent is independent of any user, and is always instantiated, ready to be contacted to set up Service Managers appropriately.

Termination of a service session

The second and final function of the Contact Agent is to terminate the service session, which entails destroying the user's Service Manager. The Service Manager notifies the

Contact Agent when the service session has been completed, and the Contact Agent terminates the session.

The Service Manager, the User Profile database, and the Contact Agent, constitute the *Service Infrastructure layer*. The name of the layer is apt: the Contact Agent and the Service Manager provide the framework for the deployment, invocation and execution of the services, and form the logical infrastructure of the service domain architecture.

The Service Manager and Contact Agent are further discussed in section 5.3, with reference to the Application layer API set.

4.6. Databases

Two general categories of databases are identified in figure 4.1:

- The User Profile database
- Service databases

The databases are shown as many discrete structures to aid the logical understanding of the diagram. In reality, many database requirements could be consolidated, and reside on only a few databases; additionally, a network may have database functionality repeated on multiple databases throughout the network.

4.6.1. User Profile database

The User Profile database is a repository containing the user profiles of all subscribers under the jurisdiction of the service domain architecture. The User Profile database plays a simple role at the beginning of every service session, providing a user profile for the instantiation of a Service Manager. The Contact Agent is the only component that communicates with the User Profile database. The User Profile database is updated if a user changes the services to which he/ she is subscribed, in an offline process.

The User Profile database is shown to logically lie within the service domain; physically, however, it is located in a physical database external to the application server.

4.6.2. Service databases and the Central repository

Services often require a repository for the storage of service-specific information. For example, the “Abbreviated dialling” service needs to store the subscribers’ number pairs, which map the abbreviated number to a routable number. Figure 4.1 shows these service databases residing in the Central repository.

Service databases are updated whenever the service settings or parameters of a subscriber are changed. The configuration of service settings is referred to as “service management”, and is further discussed in section 7.1. For example, a subscriber may add a new number pair for the Abbreviated dialling service. Service databases are updated in a real-time, online process. Service management for the Abbreviated dialling service is demonstrated in section 7.2.

4.7. Functional context of architectural components

Figure 4.1 showed a logical view of the service domain architecture, highlighting the client – peer layering structure employed in the service domain. In this section, the service domain architecture is presented using a different approach, which highlights the functional and spatial structure of the service domain architecture, and the service provisioning environment as a whole. The alternate approaches to the decomposition of the service provisioning environment emphasise important aspects in different ways.

The functional and spatial perspective of the service provisioning environment forwarded in this section utilises a network-neutral framework for its presentation. The network-neutral framework that is used is that proposed by Hanrahan [7].

The framework consists of horizontal layers and vertical domains. Each layer has characteristic functionality that is not generally found in another layer. Functional entities do not straddle layers, while physical entities may do so. Vertically, technical functional domains are found. A functional domain has a related set of distinctive technical functions, which are performed in one or more layers. [7]

For more information on the definition of the neutral framework, the horizontal layers or the vertical domains, refer to [7].

Figure 4.2 shows the proposed service provisioning environment inserted in the network-neutral framework. The graphical representation of the service provisioning environment shown in figure 4.2 illustrates the same architecture that was presented in figure 4.1, but from a functional and spatial perspective.

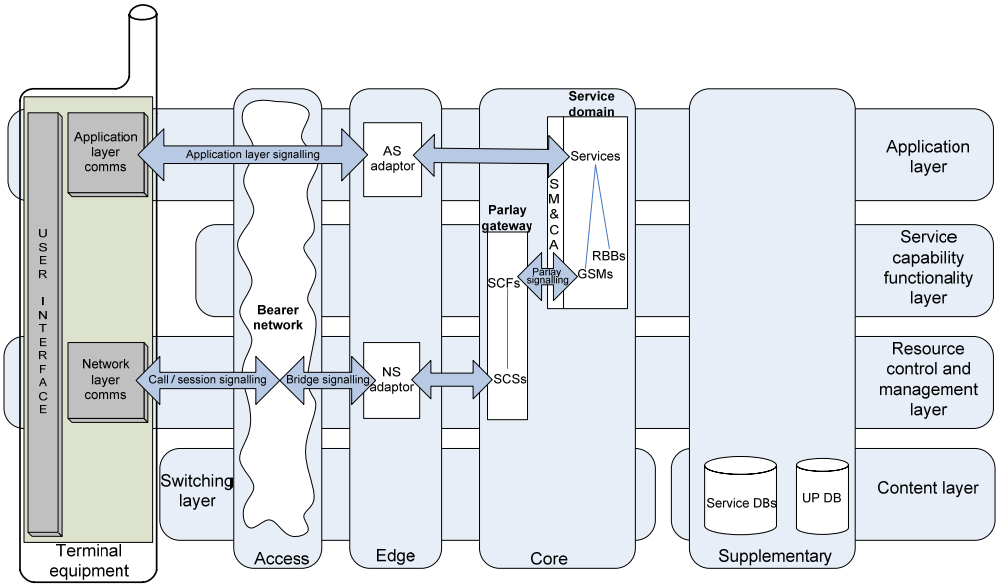


Figure 4-2: The proposed service provisioning inserted in a neutral framework

A discussion of the salient features of figure 4.2 follows. The first, most striking (and easily overlooked) observation, concerns the designations of what constitutes the Access domain and what constitutes the Core domain. While, ordinarily, the bearer network would be shown to be located in the Core domain, it is now shown to be the exclusive element in the Access network.

Instead, the Parlay gateway and the service domain are located in the Core domain. In existing service environments, these elements would typically be thought of as being peripheral and supplementary in nature. The reason for this classification is that the proposed service provisioning environment is service-centric. That is, the service provisioning environment's primary objective is to provide services to users, and thus places the elements associated with service provisioning at its heart, in the Core domain. In contrast, existing approaches consider the bearer network to be the primary focus of the telecoms environment, and place the bearer network in the Core domain.

Since the service domain and the Parlay gateway are shown as Core domain elements, and the bearer network as an Access domain element, the signalling adaptors are shown in the Edge domain, and provide the link between the Access and Core domains.

The NS adaptor, which adapts signals originating in the network for processing by the Parlay gateway, is shown in the Resource control and management layer because the intention of signalling to the service domain elements is to achieve bearer network resource control. The AS adaptor, which adapts the signals originated by terminals for assimilation by the service domain, is shown in the Application layer due to the high-level nature of the communications processed by the AS adaptor.

The service infrastructure components of the service domain, viz. the Service Manager and the Contact Agent, are shown in the Application layer and the Service capability functionality layer, since they receive signalling from the Parlay gateway (located in the Service capability functionality layer), and from the terminal, using application layer signalling.

The services, found in the service domain, are shown in the Application layer. According to [7], the Service capability functionality layer contains functionality that is generic to a number of services and applications; since the objectives of the RBBs and the GSMs are to serve the services, they are shown in this layer.

Figure 4.2 also identifies the four categories of service signalling. Application layer signalling is shown in the Application layer, between the terminal equipment and the AS adaptor, call / session signalling is shown between the terminal and the network, bridge signalling is shown between the bearer network and the NS adaptor, and Parlay signalling is shown between the Parlay gateway and the service domain.

Looking ahead

This chapter provided a qualitative understanding of the components and API sets in the proposed service domain architecture. Before the dynamic operation and behaviour of the architecture can be presented (chapter 6), additional technical detail of certain components is required. Chapter 5 provides detailed specifications of critical components, showing their physical implementation and use of the API sets.

Chapter 5

CRITICAL IMPLEMENTATION ISSUES

All of the components in the *Service Infrastructure layer* (i.e. the Service Manager and Contact Agent) and in the *Reuse and Abstraction layer* (i.e. the GSMs and RBBs) are unique to the proposed service domain architecture. These components need to be provided by the service domain operator (or the network operator) in a typical deployment. The implementation detail and technical characteristics of these critical components provides the setting in which third party application developers develop services.

In this chapter, basic implementations of various critical components are presented. In chapter 6, their dynamic operation is illustrated. Examples of services developed in the proposed service provisioning environment, which rely on the technical specifications of the critical components presented in this chapter, are presented in chapter 7.

5.1. Service infrastructure components and the Application layer API set

The service infrastructure components are those components residing in the Service Infrastructure layer, namely the Service Manager and the Contact Agent. This section provides a full definition of the Application layer API set, and the implementations for all objects that use application layer signalling.

5.1.1. Overview

The Application layer API set defines all permitted communication on the application layer signalling channel, between the terminal and various objects in the service domain. There are three categories of signalling that make up the Application layer API set, shown in figure 5.1.

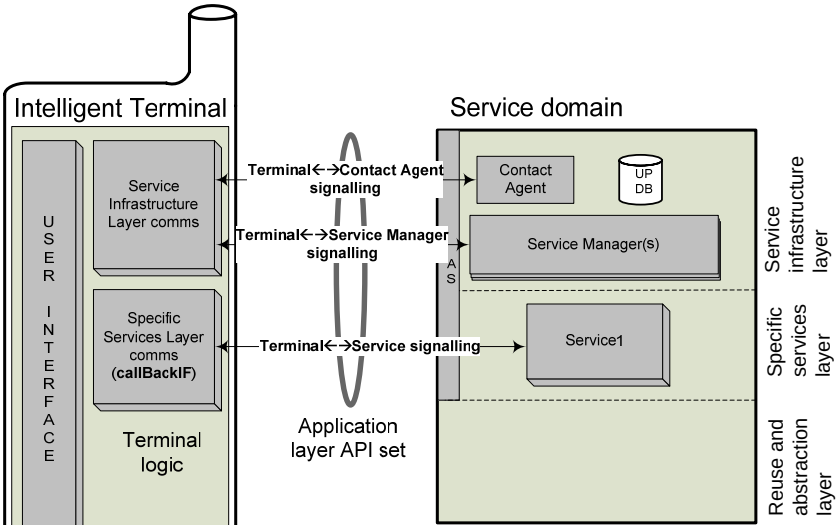


Figure 5-1: The Application layer API set: 3 categories of signals

Terminals ←→ Contact Agent signalling. This signalling is only required at the beginning of every service session, and is used instantiate the appropriate Service Manager for the user.
Terminal ←→ Service Manager signalling. This signalling is required throughout the service session. This signalling is used for the management of services, and for the invocation of services.
Terminal ←→ Service signalling. This signalling is used during the execution of services, when services require additional information from the user for the successful execution of the service.

Signalling at the application layer is achieved through method calls on other objects. Thus, in provisioning application layer signalling, every object involved in application layer signalling provides a set of public methods on its interface, which can be called by other objects over the application layer signalling channel. It is the set of methods offered by all of the objects involved in application layer signalling that defines the Application layer API set.

The full specification of the Application layer API set is thus achieved through the specification of the interfaces offered by the terminal, the Service Manager and Contact Agent.

5.1.2. Interface diagrams

Since the terminal, Service Manager and Contact Agent only communicate with other objects over the application layer signalling channel, the specification of the public methods that these objects offer to the Application layer API set in fact provides the complete specification of their public interfaces. Therefore, defining the complete Application layer API set and defining the public interfaces of the terminal, Service Manager and Contact Agent is achieved concurrently.

Interface diagram for the Contact Agent

The interface diagram for the Contact Agent is shown in figure 5.2.

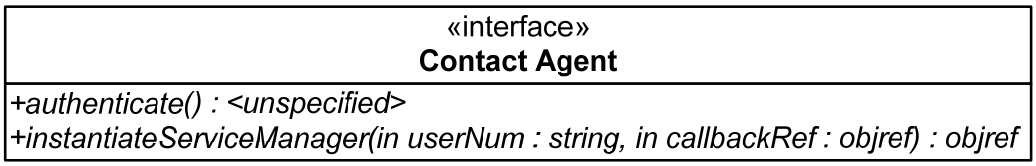


Figure 5-2: Interface diagram for the Contact Agent

The Contact Agent has two public methods. Both are offered on the Application layer API set, and are used at service session initiation.

- authenticate() is called by the terminal before it requests for the instantiation of a Service Manager. This method instructs the service domain (as represented by the Contact Agent) to authenticate itself to the terminal. The Contact Agent then requests the terminal to authenticate itself. (The exact authentication procedure is an implementation issue, and is not specified in this report. The method authenticate() represents a generic authentication process.)
- instantiateServiceManager() is called by the terminal when a service session is required, after authentication has taken place. The terminal sends its *userNum* (e.g. a unique E.164 number) and a reference to its callback interface. The Contact Agent instantiates the user's Service Manager, and returns the reference of the Service Manager to the terminal.

The Contact Agent instantiates the Service Manager using the user's subscription profile, which is retrieved from the User Profile database using the *userNum*, and with a reference

to the terminal's callback interface. (The initiation of a service session is fully described section 6.1.1.)

Interface diagram for the terminal's callback interface

The interface diagram for the callback interface of the terminal is shown in figure 5.3.

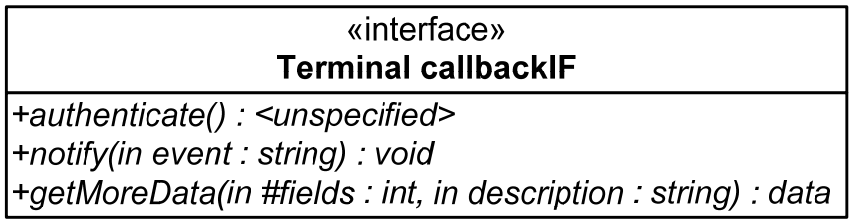


Figure 5-3: Interface diagram for the terminal's callback interface

The callback interface has three public methods, all of which are offered on the Application layer API set:

- authenticate() is called by the Contact Agent after the terminal calls a similar method on the Contact Agent. This method instructs the terminal to authenticate itself to the Contact Agent. (The exact authentication procedure is an implementation issue, and is not specified in this report. The method authenticate() represents a generic authentication process.)
- notify() and getMoreData() are methods used by services to either notify the user of an event, or obtain additional information from the user for the completion of the service. Note that both notify() and getMoreData() are independent of any particular service. (See section 6.2.2 for more details.)

Interface diagram for the Service Manager

The interface diagram for a Service Manager is shown in figure 5.4.

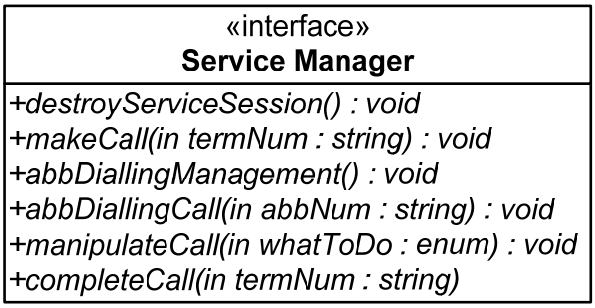


Figure 5-4: Interface diagram for a Service Manager

The methods offered by any user's Service Manager can be classified into two categories: methods that are used for controlling the service session (which are independent of any particular service), and methods that correspond directly to specific services. The only method shown that does not correspond to a specific service is *destroyServiceSession()*, which is used for service session control.

The methods that correspond directly to specific services are implemented to allow application layer service invocation, using application layer signalling. The invocation of terminal initiated services requires that the terminal explicitly request the service that the user selects on the Service Manager. Therefore, the Service Manager must be instantiated with a set of methods, each corresponding to a particular service to which the user subscribes. Thus, every user's Service Manager is different: each user's Service Manager contains the methods of only those services to which the user subscribes (in addition to those methods that are independent of any particular service, used to control the service session).

By offering methods corresponding only to those services that a user subscribes, a user will not be able to invoke a service that has not been subscribed to, since the Service Manager will not recognise the method call made by the terminal.

The Application layer API set thus includes a method for every service offered on the service domain. A Service Manager could potentially have as many methods as the service domain has services (in addition to the methods used to affect service session control), if the user subscribes to every service. This would most likely not be the case, and only a subset of the full Application layer API set would be relevant to any particular user.

The addition of a new service to the service domain requires that a new method be offered on the Application layer API set, representing that service, which could potentially introduce a scaling problem. However, the alternative of offering a single method on the Service Manager representing every service negates the potential power of application layer signalling. For example, assume a single generic invocation message of the following form is used:

serviceInvocationRequest(whichService, argument1, argument2)

The generic service invocation message would have a predetermined number of arguments (one of which being the specific service which is being requested). This reduces the flexibility inherent in application layer signalling, and places a limit on the capabilities of services that can be deployed. Therefore, the approach of allowing each service the flexibility to define the number and type of arguments it specifically needs in its own method is used in this report.

The Service Manager in figure 5.4 indicates that that user has subscribed to three services: the BCS, Abbreviated dialling, and a call manipulation service. These particular services were chosen because they provide a good representation of other services which could be offered, and because they are used in the examples presented in chapter 7.

The particular Service Manager shown in figure 5.4 has six public methods, all of which are offered on the Application layer API set:

- makeCall(), abbDiallingCall(), completeCall() and manipulateCall() are methods that correspond directly to the specific services the user has subscribed to. These would be called by the terminal when the user requires the corresponding service to be invoked.
- abbDiallingManagement() would be called by the terminal to perform service management on the Abbreviated dialling service. For example, if the user wishes to add an additional number pair, this method would be called. Of those services to which the user subscribes, the Abbreviated Dialling service is the only service for which service management is shown to be possible.
- destroyServiceSession() is the method called by the terminal for the termination of the service session. (The termination of a service session is fully described section 6.1.2.)

5.1.3. Methods of the Application layer API set

The interface diagrams presented in the previous section introduce the methods that are offered on the Application layer API set by each of the relevant objects that communicate over the application layer signalling channel. In this section, the methods are re-classified into the three categories of signals identified in section 5.1.1, namely:

- Terminal ←→ Contact Agent signalling
- Terminal ←→ Service Manager signalling
- Terminal ←→ service signalling.

Figure 5.5 shows the breakdown of the Application layer API set methods into their respective categories.

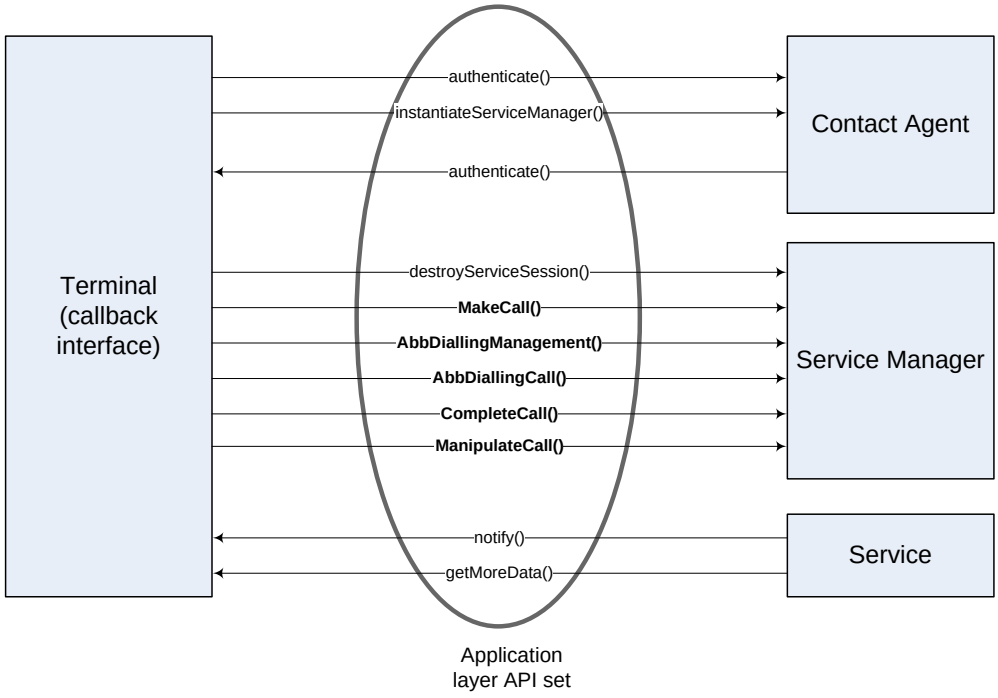


Figure 5-5: The methods of the Application layer API set

Terminal ←→ Contact Agent signalling

Terminals ←→ Contact Agent signalling is only required at the beginning of every service session. Further details of the interaction between the terminal and the Contact Agent for the initiation of a service session can be found in section 6.1.1.

Terminal ←→ Service Manager signalling

Terminal ←→ Service Manager signalling is required throughout the service session. This signalling is used for either the invocation of services, or for the management of services.

Terminal ←→ service signalling

Terminal←→ service communications are sometimes during service execution, but most often used for service management.

Whereas services can make method calls on terminals (using their callback interfaces), terminals can never make direct calls on services. This allows terminals to be independent of the implementation of any particular service.

Terminals communicate with services either through the Service Manager, or using the return parameter of methods that services call on terminals. This is the reason why no interface diagram for any service was shown in section 5.1.2.

5.1.4. A comparison of the Application layer API set and the Parlay APIs

From the discussion of the Application layer API set and the service infrastructure components, many similarities can be seen between the structure of the Parlay gateway and the structure of the service domain. Similarly, there are many parallels between the Application layer API set and the Parlay APIs.

The table below makes a number of comparisons between the Parlay gateway (and the Parlay APIs) and the service infrastructure components (and the Application layer API set). Note that SIC (seen in the proposed architecture column) stands for Service Infrastructure Components.

Table 5.1: A comparison of Parlay and the proposed architecture

	Parlay	Proposed architecture
Initial access		
Point of first contact	IpInitial (Framework)	Contact agent (SIC)
Authentication	IpAPILevelAuthentication (Framework)	Contact agent (SIC)
Session management		
Initiation of relationship	IpAccess (Framework)	Service manager (SIC)
Termination of relationship	IpAccess (Framework)	Service manager (SIC)
Service invocation and execution		
Service invocation	IpAccess (Framework)	Service manager (SIC)
Service execution	IpServiceXManager	ServiceX
Environment independence		
Interface	Service domain<>Network	Service domain<> Terminals

In the Parlay context, the client is the service application, which is served by the Parlay gateway using the Parlay APIs. In the proposed architecture, the client is the telecoms user's terminal, which is served by the proposed architecture using the Application layer API set.

In both the Parlay and the proposed architecture contexts, the client initially has only access to a single object in the domain of the server, which is its point of first contact (Framework vs. Contact Agent). Also, in both cases, authentication is required between the client and the server before access to more objects in the server's domain is granted.

In both cases, the client does not initially have direct access to the service it wishes to use (Parlay SCF vs. application layer service). Finally, in both cases, service execution occurs outside of the point of first contact (Framework SCF vs. SIC).

The standardisation of the Application layer API set allows application layer signalling between terminals and the service domain to be independent of any network-specific protocols. This allows both the terminals and the services to be implemented without requiring knowledge of the technological attributes of the underlying network. Thus, the Application layer API set and the Parlay APIs provide similar functions: both provide the service domain with a network independent view of its operational environment. The difference is that the Parlay APIs standardise the interface between the service domain and the network, whereas the Application layer API set standardises the interface between the service domain and the terminals.

There are thus many parallels between the Framework SCF in the Parlay gateway and the service infrastructure components in the proposed architecture, and between the Parlay APIs and the Application layer API set.

5.2. The Generic Service Modules and the GSM API set

This section presents the general structure of GSMs, shows how they are implemented in the proposed service domain architecture, and illustrates how services take advantage of the GSMs using the GSM API set. Specific examples of GSMs are also provided, highlighting points of a more specific nature.

5.2.1. Overview

The GSMs were developed for the following two reasons:

- Abstraction: GSMs provide an additional level of abstraction to the service, removing the need for the service to implement low-level Parlay APIs.
- Software reuse: GSMs encapsulate commonly required Parlay messages into reusable building-blocks.

GSMs reduce the method calls and instructions that the service needs to execute to achieve the required actions. Single, high-level calls on GSMs accomplish extensive and complex Parlay functionality.

Encapsulation of Parlay functionality into reusable building-blocks is possible due to the recognition of recurring patterns of messages to and from the Parlay gateway needed to achieve simple, generic operations. For example, in the case of creating a simple 3rd party call, 15 Parlay messages are exchanged between the Parlay gateway and the application providing the 3rd party call service [5]. If the service itself was responsible for directly interacting with the Parlay gateway, it would be responsible for the processing of these 16 messages. However, if a GSM offered a simple method to services called “MakeACall”, the service would only be required to implement a single method call. The 16 Parlay messages necessary to achieve this task in the underlying network would then be carried out by the GSM, freeing the service of its implementation. In addition to the computational relief GSMs offer, the method “MakeACall” is also intuitive and logical due to its high-level nature.

Parlay categorises the network functionality it offers to service developers into many distinct groupings, broadly corresponding to each of the Parlay SCFs. Following in this vein, multiple GSMs are proposed, each implementing a group of related operations. For

example, the “MakeACall” method in the above example would be offered by a GSM whose task is to control call setup, modification, take down and other call manipulations. Such a GSM is proposed in section 5.2.3, and is named the MPCC (Multi Party Call Control) GSM. Other methods the MPCC GSM offers to services include “EndCall” and “AddParty”, both of which perform operations related to the broad function of (multi party) call control.

5.2.2. Generic Service Module implementation

The practical implementation of the GSMs, and the examples presented in the next section, are largely based on the work of [8].

The architectural features of the proposed service domain architecture shown in figure 5.1 provides only sufficient detail of the Reuse and Abstraction layer for the architecture, as a whole, to be understood. The Reuse and Abstraction layer is slightly more complicated than figure 5.1 suggests, and an understanding of the operation and behaviour of the GSMs requires a more comprehensive diagram as a reference.

Figure 5.6 shows the Reuse and Abstraction layer in more detail, and other elements of the service domain that are relevant to the operational environment of the GSMs are also shown. Since the service infrastructure components (in the Service Infrastructure layer) and the RBBs have no impact on the implementation or operation of the GSMs, they are excluded from the diagram.

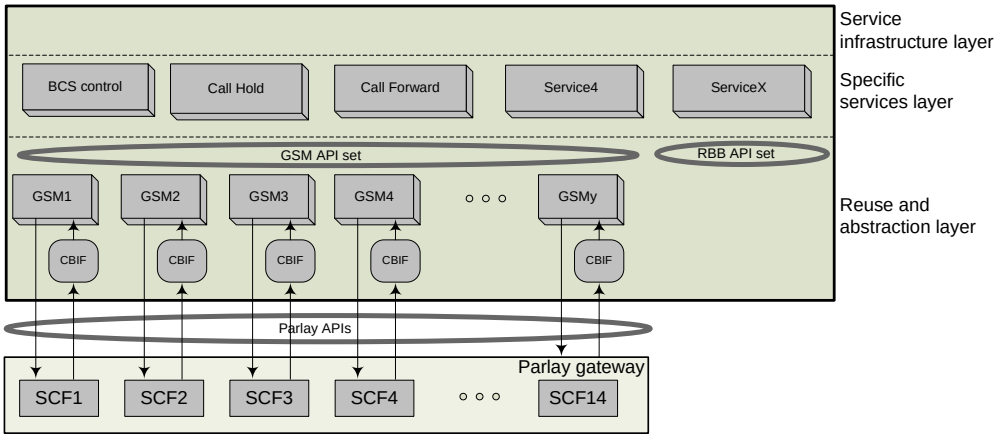


Figure 5-6: The static structure of the GSM layer in detail

The major different between the GSMs shown in figures 5.1 and 5.6 is that figure 5.6 depicts each and every GSM coupled with a callback interface. Callback interfaces serve to provide a means for the SCFs in the Parlay gateway to contact the applications in the service domain, and are a convention used throughout the Parlay standards. Whereas the GSMs are able to contact the SCFs directly, the SCFs contact the GSMs using their callback interfaces.

Each GSM is shown to be associated to a particular SCF: the grouping of functions employed by Parlay in compiling the SCFs is loosely followed by the GSMs. For example, all of the Parlay methods required by the MPCC GSM are offered by the MPCC SCF. There is thus a strong relationship between these two components. However, the MPCC GSM (and GSMs in general) is able to make use of any of the Parlay SCFs.

As will be shown by the GSM examples in section 5.2.3, the dynamic behaviour of all the GSMs follows a basic pattern, or template. Before the specific GSMs are presented in full detail, operational behaviour for a generic GSM is shown. The Interaction Diagram (ID) and the Message Sequence Chart (MSC) are two illustrative conventions that are highly effective means for depicting operational behaviour and object interaction, and are used throughout the rest of this report. To illustrate any specific object interaction, the ID is given first, which shows the interaction of the objects in the context of their logical and spatial structure, and then the MSC is shown, which provides additional detail using a more abstracted view. Figures 5.7 and 5.8 show the ID and MSC, respectively, for a generic GSM.

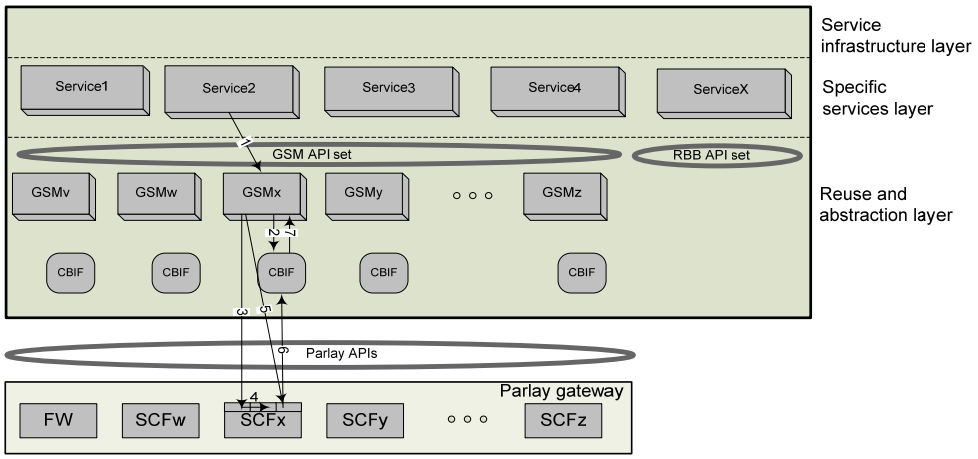


Figure 5-7: ID showing the behaviour of a generic GSM

The specific messages in the diagrams are now discussed. Although specific services, GSMs and SCFs are referred to (corresponding to those used in the diagram), the scenarios presented are generic and represent any particular service / GSM / SCF interaction.

1. Service2 calls a specific method on GSMx, requesting that it “Perform task”. After this first message, service2 is free to continue with other operations, oblivious to the operations that GSMx is required to perform on the Parlay gateway to achieve this task. Thus, the involvement of service2 in the execution of the task ends after the initiating method call (although further interaction between service2 and GSMx can be supported if required).

2. In preparation for its interaction with the Parlay gateway, GSMx instantiates a callback interface. The callback interface is used to receive communication from the Parlay SCFs.

3,4,5. In message 3, GSMx signals to an object associated with SCFx. (Each SCF can have multiple objects associated with it at any time, depending on the method calls made on it.) Message 4 shows signalling within the Parlay gateway, and message 5 shows a signal from GSMx to a different object of SCFx.

6,7. Messages 6 and 7 show communication in the reverse direction, from SCFx to GSMx. Message 6 is labelled “result”, and it represents the result of an operation performed by GSMx on SCFx (using either message 3 or 5). Note that SCFx signals to the callback interface of GSMx, and the callback interface forwards the message to the actual GSMx object.

8. Message 8 may or may not be required. The function offered by a GSMx may not return any information to service2 at all; in this case, message 8 will not be required. In other cases, GSMx may pass information back to service2 in the form of a return parameter of the method that was originally called (message 1). (In a third, more improbable case, GSMx may explicitly call a method offered by service2.)

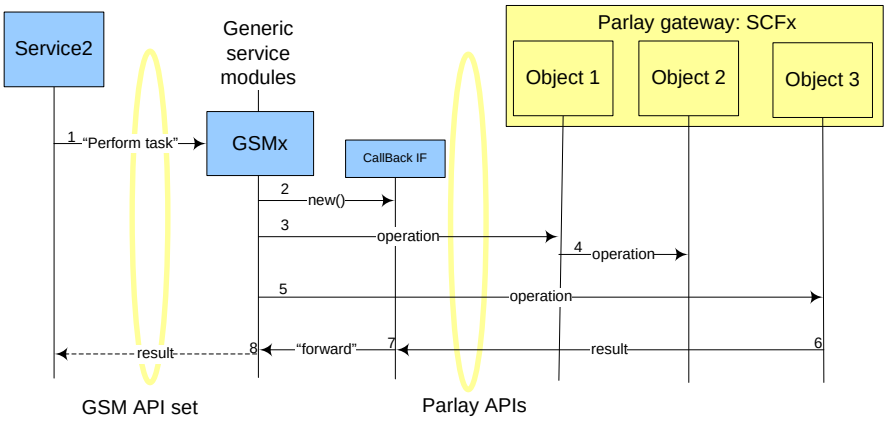


Figure 5-8: MSC showing the behaviour of a generic GSM

To summarise the key points of the generic use of a GSM by a service: the service makes a single, high-level call on a GSM for a specific operation. Following this request, the service plays no further role in the execution of this operation (besides possibly receiving the return parameter of the called method). To execute the operation, the GSM instantiates a callback interface, and proceeds to interact with the Parlay gateway, making the necessary calls to implement the operation it was called to execute. At no point is the service required to interact with the Parlay gateway.

5.2.3. Examples of GSMs

Three examples of GSMs are provided: the Framework GSM (FW GSM), MPCC GSM and the Service Infrastructure GSM (SI GSM). The FW GSM and the MPCC GSM are taken from [8], while the SI GSM is developed especially for the proposed service domain architecture.

The FW GSM was developed to provide reuse and abstraction of the functionality offered by the Framework SCF. The FW GSM was chosen for two reasons. Firstly, since it represents the functionality of the Framework SCF, it would be used by every service. Secondly, since the nature of the typical interactions with the Framework SCF allows extensive Parlay message patterns to be easily encapsulated into simple GSM methods, services using the FW GSM would be significantly simpler.

The MPCC GSM was developed to provide reuse and abstraction of the functionality offered by the MPCC SCF. The MPCC SCF is perhaps the most frequently needed Parlay

SCF, and thus the functionality offered by the MPCC GSM would potentially be the most useful and widely used.

The SI GSM is developed to facilitate interaction between the service infrastructure components (viz. the Contact Agent and the Service Manager) and the Parlay gateway, and is not intended for use by any service.

In the MSCs, the callback interfaces corresponding to the GSMs are not explicitly shown. For illustrative simplicity, they are imagined to be logically integrated into the GSM objects.

FW (Framework) GSM

The Parlay standards specify the use of a Framework SCF, which provides applications with basic mechanisms that enable them to make use of the service capabilities of the network [4]. Examples of the functions provided by the framework include authentication, authorisation, and service discovery. A detailed description of the framework can be found at [9].

The FW GSM has been developed to handle the interaction with the Framework SCF, and is thus responsible for performing tasks such as service discovery, authorisation etc. The FW GSM is especially useful since all services would utilise it, and because great computational efficiency can be realised with its use.

The FW GSM manages the creation and deletion of services:

1. Service creation. The FW GSM provides services with service lists, service descriptions, and references to new SCF managers.
2. Service deletion. The FW GSM controls the disposition of SCF managers and resources.

The interface diagram for the FW GSM is shown in figure 5.9.

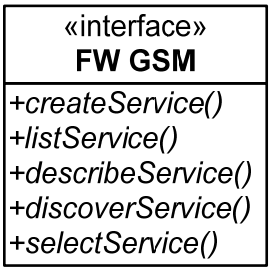


Figure 5-9: Interface diagram for the FW GSM

A discussion of the purpose of each of the FW GSM methods follows, accompanied by MSCs showing how the methods are implemented.

- createService()

This method performs the necessary steps with the Parlay gateway when a service is accessing the Parlay gateway for the first time. The MSC, showing the actions taken by the FW GSM to execute the createService() method, is shown in figure 5.10.

Before being authorised to use the Parlay SCFs, the GSM must first authenticate itself with the Framework SCF. For this purpose, the GSM needs a reference to the Initial Contact interfaces for the Framework SCF. Once the GSM has been authenticated by the Framework SCF, it can gain access to other Parlay interfaces and SCFs. This is done by invoking the *requestAccess()* method, by which the GSM requests a certain type of access SCF. [9]

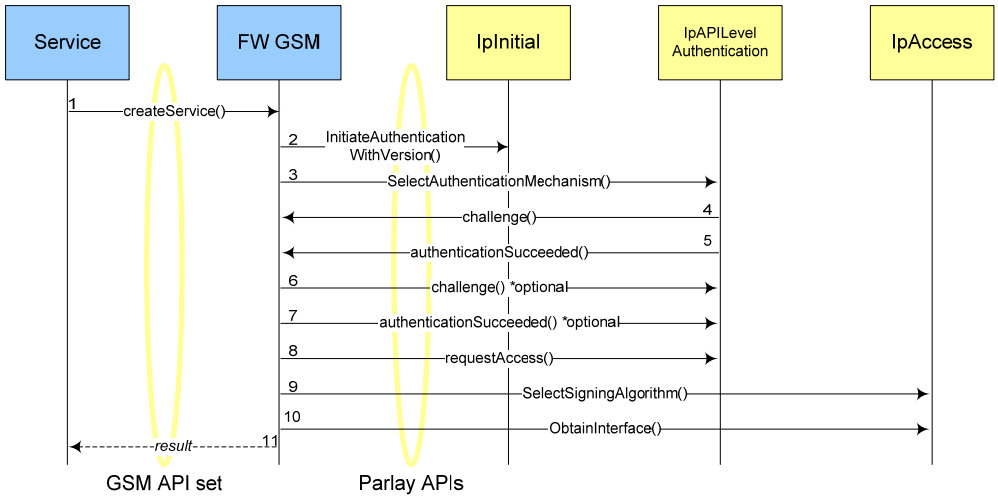


Figure 5-10: MSC for FW GSM createService() method

The messages in the MSC shown in figure 5.10 are now discussed.

1. The service executes *createService()* on the FW GSM.

Messages 2-10 constitute the standard sequence of messages used to create a service, as presented in the Parlay standards, and are taken directly from [9].

2. The FW GSM triggers *initiateAuthenticationWithVersion()*, which initiates authentication with a publicly available framework by passing the client domain's identifier, the authentication type, the framework version number and returns the Framework identifier.

3. *selectAutheticationMechanism()* is then invoked, which provides the framework with a list of authentication mechanisms the FW GSM supports. The framework selects one from the list and passes it back as the return parameter.

4. Depending on the authentication mechanism, a number of challenges are invoked on the FW GSM, which it must respond correctly to using the return parameter.

5. The *authenticationSucceeded()* is then invoked to inform the FW GSM of its success.

6, 7. Thereafter, depending on the FW GSM's policy, it might authenticate the framework with a similar process.

8. The FW GSM invokes the *requestAccess()*, which returns the reference to the framework access interface. The framework access interface provides access to other framework interfaces.

9. Before the FW GSM can use the access interface, it must provide a list of all signing algorithms it supports for use in cases where digital signature is required through the operation *selectSigningAlgorithm*(). The framework selects one from the list and passes it back as a return parameter.

10. The FW GSM invokes *obtainInterface()*, which returns a reference to the required framework interface (service discovery, in this case).

11. The FW GSM notifies the service that the operation was carried out successfully.

The FW GSM encapsulates messages 2-10 into a single method, *createService()*. Without the use of the FW GSM, the service would need to implement every message; with the introduction of the FW GSM, it now only needs to make a simple call on the FW GSM to accomplish the same result.

- listService(), describeService() and discoverService()

The Service Discovery interface of the Framework SCF is used to obtain a serviceID for the SCF of interest, as shown in the MSC in figure 5.11.

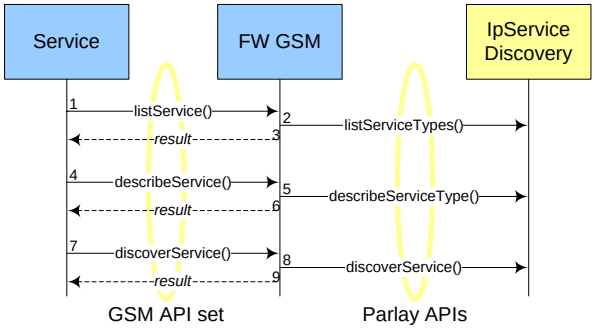


Figure 5-11: MSC for various FW GSM methods

This serviceID is obtained by performing a number of operations, which might include obtaining a list of available SCFs through *listServices()* (messages 1-3) and asking the Framework SCF to describe the service types through *describeService()* (messages 4-6), which returns properties of the services of interest. Thereafter, the application fine-tunes these properties to reflect its requirements, and passes it to the Framework SCF through *discoverService()* (messages 7-9). The Framework SCF then returns a list of serviceID matching the needs put forward by the service.

Because of the one-to-one mapping between the message calls on the FW GSM and the message calls on the Framework SCF, no additional abstraction is achieved.

- selectService()

The MSC showing the implementation of service selection is shown in figure 5.12.

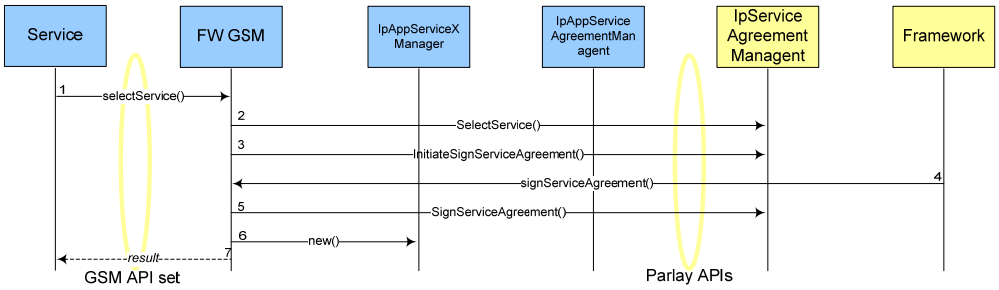


Figure 5-12: MSC for FW GSM selectService() method

The messages in the MSC shown in figure 5.12 are now discussed.

1. The service selects the required service by passing the serviceID (e.g. for the MPCC SCF) to the FW GSM.

Messages 2-6 constitute the standard sequence of messages used to select a service, as presented in the Parlay standards, and are taken directly from [9].

2-5. The FW GSM handles the service agreement signing between the application domain and the framework. A reference to the SCF manager (e.g. MPCC SCF manager) in the Parlay gateway is returned to the FW GSM.

6. FW GSM creates a corresponding SCF manager, on the application side, for the particular service it selected for use (e.g. MPCC GSM manager). A manager ID is assigned to the newly created SCF manager. The reference to the manager and the manager ID is returned to the service.

7. The FW GSM notifies the service that the operation was carried out successfully.

The FW GSM encapsulates messages 2-6 into a single method, *selectService()*. Without the use of the FW GSM, the service would need to implement every message; with the introduction of the FW GSM, it now only needs to make a simple call on the FW GSM to accomplish the same result.

MPCC (Multi Party Call Control) GSM

The MPCC GSM offers methods to services to initiate, control, manage and terminate calls with a number of parties. The interface diagram for the MPCC GSM is shown in figure 5.13.

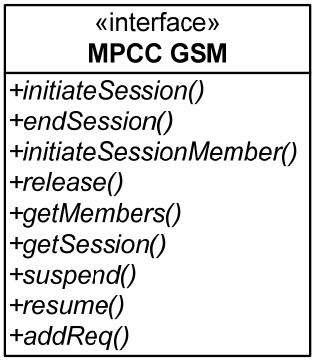


Figure 5-13: Interface diagram for the MPCC GSM

The methods *initiateSession()* and *endSession()* are offered by every GSM (besides the FW GSM), and are thus inherited from a common, parent GSM. Since no other examples of GSMs are presented in this report, *initiateSession()* and *endSession()* are shown to be implemented by the MPCC GSM.

A discussion of those methods that are unique to the MPCC GSM follows, accompanied by the MSCs showing how the methods are implemented

- initiateSessionMember() and addReq()

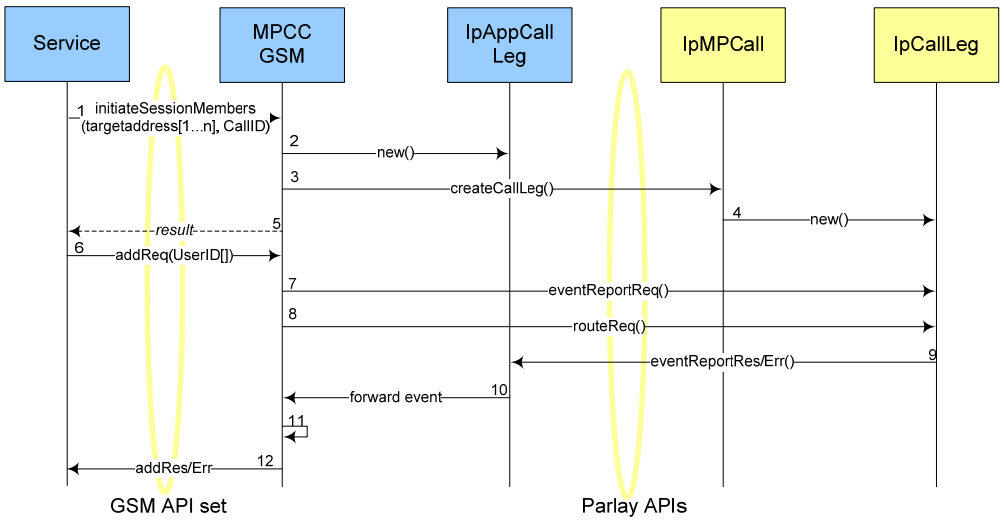


Figure 5-14: MSC for MPCC GSM initiateSessionMembers() method

In the MSC shown in figure 5.14, two logical blocks of messages are shown, each initiated by a method call by the service on MPCC GSM:

Block 1. *initiateSessionMembers()* (message 1) begins the first logical block. *initiateSessionMember()* is used to create one or more call legs, each represented by an IpAppCallLeg object. *initiateSessionMembers()* has as input parameters the target address(es) in an array and the call ID (for the call to add the users to, using *addReq()*).

Block 2. *addReq()* (message 5), begins the second logical block. This logical block that allows the addition of one or more parties to the call. The user invokes *addReq()* on MPCC GSM, which then executes messages 6 and 7 one or more times depending on the number of User IDs in the array given as the input parameter to a*ddReq()*. If an error occurs during the routing of a leg, it is forwarded to the service.

- release()

In the existing approach to service provisioning, a call can be terminated in one of two ways. Firstly, either party to the call could end the call, e.g. by hanging up. Secondly, a Parlay service could terminate the call from the application layer. Figure 5.15 shows both of these scenarios.

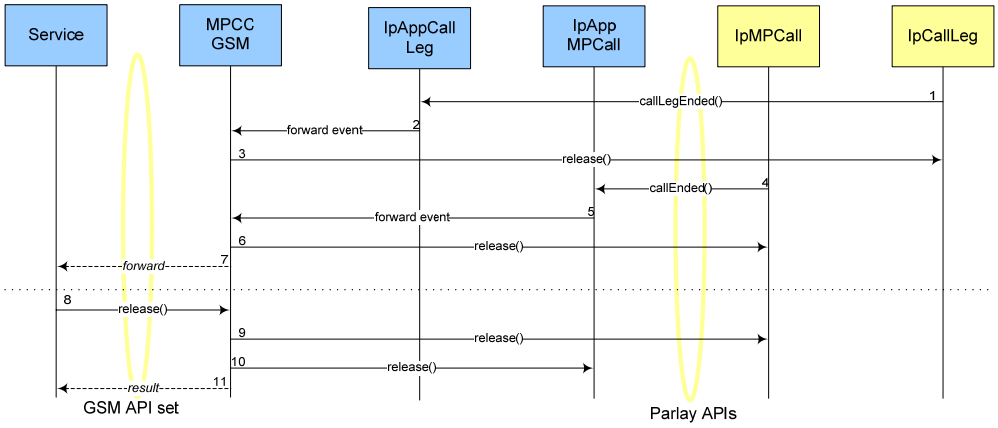


Figure 5-15: MSC for MPCC GSM release() method

Messages 1-7 show the series of messages that occur when a party to the call hangs up. IpMultiPartycall and IpMultiPartyCallLeg trigger *callEnded()* and *callLegEnded()* respectively on the GSM interfaces, which signals that the call has been terminated by a party. The MPCC GSM then issues a release on the associated call objects to free resources used for the call.

Messages 8-11 show the messages that occur when the call is terminated using a service in the application layer. *release()* is called by the service on the MPCC GSM (message 8) indicating that the call should be terminated, and the MPCC GSM then terminates the call by releasing the associated call objects to free resources with the call (messages 9 and 10).

The second scenario to call termination, where an application layer service initiates the termination of the call, has special significance in the proposed service provisioning environment. Since the proposed service provisioning environment advocates the control of the BCS in the application layer, a party terminates its involvement in a call / session by signalling to the BCS controller in the service domain that the call / session should be terminated. The call termination is then controlled by the BCS controller (which is an application layer service), and proceed using messages 8-11. In the proposed service provisioning environment, call termination using messages 1-7 would not occur.

- getMembers()

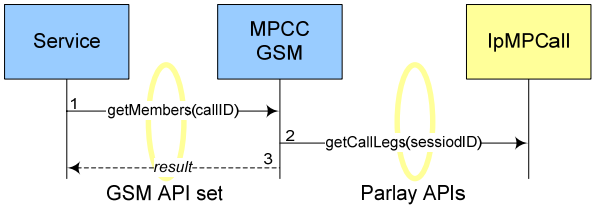


Figure 5-16: MSC for MPCC GSM getMembers() method

Figure 5.16 shows the use of *getMemebers()* to obtain the members participating in a specific call session, which returns a list of the call members.

- getSession()

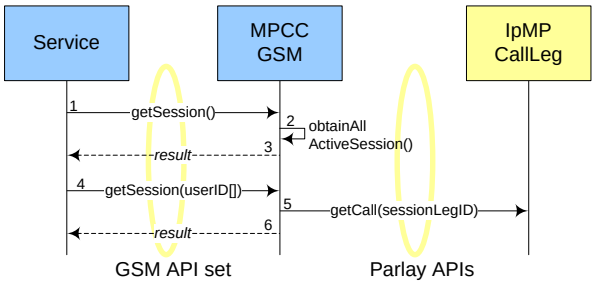


Figure 5-17: MSC for MPCC GSM getSession() method

Figure 5.17 shows the use of the *getSession()* MPCC GSM method. *getSession()* returns the callID for all active sessions on MPCC SCF. Alternatively, if an argument is provided, *getSession(userID[])* returns a callID array, matching the sequence of userIDs in userID[]. Message 4 may be performed a number of times depending on the number of user IDs supplied.

- suspend() and resume()

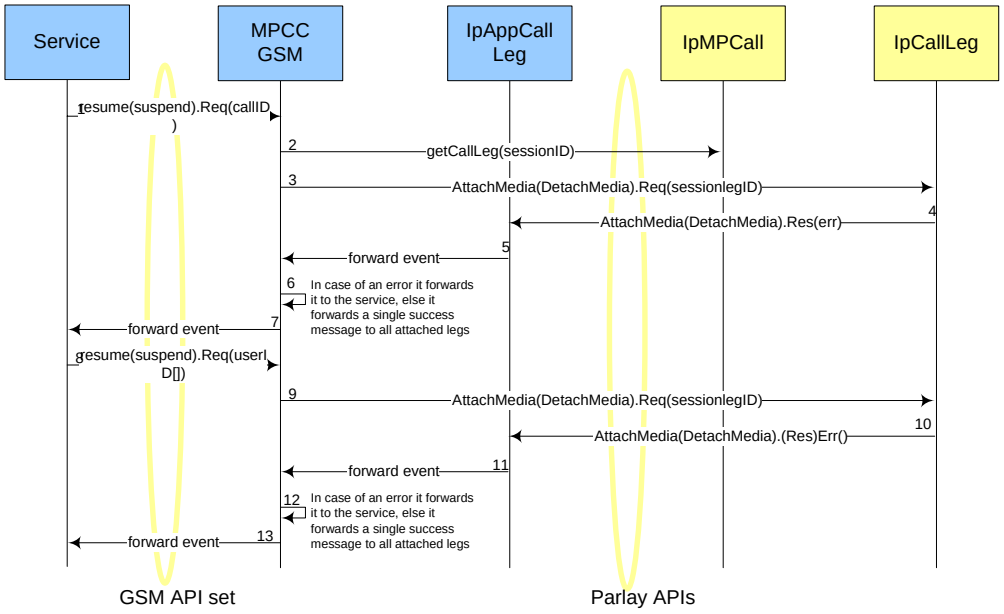


Figure 5-18: MSC for MPCC GSM for suspend() and resume() methods

Figure 5.18 shows how a service can suspend (resume) the participation of a member(s) from the call. There are two way to accomplish this. The user can invoke s*uspend(resume).Req()* with the *callID* (which implies suspension (resumption) of the call as a whole), or *userID[]* (which suspends the users specified in the array).

The former scenario is accomplished by obtaining all the call legs related to the call and detaching (attaching) them, which implies message 3 is invoked a number of times (equal to the number call legs). In the latter scenario, the MPCC GSM performs the translation of user IDs to *sessionLegIDs* and the rest of the sequence (9-13) is the same as in the former scenario (3-7).

SI (Service Infrastructure) GSM

The SI GSM is designed to abstract the service infrastructure components (viz. the Contact Agent and the Service Manager) from the Parlay gateway, thereby relieving them from having to implement Parlay API methods, callback interfaces etc. The interface diagram for the SI GSM is shown in figure 5.19.

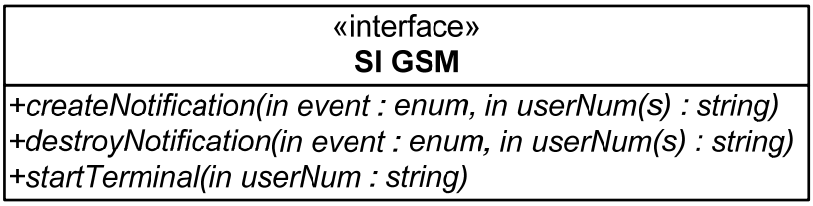


Figure 5-19: Interface diagram for the SI GSM

None of the methods shown in figure 5.19 is used during service execution; they are used in either the creation or termination of a service session. The initiation of a service session is presented in section 6.1.1, and the use of the SI GSM is described in further detail. A discussion of each of the methods follows.

- createNotification() and destroyNotification()

createNotification() creates a notification in the Parlay gateway for a network event (specified by the argument *event*) involving any of the stated users (specified by *userNums*). If the specified event occurs in the network, involving any of the specified

users, the Parlay gateway notifies the SI GSM of the event, and the SI GSM forwards the notification to the object (either the Contact Agent or the Service Manager) that originally called the method.

The notification that is created in the gateway by the SI GSM is always made in the *interrupt* mode. With the notification set to the interrupt mode, when a network event is triggered in the gateway, all processing of the session in the underlying network is paused, and full control is handed over to the service domain for further processing. Refer to [5] for further information on Parlay notifications.

The MSC showing the use of the createNotification() method is shown in figure 5.20.

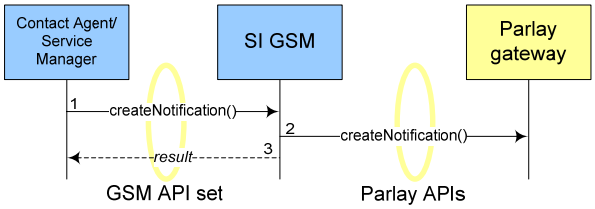


Figure 5-20: MSC for SI GSM for createNotification() method

createNotification() is used by both the Contact Agent and the Service Manager, and its use in each case is discussed separately.

The **Contact Agent** calls createNotification() on the SI GSM only once, when the service domain architecture is first provisioned. This method is never used by the Contact Agent ever again. Thus, it is never required during the instantiation or execution of any service session.

Recall that a service session can be initiated under either of two scenarios: either a user elects to use a telecoms service, or the user is invited to use a service by another party. In the former case, the terminal initiates the service session, and in the latter case the network initiates the service session.

When the terminal initiates the service session, the Contact Agent is contacted by the terminal directly. In this case, the Contact Agent explicitly is made aware of the need for a service session. However, when events in the network indicate that a service session in

the service domain is required, the Contact Agent needs a way of being notified of this. To accomplish this, the Contact Agent instructs the Parlay gateway to notify it if any subscriber is invited to use any service in the underlying network. It achieves this by calling createNotification() on the SI GSM.

The Contact Agent calls createNotification() on the SI GSM only once, when the service domain architecture is first provisioned. The *event* and *userNums* arguments of the method are set such that the Contact Agent is notified of *any* attempt to invite *any* user to use *any* service.

With the notification appropriately set up, any attempt by another party to invite a user to use any service triggers the network event. The Parlay gateway notifies the SI GSM of the event, and the user it concerns, and the SI GSM forwards this notification to the Contact Agent. Since the notification is set to the *interrupt* mode, the Contact Agent is able to control the further processing of the situation. (The Contact Agent then proceeds to establish the service session for the user by calling the startTerminal() method on the SI GSM, as explained later.)

The **Service Manager** calls createNotification() when it is first instantiated, at the beginning of every service session, to establish the appropriate notifications in the Parlay gateway corresponding to the network initiated services to which a user subscribes. Recall that network initiated services are those services whose invocations originate in the underlying network; thus, the notifications created by the Service Manager in the Parlay gateway ensure that it is notified if any network initiated service to which the user subscribes requires invocation. These notifications are also set to the interrupt mode, allowing the control of the processing of the service to be maintained in the service domain.

The Service Manager calls destroyNotification() on the SI GSM at the end of every service session, to destroy all of the notifications it created at the beginning of the service session corresponding to the network initiated services to which the user subscribes. The practical use of the destroyNotification() method is the same as the use of the createNotification() method, as shown in figure 5.20.

- startTerminal()

As described previously, the Contact Agent uses createNotification() to ensure that it is notified whenever an event in the underlying network indicates that a new service session is required. startTerminal() is another method that is used by the Contact Agent to instantiate a service session that is initiated in the underlying network.

A requirement for the instantiation of any service session (whether it is initiated by the terminal or a network event) is that the service domain and the terminal are be able to communicate at the application layer. While the terminal always has a reference to the Contact Agent (in the service domain), the service domain only maintains a reference to the terminal during a given service session. Thus, the service domain has no way of signalling to the terminal at the application layer outside of a service session.

If the terminal initiates the service session by contacting the Contact Agent directly, the service domain immediately obtains a reference to the terminal, and the service session can commence since the terminal and the service domain are able to communicate, in both directions, at the application layer. However, if the Contact Agent is notified of the need for a service session through a network event, the Contact Agent has no way of contacting the terminal at the application layer to notify the terminal of the need for a service session.

The startTerminal() method, offered by the SI GSM, enables the Contact Agent to contact a terminal at the network level. A terminal can always be contacted at the network level using its unique network identifier (e.g. an E.164 number). If the terminal can be contacted at the network level, it can be notified of the need for a service session, and can then initiate the service session by contacting the Contact Agent.

The MSC showing the use of the startTerminal() method is shown in figure 5.21.

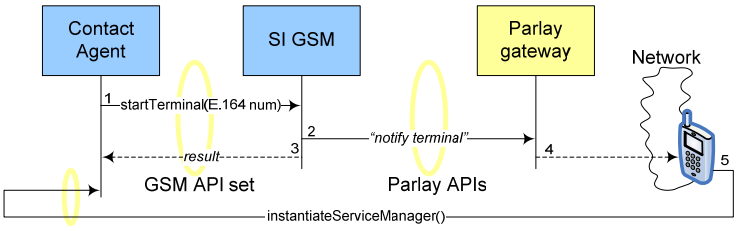


Figure 5-21: MSC for SI GSM for startTerminal() method

Messages 1-3 show the operation of the startTerminal() method. Messages 4 and 5 are not strictly part of the startTerminal() method, but illustrate the subsequent effects the method has: message 4 shows the Parlay gateway contacting the terminal at the network level; message 5 shows the terminal contacting the Contact Agent at the application layer level, requesting the initiation of a new service session.

5.2.4. Methods of the GSM API set

The GSM API set is comprised of all of the methods offered by every GSM. The GSM API set offers a well-defined interface to service developers by providing a comprehensive definition of all of the arguments, parameters, data types etc. of all of its methods.

The size of the GSM API set is determined by the range of GSMs that are implemented. Figure 5.22 shows the GSM API set where only the FW GSM, MPCC GSM and the SI GSM are implemented. Excluding the SI GSM, the GSM API set shown consists mainly of 'signals' directed from services to the GSMs; thus, components that use the GSMs do not need to offer the GSMs any public methods. The fourth, unnamed GSM, represents the set of all other GSMs which haven't been specified in this report, whose methods will also contribute to the GSM API set.

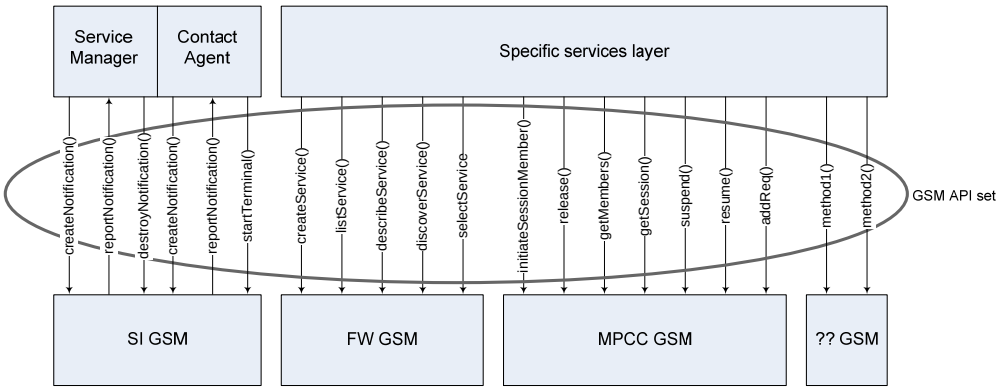


Figure 5-22: The methods of the GSM API set

5.3. The Reusable Building-Blocks and the RBB API set

This section describes the implementation of RBBs and use of the RBB API set by means of a representative example.

5.3.1. Overview

RBBs encapsulate commonly required functionality into reusable building-blocks, allowing services to implement only the functionality that is unique to that specific service, and outsource all other operations to RBBs.

Whereas GSMs interface with the Parlay gateway, offering software reuse and abstraction specifically related to the Parlay SCFs, RBBs do not interface with the Parlay gateway at all. The functionality offered by RBBs does not necessarily even have to be oriented towards telecoms; many RBBs provide generic software functionality, such as database manipulation.

5.3.2. Examples of RBBs

The “NumSearch RBB” is used to illustrate and exemplify the function of a typical RBB.

The NumSearch RBB is designed to be used for all number translation services. Its use is illustrated here through its application to the Abbreviated Dialling service, which is one of many possible services that would require number translation. Its operation in this case is as follows. It receives a subscriber’s number and an abbreviated number as its inputs. Using this data, it searches a database for the subscriber’s profile, and matches the abbreviated number with a routable E.164 number. The NumSearch RBB then returns this routable E.164 number to the service that called it (the Abbreviated Dialling service).

Figure 5.23 depicts the ID, showing the NumSearch RBB operating in the service domain, and figure 5.24 shows the associated MSC.

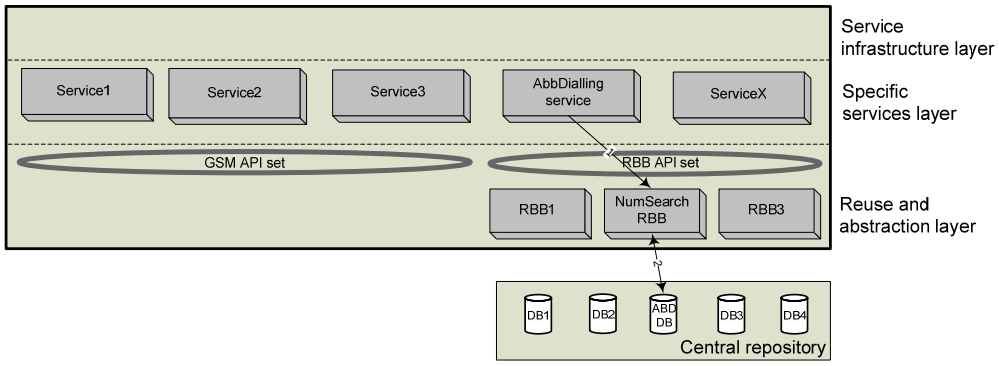


Figure 5-23: ID for "NumSearch RBB"

Message 1 shows the Abbreviated Dialling service invoking the NumSearch RBB, and message 2 shows the RBB performing the subsequent interactions with the appropriate database.

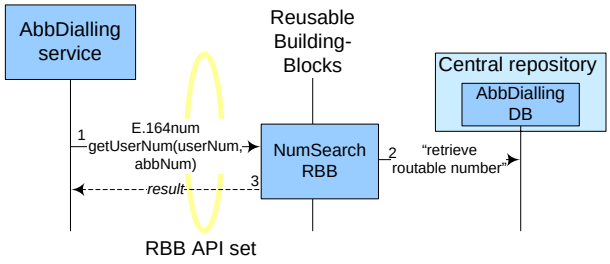


Figure 5-24: MSC for "NumSearch RBB"

5.3.3. Methods of the RBB API set

The RBB API set is comprised of all of the methods offered by every RBB. The RBB API set offers a well-defined interface to service developers by providing a comprehensive definition of all of the arguments, parameters, data types etc. of all of its methods.

The size of the RBB API set is determined by the range of RBBs that are implemented. With only the NumSearch RBB defined in this section, the RBB API set would only contain the single method offered by the NumSearch RBB: *getUserNum()*.

Looking ahead

Chapter 6 presents a dynamic view of the service domain architecture, showing the interaction between critical components in a variety of common scenarios. The dynamic view of the architecture presented in chapter 6 builds on the static view of the service domain architecture from chapter 5 and the critical implementation issues from this chapter.

Chapter 6

A DYNAMIC VIEW OF THE SERVICE DOMAIN ARCHITECTURE

Many of the tasks required for the invocation and execution of different services in the proposed service domain architecture are the same, despite the different objectives of the services themselves. For example, the structure of the service domain architecture prescribes that all services are invoked in the same manner, communicate with terminals in the same way, and affect their control on the resources of the network using the same approach.

In this chapter, these generic operations are identified. Through the recognition of these generic operations, the task of developing services for the proposed service domain architecture is simplified. The examples of services developed for the proposed service domain architecture presented in chapter 7 make frequent reference to these generic operations, highlighting and isolating only those operations that are unique to that service.

Through the explanation and understanding of the generic operations shown in this chapter, the operational inter-relationships and the dynamic behaviour of the service domain components, introduced statically in the previous two chapters, is made clearer. From these dynamic illustrations, the reasons for the structure of the service domain architecture become apparent, and the architecture justified.

The generic operations identified in this chapter are separated and presented in two categories: *service session routines*, and *fundamental interactions*.

Service session routines concern the generic operations related to the initiation and termination of service sessions, and are independent of the execution of any service. Service session routines involve the service infrastructure components in the Service

Infrastructure layer, and their presentation aids in the understanding of the role the key role service infrastructure components play in the proposed architecture.

The *fundamental interactions* concern the generic operations related to the functioning of services. Fundamental interactions identify the standard approaches used in the proposed architecture for service invocation, communication between services and terminals, and for the control of network resources.

IDs and MSCs are used to illustrate the dynamic nature of both the service session routines and the fundamental interactions. While the MSCs show detailed technical interactions in an abstracted view, it is through the depiction of the interactions on the architecture's logical structure, using IDs, that the relations between the horizontal layers and between the various components become clear.

6.1. Service session routines

Service session routines are the sequences of messages that are required for the management of service sessions. There are two service session routines: the "Initial sequence" service session routine is used for the initiation of every service session, and the "Terminating sequence" service session routine is used for the subsequent termination of every service session.

6.1.1. Initiating service session routine

The description of the initiating service session routine requires the understanding of the service infrastructure components. Although the functions of the Contact Agent and the Service Manager were described in chapter 4, their roles at the beginning of a service session are briefly reviewed.

A Service Manager is required to be instantiated throughout the entire duration of any user's service session. A Service Manager is required before any service can be executed, since it is to the Service Manager that all service requests are sent, and it is only the Service Manager that can invoke services.

Before any service session is set up, the user's terminal has only reference to one component on the service domain: the Contact Agent. Before the terminal can attempt to invoke any service, it needs a reference to its Service Manager. The Contact Agent is the only component that can instantiate a Service Manager.

The Contact Agent only instantiates a Service Manager after it is requested to do so by a terminal. If this request is made, it retrieves the user's profile from the User Profile database, instantiates a Service Manager, and returns the reference of the Service Manager to the terminal.

The objective of the initiating service session routine is twofold: first, the Service Manager must first be instantiated; then, the Service Manager must prepare for service invocation requests for both terminal initiated and network initiated services, since the Service Manager is responsible for controlling the invocation for all services in the service domain architecture.

Recall from chapter 4 that the service session can be started by one of two events. Consider 'user X'. The Contact Agent instantiates the Service Manager for a user X in either of two scenarios, depending on whether user X elects to use a service, or is invited to use a service by another party:

- User X wishes to use a service, and user X's terminal contacts the Contact Agent, indicating that a service session is required. For example, user X might start a video conference (and subsequently invite other parties to join the conference). The service session in this scenario is referred to as being initiated by a terminal.
- The network notifies the Contact Agent that user X has been invited to use a service by another party, indicating that a service session is required. For example, another party might attempt to establish a video conference with user X. The service session in this scenario is referred to as being initiated by the network.

There are therefore two variations of the initiating service session routine, depending on whether the service session is initiated by a terminal or the network, and the two possibilities are presented separately. However, after the initial service session setup, the execution of the service session is the same regardless of whether it was initiated by a

terminal or the network. Therefore, besides for the initiation of the service session, no further variations are required to be presented.

Initiating service session routine: for terminal initiation

When a user selects the use of a service on a terminal, the service session is initiated by the user's terminal contacting the Contact Agent. Since terminals always maintain a reference to the Contact Agent, even when they are not involved in any service session, the *initiating service session routine for terminal initiation* is a simple process.

The ID and the MSC for the initiating service session routine, which is initiated by a terminal, is shown in figures 6.1 and 6.2, respectively.

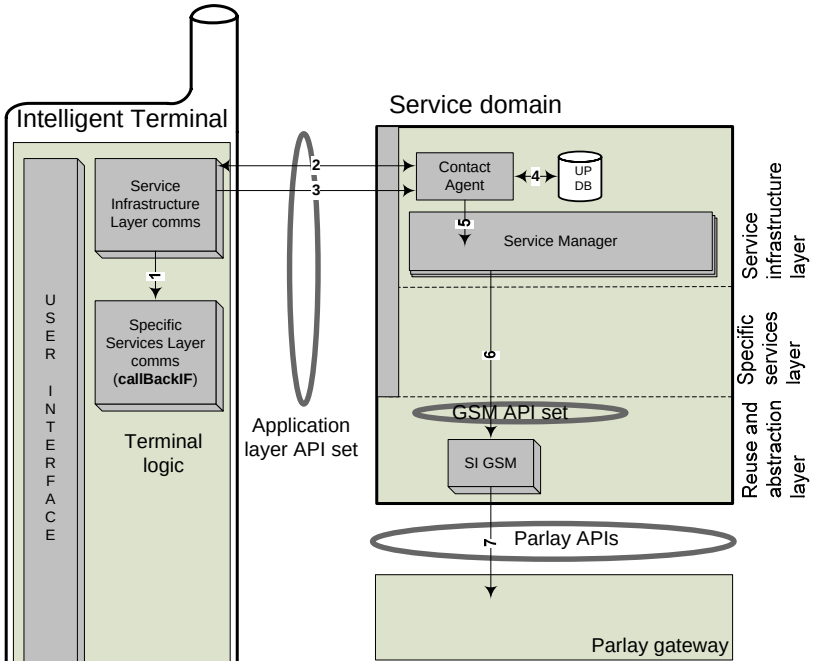


Figure 6-1: ID for terminal-initiated *initiating service session routine*

The description of each of the messages shown in figures 6.1 and 6.2 for the initiation of a service session by a terminal follows:

1. In preparation for the impending service session, the terminal creates a callback interface. The ID shows that the callback interface resides within the

terminal. It is using this callback interface that services contact the terminal, and potentially request additional information or notify the terminal of an event, etc.

2. This ‘signal’ represents the authentication sequence between the terminal and the service domain (as represented by the Contact Agent). The authentication can be performed on one or both objects. Since the authentication is between the terminal equipment and the service domain, the user is oblivious to its execution. The precise authentication sequence is an implementation detail, and is not shown here for reasons of clarity.

3. After authentication has been successfully completed, the terminal contacts the Contact Agent, requesting a new service session to be started. From the ID, this request is seen to utilise application layer signalling. From the MSC, the terminal is shown to call the instantiateServiceManager() method on the Contact Agent. Requesting the start of a new service session is therefore achieved by requesting the instantiation of a new Service Manager. As arguments to the method, the terminal provides the Contact Agent with a reference to its callback interface, and its unique network level address (e.g. E.164 number).

The return parameter of instantiateServiceManager() is the reference to the Service Manager, which is passed to the terminal after the Service Manager has been instantiated. Messages 4 and 5 show the steps taken by the instantiateServiceManager() method to instantiate the Service Manager.

4. Using the terminal’s unique network level address, the Contact Agent retrieves the user’s subscription profile from the User Profile database.

5. The Contact Agent instantiates a new Service Manager, with methods and data dictated by the user’s user subscription profile. (The methods that the Service Manager is instantiated with are determined by the terminal initiated services to which the user subscribes. Thus, the Service Manager implicitly prepares for the potential invocation of any subscribed terminal initiated services through the very methods it is instantiated with. See section 5.1 for more details.) The Contact Agent also passes the reference to the terminal’s callback interface to the Service Manager.

Now that the Service Manager has been instantiated, the instantiateServiceManager() method has completed, and the Contact Agent is returns the reference of the Service Manager to the terminal (using the terminal's callback interface). For all subsequent dealings with the service domain, the terminal contacts its Service Manager directly.

6. The final step in the instantiation of a service session is for the Service Manager prepare for the potential invocation of any network initiated services to which the user subscribes. To do this, it needs to establish the appropriate triggers in the Parlay gateway corresponding to those network events which should cause the invocation of the subscribed network initiated services. However, the Service Manager does not interact with the Parlay gateway directly. Instead, it instructs the SI GSM to create the notification in the Parlay gateway.

7. The SI GSM interacts with the Parlay gateway, and creates all of the requested notifications.

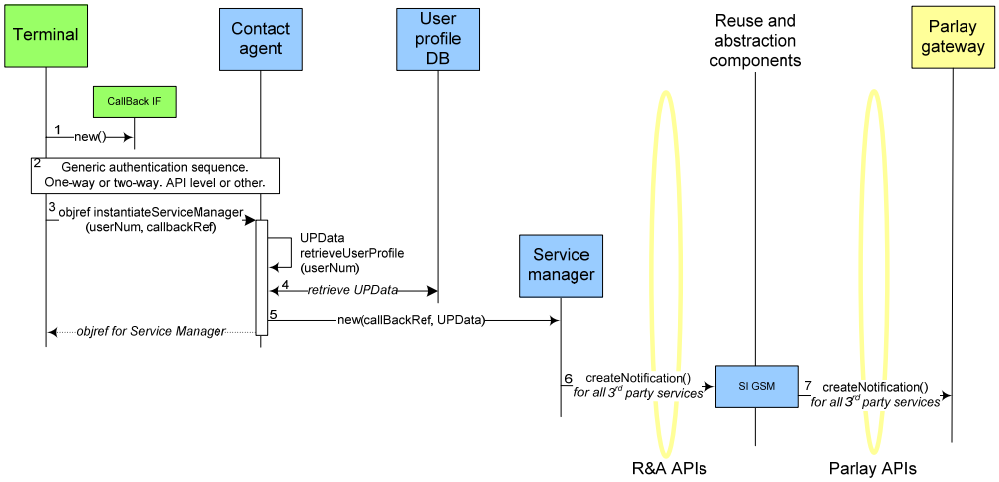


Figure 6-2: MSC for terminal-initiated *initiating service session routine*

Note the different approaches the Service Manager uses to prepare for the invocation of terminal initiated and network initiated services. The preparation for terminal initiated services is implicitly achieved when the Service Manager is instantiated, since it is instantiated with only those methods corresponding terminal initiated services to which the user subscribes (see message 5).

The preparation for network initiated services is achieved explicitly through the creation of notifications in the Parlay gateway, shown by messages 6 and 7. Notifications are created for the various network events required for the invocation of those network initiated services to which the user subscribes.

Note that, traditionally, the individual services themselves set up the required notifications in the Parlay gateway, and it is to these services that the notifications from the Parlay gateway are directly sent. In the proposed approach, the Service Manager establishes all of the Parlay gateway notifications for every subscribed network initiated service, and the reporting of the notifications for all subscribed network initiated services is always made to the Service Manager.

Initiating service session routine: for network initiation

When another party invites a user to use a specific service, the service session is said to be initiated by the network. However, a requirement for the successful initiation of a new service session is that the terminal and the service domain are able to communicate, in both directions, in the application layer. Since the network initiated the service session, the terminal is oblivious to the impending service session; and, since the service domain has no way of contacting the terminal in the application layer (since it does not maintain a reference to the terminal outside of a service session), no application layer signalling can be established (which is a requirement for the successful initiation of a service session).

To get around this problem, the terminal is notified of the impending service session at the network level. After the Contact Agent receives notification from the Parlay gateway that a user has been invited to use a service by another party, e.g. join a video conference, the Contact Agent uses the Parlay gateway to contact the user's terminal at the network level, notifying it of the impending service session. The terminal then contacts the Contact Agent in the application layer, which brings the process to the exact starting point of the *initiating service session routine for terminal initiation*.

The *initiating service session routine for network initiation* is therefore similar to the *initiating service session routine for terminal initiation*; it just requires an additional sequence of messages to occur at the beginning of the service session initiation process.

After this initial sequence of messages, the *initiating service session routines* in both cases use the same sequence of messages.

The ID and the MSC for the initiating service session routine, which is initiated by the network, is shown in figures 6.3 and 6.4, respectively. The ID in figure 6.3 shows only those messages which are unique to the case where the service session is initiated by the network, and hides the subsequent message sequence that is common to both.

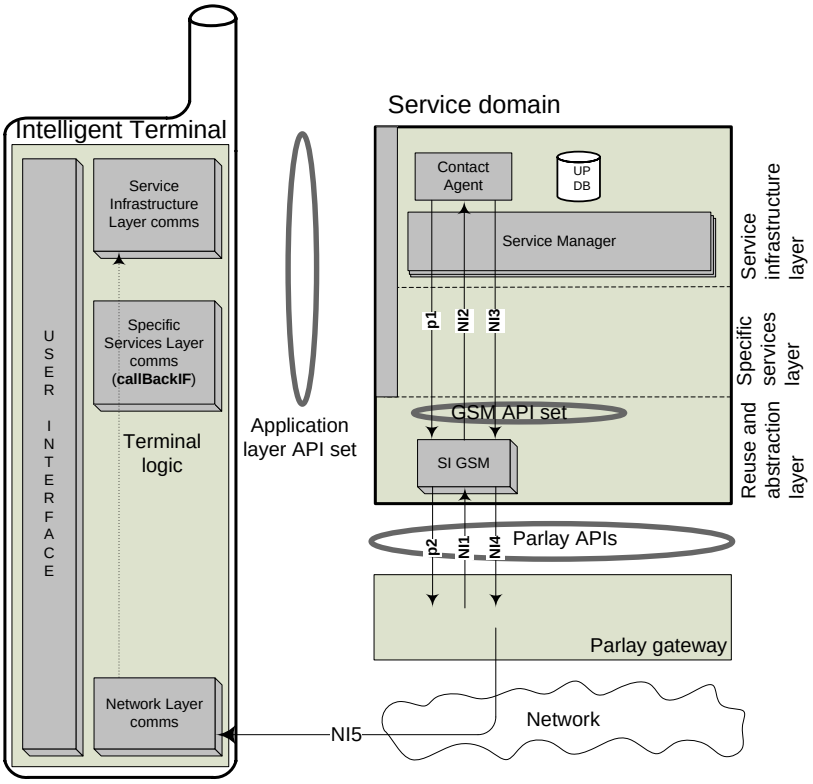


Figure 6-3: ID for network-initiated *initiating service session routine*

All signals that are labelled with names beginning with a “NI” (NI1 – NI5) indicate those signals that are unique to the network-initiated case.

Signals “p1” and “p2” identify those signals that are use “pre” service session, and are not used during the initiation of any specific service session. Recall that, to allow for the initiation of a service session caused by a user being invited to use a service by another party, the Contact Agent needs a mechanism of detecting such events in the underlying network.

Signals p1 and p2 provide just that mechanism. Signals p1 and p2 establish the notification in the Parlay gateway for any network event involving any party, ensuring that if a user is invited to use a service in the network, the Parlay gateway will detect the event and notify the Contact Agent of the need for a new service session. This notification is established only once, when the service domain architecture when it is first provisioned, and never again.

Note that, in establishing the notification in the Parlay gateway, the Contact Agent makes use of the SI GSM, and does not have to interact with the Parlay gateway directly. Similarly, when a user is invited to use a service in the network and the Parlay gateway detects the event, the SI GSM is sent the notification, which is forwarded to the Contact Agent.

A discussion of the messages that are unique to the network-initiated case of service session initiation in figure 6.3 and 6.4 follows:

NI1: The Parlay gateway detects that a user has been invited to use a service in the underlying network, and sends the notification to the SI GSM, with the user's network level address as a parameter.

NI2: The SI GSM forwards this notification to the Contact Agent, notifying it of the need for a new service session.

NI3: The Contact Agent instructs the SI GSM to contact the terminal, using the network level address provided.

NI4: The SI GSM performs the necessary interactions with the Parlay gateway to execute this task.

NI5: The terminal is contacted via the underlying network, and is informed of the need for a new service session. (Note that the user of the terminal is oblivious to the terminals receipt of this signal, and to its subsequent actions.)

The sequence of messages that then follows are exactly the same as in the *initiating service session routine for terminal initiation*, given by messages 1-7 in the MSC (but excluded in the ID).

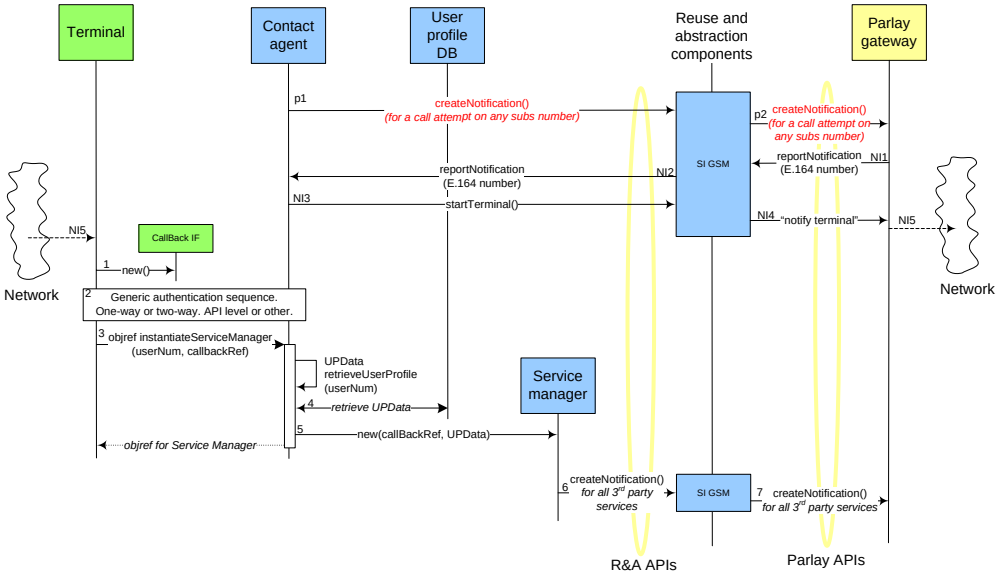


Figure 6-4: MSC for network-initiated *initiating service session routine*

Initiating service session routines: commonalities

The only difference between the MSCs shown in figures 6.2 and 6.4 are the additional messages shown at the beginning of the sequence in figure 6.4. Thus, the *initiating service session routine for network initiation* uses the *initiating service session routine for terminal initiation*, and adds a sequence of messages at the beginning. The MSC in figure 6.5 shows the messages that the two variations have in common.

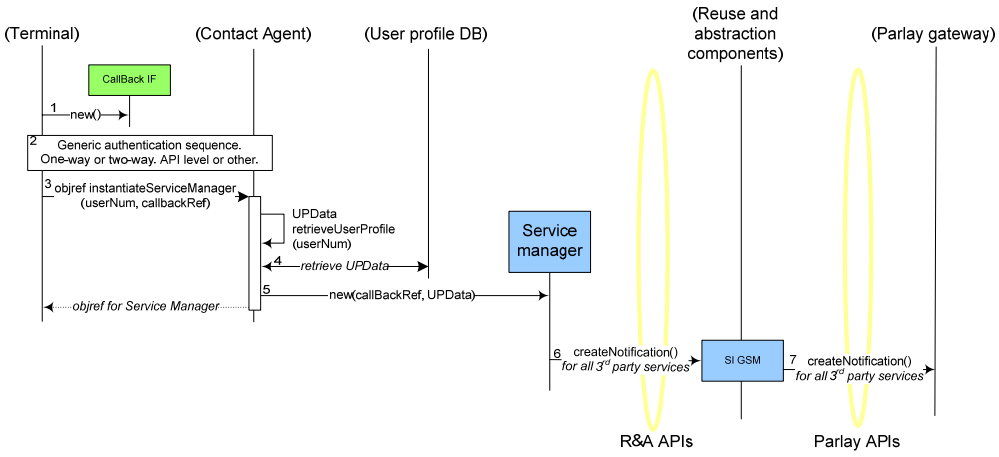


Figure 6-5: The common messages in the two initial service session routines

Figure 6.5 effectively shows that entire *initiating service session routine for terminal initiation*, and the latter half of the *initiating service session routine for network initiation*. The sequence of messages shown in figure 6.5 is referred to as the "common sequence".

The former half of the *initiating service session routine for network initiation* (that is not included in figure 6.5) effectively instructs the terminal to contact the Contact Agent when relevant events in the underlying network have been detected. After this has been performed, the common sequence can commence.

With the recognition of the common sequence, the *initiating service session routine for terminal initiation* is shown in an abstracted view, in figure 6.6.

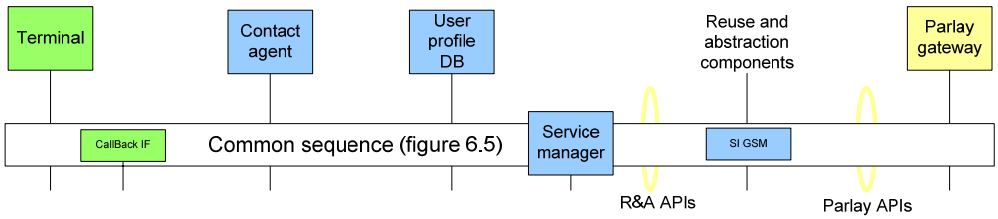


Figure 6-6: Abstracted *initiating service session routine for terminal initiation*

The common sequence comprises the entire *initiating service session routine for terminal initiation*. The callback interface for the terminal and the Service Manager are shown to lie within the "Common sequence" block because they are only instantiated in the Common sequence, and do not exist prior to its execution.

The *initiating service session routine for network initiation* is shown in an abstracted view in figure 6.7.

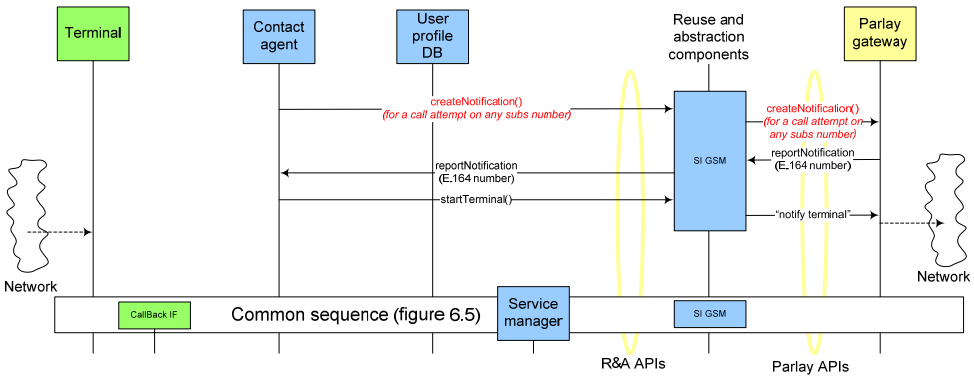


Figure 6-7: Abstracted *initiating service session routine for network initiation*

Figure 6.7 shows that the *initiating service session routine for network initiation* requires a sequence of messages before the common sequence can commence, which is used to inform the terminal of the need for it to contact the Contact Agent. The common sequence that follows, which is illustrated in figure 6.5, is also the *initiating service session routine for terminal initiation*.

While the *initiating service session routine* message sequence is dependent on whether it is network or terminal initiated, it is independent of any particular user or service. For this reason, the examples of services implemented in the proposed service provisioning environment (in chapter 7) will not show the individual messages, but will refer to the initiating sequence abstractly as either the *initiating service session routine for network initiation* or the *initiating service session routine for terminal initiation*, whichever is appropriate.

6.1.2. Terminating service session routine

Two actions are required to terminate a service session. Firstly, the notifications in the Parlay gateway that were established for the particular network initiated services that the user subscribes to must be destroyed. Secondly, the user's Service Manager must be destroyed. By destroying the Service Manager, the means of achieving service invocation for terminal initiated services is also destroyed.

The MSC for the *terminating service session routine* is shown in figure 6.8.

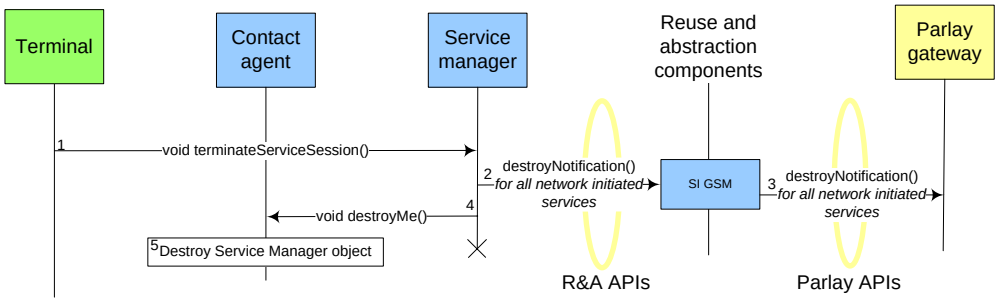


Figure 6-8: MSC for *terminating service session routine*

All service sessions are terminated using this same sequence.

A discussion of the specific messages follows:

1. Using application layer signalling, the terminal calls terminateServiceSession() on the Service Manager, indicating that a service session is no longer required. The end of a service session is always signalled by the terminal.
2. The Service Manager instructs the SI GSM to destroy all notifications in the Parlay gateway that were specifically created for the network initiated services of that user.
3. The SI GSM interacts with the Parlay gateway, and destroys all of the requested notifications.
4. Once the network notifications have been cancelled, the Service Manager instructs the Contact Agent, which was responsible for the original instantiation of the Service Manager, to destroy it. The Contact Agent destroys the Service Manager, and with it, the service session.

Since the message sequence is independent of any particular user or service, the examples of services implemented in the proposed service provisioning environment (in chapter 7) will not show the individual messages, but will refer to the sequence abstractly as the *terminating service session routine*.

6.2. Fundamental interactions

The prescribed structure of the service domain architecture defines how objects within the service domain interact, and how objects in the service domain interact with objects external to the service domain. For example, the way services control the resources of the network is partly determined by the inclusion of GSMs in the proposed service domain architecture. Also, the way terminals communicate with services is partly determined by the existence of the Service Manager, and the introduction of application layer signalling in the proposed service provisioning environment. Thus, the environment in which services are provisioned has a marked impact on how services accomplish commonly required tasks.

Fundamental interactions are those interactions that occur between objects in the service provisioning environment to accomplish simple, yet fundamental, operations. The fundamental interactions presented in this section essentially provide a demonstration of how key tasks are accomplished in the proposed service provisioning environment. These

fundamental interactions are used by most services, no matter how diverse or unique their objectives, as the examples in chapter 7 will demonstrate.

Three fundamental interactions are shown, illustrating how the proposed service provisioning environment achieves service invocation, communication between the service domain and the terminals, and network resource control.

6.2.1. Fundamental interaction 1: Service invocation

Recall that, in the proposed service provisioning environment, the invocation of all services is controlled by the Service Manager. Figure 6.9 shows the invocation of both terminal and network initiated services, using application layer and triggered invocation, respectively, through the use of a Service Manager.

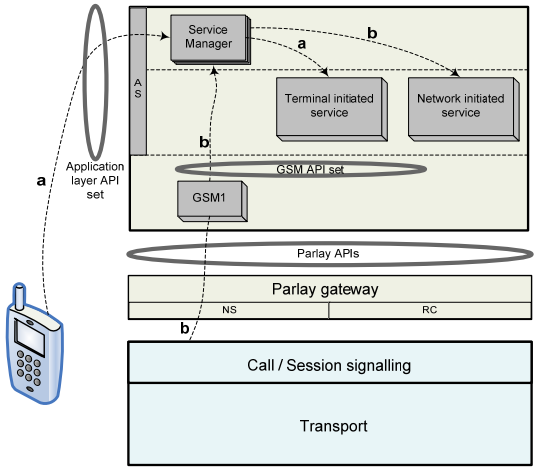


Figure 6-9: ID showing service invocation using a Service Manager

The MSCs for the invocation of terminal and network initiated services are shown in figure 6.10 and 6.11, respectively.

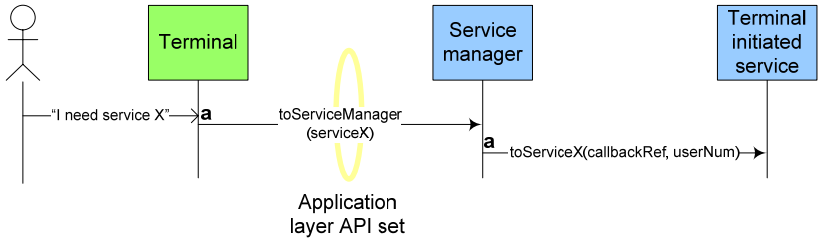


Figure 6-10: MSC showing application layer invocation of terminal initiated services

The signals labelled “a” in figure 6.9, showing the invocation of terminal initiated services, are shown in figure 6.10. Similarly, the signals labelled “b” in figure 6.9, showing the invocation of network initiated services, are shown in figure 6.11.

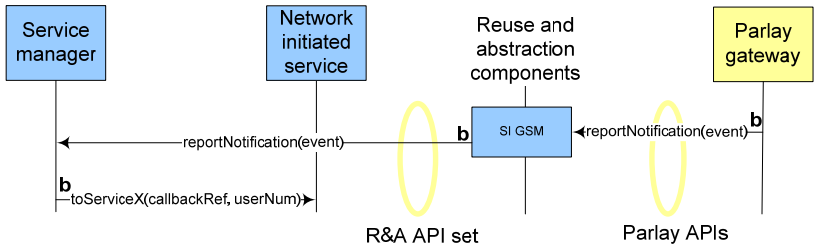


Figure 6-11: MSC showing triggered invocation of network initiated services

Note that in both figures 6.10 and 6.11, the service invocation requests are sent to the Service Manager, and the Service Manager invokes the appropriate service. For terminal initiated services (figure 6.10), the invocation request originates in the terminal, and the Service Manager is contacted using application layer signalling. For network initiated services (figure 6.11), the invocation request originates in the network, and the Service Manager is informed of the invocation request from the network, via the SI GSM.

The method call made by the terminal on the Service Manager in figure 6.10 called “toServiceManager” is not a specific method, but is a generic name representing all possible methods that a terminal could call on the Service Manager. Similarly, the method call made by the Service Manager on a service in figures 6.10 and 6.11 called “toServiceX” is not a specific method, but is a generic name representing all possible method calls that the Service Manager could make on services.

The fundamental interaction shown in figure 6.10 is always used to invoke terminal initiated services, and the fundamental interaction shown in figure 6.11 is always used to invoke network initiated services.

6.2.2. Fundamental interaction 2: Terminal ←→ Service communication

Terminal ←→ service communication refers to all communication between a terminal and a service, excluding the signalling that is used for service invocation. Specifically,

terminal service communication is used for service management (e.g. to set up the abbreviated dialling number pairs for the Abbreviated Dialling service), and is sometimes required during the execution of a service (if, for example, the service requires additional information from the terminal for the successful execution of the service).

Figure 6.12 shows the proposed approach to terminal ←→ service communication.

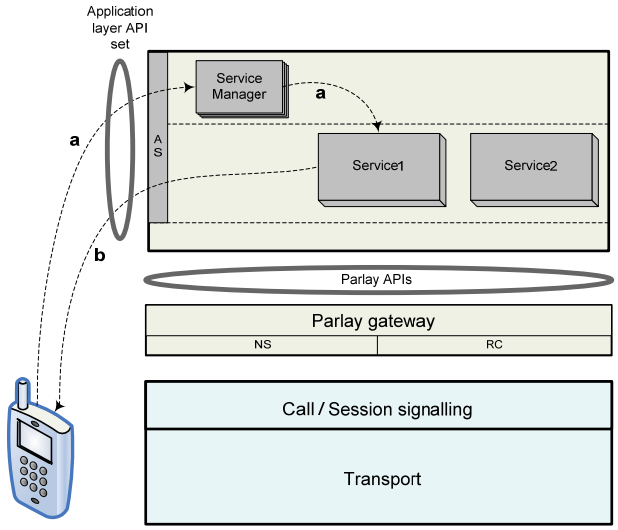


Figure 6-12: ID showing the proposed approach to terminal ←→ service communication

In the proposed approach to terminal ←→ service communication, terminals and services communicate directly, using application layer signalling. However, there is a slight difference as to how terminals signal to services, and how services signal to terminals.

Services do not have references to services, and signal only to a single object in the service domain during any given service session: the Service Manager. The Service Manager then forwards this signal to the appropriate service. Signal (a) shows the terminal signalling to a service in the service domain via the Service Manager. In contrast, services can signal to the terminal without the mediation of the Service Manager during service execution (b), since they received a reference to the terminal's callback interface when they were invoked.

However, whether or not the Service Manager plays an intermediary role, all communication between the terminals and the service domain in the proposed approach uses application layer signalling (using high-level, service oriented signalling protocols).

The MSC in figure 6.13 illustrates the proposed approach to terminal ←→ service communications, corresponding to the ID shown in figure 6.12. Again, note that the method calls shown in the MSC are not specific, but are generic methods representing all possible method calls between any two objects.

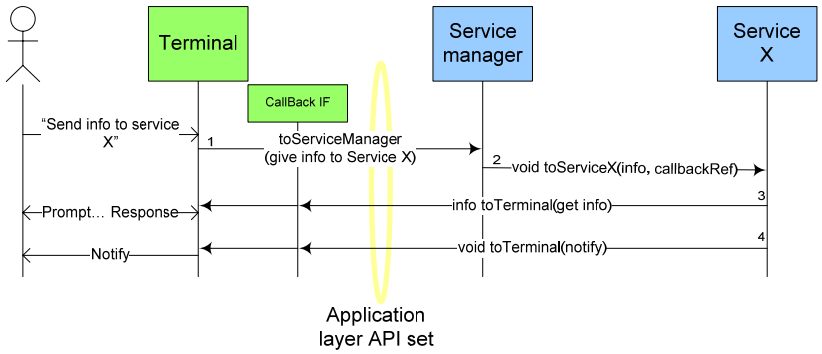


Figure 6-13: MSC showing terminal ←→ service communication using application layer signalling

Typical signalling from a terminal to a service (represented by signal (a) in figure 6.12), is shown in messages 1 and 2:

1. In response to the user's instruction, message 1 shows the terminal's signal to the Service Manager, requesting that the information be forwarded to service X.
2. The Service Manager performs this task.

Terminal → service communications is thus achieved using application layer signalling.

Two variations of signalling from a service to a terminal (represented by signal (b) in figure 6.12), are shown by messages 3 and 4.

3. This message shows the service requesting additional information from the terminal. The service requests (via the terminal's callback interface) that the terminal provide it with more information that is required for the successful execution of the service. The terminal prompts the user for this information, which it returns to the service.

4. This message shows the service providing a notification to the terminal, and not requiring any further information. The service notifies the terminal (via the callback interface) of some event, and the terminal notifies the user.

Service → terminal comms is thus achieved using application layer signalling.

The fundamental interactions shown in figure 6.13, for communications between the terminal and the service in either direction, are used regularly by terminals services to achieve terminal ←→ service communications.

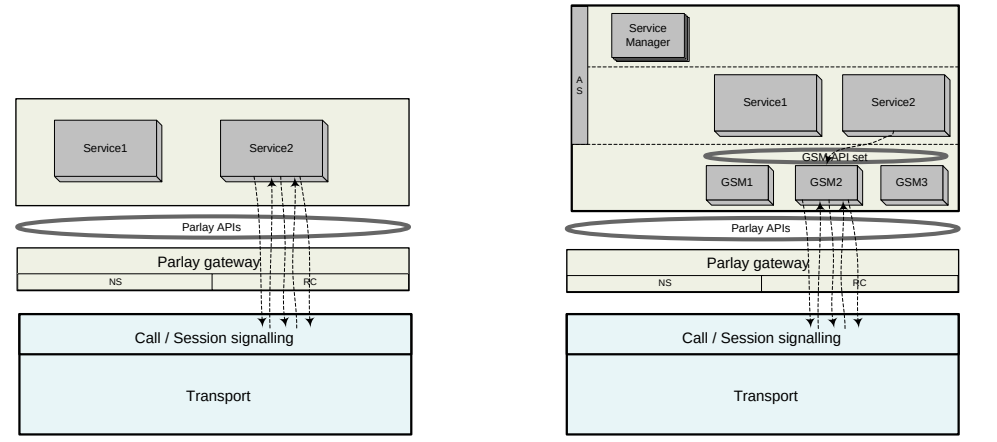
6.2.3. Fundamental interaction 3: Network resource control

The execution of any typical service requires the use of the underlying network. Fundamental interaction 3 concerns the way that services control and manipulate network resources. In the OSA approach to service provisioning (such as with Parlay), the approach used for network resource control essentially reduces to the way the services interact with the Parlay gateway.

GSMs play a major role in determining how services achieve network functionality in the proposed approach. The proposed service provisioning environment introduced GSMs to the service domain to provide a layer of abstraction between the services and the Parlay gateway. The abstraction layer provided by GSMs offers two advantages:

- GSMs relieve services from having to interact with the Parlay gateway, and are thus shielded from the relatively complex Parlay APIs.
- GSMs offer services simple, high-level methods, which encapsulate sequences of Parlay messages, to achieve commonly required network functionality.

To highlight the impact GSMs have on the way services interact with the Parlay gateway, the proposed approach is contrasted to the approach implied in the Parlay standards, where services are required to interact directly with the Parlay gateway. The approach implied in the Parlay standards is shown on the left of figure 6.14, and the proposed approach, which uses GSMs, is shown on the right.



Existing approach to network resource control **Proposed approach to network resource control**

Figure 6-14: Network resource control with and without GSMs

The left side of figure 6.14 shows that without the use of GSMs, services are required to communicate directly with the Parlay gateway, using the Parlay APIs. When GSMs are used, the service communicates with the GSM, and the GSM performs the necessary interaction with the Parlay gateway.

Also note that, to accomplish a certain goal, the GSM requires the service to make only a single method call, whereas many messages are required when no GSM is available.

Figure 6.15 shows the MSC of the proposed approach to achieving network resource control, corresponding to the right side of figure 6.14.

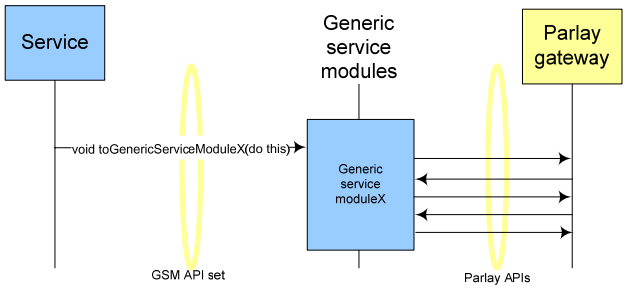


Figure 6-15: Network resource control using a GSM

The method call “toGenericServiceModuleX” is the generic method name used to represent all method calls made by a service on a GSM. The various method calls represented by “toGenericServiceModuleX” constitute the GSM API set.

As depicted by the right side of figure 6.14, the MSC shows how the GSM shields the service from the Parlay APIs, and from the onerous implementation detail that is required to carry out the operation. This fundamental interaction, showing the approach that services follow to control and manipulate network resources, is used in the implementation of almost every telecoms service.

An understanding of the service session routines and the fundamental interactions presented in this chapter greatly simplifies the development of services for the proposed service provisioning environment.

Looking ahead

Chapters 3, 4, 5 and 6 provided a complete description of the proposed service provisioning environment. Chapter 3 discussed the principal concepts underlying the proposed service provisioning environment; chapter 4 presented a static view of the proposed service domain architecture; chapter 5 discussed critical implementation issues of the architecture; and this chapter looked at the dynamic nature of the proposed environment.

The practical feasibility of the theoretical framework laid-out in chapters 3 to 6 is tested in chapter 7, where the implementation of various services in the proposed service provisioning environment is presented.

Chapter 7

EXAMPLES OF SERVICES IMPLEMENTED IN THE PROPOSED SERVICE PROVISIONING ENVIRONMENT

In this chapter, the implementation of three services in the proposed service provisioning environment is presented: the Abbreviated Dialling service, the Call Completion service, and the Call Manipulation service. These services were chosen because they include many of the features commonly found in other services, and because they utilise most aspects of the service provisioning environment, thus aptly demonstrating the nature of the service provisioning environment.

In the description of these services, extensive use will be made of the service session routines and the fundamental interactions of the previous chapter to simply the presentation of the examples, and to show how specific services can be decomposed into repetitive patterns.

7.1. Service session life-cycles

Before the service examples are presented, *service session life-cycles* are described.

Certain services require service management, which is the configuration of service data and other parameters by a subscriber. For example, the Abbreviated Dialling service (discussed in section 7.2) requires subscribers to setup their personalised number pairs, which map the abbreviated numbers onto routable numbers. As has been previously discussed, a service session is required to be created for service management to occur.

The reason a service session is established for the configuration of services is that the user requires a Service Manager in order to communicate with the services he/ she wishes to

configure, and the services to be configured require the user's Service Manager to provide them with a reference to the callback interface of the user's terminal, so that the service can communicate with the user.

There are therefore two instances when a service session needs to be established: for service management (for certain services), and for service execution (for all services). Service sessions used for service management would be used far less frequently than service sessions for service execution. For example, a subscriber of the Abbreviated Dialling service might only update her abbreviated number pairs (perform service management) once a month, whereas she might require the execution of the Abbreviated Dialling service a number of times a day.

Figure 7.1 shows the two possible *service session life-cycles*, depending on whether the service session is created for the purpose of service management or service execution.

The service session life-cycles depicted in figure 7.1 do not show any specific life-cycle, but represent the general phases through which the service session progresses. The initiating and terminating service session routines, which are required at the beginning and end of every service session, are shown in the context of the service session life-cycles.

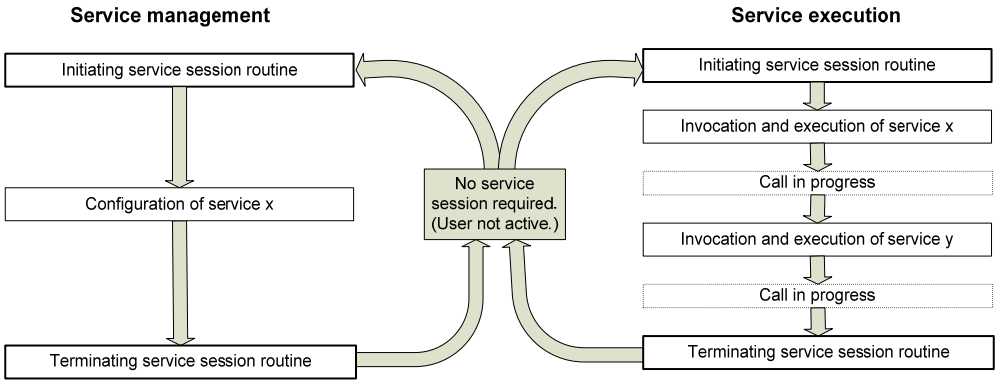


Figure 7-1: Typical service session life-cycles

On the left of figure 7.1 is the typical service session life-cycle for service management, and the life-cycle for the invocation and execution of a number of services is shown on the right. The middle of the diagram represents the state where the user is not actively using any telecoms service, and no service session is required.

Within the service session used for the invocation and execution of services, a bearer connection is usually in existence, and a small number of services may be required.

7.2. Service example 1: Abbreviated Dialling service

The Abbreviated Dialling service allows a user to dial another user by means of a short (abbreviated) number. The Abbreviated Dialling service requires user data to be set up before the service can operate, and thus uses both the service management and service execution life-cycles identified in figure 7.1.

Service management for the Abbreviated Dialling service is discussed first, followed by a discussion on service execution.

Service management for the Abbreviated Dialling service

Service management for the Abbreviated Dialling service allows subscribers to add, remove and edit personalised number pairs, which link abbreviated numbers with the routable numbers. For example, a user might use service management to associate a routable E.164 number with the abbreviated number ‘05’.

The ID and the MSC illustrating Abbreviated Dialling service management are shown in figures 7.2 and 7.3, respectively.

Service management for the Abbreviated Dialling service is always initiated by a terminal. The service session that is established for the management of the Abbreviated Dialling service is thus instantiated using a terminal-initiated *initiating service session routine* (assuming that a service session isn’t already in existence). For clarity, the initiating service session routine omitted entirely in the ID, and is shown only abstractly in the MSC. (After the initiating service session routine, the service session has been initiated, and the terminal has a reference to its Service Manager.)

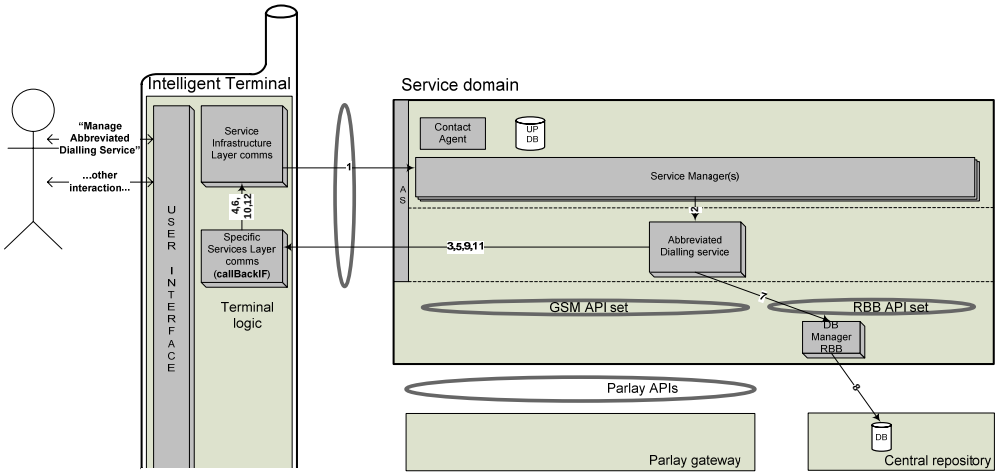


Figure 7-2: Service management ID for the Abbreviated Dialling service

The messages shown in figures 7.2 and 7.3, which are used for Abbreviated Dialling service management specifically, are now described.

1. Based on the instruction by the user, the terminal requests that the Service Manager set up the Abbreviated Dialling service (to which the user is subscribed). This method call is a standardised method contained in the Application layer API set.

2. The Service Manager informs the Abbreviated Dialling service that a user wishes to configure his / her profile. The Service Manager gives the service the user's identity and the reference of the terminal's callback interface.

Messages 1 and 2 constitute fundamental interaction 1: Service invocation.

3-12. Using the callback interface of the user's terminal, the service strikes up communication with the terminal. A dialogue between the service and the terminal then ensues, making the appropriate additions / changes, and the service records the changes to its database. Messages 3, 5, 9 and 11 are standardised methods contained in the Application layer API set.

Messages 3 to 12 (excl. 7 and 8), constitute fundamental interaction 2: Terminal ←→ service communication.

Once the terminal has completed the service management process (and assuming that no further use of the service session is required), it initiates the *terminating service session routine* (shown abstractly in figure 7.3), and the service session is ended.

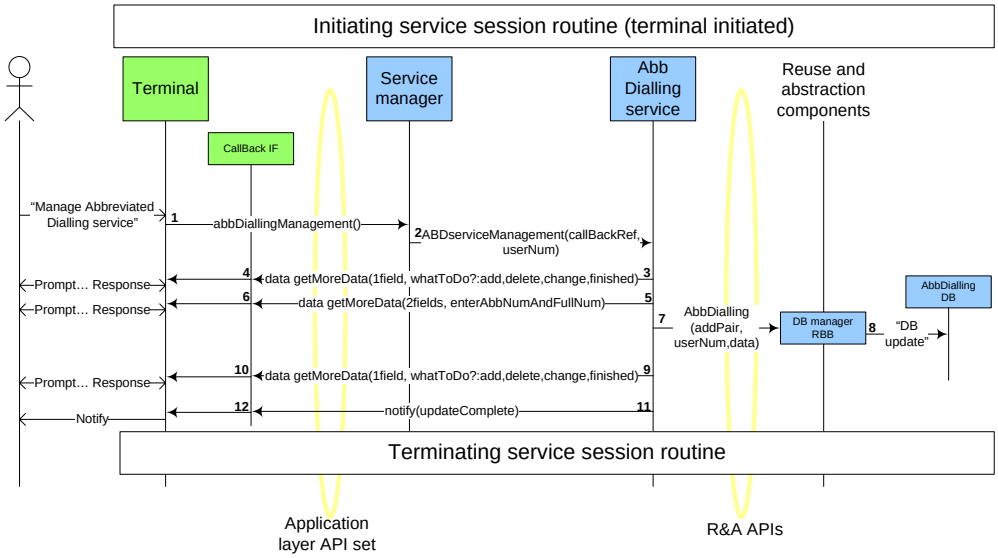


Figure 7-3: Service management MSC for the Abbreviated Dialling service

Service execution for the Abbreviated Dialling service

The Abbreviated Dialling service is requested by a terminal when a user attempts to establish a call using an abbreviated number. The ID and the MSC illustrating Abbreviated Dialling service execution are shown in figures 7.4 and 7.5, respectively.

Since the invocation request for the Abbreviated Dialling service originates in the terminal, it is classified as a terminal initiated service, and, through the use of application layer signalling, is invoked using application layer service invocation. The service session that is established for execution of the service (assuming that a service session isn't already in existence) is thus instantiated using a terminal-initiated *initiating service session routine*. Again, for clarity, the initiating service session routine omitted entirely in the ID, and is shown only abstractly in the MSC.

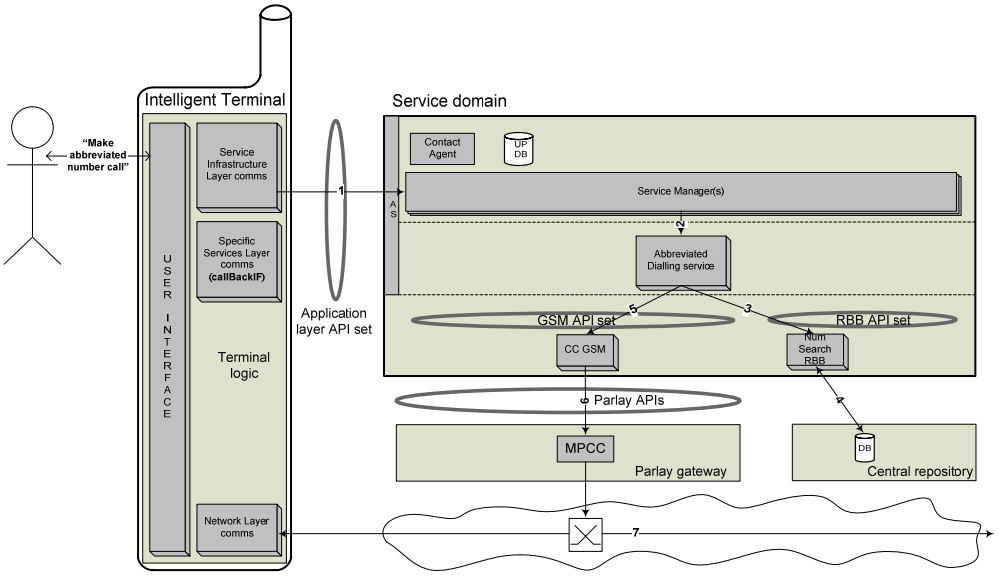


Figure 7-4: Service execution ID for the Abbreviated Dialling service

The messages shown in figures 7.4 and 7.5, which are used for Abbreviated Dialling service execution specifically, are now described.

1. Based on the instruction by the user, the terminal requests that the Service Manager invoke the Abbreviated Dialling service. This method call is a standardised method contained in the Application layer API set.

2. If the user is subscribed to the Abbreviated Dialling service, the Service Manager recognises the method that the terminal called, and invokes the Abbreviated Dialling service. The MSC shows that, as arguments in the invocation method called on the Abbreviated Dialling service, the Service Manager passes the subscriber's number (*userNum*), the abbreviated number (*abbNum*), and the reference to the callback interface of the subscriber's terminal (*callbackRef*).

Messages 1 and 2 constitute fundamental interaction 1: Service invocation.

3. The Abbreviated Dialling service requests that the NumSearch RBB determine the routable E.164 number corresponding to the abbreviated number (*abbNum*) for that user (*userNum*).

4. The NumSearch RBB searches the database used by the Abbreviated Dialling service for the routable number, and then returns routable number to the service.

5. The service has now determined the routable number of the terminating user, and requires the call to be routed. The Abbreviated Dialling service makes a single, high-level call to the MPCC GSM, to route the call. (Refer to section 5.2.2 for a complete description of the MPCC GSM.) This method call is a standardised method contained in the GSM API set.

6. The MPCC GSM establishes the call by performing the necessary interactions with the appropriate SCF in the Parlay gateway. The exact interactions between the MPCC GSM and the Parlay gateway used to establish a call were shown in figure 5.14 of section 5.2.2.

Messages 5 and 6 constitute fundamental interaction 3: Network resource control.

7. The call between the subscriber (served by the service session in question) and the terminating user (specified by the abbreviated number) is then established.

The Abbreviated Dialling service has now executed to completion. On completion of the call, and assuming that no further use of the service session is required, the terminal instructs the Service Manager to proceed with the *terminating service session routine*, as shown in the MSC.

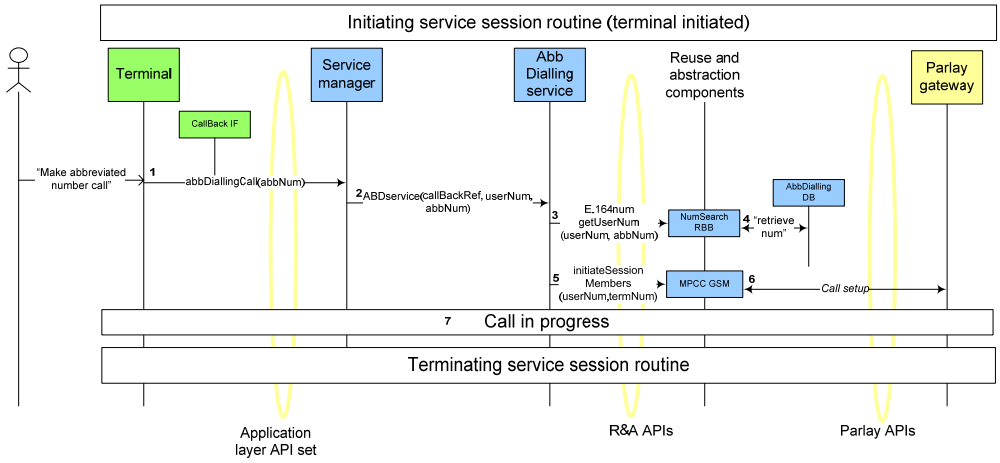


Figure 7-5: Service execution MSC for the Abbreviated Dialling service

7.3. Service example 2: Call Completion service

The Call Completion service enables a calling user, encountering a busy destination, to have the call completed when the busy destination becomes not busy, without having to make a new call attempt [10]. Unlike the Abbreviated Dialling service discussed previously, the Call Completion service does not require service any management; accordingly, only the execution of the Call Completion service is demonstrated.

Before the execution of the Call Completion service is illustrated and discussed in full detail, a qualitative overview of the general operation of the service is provided. The typical operation of the Call Completion service is as follows:

- If the subscriber to the Call Completion service wants to ensure that an end-to-end connection is established as soon as possible, the subscriber instructs his terminal to establish a call with a specified terminating party using the Call Completion service. (If the subscriber is in no particular need to have the call established in as little time as possible, he would not use the Call Completion service, and simply instruct the terminal to establish a regular call.)
- The terminal requests that the Service Manager invoke the Call Completion service. If the service is subscribed to, the Service Manager invokes the Call Completion service, passing it the number of the subscriber and the terminating party.
- The Call Completion service attempts to establish the call. If the call is successfully established, the Call Completion service has executed to completion. However, if the terminating party is busy, the Call Completion service performs the following actions:
    - The current call attempt is aborted, but the subscriber's service session remains in existence.
    - The Call Completion service constantly checks the terminating terminal to ascertain whether it is still busy.
    - On determining that the terminating terminal is no longer busy, the Call Completion service establishes the call between the service subscriber and the terminating party, and the Call Completion service has executed to completion.

The operation of the Call Completion service described above assumed that the user consciously decided to utilise the Call Completion service, and explicitly instructed the terminal to request that the service be invoked from the start. Using this approach, the user would still be able to make “regular” calls, and would not be forced to use the Call Completion service on every call attempt. That is, the user would be able to abandon a call completely on encountering a busy signal.

The ID and the MSC illustrating the execution of the Call Completion service are shown in figures 7.6 and 7.7, respectively. Similar to the Abbreviated Dialling service, the Call Completion service is classified as a terminal initiated service, and is invoked using application layer signalling. The service session that is established for execution of the service (assuming that a service session isn’t already in existence) is thus instantiated using a terminal-initiated *initiating service session routine*.

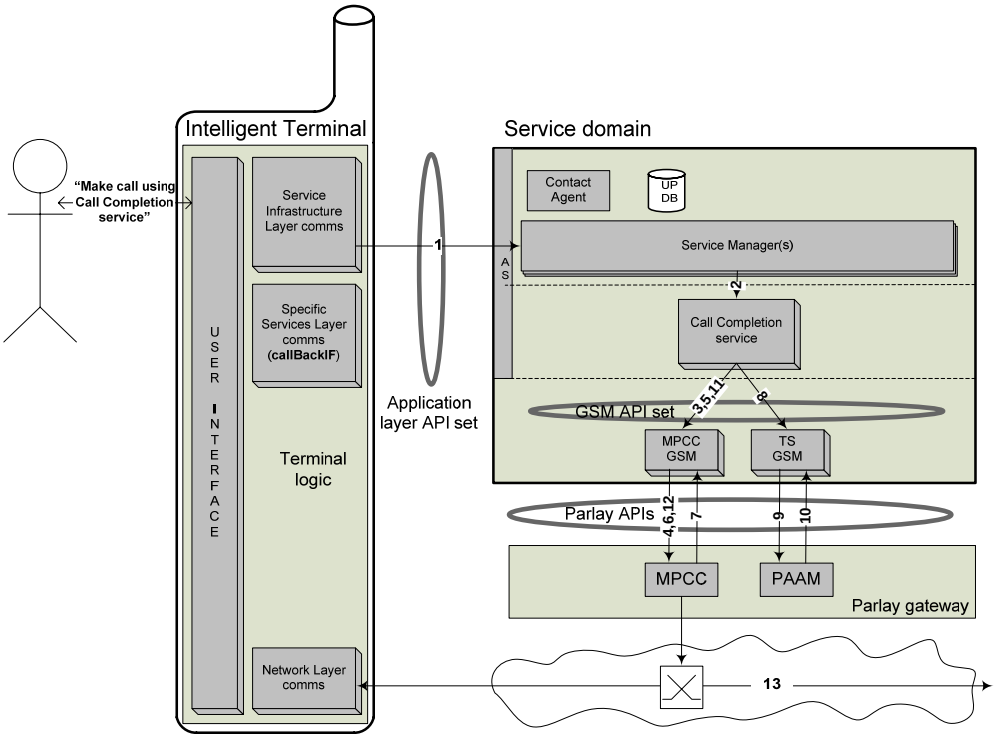


Figure 7-6: Service execution ID for the Call Completion service

The messages shown in figures 7.6 and 7.7, which are used for the execution of the Call Completion service specifically, are now described.

1. Based on the instruction by the user, the terminal requests that the Service Manager invoke the Call Completion service. This method call is a standardised method contained in the Application layer API set.
2. If the user is subscribed to the Call Completion service, the Service Manager recognises the method that the terminal called, and invokes the Call Completion service. The MSC shows that, as arguments in the invocation method called on the Call Completion service, the Service Manager passes the subscriber's number (*userNum*), the terminating party's number (*termNum*).
Messages 1 and 2 constitute fundamental interaction 1: Service invocation.

The remaining messages (5-12) illustrate the steps used in the execution of the Call Completion service, and constitute *fundamental interaction 3: Network resource control.*

3,4. The Call Completion service requests that the MPCC GSM set up a notification in the network, requesting that it be informed if the call that it is about to establish encounters a busy signal on the terminating side. Signal 3 is a standardised method contained in the GSM API set. The MPCC GSM performs the required action(s) on the Parlay gateway.

5,6. The Call Completion service requests that the MPCC GSM set the bearer connection up. Signal 5 is a standardised method contained in the GSM API set. The GSM performs the required action(s) on the Parlay gateway. The exact interactions between the MPCC GSM and the Parlay gateway used to establish a call were shown in figure 5.14 of section 5.2.2.

If the call is successfully established, the notification that was created in messages 3 and 4 will not be reported, and the Call Completion service will have executed to completion. However, if the terminating party is busy, the following sequence of events occurs.

7. The Parlay gateway informs the Call Completion service that the call could not be completed due to the terminating party being busy.

8,9. The Call Completion service requests that the TS (Terminal Status) GSM create a notification on the Parlay gateway requesting that it be informed when the terminating party goes on-hook. Signal 8 is a standardised method contained in the GSM API set. The TS GSM performs the required action(s) on the Parlay gateway.

10. After some period of time (once the terminating party has completed the existing call), the Parlay gateway informs the Call Completion service that the terminating party is now on-hook.

11,12. Similar to messages 5 and 6, the Call Completion service attempts to establish a bearer connection, using the MPCC GSM.

13. The call is successfully established, and the call proceeds.

The Call Completion service has now executed to completion. On completion of the call, and assuming that no further use of the service session is required, the terminal instructs the Service Manager to proceed with the *terminating service session routine*, as shown in the MSC.

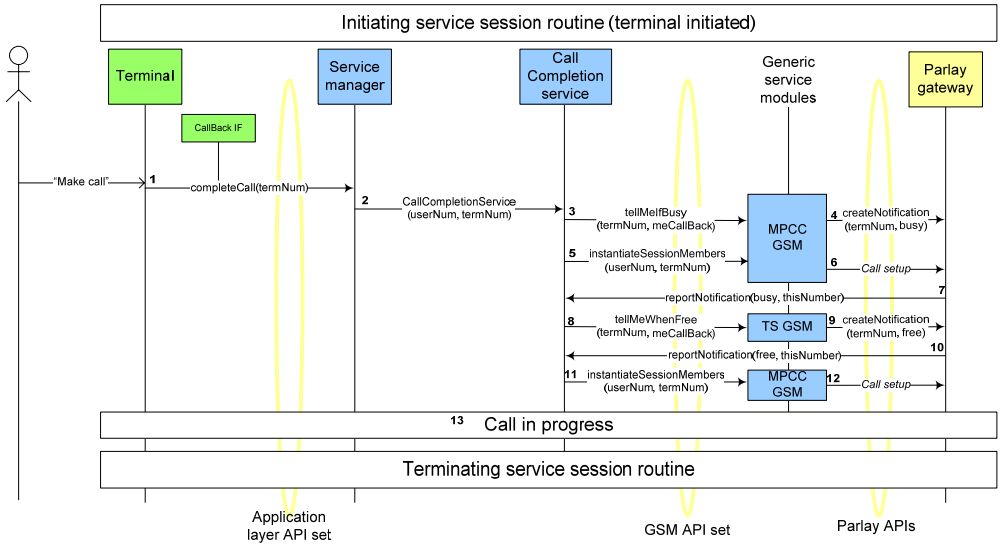


Figure 7-7: Service execution MSC for the Call Completion Service

7.4. Service example 3: Call Manipulation service

The Call Manipulation service is an 'umbrella service', comprised of a number of smaller services that are offered in existing service deployments (all of which involve the manipulation of the states of various parties in a call). The services incorporated in the Call Completion service are taken form the IN CS2 (IN Capability Set 2) specifications [10], and include the following services:

- Call Hold. Call Hold allows a user to interrupt his / her connection to an existing call, without releasing that call.
- Call Retrieve. Call Retrieve allows a user to re-establish his / her connection to a call previously placed on hold.
- Call Toggle. Call Toggle is applicable to a user who has one active call, and one on hold. It allows him / her repeatedly to select the currently held party as the new connection, with the previously connected party being put on hold.

Since the Call Manipulation service does not require service any service management, only the execution of the Call Manipulation service is demonstrated. There are a vast number of possible ways a subscriber could use the service; thus, the operation of the Call Manipulation service presented in this section is only intended to provide an example of some ways the service can be utilised.

Before the execution of the Call Manipulation service is illustrated and discussed in full detail, a qualitative overview of the general operation of the service is provided. The operation of the Call Manipulation service, as illustrated later in this section, is as follows:

- (The Call Manipulation service requires that a call, and thus a service session, be in existence before it can operate.)
- The subscriber requests the use of the Call Manipulation service. The terminal signals to the Service Manager, and the Service Manager invokes the Call Manipulation service. The subscriber uses the Call Manipulation service as follows:
    - The subscriber requests that the current call be placed on hold. The Call Manipulation service places the current call on hold, and prompts the user (via the terminal's callback interface) for the number of the new terminating party.

- The subscriber provides the Call Manipulation service with the number of the new terminating party, and the Call Manipulation service establishes the new call (while maintaining the original terminating party on hold).
- During the new call, the subscriber requests that the original and new terminating parties be toggled. The Call Manipulation service places the terminating party in the current call on hold, retrieves the terminating party of the original call (who was put on hold), and reinstates the original call.

Since the Call Manipulation service is controlled using application layer signalling, it is classified as a terminal initiated service. The ID and the MSC illustrating the execution of the Call Manipulation service are shown in figures 7.8 and 7.9, respectively.

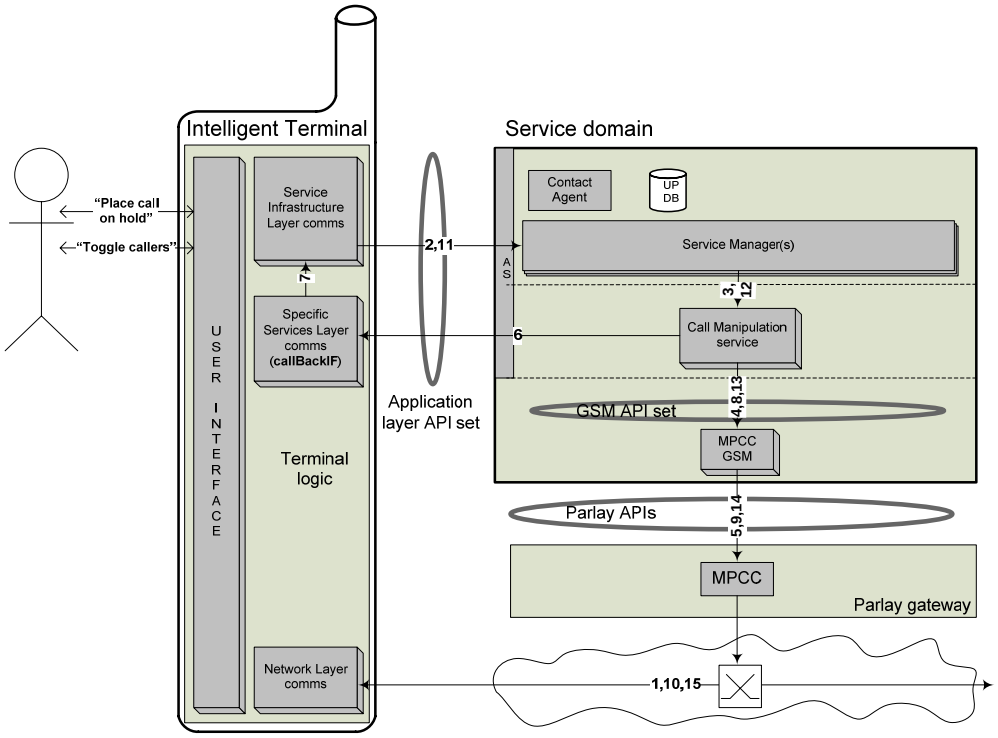


Figure 7-8: Service execution ID for the Call Manipulation service

The messages shown in figures 7.8 and 7.9, which are used for the execution of the Call Manipulation service specifically, are now described.

1. A service session is inexistence, and the terminal has a reference to its Service Manager. A call is in progress between the subscriber and user 'A'.

2. Based on the instruction by the user, the terminal requests that the Service Manager invoke the Call Manipulation service. This method call is a standardised method contained in the Application layer API set.

3. If the user is subscribed to the Call Manipulation service, the Service Manager recognises the method that the terminal called, and invokes the Call Manipulation service, requesting that it place the current call on hold. The MSC shows that, as arguments in the invocation method called on the Call Manipulation service, the Service Manager passes the subscriber's number (*userNum*), the terminating party's number (*termNum*), the reference to the terminal's callback interface (*callbackRef*), and the type of call manipulation that is required (*putOnHold*).
Messages 2 and 3 constitute fundamental interaction 1: Service invocation.

4,5. The Call Manipulation service instructs the MPCC GSM to perform the necessary object manipulations to place the terminating party on hold, which the GSM does. Signal 4 is a standardised method contained in the GSM API set.
Messages 4 and 5 constitute fundamental interaction 3: Network resource control.

6,7. The Call Manipulation service prompts the terminal for the number of the new party with whom a connection is to be established. The terminal prompts the user for the number, and the new terminating number (user 'B') is returned to the Call Manipulation service. Message 6 is a standardised method contained in the Application layer API set.
Messages 6 and 7 constitute fundamental interaction 2: Terminal ←→ service communication.

8,9. The Call Manipulation service requests that the MPCC GSM establish the new call between the subscriber and user B. Signal 8 is a standardised method contained in the GSM API set. The GSM performs the required action(s) on the

Parlay gateway. The exact interactions between the MPCC GSM and the Parlay gateway used to establish a call were shown in figure 5.14 of section 5.2.2. *Messages 8 and 9 constitute fundamental interaction 3: Network resource control.*

10. The new call between the subscriber and user ‘B’ is established.

11. The user now requests that the terminal instruct the Call Manipulation service to toggle between the existing terminating party (user ‘B’) and the original terminating party (user ‘A’). This method call is a standardised method contained in the Application layer API set.
12. The Service Manager passes the request to the Call Manipulation service. The MSC shows that, as an argument in the method called on the Call Manipulation service, the Service Manager indicates that the Call Manipulation service needs to toggle the parties (*togglePartу*).
Messages 11 and 12 constitute fundamental interaction 1: Service invocation.

13,14. The Call Manipulation service requests that the MPCC GSM toggle users A and B. Signal 13 is a standardised method contained in the GSM API set. The MPCC GSM performs the necessary object manipulations on the Parlay gateway to toggle the users.
Messages 13 and 14 constitute fundamental interaction 3: Network resource control.

15. The original call (between the subscriber and user A) is re-established.

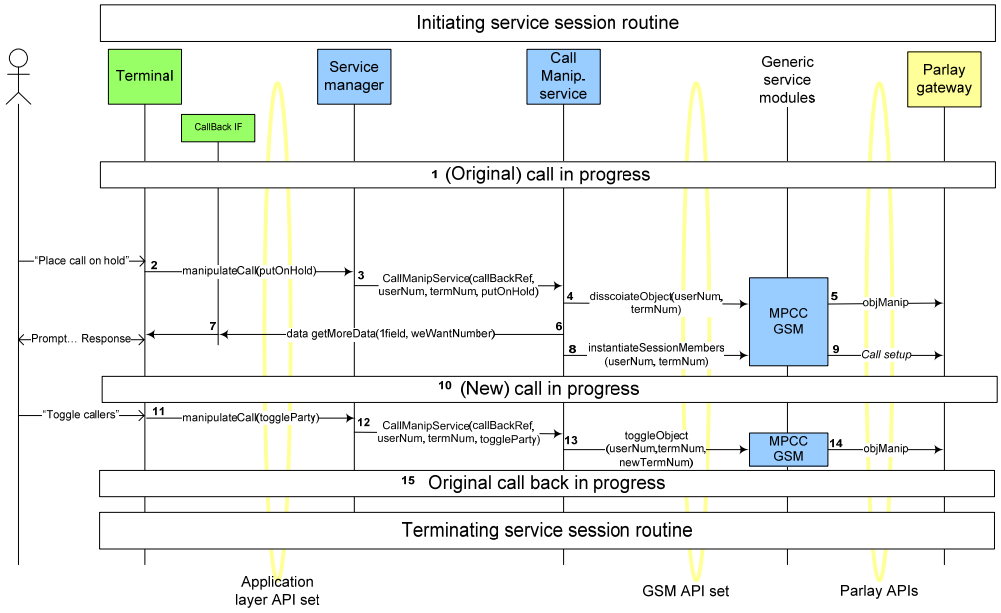


Figure 7-9: Service execution MSC for Call Manipulation service

Looking ahead

In chapter 8, the practical value and efficacy of certain aspects of the proposed service provisioning environment is investigated. By determining the practical value of the core principles, the contribution that the proposed service provisioning environment makes to telecoms can be assessed.

Chapter 8

DEMONSTRATION OF CONCEPT

This chapter sets out to demonstrate and critically evaluate the purported benefits of the proposed service provisioning environment. Due to the broad scope of the research project, only those concepts that are the most central and fundamental to the success of the proposed service provisioning environment are examined; the verification of the more elementary aspects is not explicitly performed.

Use of the proposed service provisioning environment gives rise to gains in two ways:

- Increased capabilities and complexity of services
- Increased efficiency for service invocation and service execution

Each of these types of gains is substantiated independently in the following sections.

8.1. Service capabilities and complexity

The primary determinant of the difference in the capabilities of services that can be offered in the existing and proposed service provisioning environments is the signalling channel between the terminals and the service domain. In this section, practical issues of the different signalling channels used in the existing and proposed approaches are analysed, and their impact on the capabilities of services that can be offered is evaluated.

The existing approach to service signalling, and its impact on service capabilities

The problem with communication between terminals and the service domain, in either direction, is that the signalling path used is not intended for service signalling; thus, the various parts of the signalling path need to be manipulated to allow service signalling.

Signalling from the terminal to the service domain (e.g. for the invocation of terminal initiated services) requires that the signal, which originates in the terminal, reach the service domain by travelling through the network and the Parlay gateway. Specifically, any message originating in the terminal reaches the service domain by being transported using call / session layer protocols in the network, and then using Parlay API messages between the Parlay gateway and the service domain.

In the opposite direction, services signal to terminals using a similar path; using the Parlay APIs first, and then the network to contact the terminal. Specifically, for a service to communicate with a terminal, it effectively needs to cause the network to perform certain actions which somehow impact on the target terminal, thereby allowing (some degree of) communication.

Each part of the communication path linking the terminals to the service domain in the existing approach introduces unique problems.

Network signalling

The signalling protocols that are used in the network layer are intended for call / session signalling, and are designed for the establishment, management and tear-down of bearer connections. Call / session layer protocols and are not oriented towards service signalling, and when using call / session layer signalling to invoke and control applications, the call / session signalling protocol, and the rules for interpreting messages, becomes complex.

To achieve the flow of service-related information through the network, any flexibility in the call / session layer protocols needs to be manipulated. For example, the header and message content in a SIP message can be manipulated to permit service signalling. However, having to exploit the limited amount of flexibility in the call / session layer protocols (e.g. SIP) to achieve service signalling is restrictive, and only basic information transfer can be achieved.

Network ←→ service domain signalling

Of the complete communication path between the terminals and the service domain, network layer signalling governs the section of the path between the terminals and the switches in the network. The remaining part of the communication path, between the network switches and the service domain, is analysed here. This section of the communication path, between the network and the service domain, poses the greatest constraint to service signalling in the existing approach. The reasons for this follow.

The channel between the network and the service domain is governed by the Parlay gateway, and the Parlay APIs. Parlay was developed to detect activity in the underlying network, and, based on this activity, generate and transmit messages to the service domain. It was not developed to 'relay' message flows from the underlying network to the service domain. The Parlay APIs actually dictate the message set that it can pass, and do not contain any flexibility that can be manipulated for ad hoc purposes.

The message set that can be passed by the Parlay gateway is determined by the *notification* Parlay APIs. For example, the Parlay method *reportNotification()* reports to the service when a specified user performs specified operations in the network, e.g. attempts to make a call, receives a call, is engaged, or terminates a call. With the communication channel between the network and the service domain quantised to such a limited set of possibilities, the information that can be passed between the network and the service domain is limited to the most basic of signals.

The complete communication path

Combining the analyses on network signalling, and network ←→ service domain signalling, the following conclusions can be made.

Achieving terminal→ service signalling: *For a terminal to make a request to a service, trigger information must be present in the call / session layer signalling methods, and the event relating to that trigger must be enabled in the Parlay gateway.* That is, with the knowledge of the events that a network switch is set to detect, the terminal creates the appropriate network conditions / events that will trigger the switch to notify the Parlay

gateway of the network event. The Parlay gateway then notifies the service, and the communication path between the terminal and the service is completed.

Achieving service→ terminal signalling: with the knowledge of how certain network conditions affect certain terminals, the service uses the Parlay APIs to manipulate the network resources. By manipulating network resources involving the target terminal in certain ways, the service is able to signal to the terminal. Note that once an event is triggered, any dialogue must be controlled by the service.

Note that a service can still obtain information from a user (as opposed to a terminal) by using the UI (User Interaction) Parlay SCF. For example, a service could request that a user enter a username and a password. However, it is communication between the service and the terminal (for control signals, etc.) which is more severely impeded.

The consequence for service capabilities and complexity

Obviously, in both of these cases, the 'information' that can be passed between the terminals and the service domain is severely constrained to the most basic signalling. This has marked consequences on the services that can be successfully deployed in this environment.

- Since terminals invoke terminal initiated services by creating the necessary network condition that triggers a network event in a switch (which results in a Parlay notification being sent to the service), only very simple services, which don't require extensive user data, can be deployed.
- Since services communicate with terminals by manipulating bearer network states (which involve the target terminal in certain ways), all services that require message exchange with the terminal during service execution have severe limitations.

The proposed approach to service signalling, and its impact on service capabilities

In the proposed service provisioning environment, the communication channel between the terminals and the service domain is maintained in the application layer. Application layer signalling completely bypasses the network and the Parlay gateway, negating all of

the problems outlined above. Specifically, in the proposed approach, communication does not rely on having to manipulate bearer network resources, or rely on network event detections and network notifications, to allow contact between terminals and the service domain.

Using application layer signalling, terminals and services can make high-level, explicit calls on each other, requesting and providing all information in a simple and effective way. For example, a terminal can explicitly request that a conference call be established by calling the following method on a hypothetical Conference Call service:

establishConferenceCall(mediaType, parties[], billingInfo)

In a single method call, the terminal is able to completely specify all of the required details of the conference call.

Also, services which require the exchange of signals between the terminal and the service during service execution can be easily provisioned. For example, a service can request that a terminal provide it with a unique identifier by using the following method:

data getMoreData(fromTerminal, "Provide equipment identity number")

The communication path in the existing approach completely precludes information transfer between a service and a terminal of this sort. That is, no complex control or data signals can flow between a terminal and a service without application layer signalling.

Application layer signalling also allows communication between a service and a user (as opposed to a terminal) more easily. Whereas, in the existing approach, a service would use the UI SCF to communicate with a user (and would suffer from all the limitations of the service / switch / network / terminal signalling path), application layer signalling allows a service to request information from a user in a simple and direct manner, such as:

data getMoreData(fromUser, "Provide username and password")

Services often use call state information, such as calling party, called party, reason for forward, etc., to infer service context [11]. In the proposed approach, call state information is maintained by the user's Service Manager, and services can thus easily obtain this information. However, if other call state information is required by a service,

terminals are able to explicitly state any relevant call state information using application layer signaling.

In contrast, in the existing approach, no call state information is maintained in the service domain. Therefore, all call state information required by a service needs to be obtained through its invocation; however, only limited call state information implying the service-context can be transferred without the use of application layer signaling.

Using application layer signalling for communication between terminals and the service domain allows services with advanced capabilities to be deployed, and used, effectively. Application layer signalling also permits the introduction of certain services which are simply not possible without the use of application layer signalling.

8.2. Efficiency of service invocation and execution

The proposed service provisioning environment gives rise to gains in the efficiency of service invocation and execution. By gains in efficiency, it is meant that the same task can be achieved in a simpler way, with fewer messages, fewer processing stages, and in less time.

The increased efficiency of service invocation and execution offered by the proposed service provisioning environment is made possible through two of its fundamental principles:

- The use of application layer signalling between the terminals and the service domain. This brings about efficiency gains in service invocation (of terminal initiated services).
- The relocation of the primary view and control of the BCS to the application layer. This brings about efficiency gains in service execution.

The compound effects that application layer signalling and the relocation of the primary view and control of the BCS have on efficiency are examined by looking at the process required to establish a simple bearer call. A call between two parties is examined first, after which multiparty calls (consisting of 3 or more parties) are examined. In each case, the proposed approach (3rd party call initiation) is compared with the existing approach (1st party call initiation).

In all of the scenarios presented in this section, the degree of efficiency achieved in the scenario is quantified by the time it takes to carry out a specific task. Since the time taken to carry out a task is determined by the number of messages that are required to be exchanged in the execution of that task, the number of messages is used as a proxy for the degree of inefficiency.

As will be shown, the execution of a task often requires the use of both application layer messages and network layer messages. However, since it is assumed that the application layer messages are exchanged between software objects using CORBA as a distribution mechanism, the time taken for their distribution and processing would be negligibly small. In contrast, the contribution to the time taken for the distribution and processing of network layer signals is far greater. Therefore, the number of network layer messages used in any scenario is used as the proxy for efficiency.

SIP (as it is applied to IMS) is assumed to be used for all network layer signalling. Refer to [12] for more information on SIP.

8.2.1. Connectivity between 2 parties

Two variants of call initiation between 2 users are shown in this section, each incorporating different levels of complexity. In *simple call initiation between 2 parties*, it is assumed that both parties share the same S-CSCF (Serving – Call Session Control Function). The assumption that both parties use the same S-CSCF is dropped in *complex call initiation between 2 parties.* Refer to [13] for more information on the S-CSCF.

Simple call initiation between 2 parties

Simple call initiation between 2 parties involves the creation of a call between 2 parties that share the same S-CSCF. Figure 8.1 presents the MSC showing *simple call initiation between 2 parties*, using 1st party call initiation (as used in the existing approach). This MSC is taken from the 3GPP standards. For a full description of all of the messages, refer to [14].

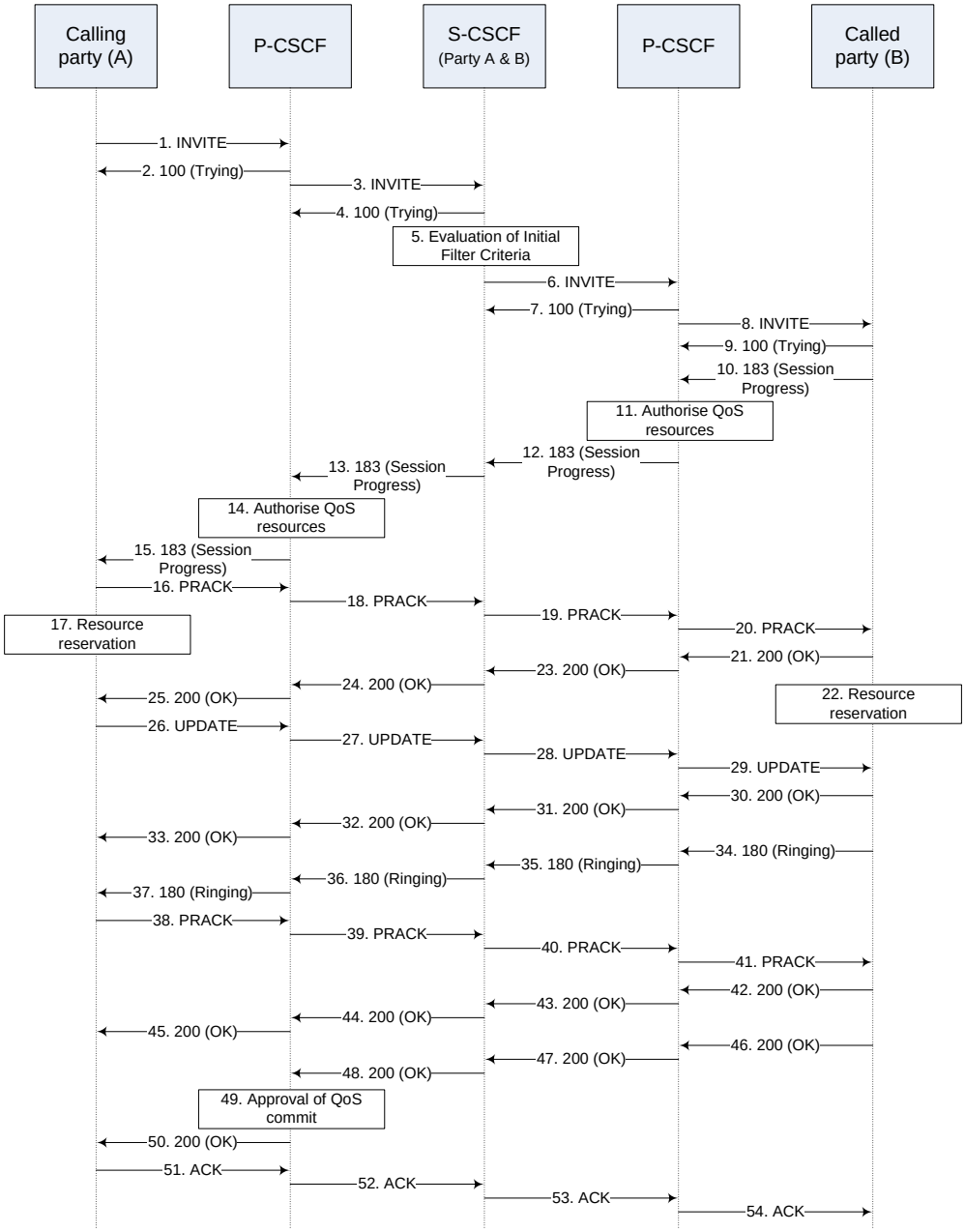


Figure 8-1: Simple 1st party call initiation between 2 parties

The 2-party call initiated using 1st party call initiation, shown in figure 8.1, requires 48 SIP messages (and 6 other actions, shown in boxes).

Figure 8.2 presents the MSC showing *simple call initiation between 2 parties*, using 3rd party call initiation (as used in the proposed approach). The mapping of Parlay API methods to SIP messages is taken from [15], and the SIP messages exchanged between

the CSCFs and parties A and B are taken from [14]. Please refer to these documents for a full description of the various messages.

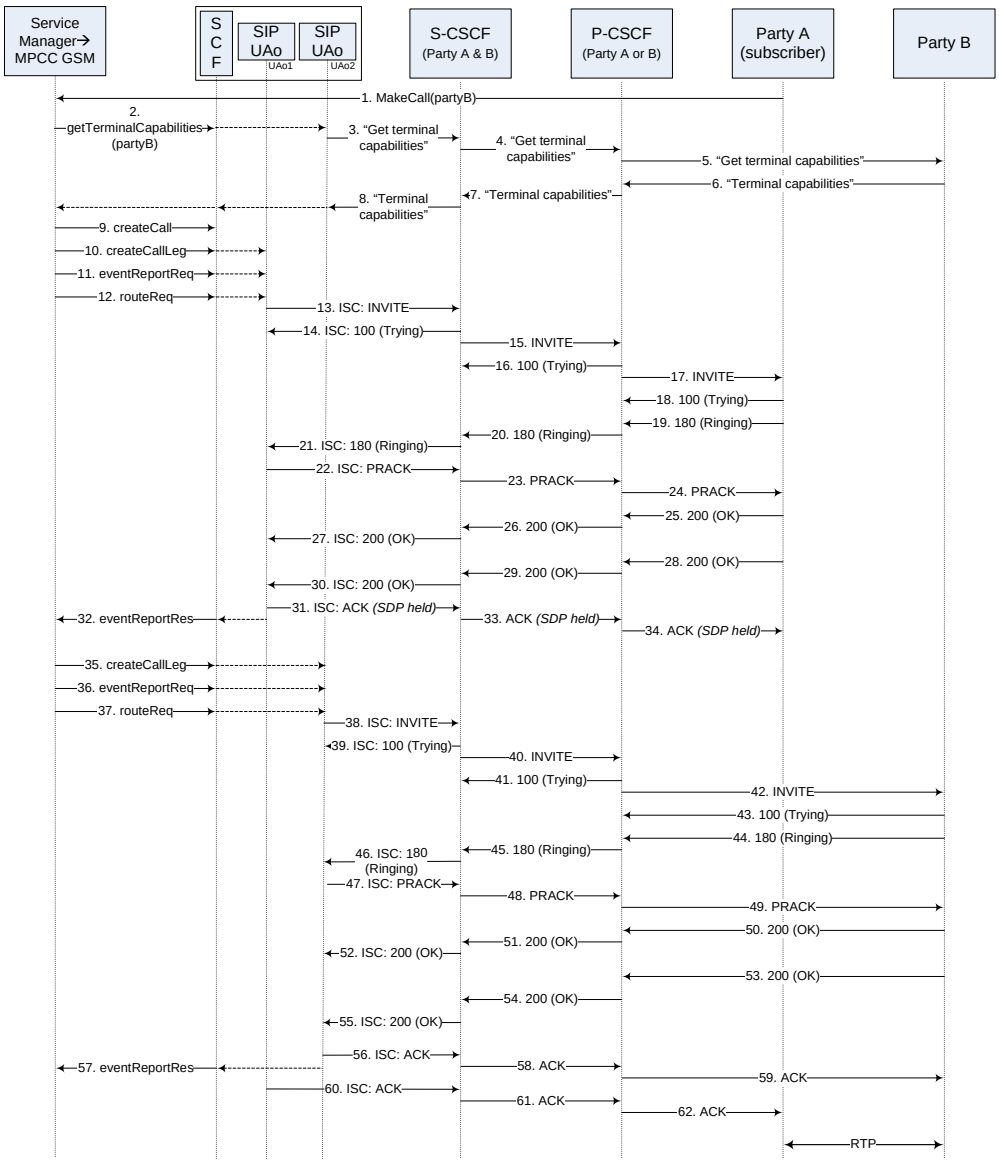


Figure 8-2: Simple 3rd party call initiation between 2 parties

A broad overview of the major parts of the MSC in figure 8.2 follows:

- Message 1: Party A requests that a call be made to Party B.
- Messages 2-8: The MPCC GSM determines the terminal capabilities of Party B.
- Message 9: The framework for the new call is created.
- Messages 10-34: The call leg relating to Party A is established.

- Messages 35-59: The call leg relating to Party B is established.
- Messages 60-62: The call between Party A and Party B is successfully created.

Note that the MPCC GSM is able to determine the Quality of Service (QoS) for the call in a simple and straightforward manner, using the terminal capabilities of both parties. (It is assumed that the Service Manager, and thus the MPCC GSM, knows the terminal capabilities of the subscriber (Party A) and thus does not have to query Party A's terminal.)

In figure 8.2, the 2-party call initiated using 3rd party call initiation is shown to require 51 SIP messages. (Figure 8.2 also shows that the 3rd party call requires 11 application layer messages. However, unlike for the SIP messages, it is assumed that the majority of these messages use CORBA as a distribution mechanism, and thus occur almost instantaneously.)

Complex call initiation between 2 parties

Simple call initiation between 2 parties, shown in figures 8.1 and 8.2, assumed that both parties used the same S-CSCF. *Complex call initiation between 2 parties* extends these examples and provides a more realistic scenario by dropping this assumption.

Figure 8.3 presents the MSC showing call initiation between 2 parties, using 1st party call initiation (as used in the existing approach). This MSC is taken from [14]. Notice that each party uses a different S-CSCF, and that an I-CSCF (Interrogating – Call Session Control Function) and an HSS (Home Subscriber Server) are used by Party A's S-CSCF to locate the S-CSCF of Party B.

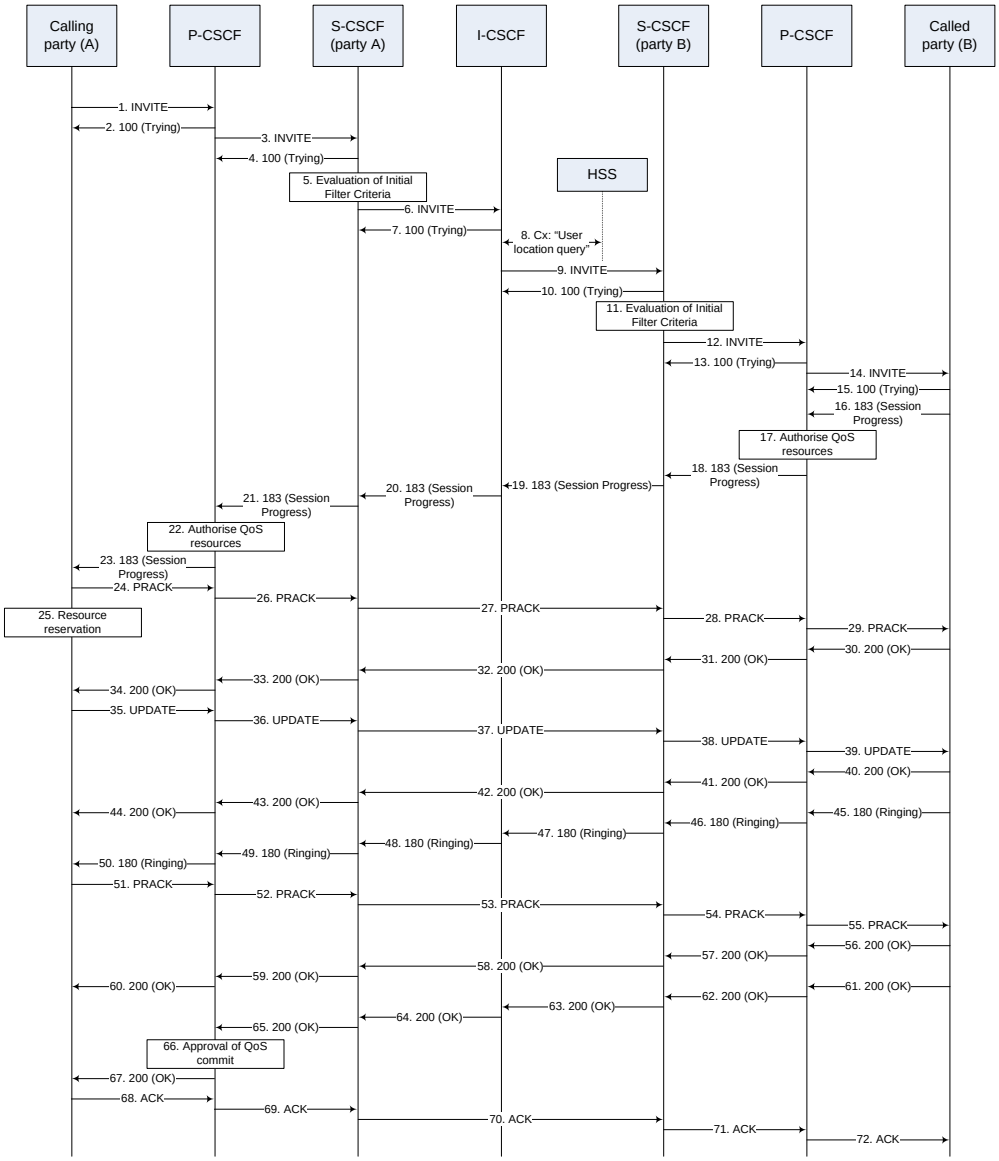


Figure 8-3: Complex 1st party call initiation between 2 parties

Figure 8.3 shows that 65 SIP messages are required (and 6 other actions, shown in boxes). Thus, when the assumption that both parties use the same S-CSCF is dropped, 17 extra SIP messages are required.

Figure 8.4 presents the MSC showing *complex call initiation between 2 parties*, using 3rd party call initiation (as used in the proposed approach). As with figure 8.2, the mapping of Parlay API methods to SIP messages is taken from [15], and the SIP messages exchanged between the CSCFs and parties A and B are taken from [14].

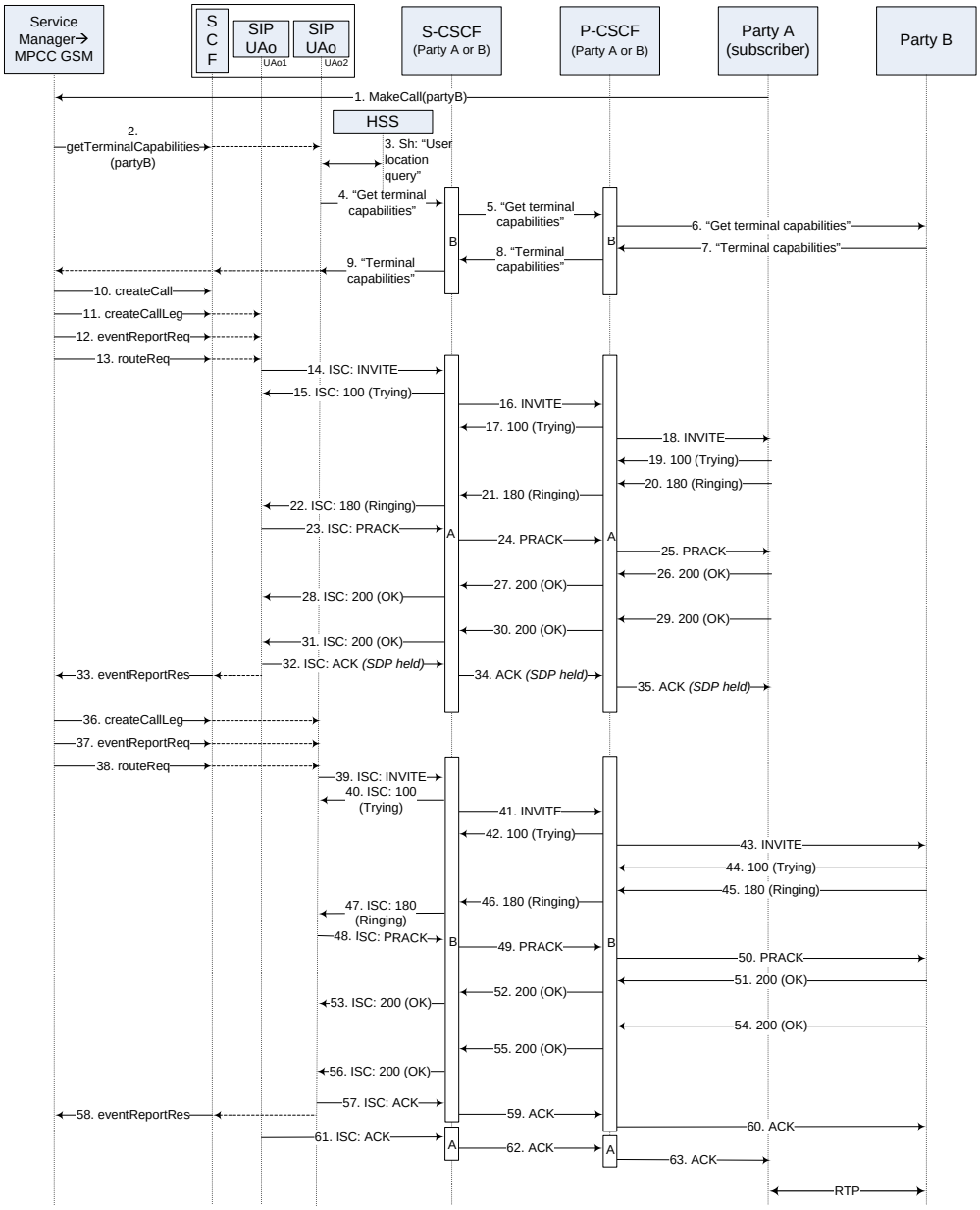


Figure 8-4: Complex 3rd party call initiation between 2 parties

The MSC in figure 8.4 follows the same pattern described for figure 8.2, with only a single exception: Message number 3 in figure 8.4 shows the Parlay gateway querying the HSS for the location of Party B. This message is necessary since the assumption that both parties are served by the same S-CSCF has been dropped. Besides this single addition, the MSCs are the same. Note again how the MPCC GSM determines the QoS for the call using knowledge of the terminal capabilities of both parties.

The 2-party call initiated using 3rd party call initiation requires only 51 SIP messages. Therefore, in setting up a 2-party call, 3rd party call initiation reduces the amount of processing by a factor of 65/51 = 1.275 (referred to hereafter as the *2-party gain*).

Note that, when it was assumed that both parties use the same S-CSCF, 51 SIP messages were also required. Dropping the assumption that both parties are served by the same S-CSCF does not affect the number of SIP messages required when the call is initiated using 3rd party call initiation.

2-party connections: A comparison between the existing and proposed approaches

When it is assumed that both parties use the same S-CSCF, 1st and 3rd party call initiation use approximately the same number of SIP messages (48 and 51, respectively). Thus, in the unlikely case that these restrictive assumptions hold, the proposed approach, using 3rd party call initiation, does not lead to any efficiency gains. (However, various other gains, such as increased control and added flexibility, are still afforded to the environment implementing 3rd party call initiation.)

When the assumption that both parties use the same S-CSCF is dropped, the proposed approach (using 3rd party call initiation) does realise efficiency gains over 1st party call initiation. In this case, 3rd party call initiation requires 14 fewer SIP messages than does 1st party call initiation.

More importantly, dropping the assumption that both parties use the same S-CSCF does not increase the number of SIP messages required in the proposed approach at all. Continuing this trend, as further complexities are introduced which complicate the call initiation process, the efficiency gains provided by the proposed approach over the existing approach continue to increase.

How the number of SIP messages required to initiate a call ultimately affects the post-dial delay is a function of numerous practical considerations, including network congestion. In practical experiments on call initiation using SIP over the internet, it was found that the post-dial delay is increased by approximately 200ms for every hop, and that two terminals would be an average of 6 hops apart from each other [16]. Therefore, the

average 2-party call that is established using 1st party call initiation over the internet has a post-dial delay of approximately 1.2 seconds.

However, since 3rd party call initiation ensures that the number of hops in any 2-party call is limited to 2 (one for each party, where each hop is from the Parlay gateway to the S-CSCF serving that party), the average call that is established using 3rd party call initiation over the internet has a post-dial delay of approximately 400ms.

Although the estimate of 200ms per hop is a function of various network conditions, the number of hops required to initiate a call is only a function of network topology. Since 3rd party call initiation requires only 2 hops, and 1st party call initiation of a 2-party call requires approximately 6 hops, 3rd party call initiation requires about a third of the time required for 1st party call initiation to establish a 2-party call over the internet (independent of the delay per hop).

Note that, in figure 8.4, two major blocks of messages can be identified:

- Messages 11-35: Setup of Party A
- Messages 36-60: Setup of Party B

Reference to these blocks is made in the next section.

This section analysed the relative performance of 1st and 3rd party call initiation under different sets of assumptions, where the number of parties participating in the call was maintained at 2. In sections 8.2.2 and 8.2.3, the relative performance of 1st and 3rd party call initiation is analysed again; however, this time, the assumptions used for call initiation are kept constant, and the number of parties participating in the call is varied. (The assumptions that are used are the same as those used in the *complex call initiation* scenario, shown in figures 8.3 and 8.4: the parties to the call are not presumed to be served by the same S-CSCF.)

8.2.2. Multiparty connectivity without a bridge

Multiparty conferences can generally be structured in one of two topologies; namely in a mesh topology, or a bridge topology. This section and the following section both look at the case of multiparty conferences, and each section considers multiparty conferences from a different perspective, based on the assumption of the conference topology. This

section analyses multiparty conferences with the assumption that the conference uses a full-mesh topology. The creation of a full-mesh multiparty call requires 2-party calls to be setup between every possible combination of users [17], as shown in figure 8.5.

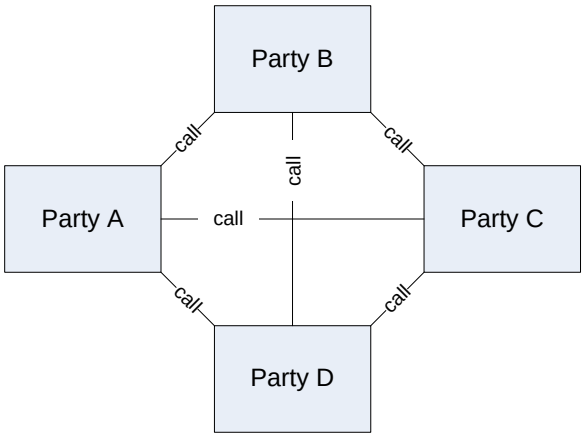


Figure 8-5: Full-mesh conference topology

Multiparty connections in the existing approach (using 1st party call initiation)

Figure 8.3 illustrated the message sequence required to establish a connection between 2 parties in the existing approach (using 1st party call initiation). This case is extended to include additional parties to the call. For full-mesh conferences, the existing approach effectively requires that each and every party in the multiparty call establish a regular 2-party connection with every other party.

Consider the initiation of a 3-party call (a multiparty call with 3 parties) in the existing approach, as shown in figure 8.6. For a full-mesh 3-party call to be established, the following 2-party calls need to be established:

- Party A ←→ Party B
- Party A ←→ Party C
- Party B ←→ Party C

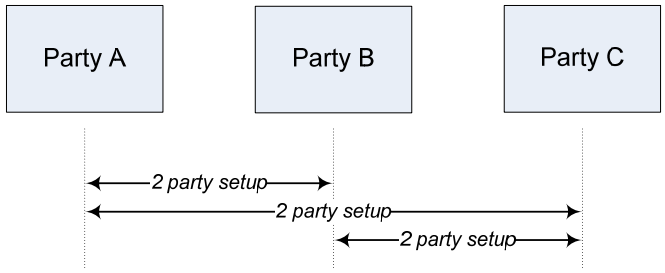


Figure 8-6: The creation of a 3-party call using 1st party call initiation

Since, in the existing approach, the creation of a 3-party call requires the three 2-party calls to be established, a 3-party call requires 3 times the number of messages and 3 times the amount of processing that would be required to set up a single, regular, 2-party call.

In figure 8.7, the 3-party call example is extended to show the process required to set up a 4-party conference (multiparty call) in the existing approach.

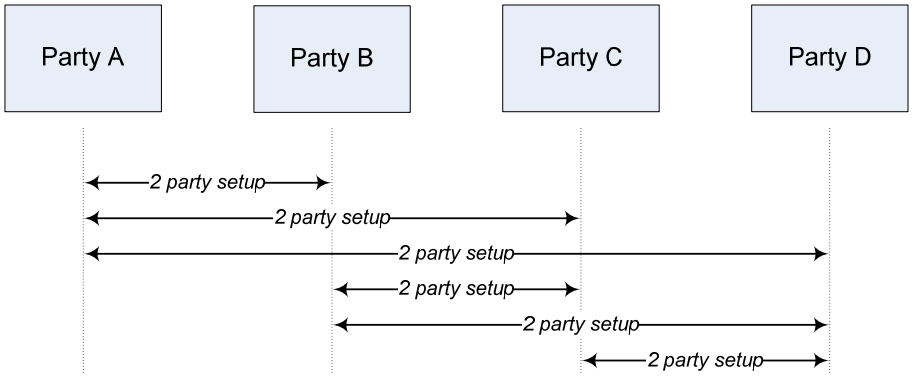


Figure 8-7: The creation of a 4-party call using 1st party call initiation

Again, when 1st party call initiation is used, 2-party calls need to be established between every possible combination of users to effectively create a 4-party conference. Whereas the setup of a 3-party call requires the use of three 2-party calls, a 4-party call requires six 2-party calls.

As more parties are added to the conference call, this exponential trend in the processing requirements continues, and can be explained by the following equation:

$$\text{Processing}_{\text{X party call}} = {}^{x}C_{2} * \text{Processing}_{\text{2 party call}} \quad (1)$$

Equation (1) describes the amount of processing that is required to initiate a multiparty call with X parties as a function of the number of 2-party calls that need to be established.

At the margin, the processing required to add a single additional party to the multiparty call in the existing approach is described by the following equation:

$$\text{Marginal processing}_{\text{to include user \# X}} = (X-1) * \text{Processing}_{\text{2 party call}} \quad (2)$$

Equation (2) describes the amount of additional processing that is required to add a single party to a multiparty call, consisting of (X-1) parties, as a function of the additional number of 2-party calls that need to be established.

From this marginal equation, the following equation can be deduced:

$$\text{Processing}_{\text{X party call}} = \text{Processing}_{\text{(X-1) party call}} + (X-1) * \text{Processing}_{\text{2 party call}} \quad (3)$$

Equation (3) implies that, in the existing approach, the total amount of processing required to establish a multiparty conference increases exponentially with a linear increase in the number of parties to the call.

Translating the number of SIP messages that are required to initiate a (multiparty) call to a post-dial delay (or setup time) is highly dependent on various practical considerations; specifically, the setup time is determined by the number of hops between terminals (which is a function of network topology), and by the average delay per hop (which is a mainly a function of network congestion). Assuming the multiparty call is a full-mesh call (where every party has a 2-party connection with every other party), and assuming all parties in the multiparty call are added at the same time, the time taken to establish the multiparty call is given by the following equation:

$$\text{Setup time}_{\text{X party call}} = {}^{x}C_2 \text{ (2-party calls)} * \text{(Hops betw. terminals)} * \text{(time/hop)} \quad (4)$$

Multiparty connections in the proposed approach (using 3rd party call initiation)

Figure 8.4 illustrated the message sequence required in the proposed approach to establish a connection between 2 parties, using 3rd party call initiation. Figure 8.8 extends the 2-party case to show the inclusion of an additional party in the call.

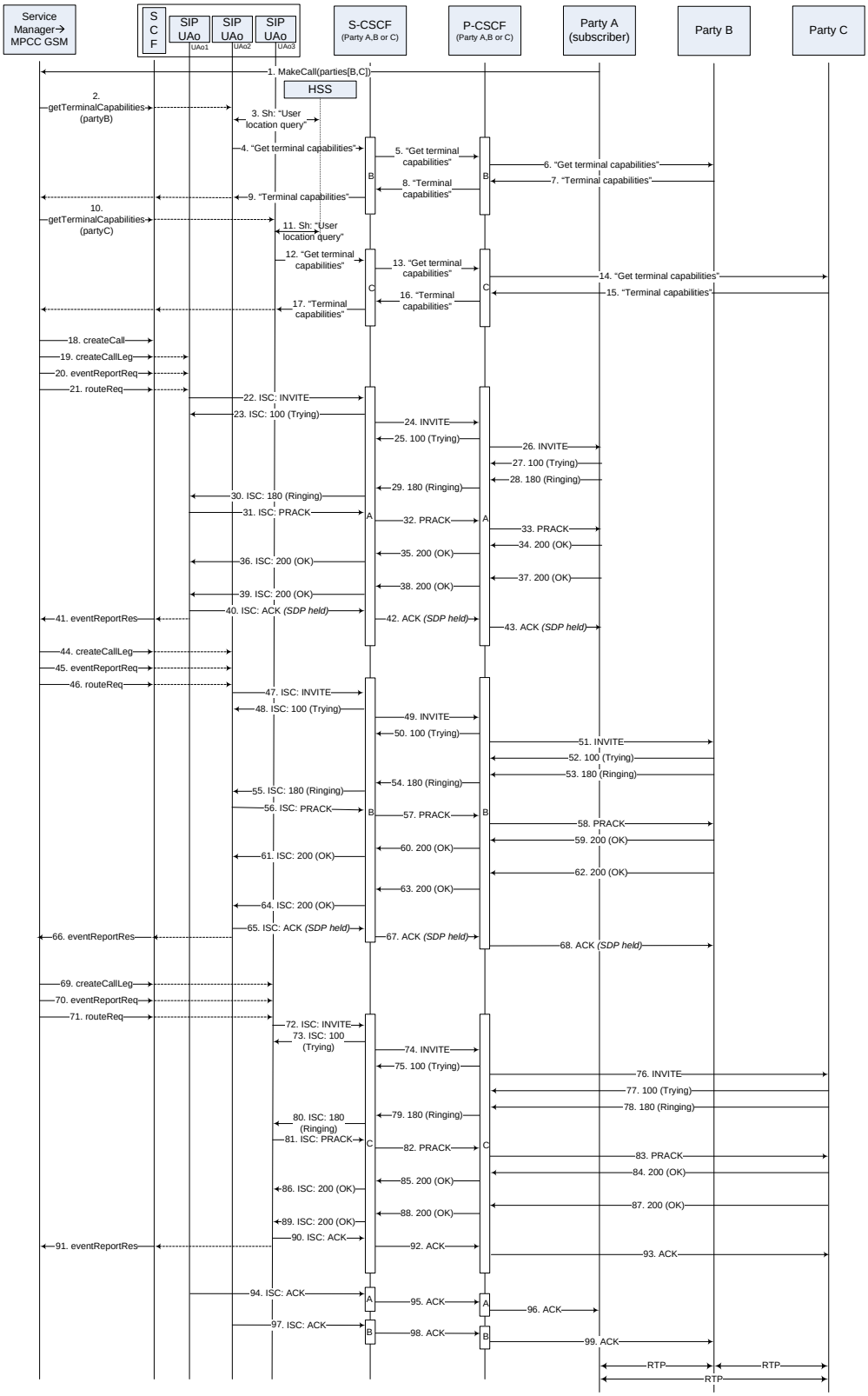


Figure 8-8: 3rd party call initiation between 3 parties

In figure 8.7, three major blocks of messages can be identified:

- Messages 19-43: Setup of Party A
- Messages 44-68: Setup of Party B
- Messages 69-93: Setup of Party C

Recall that in figure 8.4, where a 2-party call was established using 3rd party call initiation, 2 blocks of messages were required; one block per party. In figure 8.8, where a 3-party call is established using 3rd party call initiation, 3 blocks of messages are required; again, one block per party.

Figure 8.9 shows the creation of a 3-party call using 3rd party call initiation, where the message blocks used to setup each party to the call are shown abstractly.

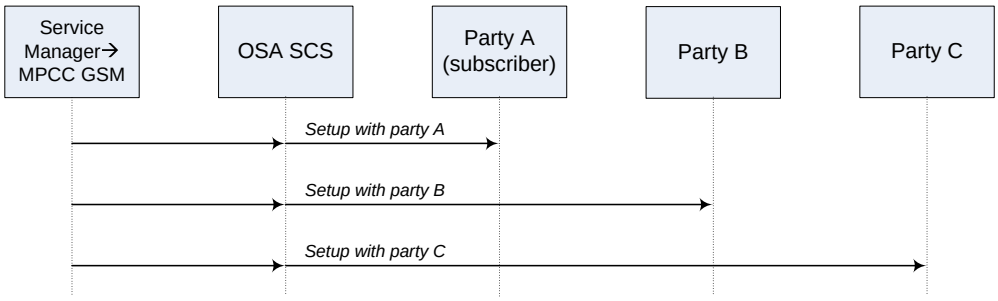


Figure 8-9: The creation of a 3-party call using 3rd party call initiation

In figure 8.10, the 3-party call is extended to show the process required to set up a 4-party conference, using 3rd party call initiation. Note that the call setup depicted in both figures 8.9 and 8.10 effectively utilise a centralised bridge.

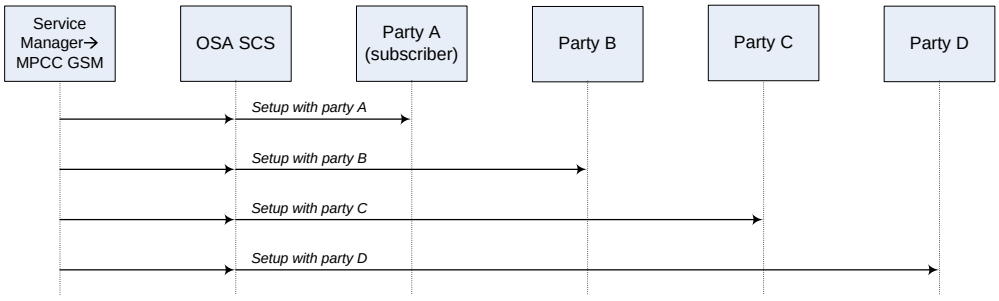


Figure 8-10: The creation of a 4-party call using 3rd party call initiation

Notice that the trend continues: as additional parties are added to the multiparty call, the processing required increases linearly. This is in contrast to the existing approach, where a linear increase in the number of parties in a multiparty call resulted in an exponential increase in the amount of processing required.

To quantify this relationship in a way which is conducive to comparison with the relationships derived for the existing approach, the amount of processing that is required to establish a multiparty call in the proposed approach is also described as a function of an equivalent number of 2-party calls. Using 3rd party call initiation, the following equation holds:

$$\text{Processing}_{\text{X party call}} = X/2 * \text{Processing}_{\text{2 party call}} \quad (5)$$

At the margin, the processing required to add a single additional party to the multiparty call in the proposed approach is described by the following equation:

$$\text{Marginal processing}_{\text{to include user \# X}} = 0.5 * \text{Processing}_{\text{2 party call}} \quad (6)$$

From this marginal equation, the following equation can be deduced:

$$\text{Processing}_{\text{X party call}} = \text{Processing}_{\text{(X-1) party call}} + 0.5 * \text{Processing}_{\text{2 party call}} \quad (7)$$

Equation (7) implies that, in the proposed approach, the processing required to add an additional user to a conference increases linearly as the conference grows in size.

In estimating the setup time for a multiparty call established using 3rd party call initiation, recall that each party requires only a single hop for its setup. Therefore, the setup time for a multiparty call consisting of X parties is given by:

$$\text{Setup time}_{\text{X party call}} = X * (\text{time/hop}) \quad (8)$$

Multiparty connections: A comparison between the existing and proposed approaches

The existing and proposed approaches are compared under 3 categories: total processing required to establish a multiparty call, marginal processing required for the addition of one more party to the call, and total time required to establish a multiparty call. The first two categories, viz. total and marginal processing, are based on theoretical relationships

developed in this report. The third category, total setup time, is based on practical measurements made in [16].

Total processing required to establish a multiparty call

At the beginning of this section, it was stated that the number of SIP messages required to carry out a task would be used as a proxy for efficiency, where fewer SIP messages translate into greater efficiency. In section 8.2.1, it was determined that, for a 2-party call, using 1st party call initiation requires 65 SIP messages, and that 3rd party call initiation requires 51 SIP messages. Using these values in equations 1 and 5, respectively, an estimate can be made for the number of SIP messages required to establish a multiparty call, with a varying number of parties.

Figure 8.10 compares the total number of SIP messages required to establish a multiparty call of various sizes in the existing and proposed approaches. (It is assumed that the initiating party uses a different S-CSCF to the other parties to the call.)

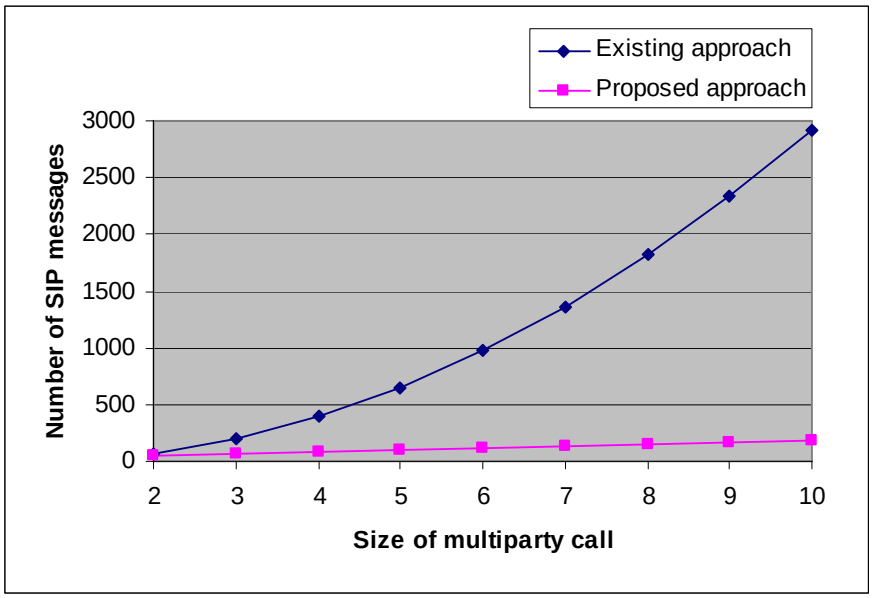


Figure 8-11: Number of SIP messages required in a multiparty call

Although similar efficiency is achieved using either 1st or 3rd party call initiation for a 2-party call, the use of 3rd party call initiation (in the proposed approach) results in

increasingly larger efficiency gains as the number of parties in the multiparty call increases, as compared to using 1st party call initiation (in the existing approach).

Figure 8.11 provides absolute estimates of the processing required to establish multiparty calls using 1st and 3rd party call initiation. The relative performance of the two approaches is now determined. Comparing equations 1 and 5, the proposed approach, using 3rd party call initiation, reduces the total amount of processing required to establish an X-party call by a factor of

$$(\text{2-party gain}) * (2 * {}^{x}C_2)/X = (2.55 * {}^{x}C_2)/X \qquad (9)$$

For example, as compared with the existing approach, the following processing reductions are realised by using the proposed approach:

- For a 3-party call: 2.6x less processing
- For a 4-party call: 3.8x less processing
- For a 5-party call: 5.1x less processing

In figure 8.12, the bottom line provides a graphical depiction of this relationship, where the x-axis shows the number of parties in the multiparty call, and the y-axis shows the factor reduction in total processing.

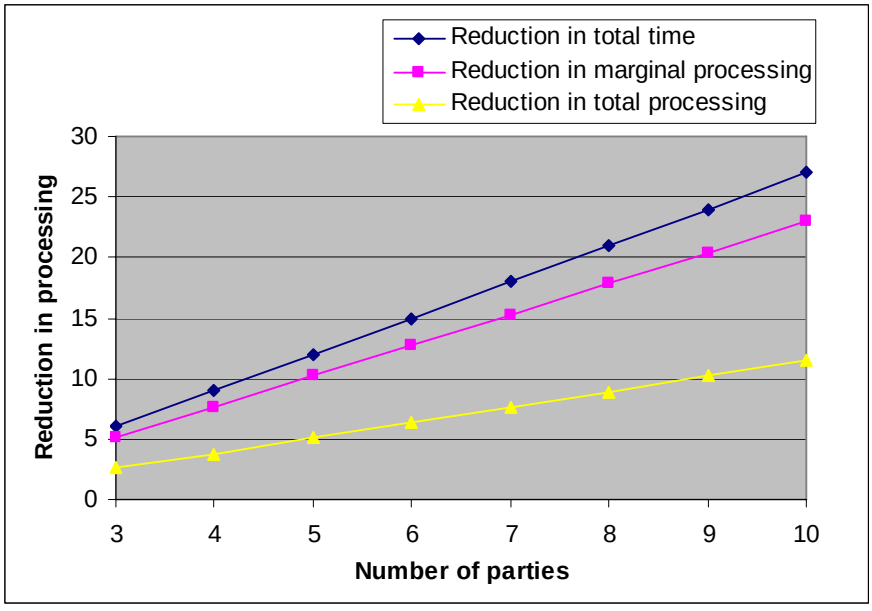


Figure 8-12: Processing gains for 3rd party call initiation

Marginal processing required for the addition of one more party to a multiparty call

Comparing equations 2 and 6, the proposed approach, using 3rd party call initiation, reduces the marginal processing required to include an additional party to an existing (X-1)-party call by a factor of

$$(\text{2-party gain}) * 2 * (X-1) = 2.55 * (X-1) \qquad (10)$$

For example, as compared with the existing approach, the following processing reductions are realised by using the proposed approach:

- For the addition of a 3rd party: 5.1x less processing
- For the addition of a 4th party: 7.7x less processing
- For the addition of a 5th party: 10.2x less processing

The middle line of figure 8.12 provides a graphical depiction of this relationship, where the x-axis shows the number of parties in the multiparty call, and the y-axis shows the percentage reduction in the marginal processing required to add the last party to the conference.

Total time required to establish a multiparty call

Comparing equations 4 and 8, the proposed approach, using 3rd party call initiation, reduces the total time required to establish a multiparty call by a factor of

$$(\text{Hops between terminals}) * {}^{x}C_2 / X \qquad (11)$$

Using the assumption given in [16] that the average number of hops between terminals in the internet is 6, 3rd party call initiation reduces the total time required to establish a multiparty call by a factor of

$$6 * {}^{x}C_2 / X \qquad (12)$$

For example, the following setup time reductions are realised by using the proposed approach (assuming the number of hops between terminals is 6):

- For a 2-party call: 3x less time
- For a 3-party call: 6x less time
- For a 4-party call: 9x less time

The top line of figure 8.12 provides a graphical depiction of this relationship, where the x-axis shows the number of parties in the multiparty call, and the y-axis shows the percentage reduction in the total time required in the setup of the multiparty call.

The analysis above is based on relation number 12, and removed one of the variables from relation number 11 by assuming that the number of hops between terminals remained constant (and equal to 6). From relation 12, the resulting gains from using 3rd party call initiation was calculated as a function of the number of parties in the call.

Relation number 11 is now analysed from a different perspective: it is assumed that the number of parties in the call remains constant, and the resulting gains by using 3rd party call initiation is calculated as a function of the number of hops between terminals. Specifically, 3rd party call initiation of a 3-party call reduces the total time required to establish a multiparty call by a factor of

$$(\text{Hops between terminals}) * {}^{3}C_{2} / 3 = (\text{Hops between terminals}) \quad (13)$$

For example, the following setup time reductions are realised by using the proposed approach (assuming the number of parties in the call is 3):

- For 3 hops between terminals: 3x less time
- For 4 hops between terminals: 4x less time
- For 5 hops between terminals: 5x less time

Therefore, as the number of hops between terminals increases, the reduction in the setup time of a 3-party call realised by using 3rd party signalling increases (proportionally).

Note that, although the reductions in the total time achieved by using 3rd party signalling are dependent on the prevailing network conditions, the relationship between the relative performance of 1st and 3rd party call initiation remains the same. Therefore, the setup time reductions, implied by relations 11, 12 and 13, will hold, irrespective of prevailing network conditions.

To make absolute time estimates of the post-dial delay (or setup time), assumptions need to be made of the number of hops between terminals, and the amount of delay incurred per hop. Both of these estimates are dependent on the network that is used for the call. Figure 8.13 depicts the setup time for the initiation of a 3-party call for both 1st and 3rd

party call initiation, using the estimates made in [16] for call initiation over the internet (hops between terminals = 6, delay per hop = 200ms).

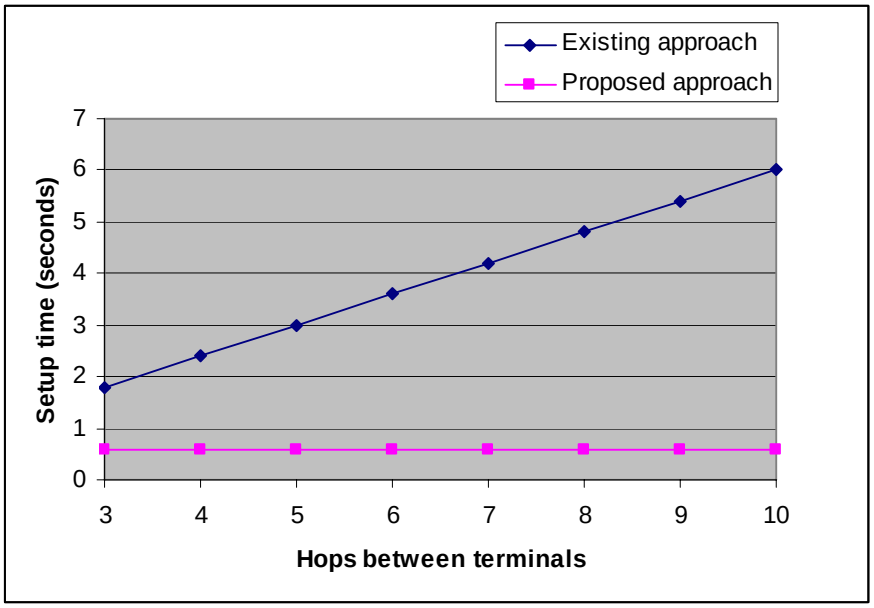


Figure 8-13: Setup time for a 3-party call over the internet

Note that the setup time for 3rd party call initiation is independent of the number of hops between terminals.

Assuming there are approximately 6 hops between terminals over the internet, it would take approximately 3.6 seconds to set up a typical 3-party call using 1st party call initiation (vs. 0.6 seconds using 3rd party call initiation). To set up a 4-party call using 1st party call initiation, it would take approximately 7.2 seconds (again, vs. 0.6 seconds using 3rd party call initiation). It appears that full-mesh conferences over the internet with more than 3 parties become an increasingly impracticable approach when 1st party call initiation is used.

8.2.3. Multiparty connectivity with a bridge

Multiparty conferences are now analysed from the viewpoint that the conference uses a bridge topology, as shown in figure 8.14. In a conference set up as with a bridge topology, each party maintains only a single connection: to the bridge.

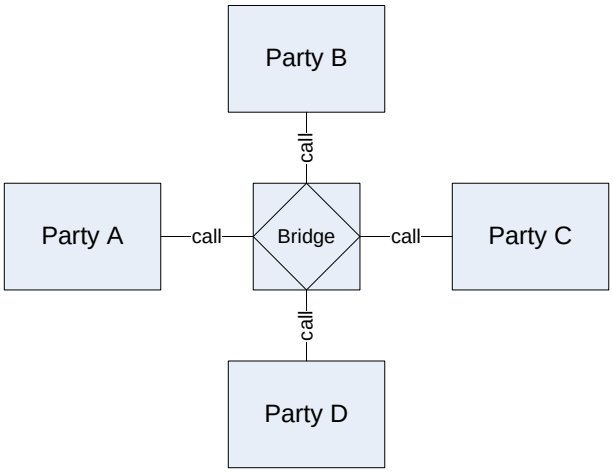


Figure 8-14: Bridge conference topology

A comparison between figure 8.5 and figure 8.14 shows that, while a full-mesh conference between *X* parties requires *X(X-1)/2* connections, an equal sized conference using the bridge topology requires only *X* connections.

Multiparty connections in the existing approach (using 1st party call initiation)

To illustrate the existing approach used to establish a multiparty conference with a bridge topology, consider the example shown in figure 8.15.

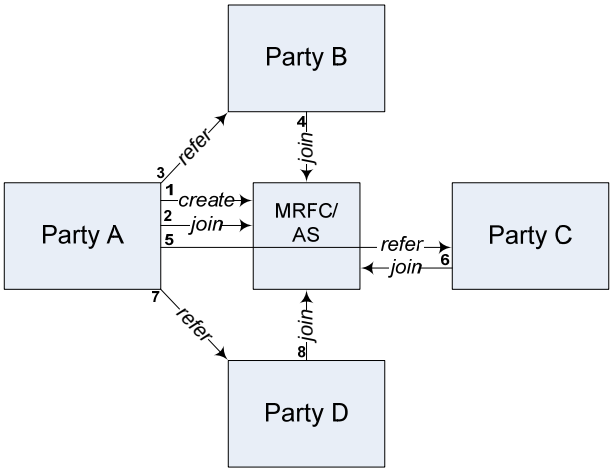


Figure 8-15: Existing approach to 4-party conference initiation using a bridge

In this example, Party A wishes to establish a 4-party conference. The steps needed to establish the conference are as follows:

1,2. Party A creates and joins the conference

3,4. Party A sends a *refer* message to Party B, inviting Party B to join the specified conference which has been created. Party B joins the conference.

5,6. Party A sends a *refer* message to Party C, inviting Party C to join the specified conference which has been created. Party C joins the conference.

7,8. Party A sends a *refer* message to Party D, inviting Party D to join the specified conference which has been created. Party D joins the conference.

The process can be easily generalised to allow for an X-party conference call, where each additional party needs be referred to the conference by Party A, and then subsequently join the conference.

To facilitate a comparison between the relative efficiencies of the existing and proposed approaches to conference setup, a measure needs to be made of the total amount of processing that is required to establish the conference in any given approach. Equation (14) quantifies the total amount of processing required to establish an X-party call in the existing approach (using 1st party call initiation):

$$\text{Processing}_{\text{X party call}} = \text{(Conf. creation)} + \text{(Referral process)}*(X-1) + \text{(Joining process)}*X \qquad (14)$$

Equation (14) describes the amount of processing that is required to initiate a multiparty call with X parties as a function of the number of referrals between parties and the number of joining processes that are required.

As in the previous section, the number of SIP messages required to perform the conference setup is used as a proxy for the degree of inefficiency of the approach. The dependent variable of equation (14), *Processing*, provides a measure of the number of SIP messages required for the conference setup. The number of SIP messages required for the *joining process* and for the *referral process* are now determined.

Figure 8.16 presents the MSC showing the process that is required for a party to join a bridge-topology conference call (the *joining process*).

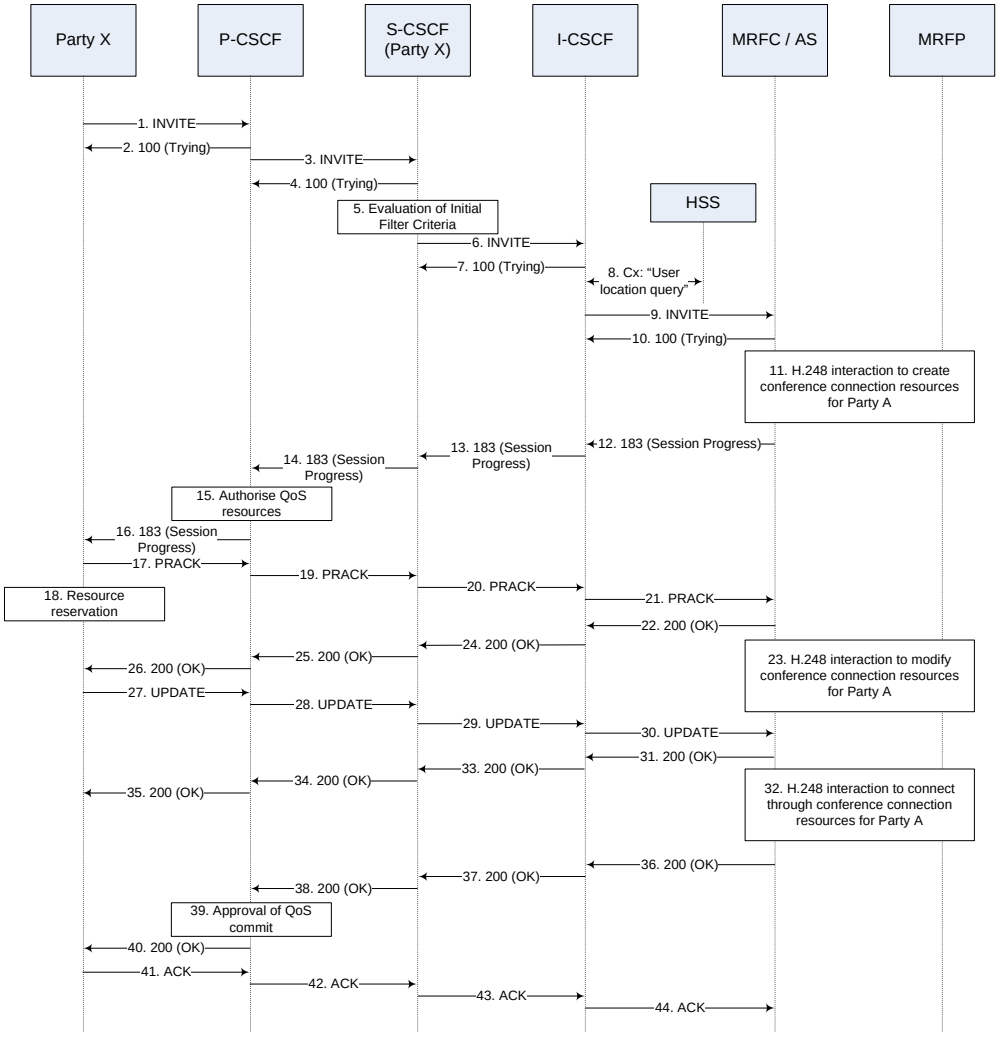


Figure 8-16: MSC showing single party joining a bridge conference

Figure 8.16 is taken directly from [18]. Please refer to this document for a full description of any of the messages. Figure 8.16 shows that 36 SIP messages (and a number of other operations) are required to be exchanged to allow a party to join a conference that uses a bridge topology.

Figure 8.17 presents the MSC showing the process that is required for a party to refer another party to an existing conference (the *referral process*) that uses a bridge topology. As with figure 8.16, figure 8.17 is taken directly from [18], which provides a full description of each of the messages.

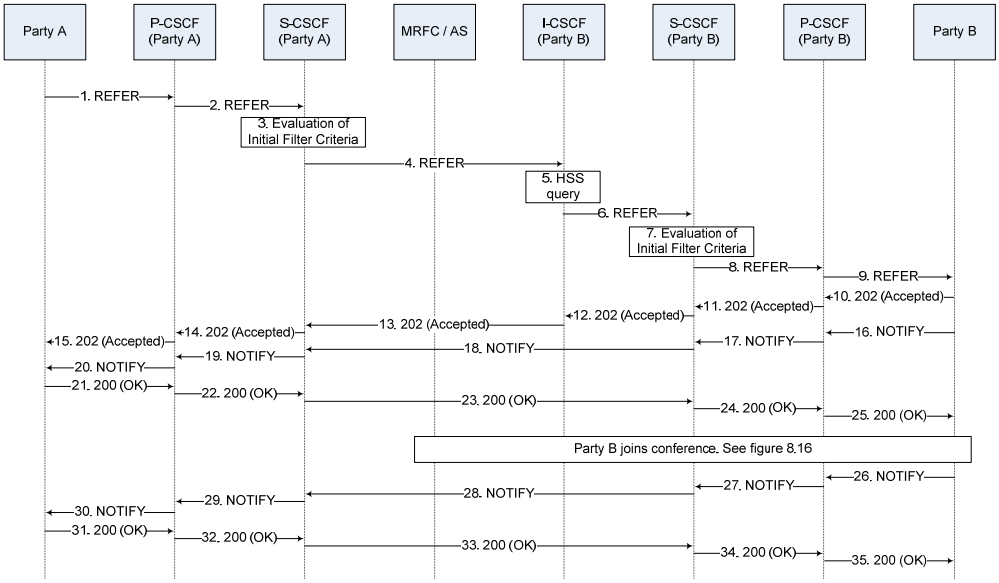


Figure 8-17: MSC showing the referral of a party to a bridge conference

Figure 8.17 shows that 32 SIP messages (and a number of other operations) are required to be exchanged to allow a party to refer another party to an existing conference (that uses a bridge topology).

Reference is made in a later subsection to the number of SIP messages required for the *joining process* and the *referral process*, when the existing approach to conference creation (discussed in this section) is compared to the proposed approach, using the relevant equations.

Multiparty connections in the proposed approach (using 3rd party call initiation)

The proposed approach to establishing multiparty conferences using a bridge topology is illustrated by the example shown in figure 8.18.

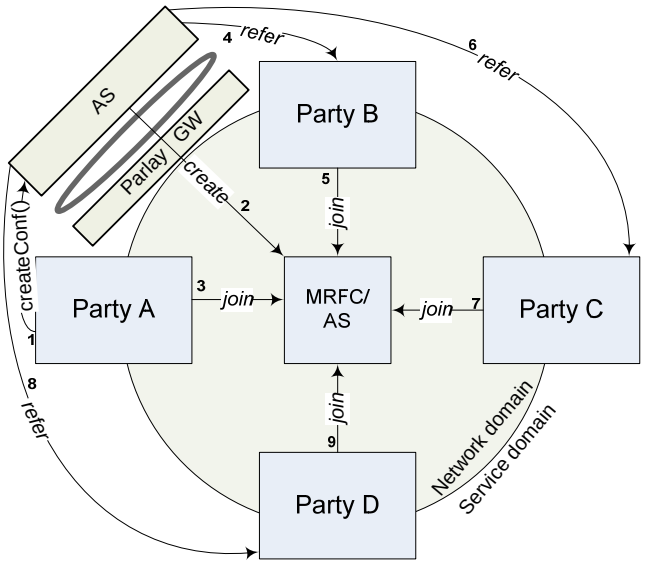


Figure 8-18: Proposed approach to 4-party conference initiation using a bridge

Similar to the example shown in figure 8.15, Party A wishes to establish a 4-party conference. The steps needed to establish the conference, in the proposed approach, are as follows:

1. Using application layer signalling, Party A signals to a conferencing service (provisioned as an application) in the service domain.
2. Using the Parlay APIs and the Parlay gateway, the conference service signals to the conference bridge in the network to create the conference. The conference service obtains the conference URI for the newly created conference, and returns the URI to Party A.
3. Using the conference URI, Party A joins the conference (using SIP signalling).
4. Using application layer signalling, the conference service invites Party B to join the conference, and provides Party B with the conference URI.
5. Using the conference URI, Party B joins the conference (using SIP signalling).

6,7; 8,9. Similar to messages 5 and 5 (for Party B), Parties C and D receive an invite message from the conference service (at the application layer), and then proceed to join the conference (at the network layer).

This process can be easily generalised to allow for an X-party conference call. To compare the existing approach used to establish a conference call (using a bridge topology) to the proposed approach, the number of SIP messages that are required to be exchanged to establish the conference needs to be quantified. Note that, in the proposed

approach (shown in figure 8.18), all referrals are made in the application layer, and are thus assumed to require negligible time for their processing. The only tasks which require the exchange of SIP messages are the initial conference creation and the joining procedures. Equation (15) quantifies the total amount of processing required to establish an X-party call in the proposed approach (using 3rd party call initiation):

$$\text{Processing}_{\text{X party call}} = (\text{Conf. creation}) + (\text{Joining process})*X \qquad (15)$$

Multiparty connections: A comparison between the existing and proposed approaches

The primary difference between the existing and proposed approaches to conference creation (using a bridge topology) is that the proposed approach (using 3rd party call initiation) performs the *referral processes* in the application layer, as opposed to the network layer. Since messages in the application layer are exchanged far more efficiently and quickly than SIP messages in the network layer, the proposed approach to establishing multiparty conferences (based on a bridge topology), using 3rd party call initiation, offers greater efficiency than the existing approach.

When the actual number of SIP messages that are required to be exchanged for the *joining process* and the *referral process* (as determined in figures 8.16 and 8.17) is substituted into equations 14 and 15, the difference in the efficiency of the existing and proposed approaches can be quantified. (In the comparison presented next, it is assumed that the number of SIP messages required for conference creation in each approach is comparable, and insignificant in the context of the conference setup as a whole.)

Comparing equations 14 and 15, with the appropriate number of SIP messages substituted for the *joining process* and the *referral process*, the proposed approach, using 3rd party call initiation, reduces the total amount of SIP messages required to establish an X-party call by a factor of

$$(68*X - 32) / (36*X) \qquad (16)$$

For example, as compared with the existing approach, the following processing reductions are realised by using the proposed approach:

- For a 3-party call: 1.6x less processing
- For a 4-party call: 1.67x less processing

- For a 5-party call: 1.71x less processing
- For conference call of a larger size, the efficiency gain provided by the proposed approach tends to a factor 1.81.

The proposed approach to the setup of a conference (that uses a bridge topology) using 3rd party call initiation therefore offers a significant efficiency gain over the existing approach, but, in contrast to full-mesh conferences, is not highly dependent on the size of the conference call.

Looking ahead

Chapter 9, next, is the final chapter of this project report, and concludes the report by providing a brief summary of the work performed, and highlights the major results and conclusions.

Chapter 9

IN CLOSING

9.1. The proposed service provisioning environment: A contribution to the Next Generation Network

The objective of any communications network is to offer its users valuable services. The value of any service is measured by the contribution it makes to the users' lives, and the contribution a service can make is dependent on its capabilities. Additionally, the existence of value-added services is a function of their ease of implementation, and the number of service developers possessing the skills to develop these services.

The operational environment of services determines both the capabilities of services and the ease of service implementation.

The changes proposed in this research report go beyond defining a new platform for the implementation and execution of services: the entire operational environment of services is modified. The service-centric approach of the proposed service provisioning environment advocates changes to both the architectural and logical structure of existing telecoms networks in many areas, orienting their focal-point towards service provisioning.

If the ultimate objective of telecoms networks is to offer its users a set of value-added services, then the proposed service provisioning environment, advocated in this research report, constitutes a major contribution to any NGN (Next Generation Network). An NGN that employs the proposed service provisioning environment will be able to deploy more complex services, faster, and in an easier way, thus maximising the value it creates for users.

9.2. Summary of work

The primary objective of this research project was to create an operational environment for services which ultimately creates the most value for the end user. This was attempted through two efforts:

- Design the services' operational environment so that services can be developed and deployed in the simplest way. This ensures that the skills that are required to develop services for communications networks are possessed by the maximum number of service developers.
- Design the services' operational environment to maximally exploit the capabilities of the underlying network. This maximises the range of features and complexity of services that can be supported.

These two objectives were attempted to be met by:

- Redesigning the service domain architecture
    - Instituting structure in the service domain through the use of a layered approach
    - Implementing centralised service control and management
    - Implementing a generic invocation interface for services
    - Advocating software reuse, using RBBs and GSMs
- Modifying the operational environment of the service domain, in the context of the rest of the network
    - Relocation of the primary view and control of the BCS to the application layer
    - Introducing application layer signalling

Each of the areas of work conducted in this research project, which are outlined above, are summarised in the paragraphs that follow.

Redesigning the service domain architecture

This research report presented a well-defined service domain architecture, comprising 3 layers. The *Reuse and Abstraction layer* contains RBBs and GSMs, which offer services (in the *Specific Services layer*) simple, generic and commonly used functionality.

Whereas RBBs offer services reusable blocks of logic that may or may not contain typical telecoms functionality, GSMs provide services with functionality to specifically control network resources in a simple and abstracted way, alleviating services from implementing onerous Parlay message sequences. Together, the GSMs and the RBBs constitute the software reuse framework advocated by the proposed service domain architecture.

The Service Manager, located in the *Service Infrastructure layer*, is responsible for the invocation of all services and maintains a single point of control for services, enabling holistic service management. By ensuring that services have only a single source of possible invocation, the Service Manager provides services with a generic invocation interface, shielding the services from the technical attributes of the invoking domains and methods.

Modifying the service provisioning environment

Application layer signalling provides a direct communication path between the terminals and the service domain, and, by negating the need for the signals to be processed in the network, offers the use of a more powerful, and service-oriented, signalling protocol. The application layer signalling channel, between the terminals and the service domain, is defined by the Application layer API set. The Application layer API set offers services a standardised, network-independent interface, freeing them from the specific technology of the underlying network.

Finally, it was proposed that the primary view and control of the BCS be relocated from the network to the service domain, in the application layer. Having the view and control of the BCS located in the service domain allows a centralised point for connectivity management, facilitating the manipulation and control of bearer connectivity.

9.3. Conclusions

The proposed service provisioning environment allows telecoms services that incorporate a wide range of advanced capabilities and complex features to be developed in a fast and simple way by application developers without detailed telecoms knowledge.

The application layer communication channel provides a high-level, powerful and flexible signalling path between terminals and services. The network-independent nature of application layer signalling renders terminals increasingly network-independent, and promotes the support of multiple types of terminals on any underlying network.

The use of 3rd party call initiation in the proposed service provisioning environment compounds the benefits accorded by application layer signalling and the relocation of the control of the BCS: 3rd party call initiation provides additional control and flexibility to the management of bearer connections, and realises significant performance enhancements in the setup of multiparty calls, as compared with 1st party call initiation used in existing networks.

9.4. Recommendations for future work

The scope of this project was fairly large, as evidenced by the summary of work provided in section 9.2. The stated objectives of the project centred on the development of a structured service domain, and the modification of the service domain's operational environment to the end of creating an environment that is more conducive for the implementation and deployment of services.

In attempting these goals, many new features were proposed. Often, descriptions of certain features were only provided in broad, qualitative terms, sufficient for the understanding of their function. It is considered beyond the scope of this project to fully and definitively define and specify each of these features in detail.

Many of the ideas advanced in this project therefore provide fertile areas for further research and work. In the following paragraphs, certain specific aspects warranting further study have been identified, and are the present author's recommendations for future work.

Work concerning the service domain architecture:

- The structure of the proposed service domain architecture indicates two areas that can be exploited for software reuse: RBBs and GSMs. This project has fully defined their general objectives of and their contexts of operation, and provided some examples. However, a more complete set of RBBs and GSMs was not

provided. The development of a more comprehensive set of RBBs and GSMs would contribute to the practical applicability of the theoretical framework developed in this project.

Work concerning the operational environment of the service domain:

- A major development proposed by the project is the use of application layer signalling. The technical details and actual implementation of application layer signalling were not dealt with, and were thought to be handled by a theoretical "signalling plane". Thus, a technical specification of application layer signalling is needed for the practical implementation of the proposed service provisioning environment.

Work concerning the terminal ←→ service domain interface

- The Application layer API set, and the application layer interface between the service domain and the terminals in general, was designed first and foremost for the expedience of service invocation and execution. The interface was not specifically designed for service management. As the Abbreviated Dialling service example in chapter 8 demonstrated, the interface is able to handle service management. However, for advanced service management requirements, the definition of the interface may require modification.
- At the outset of the project, certain assumptions were made regarding the capabilities of the generic "intelligent terminal". These assumptions led to the Application layer API set, as it has been presented. Different assumptions concerning the intelligent terminal, and concerning the distribution of the service logic between the terminal and the service domain, allow other approaches for achieving application layer communications to be implemented. For example, application layer signalling could be implemented using web-forms containing multiple data fields, Java applets etc. The feasibility and efficacy of such approaches could be the focus of further study.

REFERENCES

[1] Atis Telecom Glossary. http://www.atis.org/tg2k/_communications_network.html, Last accessed 12 February 2006.

[2] Berinato B., *History of telecommunications and data networks.* http://williamstallings.com/Extras/Telecom.html, Last accessed 12 February 2006.

[3] Aepona. http://www.aepona.com/learn_about/questions2.html, Last accessed 12 February 2006.

[4] ETSI ES 203 915-1, *Open Service Access (OSA); Application Programming Interface (API); Part 1: Overview (Parlay 5)*, April 2005.

[5] ETSI ES 203 915-1, *Open Service Access (OSA); Application Programming Interface (API); Part 4: Call Control; Sub-part 3: Multi-Party Call Control SCF (Parlay 5)*, April 2005.

[6] ETSI ES 203 915-1, *Open Service Access (OSA); Application Programming Interface (API); Part 9: Generic Messaging SCF (Parlay 5)*, April 2005.

[7] Hanrahan H., *Convergence: Networks, Services and Applications*. School of Electrical and Information Engineering, University of the Witwatersrand, 2005.

[8] Oni O., *Reusable Block Provisioning for Application Service Providers with Parlay/OSA*, MSc(Eng) Project Report, University of the Witwatersrand, Johannesburg, 2006.

[9] ETSI ES 203 915-1, *Open Service Access (OSA); Application Programming Interface (API); Part 3: Framework (Parlay 5)*, April 2005.

[10] ITU-T Q.1221, *Introduction to Intelligent Network Capability Set*, September 1997.

[11] IEFT RFC 3087, *Control of Service Context using Request-URI*, April 2001.

[12] IETF RFC 3261, *SIP: Session Initiation Protocol*, June 2002.

[13] 3GPP TS 23.228, *IP Multimedia Subsystem (IMS); Stage 2 V. 7.3.0*, March 2006.

[14] 3GPP TS 24.228, *Signalling flows for the IP multimedia call control based on Session Initiation Protocol (SIP) and Session Description Protocol (SDP); Stage 3 V.5.14.0*, December 2005.

[15] 3GPP TS 29.998, *Open Service Access (OSA); Application Programming Interface (API) Mapping for Open Service Access; Part 4: Call Control Service Mapping; Subpart 4: Multiparty Call Control ISC V.6.0.4*, December 2004.

[16] Singh K., Schulzrinne H., *Peer-to-Peer Internet Telephony using SIP*. Department of Computer Science, Columbia University.

[17] Miladinovic I., Stadler J., *Multiparty Conference Signalling using the Session Initiation Protocol (SIP)*. Institute of Communication Networks, Vienna University of Technology.

[18] 3GPP TR 29.847, *Conferencing based on SIP, SDP and other protocols; Functional models, information flows and protocol details*, June 2004.