

3GPP TSG_CN#7
ETSI SMG3 Plenary Meeting #7,
Madrid, Spain
13th – 15th March 2000

NP-000059

Agenda item: 5.5
Source: TSG_N OSA ad hoc
Title: Approved Liaison Statements

LIST OF APPROVED LIAISON STATEMENTS
from the OSA ad hoc Group

Tdoc OSA	Destination group(s)	Subject
00022	S3	Statement on security issues in VHE/OSA
00024	S2	Annex A of VHE/OSA Stage 2 (3G TS 23.127 v 1.1.1)
00054	S2	VHE OSA time schedule
00102	S2	Aligment of Stage 2 material
00103	S2	Sequence diagrams for Release 2000

**OSA Ad-hoc Meeting, Tilburg, Holland
6 ÷ 7 January, 2000**

From: CN OSA ad-hoc
To: TSG SA WG3
Cc: TSG SA WG1, TSG SA WG2
Title: Statement on security issues in VHE/OSA

TSG CN OSA ad-hoc thanks TSG SA WG3 for their response to the Liaison Statement on security issues in VHE/OSA and would like to respond on 2 issues:

- **Secure user identity:**
Within CN there has been work on a secure (encrypted) IMSI. This is used for communication between visited and home network. CN OSA ad-hoc does not see why this identity should be used for VHE, which as stated correctly is only used for the Home environment.

CN OSA ad-hoc would also like to point out that other identities (such as e-mail address) might be used within VHE.
- **Secure User Authorization to the Application via secure access to User Profile Data:**
It is true that the application can trust that the user can use it, however :
 1. An application can access user data (e.g. location information) and it has to be secured that the application only access data from subscribed users.
 2. Mechanisms are needed to either inform the application when a user subscribes to it or the application needs to get the possibility to check if the user has a subscription.

3GPP TSG_CN OSA#03
OSA Ad-hoc Meeting, Tilburg, Holland
6 - 7 January, 2000

Tdoc OSA-00024

Liaison Statement

From: TSG CN OSA ad-hoc
To: TSG SA WG2
Title: Annex A of VHE/OSA Stage 2 (3G TS 23.127 v 1.1.1)

In the review of version 1.1.1 of the VHE/OSA Stage 2 specification it was noted that Annex A includes a mapping between the OSA interface Class Methods and some MAP/CAP information flows. Analysis of this table identified a number of problems, namely:

- The MAP/CAP information provided is neither correct or complete;
- This type of information is not appropriate to a stage 2 description;
- There is no indication concerning precedence in case of inconsistency between the information flows as described in the Stage 2 and the same flows described in the protocol descriptions.

As we will be producing a more detailed mapping document for both CAP and MAP to the OSA and in view of the problems identified above, TSG CN OSA ad-hoc request that Annex A is removed from the Stage 2 specification.

3GPP TSG_CN OSA#04
OSA Ad-hoc Meeting, *Sophia Antipolis*, France
28-29 January, 2000

Tdoc NP-OSA-00054

From: TSG CN OSA ad-hoc
To: TSG SA2

Liaison Statement on VHE OSA time schedule

TSG OSA ad hoc has met for two days in Sophia Antipolis. We have experienced some procedural difficulties caused by the fact that the S2 material (input contributions and output documents) has not been available to members for their consideration.

It would be very beneficial for the progress in our meeting if the input and output of the S2 OSA meeting available well before the next CN OSA ad hoc meeting. We have noticed that for the next meeting there is again only one working day between the end of the S2 and S3 OSA meetings.

Given the tight schedule it is not appropriate for S3 to shift the its meeting forward. In order for CN OSA ad hoc to have sufficient time to take the S2 requirements into account, CN OSA ad-hoc requests S2 to have the output specification available at latest on the 18th of February, and therefore kindly asks S2 to reconsider the dates for their next OSA drafting meeting.

From: TSG CN OSA ad-hoc
To: TSG SA2

Liaison Statement on scope FW OSA

CN Ad-hoc would like to thank TSG SA2 for holding an OSA drafting session prior to our meeting, and making the result available. This has greatly facilitated our discussions. Following this line of collaboration, CN Ad-hoc would like to inform SA2 of some discussions that took place during our meeting, which are relevant for the SA2 work. It mainly concerns the Framework. Some of the issues were raised by your liaison towards our meeting, some were triggered by contributions to our meeting.

1. Event Notification

Concerning the Event Notification, the OSA Ad-hoc group would like to point out that the Framework Event Notification interface considers generic events only related to the Framework, i.e. no call related events. As there are currently no specific events defined for the Framework Event Notification interface the OSA Ad-hoc group recommends removal of the Event Notification interface.

2. Definition of OSA Access

The OSA Ad-hoc group would like to thank the TG SA2 group for pointing out the new term OSA_Access. The OSA Ad-hoc group has allocated the functionality of OSA_Access to the IpAccess interface.

3. Interfaces between Framework and Service Capabilities

The OSA Ad-hoc group would like to point out that it considers the scope of release 99 the standardization of the API between the gateway and the applications. The gateway in release 99 is all owned by a single party. The possibility of standardizing a second open interface, between the provider of the framework and the provider of the services (which is called "framework service view" in Parlay), is left open for release 2000.

This means that all interactions between the framework and the services are internal in release 99, and it is confusing to explicitly mention one possible method, as it is done in 23.127, v1.3.0, Annex A.

3GPP CN OSA/VHE and SPAN3 request that Annex A, together with all references to methods between the framework and the services, be deleted from release 99.

5. Relation between Framework and Service Capability Servers

The CN OSA Ad-hoc group would like to point out that the current architecture description in section 5.1 of the document 3G TS 23.127 1.3.0 can be improved by showing that the Framework can relate to multiple Service Capability Servers. This issue was raised by document OSA-00095, the suggestion after discussions is attached.

6. Call Control State Transition Diagram

The CN OSA Ad-hoc group identified mismatches with the "Call Interface" State Transition Diagrams. Also the CN OSA identified a number of errors within this state transition diagram. To avoid extensive discussions to correct the "Call Interface" State Transition Diagram and to ensure alignment between the stage 2 and stage 3 Call Interface State Transition Diagrams, CN OSA proposes to remove the state

transition diagram from the stage 2 document. This will also ease future evolvement of the “Call Interface” State Transition Diagram.

7. 3G TS 23.127 1.3.0

3GPP CN OSA Ad-hoc group would like to point the TSG SA2 group to document OSA-00104 (attached) that contains comments / improvements on document 3G TS 23.127 1.3.0 regarding missing parameters, method names and other editorial issues.

8. Charging

3GPP CN OSA Ad-hoc group has discussed the charging extensively, and proposes to include the charging operations in the Call class, rather than having a dedicated Call with Charging class as presently in the S2 documentation.

In future releases there will be more advanced call classes as well, like multimedia and multiparty additions. Having a separate Call with Charging would require similar specialisations for those more advanced calls.

Furthermore CN Ad-hoc proposes to change the method name ‘setCallDetailRecord’ to ‘setCallChargePlan’. Previously S2 documentation included both methods, the ‘setCallChargePlan’ has been removed by S2 after it had been identified that both have the same parameters. However, ‘setCallChargePlan’ is the more general method, which can be used to directly provide CDR information as one option.

So the CN OSA Ad-hoc proposal is to add the following methods to the “Call Interface” Class

- setAdviceOfCharge()
- setCallChargePlan()

. And to remove the “Charging” Interface Class

From: TSG CN OSA ad-hoc
To: TSG SA2

Liaison Statement on scope FW OSA

CN Ad-hoc would like to thank TSG SA2 for holding an OSA drafting session prior to our meeting, and making the result available. This has greatly facilitated our discussions. Following this line of collaboration, CN Ad-hoc would like to inform SA2 of some discussions that took place during our meeting, which are relevant for the SA2 work. It mainly concerns the Framework. Some of the issues were raised by your liaison towards our meeting, some were triggered by contributions to our meeting.

1. Event Notification

Concerning the Event Notification, the OSA Ad-hoc group would like to point out that the Framework Event Notification interface considers generic events only related to the Framework, i.e. no call related events. As there are currently no specific events defined for the Framework Event Notification interface the OSA Ad-hoc group recommends removal of the Event Notification interface.

2. Definition of OSA Access

The OSA Ad-hoc group would like to thank the TG SA2 group for pointing out the new term OSA_Access. The OSA Ad-hoc group has allocated the functionality of OSA_Access to the IpAccess interface.

3. Interfaces between Framework and Service Capabilities

The OSA Ad-hoc group would like to point out that it considers the scope of release 99 the standardization of the API between the gateway and the applications. The gateway in release 99 is all owned by a single party. The possibility of standardizing a second open interface, between the provider of the framework and the provider of the services (which is called "framework service view" in Parlay), is left open for release 2000.

This means that all interactions between the framework and the services are internal in release 99, and it is confusing to explicitly mention one possible method, as it is done in 23.127, v1.3.0, Annex A.

3GPP CN OSA/VHE and SPAN3 request that Annex A, together with all references to methods between the framework and the services, be deleted from release 99.

5. Relation between Framework and Service Capability Servers

The CN OSA Ad-hoc group would like to point out that the current architecture description in section 5.1 of the document 3G TS 23.127 1.3.0 can be improved by showing that the Framework can relate to multiple Service Capability Servers. This issue was raised by document OSA-00095, the suggestion after discussions is attached.

6. Call Control State Transition Diagram

The CN OSA Ad-hoc group identified mismatches with the "Call Interface" State Transition Diagrams. Also the CN OSA identified a number of errors within this state transition diagram. To avoid extensive discussions to correct the "Call Interface" State Transition Diagram and to ensure alignment between the stage 2 and stage 3 Call Interface State Transition Diagrams, CN OSA proposes to remove the state

transition diagram from the stage 2 document. This will also ease future evolvement of the “Call Interface” State Transition Diagram.

7. 3G TS 23.127 1.3.0

3GPP CN OSA Ad-hoc group would like to point the TSG SA2 group to document OSA-00104 (attached) that contains comments / improvements on document 3G TS 23.127 1.3.0 regarding missing parameters, method names and other editorial issues.

8. Charging

3GPP CN OSA Ad-hoc group has discussed the charging extensively, and proposes to include the charging operations in the Call class, rather than having a dedicated Call with Charging class as presently in the S2 documentation.

In future releases there will be more advanced call classes as well, like multimedia and multiparty additions. Having a separate Call with Charging would require similar specialisations for those more advanced calls.

Furthermore CN Ad-hoc proposes to change the method name ‘setCallDetailRecord’ to ‘setCallChargePlan’. Previously S2 documentation included both methods, the ‘setCallChargePlan’ has been removed by S2 after it had been identified that both have the same parameters. However, ‘setCallChargePlan’ is the more general method, which can be used to directly provide CDR information as one option.

So the CN OSA Ad-hoc proposal is to add the following methods to the “Call Interface” Class

- setAdviceOfCharge()
- setCallChargePlan()

. And to remove the “Charging” Interface Class

3G TS 23.127 1.3.0 (2000-02)

Technical Specification



3rd Generation Partnership Project; Technical Specification Group Services and System Aspects; Virtual Home Environment / Open Service Architecture (3G TS 23.127 version 1.3.0)

Reference

DTS/TSGS-0223xxxU

Keywords

VHE, OSA

3GPP

Postal address

3GPP support office address

650 Route des Lucioles - Sophia Antipolis
Valbonne - FRANCE
Tel.: +33 4 92 94 42 00 Fax: +33 4 93 65 47 16

Internet

<http://www.3gpp.org>

1 Scope

This document specifies the stage 2 of the Virtual Home Environment and Open Service Architecture.

Virtual Home Environment (VHE) is defined as a concept for personal service environment (PSE) portability across network boundaries and between terminals. The concept of the VHE is such that users are consistently presented with the same personalised features, User Interface customisation and services in whatever network and whatever terminal (within the capabilities of the terminal and the network), wherever the user may be located. For Release99, e.g. CAMEL, MExE and SAT are considered the mechanisms supporting the VHE concept.

The Open Service Architecture (OSA) defines an architecture that enables operator and third party applications to make use of network functionality through an open standardised interface (the OSA Interface). OSA provides the glue between applications and service capabilities provided by the network. In this way applications become independent from the underlying network technology. The applications constitute the top level of the Open Service Architecture (OSA). This level is connected to the Service Capability Servers (SCSs) via the OSA interface. The SCSs map the OSA interface onto the underlying telecom specific protocols (e.g. MAP, CAP etc.) and are therefore hiding the network complexity from the applications.

Applications can be network/server centric applications or terminal centric applications. Terminal centric applications reside in the Mobile Station (MS). Examples are MExE and SAT applications. Network/server centric applications are outside the core network and make use of service capability features offered through the OSA interface. (Note that applications may belong to the network operator domain although running outside the core network. Outside the core network means that the applications are executed in Application Servers that are physically separated from the core network entities).

2 References

References may be made to:

- a) Specific versions of publications (identified by date of publication, edition number, version number, etc.), in which case, subsequent revisions to the referenced document do not apply; or
- b) All versions up to and including the identified version (identified by "up to and including" before the version identity); or
- c) All versions subsequent to and including the identified version (identified by "onwards" following the version identity); or
- d) Publications without mention of a specific version, in which case the latest version applies.

A non-specific reference to an ETS shall also be taken to refer to later versions published as an EN with the same number.

2.1 Normative references

- [1] GSM 01.04 (ETR 350): "Digital cellular telecommunication system (Phase 2+); Abbreviations and acronyms"
- [2] GSM 02.57: "Digital cellular telecommunication system (Phase 2+); Mobile Station Application Execution Environment (MExE); Service description"
- [3] UMTS TS 23.057: "Mobile Station Application Execution Environment (MExE); Functional description - Stage2"
- [4] UMTS TS 22.078: "Customised Applications for Mobile network Enhanced Logic (CAMEL) (Phase3); Service description - Stage 1"
- [5] UMTS TS 23.078: "Customised Applications for Mobile network Enhanced Logic (CAMEL) (Phase3); Functional description - Stage 2"

- [6] GSM 11.14: "Digital cellular telecommunication system (Phase 2+); Specification of the SIM Application Toolkit for the Subscriber Identity Module - Mobile Equipment; (SIM - ME) interface" [7]UMTS TS 22.101: "Universal Mobile Telecommunications System (UMTS); Service Aspects; Service Principles"
- [8] UMTS TS 22.105: "Universal Mobile Telecommunications System (UMTS); Services and Service Capabilities"
- [9] UMTS TS 22.121: "Universal Mobile Telecommunications System (UMTS); Virtual Home Environment"
- [10] 3GPP TR 22.905: "3rd Generation Partnership Project; Technical Specification Group Services and System Aspects; Vocabulary for 3GPP Specifications"
- [11] IETF PPP Authentication Protocols - Challenge Handshake Authentication Protocol [RFC 1994, August 1996]
- [12] World Wide Web Consortium Composite Capability/Preference Profiles (CC/PP): A user side framework for content negotiation (www.w3.org)
- [13] Wireless Application Protocol, User Agent Profile Specification, (<http://www.wapforum.org>)

2.2 Informative references

- [1] UMTS TR 22.970: "Universal Mobile Telecommunications System (UMTS); Virtual Home Environment"

<Editor's note: some references may have to be aligned with their official title, e.g. 'UMTS' documents>

3 Definitions and abbreviations

3.1 Definitions

For the purposes of this TS, the following definitions apply:

Applications: software components providing services to end-users by utilising service capability features.

HE-VASP: see [9]

Home Environment: responsible for overall provision of services to users.

Local Service: see [9]

OSA Interface: Standardised Interface used by applications to access service capability features.

Personal Service Environment: contains personalised information defining how subscribed services are provided and presented towards the user. The Personal Service Environment is defined in terms of one or more User Profiles.

Service Capabilities: see [9]

Service Capability Feature: see [9]

Service Capability Server: Functional Entity providing OSA interfaces towards an application.

Services: see [9]

User Interface Profile: see [9]

User Profile: see [9]

User Services Profile: see [9].

Value Added Service Provider see [9]

Virtual Home Environment: see [9].

Further UMTS related definitions are given in 3G TS 22.101 and 3G TR 22.905.

3.2 Abbreviations

For the purposes of this TS the following abbreviations apply:

CAMEL	Customised Application For Mobile Network Enhanced Logic
CSE	Camel Service Environment
HE	Home Environment
HE-VASP	Home Environment Value Added Service Provider
HLR	Home Location Register
IDL	Interface Description Language
MAP	Mobile Application Part
ME	Mobile Equipment
MEExE	Mobile Station (Application) Execution Environment
MS	Mobile Station
MSC	Mobile Switching Centre
OSA	Open Service Architecture
PLMN	Public Land Mobile Network
PSE	Personal Service Environment
SAT	SIM Application Tool-Kit
SCP	Service Control Point
SIM	Subscriber Identity Module Short Message Service
USIM	User Service Identity Module
VASP	Value Added Service Provider
VHE	Virtual Home Environment
WGW:	WAP Gateway
WPP:	WAP Push Proxy

Further GSM related abbreviations are given in GSM 01.04. Further UMTS related abbreviations are given in 3G TR 22.905.

4 Virtual Home Environment

The Virtual Home Environment (VHE) is an important portability concept of the 3G mobile systems. It enables end users to bring with them their personal service environment whilst roaming between networks, and also being independent of terminal used.

The Personal Service Environment (PSE) describes how the user wishes to manage and interact with her communication services. It is a combination of a list of subscribed to services, service preferences and terminal interface preferences. PSE also encompasses the user management of multiple subscriptions, e.g. business and private, multiple terminal types and location preferences. The PSE is defined in terms of one or more User Profiles.

The user profiles consist of two kinds of information:

- Interface related information (User Interface Profile) and,
- Service related information (User Services profile).

Please see TS22.121 [9] for more details.

5 Open Service Architecture

In order to implement not known end user services/applications today, a highly flexible Open Service Architecture (OSA) is required. The Open Service Architecture (OSA) is the architecture enabling applications to make use of network capabilities. The applications will access the network through the OSA interface that is specified in this Technical Specification.

Network functionality offered to applications is defined as a set of Service Capability Features (SCFs) in the OSA interface, which are supported by different Service Capability Servers (SCS). These SCFs provide access to the network capabilities on which the application developers can rely when designing new applications (or enhancements/variants of already existing ones). The different features of the different SCSs can be combined as appropriate. The exact addressing (parameters, type and error values) of these features is described in stage 3 descriptions. These interface descriptions (defined using CORBA Interface Description Language) are open and accessible to application developers, who can design services in any programming language. The service logic is executed towards the OSA interfaces, while the underlying core network functions use their specific protocols.

The aim of OSA is to provide an extendible and scalable architecture that allows for inclusion of new service capability features and SCSs in future releases of UMTS with a minimum impact on the applications using the OSA interface.

The standardised OSA interface shall be secure, it is independent of vendor specific solutions and independent of programming languages, operating systems etc used in the service capabilities. Furthermore, the OSA interface is independent of the location within the home environment where service capabilities are implemented and independent of supported server capabilities in the network.

To make it possible for application developers to rapidly design new and innovative applications, an architecture with open interfaces is imperative. By using object-oriented techniques, like CORBA, it is possible to use different operating systems and programming languages in application servers and service capability servers. The different servers inter-work via the OSA interfaces. The service capability servers serve as gateways between the network entities and the applications.

The OSA API is an application layer interface, which is based on lower layers using main stream information technology and protocols. The middleware (e.g. CORBA) and lower layer protocols (e.g. IP) should provide security mechanisms to encrypt data (e.g. IP sec).

5.1 Overview of the Open Service Architecture

The Open Service Architecture consists of three parts:

- **Applications**, e.g. VPN, conferencing, location based applications. These applications are implemented in one or more Application Servers;
- **Framework**, providing applications with basic mechanisms that enable them to make use of the service capabilities in the network. Examples of framework services are Authentication, Registration and Discovery. Before an application can use the network functionality made available through the Service Capability Servers, authentication between the application and framework is needed. After authentication, the discovery service enables the application to find out which service capability features are provided by the Service Capability Servers. The service capability features are accessed by the methods defined in the OSA interface classes.
- **Service Capability Servers**, providing the applications with service capability features, which are abstractions from underlying network functionality. Examples of service capability features offered by the Service Capability Servers are Call Control and User Location. Similar service capability features may possibly be provided by more than one Service Capability Servers. For example, Call Control functionality might be provided by SCSs on top of CAMEL and MExE.

The OSA service capability features are specified in terms of a number of interface classes and their methods. The interface classes are divided into two groups:

- **framework interface classes**, describing the methods on the framework

The OSA interface classes are implemented in one or more physical entity, but separate from the physical network entities. [Figure 2](#) shows the case where the OSA interface classes are implemented in one physical entity, called “gateway” in the figure. [Figure 4](#) shows the case where the SCSs are distributed across several ‘gateways’.

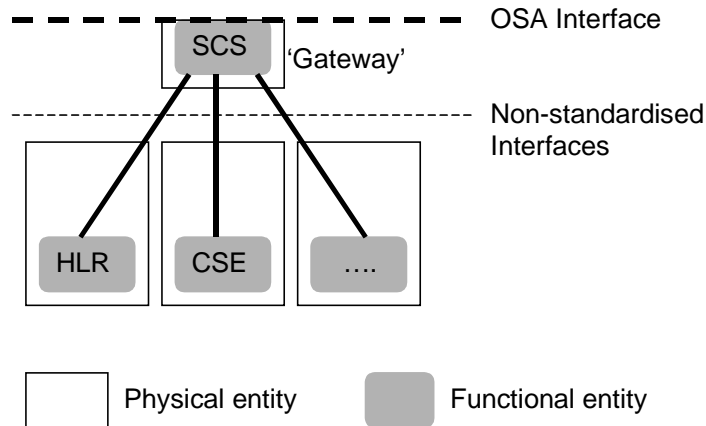


Figure 2 SCSs and network functional entities implemented in separate physical entities

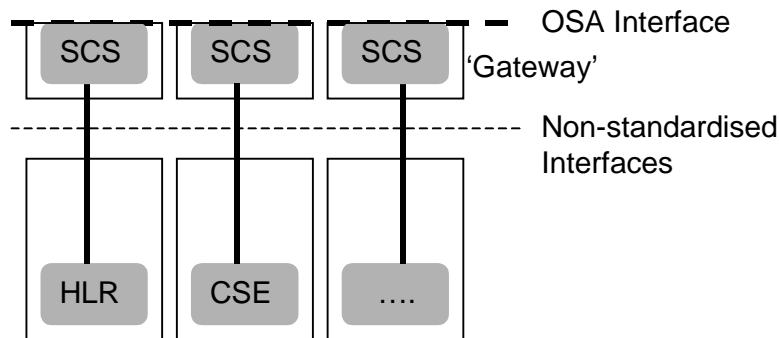


Figure 4 SCSs and network functional entities implemented in separate physical entities, SCSs distributed across several ‘gateways’.

Option 2

The OSA interface classes are implemented in the same physical entities as the traditional network entities (e.g. HLR, CSE), see [Figure 6](#).

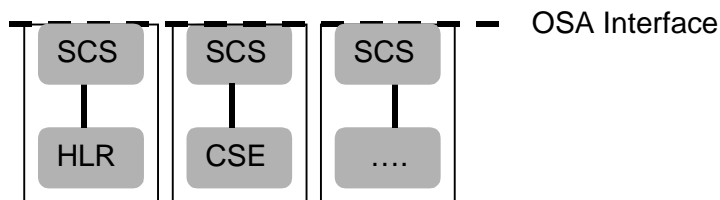


Figure 6 SCSs and network functional entities implemented in same physical entities

Option 3

Option 3 is the combination of option 1 and option 2, i.e. a hybrid solution.

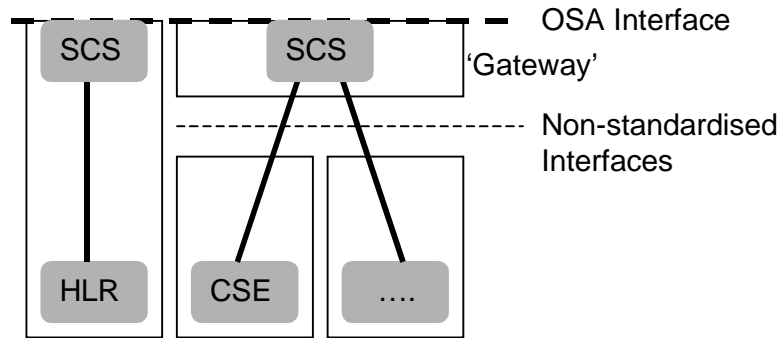


Figure 8 Hybrid implementation (combination of option 1 and 2)

It shall be noted that in all cases there is only one framework.

From the application point of view, it shall make no difference which implementation option is chosen, i.e. in all cases the same network functionality is perceived by the application. The applications shall always be provided with the same set of interface classes and a common access to framework and service capability feature interfaces. It is the framework that will provide the applications with an overview of available service capability features and how to make use of them.

5.2 Basic mechanisms in the Open Service Architecture

This section explains which basic mechanisms are executed in OSA prior to offering and activating applications.

Some of the mechanisms are applied only once (e.g. establishment of service agreement), others are applied each time a user subscription is made to an application (e.g. enabling the call attempt event for a new user).

Basic mechanisms between Application and Framework:

- **Authentication:** Once an off-line service agreement exists, the application can access the authentication interface. The authentication model of OSA is a peer-to-peer model. The application must authenticate the framework and vice versa. The application must be authenticated before it is allowed to use any other OSA interface.
- **Authorisation:** Authorisation is distinguished from authentication in that authorisation is the action of determining what a previously authenticated application is allowed to do. Authentication must precede authorisation. Once authenticated, an application is authorised to access certain service capability features.
- **Discovery of framework and service interfaces.** After successful authentication, applications can obtain available framework interface classes and use the discovery interface to obtain information on authorised service capability. The Discovery interface can be used at any time after successful authentication.
- **Establishment of service agreement.** Before any application can interact with a network service capability feature, a service agreement must be established. A service agreement may consist of an off-line (e.g. by physically passing messages) and an on-line part. The application has to sign the on-line part of the service agreement before it is allowed to access any network service capability feature.
- **Access to network service capability features:** The framework must provide access control functions to authorise the access to service capability features or service data for any API operation from a client, with the specified security level, context, domain, etc.

Basic mechanism between Framework and Service Capability Server:

- **Registering of service interfaces.** Interface classes offered by a Service Capability Server can be registered at the Framework. In this way the Framework can inform the Applications upon request about available service interface classes (Discovery). This mechanism is in general applied when installing or upgrading a Service Capability Server.

< editor's note: this mechanism is considered as of lower priority than other parts of OSA for R'99, and will be

standardised only if time permits>

Basic mechanisms between Application Server and Service Capability Server:

- **Request of event notifications.** This mechanism is applied when a user has subscribed to an application and that application needs to be invoked upon receipt of events from the network related to the user. For example, when a user subscribes to a screening application, the application needs to be invoked when the user makes a call. It will therefore request to be notified when a call setup is performed, with the user number as Called Party Number.

5.3 Handling of end-user related security

Once OSA basic mechanisms have ensured that an application has been authenticated and authorised to use network service capability features, it is important to also handle end-user related security aspects. These aspects consist of the following.

- **End-user authorisation to applications**, limiting the access of end-users to the applications they are subscribed to.
- **Application authorisation to end-users**, limiting the usage by applications of network capabilities to authorised (i.e. subscribed) end-users.
- **End-user's privacy**, allowing the user to set privacy options.

These aspects are addressed in the following subsections.

5.3.1 End-user authorisation to applications

An end-user is authorised to use an application only when he or she is subscribed to it.

In the case where the end-user has subscribed to the application before the application accesses the network SCFs, then the subscription is part of the Service Level Agreement signed between the HE and the HE-VASP.

After the application has been granted access to network SCFs, subscriptions are controlled by the Home Environment. Depending on the identity of an authenticated and authorised end-user, the Home Environment may use any relevant policy to define and possibly restrict the list of services to which a particular end-user can subscribe. At any time, the Home Environment may decide, unilaterally or after agreement with the HE-VASP, to cancel a particular subscription.

Service subscription and activation information need to be shared between the Home Environment and the HE-VASP, so that the HE-VASP knows which end-users are entitled to use its services. Appropriate online and/or offline synchronisation mechanisms (e.g. SLA re-negotiation) can be used between the HE and the HE-VASP, which are not specified in OSA release 99.

End-to-end interaction between a subscribed end-user and an application may require the usage of appropriate authentication and authorisation mechanisms between the two, which are independent from the OSA API, and therefore not in the scope of OSA standardisation.

5.3.2 Application authorisation to end-users

The Home Environment is entitled to provide service capabilities to an application with regard to a specific end-user if the following conditions are met:

- 1) The end-user is subscribed to the application
- 2) The end-user has activated the application
- 3) The usage of this network service capability does not violate the end-users privacy settings (see next section).

The service capability server ensures that the above conditions are met whenever an application attempts to use a service capability feature for a given end-user, and to respond to the application accordingly, possibly using relevant error parameters (USER_NOT_SUBSCRIBED, APPLICATION_NOT_ACTIVATED, USER_PRIVACY_VIOLATION). The mechanism used by the SCS to ensure this is internal to the HE (e.g. access to user profile) and is not standardised in

Errors

6 Framework service capability features

6.1 Trust and Security Management SCFs

The Trust and Security Management service capability features provide:

- The first point of contact for a client application to access a Framework provider;
- The authentication operations for the client application and Framework provider to perform an authentication protocol;
- The client application with the ability to select a network service capability feature to make use of;
- The client application with a portal to access other framework service capability features.

The process by which the client application accesses the Framework provider has been separated into 3 stages, each supported by a different framework service capability feature:

1. Initial Contact with the framework
2. Authentication to the framework
3. Access to framework and network service capability features

6.1.1 Initial Contact

The client application gains a reference to the Initial SCF for the Framework provider that they wish to access. This may be gained through a URL, an Application Support Broker, a *stringified* object reference, etc. At this stage, the client has no guarantee that this is a reference to the Framework provider.

The client application uses this SCF to initiate the authentication process with the Framework provider.

The Initial SCF supports the `initiateAuthentication` operation to allow the authentication process to take place (using the Authentication SCF defined in 0). This operation must be the first invoked by the client application. Invocations of other operations will fail until authentication has been successfully completed.

Once the client has authenticated with the provider, the client can gain access to other framework and network service capability features. This is done by invoking the `requestAccess` method, by which the client application requests a certain type of access service capability feature. The OSA Access service capability feature is defined in section 6.1.3.

The Initial Contact framework SCF is defined by a unique interface class, consisting of the following methods.

Method **`initiateAuthentication()`**

The client application uses this method to initiate the authentication process.

Direction Application to Framework

Parameters **clientAppID**
 This is an identifier for the client application. It is used to identify the client to the framework, (see authenticate() on Authentication). If the clientAppID cannot be found by the framework, an error code is returned by the framework. The value of the parameter fwAuthInterface is NULL in this case.

authType
 This identifies the type of authentication mechanism requested by the client. It provides operators and clients with the opportunity to use an alternative to the OSA Authentication interface, e.g. CORBA Security.

appAuthInterface
 This provides the reference for the framework to call the authentication interface of the client application.

Returns **fwAuthInterface**
 This provides the reference for the client application to call the authentication SCF of the framework.

Errors

Method **requestAccess ()**
 Once client and framework are authenticated, the client application invokes the requestAccess operation on the Initial SCF. This allows the client application to request the type of access they require. If they request OSA_ACCESS, then a reference to the OSA Access interface is returned. (Home Environments can define their own access interfaces to satisfy client requirements for different types of access.)

Direction Application to network

Parameters **accessType**
 This identifies the type of access SCF requested by the client application.

appAccessInterface
 This provides the reference for the framework to call the access interface of the client application.

Returns **fwAccessInterface**
 This provides the reference for the client to call the access SCF of the framework.

Errors

6.1.2 Authentication

Once the client application has made initial contact with the framework provider, authentication of the client application and Framework provider may be required.

The API supports multiple authentication techniques. The procedure used to select an appropriate technique for a given situation is described below. The authentication mechanisms may be supported by cryptographic processes to provide confidentiality, and by digital signatures to ensure integrity. The inclusion of cryptographic processes and digital signatures in the authentication procedure depends on the type of authentication technique selected. In some cases strong

authentication may need to be enforced by the framework provider to prevent misuse of resources. In addition it may be necessary to define the minimum encryption key length that can be used to ensure a high degree of confidentiality.

The client application must authenticate with the framework before it is able to use any of the other interfaces supported by the framework. Invocations on other interfaces will fail until authentication has been successfully completed.

1. The client application calls `initiateAuthentication` on the framework provider's Initial interface. This allows the client to specify the type of authentication process. This authentication process may be specific to the Framework provider, or the implementation technology used. The `initiateAuthentication` operation can be used to specify the specific process, (e.g. CORBA security). OSA defines a generic authentication service capability feature (Authentication), which can be used to perform the authentication process. The `initiateAuthentication` operation allows the client application to pass a reference to its `AppAuthentication` to the Framework, and receive a reference to the Authentication interface supported by the framework, in return.
2. The client application invokes the `selectAuthMethod` on the framework's Authentication SCF. This includes the authentication capabilities of the client application. The framework then chooses an authentication method based on the authentication capabilities of the client application and the framework. If the client is capable of handling more than one authentication method, then the framework chooses one option, the `prescribedMethod`. In some instances, the authentication capability of the client application may not fulfil the demands of the framework, in which case, the authentication will fail.
3. The client application and framework interact to authenticate each other. Depending on the method prescribed, this procedure may consist of a number of messages e.g. a challenge/ response protocol. This authentication protocol is performed using the `authenticate` operation on the Authentication interface. Depending on the authentication method selected, the protocol may require invocations on the Authentication SCF supported by the framework; or on the client application counterpart; or on both.

The Authentication framework SCF is defined by a single interface class, consisting of the following methods.

Method **`selectAuthMethod ()`**

The client application uses this method to initiate the authentication process. The mechanism returned by the framework is the mechanism it prefers. This should be within capability of the client application. If a mechanism that is acceptable to the framework within the capability of the client application cannot be found, the framework returns an error code (INVALID_AUTH_CAPABILITY).

Direction Application to network

Parameters **`authCapability`**

This is the means by which the authentication mechanisms supported by the client are conveyed to the framework.

Returns **`prescribedMethod`**

This is returned by the framework to indicate the mechanism it prefers for the authentication process. If the value of the `prescribedMethod` returned by the framework is not understood by the client application, it is considered a catastrophic error and the client application must abort.

Errors

Method **`authenticate ()`** (*application to network*)

This method is used by the client to authenticate the framework using the mechanism indicated in `prescribedMethod`. The framework must respond with the correct responses to the challenges presented by the client. The `clientAppID` received in the `initiateAuthentication ()` can be used by the framework to reference the correct public key for the client application (the key

management system is currently outside of the scope of the OSA specification). The number of exchanges and the order of the exchanges is dependent on the prescribedMethod.

Direction Application to network

Parameters **prescribedMethod**

This parameter contains the method that the framework has specified as acceptable for authentication (see selectAuthMethod).

challenge

The challenge presented by the client application to be responded to by the framework. The challenge mechanism used will be in accordance with the IETF *PPP Authentication Protocols - Challenge Handshake Authentication Protocol* [RFC 1994, August1996]. The challenge will be encrypted with the mechanism prescribed by selectAuthMethod().

Returns **response**

This is the response of the framework to the challenge of the client application in the current sequence. The response will be based on the challenge data, decrypted with the mechanism prescribed by selectAuthMethod().

Errors

Method **authenticate()** (*network to application*)

This method is used by the framework to authenticate the client application using the mechanism indicated in prescribedMechanism. The client application must respond with the correct responses to the challenges presented by the framework. The number of exchanges and the order of the exchanges is dependant on the prescribedMethod. (These may be interleaved with authenticate() calls by the client application on the Authentication interface. This is defined by the prescribedMethod.)

Direction Network to application

Parameters **prescribedMethod**

This parameter contains the agreed method for authentication (see selectAuthMethod on the Authentication interface.)

challenge

The challenge presented by the framework to be responded to by the client application. The challenge mechanism used will be in accordance with the IETF *PPP Authentication Protocols - Challenge Handshake Authentication Protocol* [RFC 1994, August1996]. The challenge will be encrypted with the mechanism prescribed by selectAuthMethod().

Returns **response**

This is the response of the client application to the challenge of the framework in the current sequence. The response will be based on the challenge data, decrypted with the mechanism prescribed by selectAuthMethod().

Errors

Method **abortAuthentication()** (*application to network*)

The client application uses this method to abort the authentication process. This method is invoked if the client no longer wishes to continue the authentication process, (e.g. if the framework responds incorrectly to a challenge.) If this method has been invoked, calls to the requestAccess operation on

Initial will return an error code (INVALID_AUTHENTICATION) until the client has been properly authenticated.

Direction Application to network

Parameters

Returns

Errors

Method **abortAuthentication()** (*network to application*)

The framework uses this method to abort the authentication process. This method is invoked if the framework wishes to abort the authentication process, (e.g. if the client application responds incorrectly to a challenge.) If this method has been invoked, calls to the requestAccess operation on Initial will return an error code (INVALID_AUTHENTICATION), until the client has been properly authenticated.

Direction Network to application

Parameters

Returns

Errors

6.1.3 OSA Access

During an authenticated session accessing the Framework, the client application will be able to select and access an instance of a framework or network service capability feature.

Access to framework SCFs is gained by invoking the obtainInterface, or obtainInterfaceWithCallback operations. The latter is used when a callback reference is supplied to the framework. For example, a network SCF discovery interface class reference is returned when invoking obtainInterface with “discovery” as the SCF name.

In order to use network SCFs, the client must first be authorised to do so by establishing a service agreement with the Home Environment. The client application uses the discovery SCF to retrieve the ID of the network SCF they wish to use. They may then use the accessCheck operation to check that they are authorised to use the network SCF. The selectService operation is used to tell the framework provider that the client application wishes to use the network SCF. The signServiceAgreement operation is used to digitally sign the agreement, and provide non-repudiation for both parties in agreeing that the SCF would be available for use.

Establishing a service agreement is a business level transaction, which requires the enterprise domain that owns the client application to agree terms for the use of an SCF with the framework provider (i.e. the Home Environment). Service agreements can be reached using either off-line or on-line mechanisms. Off-line agreements will be reached outside of the scope of OSA interactions, and so are not described here. However, client applications can make use of service agreements that are made off-line. Some Framework providers may only offer off-line mechanisms to reach service agreements.

After a service agreement has been established between the client and the framework provider domains, the client application will be able to make use of this agreement to access the network SCF.

The accessCheck operation allows the client application to check whether it has permission to access (read, write, etc) to a specified SCF, and specific SCF features. The client application defines the security domain and context of access to the SCF. The access control policy is based on a number of conditions, events and permissions that determine whether the client application is authorised to access the SCF/feature.

The accessCheck operation is optional, in that can be called by the client application to check that it has permission to use specific SCF features, before starting an SCF instance. It is not compulsory for the client application to make this check before selecting a network SCF and signing a service agreement to use an instance of the SCF. If the accessCheck operation confirms that the client application has permission to use a specific SCF feature, then this feature should be available to the client application when using the SCF instance. The framework provider may include the results of the accessCheck as part of the service agreement, that is signed before using an SCF instance, thereby assuring the client application that the SCF features will be available.

The selectService operation is used to identify the SCF that the client application wishes to use. A list of service properties initialises the SCF, and an SCF token is returned. The client application and framework provider must sign a copy of the service agreement to confirm the use of the SCF. The framework invokes signServiceAgreement operation on the client applications's AppAccess interface with the service agreement text to be signed. The client application uses its digital signature key to sign the agreement text, and return the signed text to the framework. The client application then calls the signServiceAgreement operation on the OSA Access SCF. The framework signs the agreement text, retrieves a reference to a network manager interface for the selected SCF (using a mechanism similar to the one described in informative annex A), and returns this reference to the client application. The OSA Access framework SCF is defined by a single interface class, consisting of the following methods.

Method **obtainInterface ()**

This method is used to obtain other frameworkSCFs. The client application uses this method to obtain interface references to other frameworkSCFs. (The obtainInterfacesWithCallback method should be used if the client application is required to supply a callback interface to the framework.)

Direction Application to network

Parameters **interfaceName**

The name of the framework SCF to which a reference to the interface is requested.

Returns **fwInterface**

This is the reference to the SCF interface requested.

Errors INVALID_INTERFACE_NAME

Method **obtainInterfaceWithCallback ()**

This method is used to obtain other frameworkSCFs. The client application uses this method to obtain interface references to other frameworkSCFs, when they are required to supply a callback interface to the framework. (The obtainInterface method should be used when no callback interface needs to be supplied.)

Direction Application to network

Parameters **interfaceName**

The name of the framework SCF to which a reference to the interface is requested.

appInterface

This is the reference to the client application interface, which is used for callbacks. If an application interface is not needed, then this method should not be used. (The obtainInterface method should be used when no callback interface needs to be supplied.)

Returns **fwInterface**

This is the reference to the SCF requested.

Errors INVALID_INTERFACE_NAME

Method **accessCheck ()**

This method may be used by the client application to check whether it has been granted permission to access the specified SCF. The response is used to indicate whether the request for access has been granted or denied and if granted the level of trust that will be applied. The securityModelID and the relevant securityLevel are available as part of the registration data for the SCF.

securityModelID:

The identity of the specific Security Model that is to be used to define a set of appropriate policies for the SCF that can be used by the framework to determine access rights. The model may include: blanket permission; session permission or one shot permission. A number of security models will be stored by the framework, and referenced by the access control module, according to the security model identifier of the SCF.

securityLevel:

The trust level required by the service for granting access. The Security Level is used by the framework's access control module when it checks for access rights.

Direction Application to network

Parameters **securityContext**

A context is a group of security relevant attributes that may have an influence on the result of the accessCheck request.

securityDomain

The security domain in which the client application is operating may influence the access control decisions and the specific set of features that the requestor is entitled to use.

group

A group can be used to define the access rights associated with all clients that belong to that group. This simplifies the administration of access rights.

serviceAccessTypes

These are defined by the specific Security Model in use but are expected to include: Create, Read, Update, Delete as well as those specific to SCFs.

Returns **serviceAccessControl**

This is a structure containing the access control policy information controlling access to the SCF, and the trustLevel that the service provider has assigned to the client application. It consists of

- policy: indicates whether access has been granted or denied. If granted then the parameter trustLevel must also have a value.
- trustLevel: The trustLevel parameter indicates the trust level that the service provider has assigned to the client application.

Errors

Method `terminateServiceAgreement()` (network to application)

This method is used by the framework to terminate a service agreement for the SCF.

Direction Network to application

Parameters `serviceToken`

This is the token passed back from the framework in a previous `selectService()` method call. This token is used to identify the service agreement to be terminated.

`terminationText`

This is the termination text describes the reason for the termination of the service agreement.

`digitalSignature`

This is a signed version of a hash of the service token and the termination text. The signing algorithm used is the same as the signing algorithm given when the service agreement was signed using `signServiceAgreement()`. The framework uses this to confirm its identity to the client. The client can check that the `terminationText` has been signed by the framework.

Returns

Errors

Method `endAccess()`

The endAccess operation is used to end the client application's access session with the framework. The client requests that its access session be ended. After it is invoked, the client application will not longer be authenticated with the framework. The client application will not be able to use the references to any of the framework SCFs gained during the access session. Any calls to these SCF interfaces will fail.

Direction Application To Network

Parameters

Returns

Errors

Method `terminateAccess()`

The terminateAccess operation is used to end the client application's access session with the framework (e.g. this may be done if the framework believes the client application is masquerading as someone else. Using this operation will force the client application to re-authenticate if it wishes to continue using the framework SCFs.)

After terminateAccess() is invoked, the client application will not longer be authenticated with the framework. The client application will not be able to use the references to any of the framework SCFs gained during the access session. Any calls to these interfaces will fail.

Direction Network to application

Parameter `terminationText`

S

This is the termination text describes the reason for the termination of the access session.

signingAlgorithm

This is the algorithm used to compute the digital signature.

digitalSignature

This is a signed version of a hash of the termination text. The framework uses this to confirm its identity to the client. The client can check that the `terminationText` has been signed by the framework.

Returns**Errors**

6.2 Discovery

The discovery SCF consists of a single interface class. Before a network SCF can be discovered, the client application must know what “types” of SCFs are supported by the Framework and what service “properties” are applicable to each SCF type. The `listServiceType()` method returns a list of all “SCF types” that are currently supported by the framework and the “describeServiceType()” returns a description of each SCF type. The description of SCF type includes the “SCF-specific properties” that are applicable to each SCF type. Then the client application can discover a specific set of registered SCFs that belong to a given type and possess the desired “property values”, using the “discoverService()” method.

Once the enterprise operator finds out the desired set of SCFs supported by the network, it subscribes (a sub-set of) these SCFs using the Subscription framework SCF. The enterprise operator (or the client applications in its domain) can find out the set of services available to it (i.e., the service that it can use) by invoking “listSubscriberServices()”.

The service discovery APIs are invoked by the enterprise operators or client applications. They are described below.

Method **discoverService ()**

The `discoverService` operation is the means by which a client application is able to obtain the IDs of the SCFs that meet its requirements. The client application passes in a list of desired service properties to describe the SCF it is looking for, in the form attribute/value pairs for the service properties. The client application also specifies the maximum number of matched responses it is willing to accept. The framework must not return more matches than the specified maximum, but it is up to the discretion of the Framework implementation to choose to return less than the specified maximum. The `discoverService()` operation returns a serviceID/Property pair list for those SCFs that match the desired service property list that the client application provided.

Direction Application to network

Parameters **serviceName**

The “ServiceTypeName” parameter conveys the required SCF type. It is key to the central purpose of “SCF trading”. By stating an SCF type, the importer implies the SCF type and a domain of discourse for talking about properties of SCF.

The framework may return an SCF of a subtype of the “type” requested. An SCF sub-type can be described by the properties of its supertypes.

desiredPropertyList

The “desiredPropertyList” parameter is a list of property name and property value pairs of service properties that the discovered set of SCFs should satisfy. These properties deal with the non-functional and non-computational aspects of the desired SCF. The property values in the desired property list must be logically interpreted as “minimum”, “maximum”, etc. by the framework.

max

The “max” parameter states the maximum number of SCFs that are to be returned in the “ServiceList” result.

Returns**serviceList :**

This parameter gives a list of matching SCFs. Each SCF is characterised by an SCF ID and a list of property name and property value pairs associated with the SCF.

Errors

ILLEGAL_Service_TYPE

UNKNOWN_Service_TYPE

Method**listServiceTypes ()**

This operation returns the names of all SCF types which are in the repository. The details of the SCF types can then be obtained using the describeServiceType() method.

Direction

Application to network

Parameters**Returns****listTypes**

The names of the requested SCF types.

Errors**Method****describeServiceType ()**

This operation lets the caller to obtain the details for a particular SCF type.

Direction

Application to network

Parameters**name**

The name of the SCF type to be described

Returns**serviceTypeDescription**

The description of the specified SCF type. The description provides information about:

- the property names associated with the SCF,
- the corresponding property value types,
- the corresponding property mode (mandatory or read only) associated with each SCF property,
- the names of the super types of this type, and
- whether the type is currently enabled or disabled.

Errors

ILLEGAL_SERVICE_TYPE

UNKNOWN_SERVICE_TYPE

Method	listSubscribedServices ()
	Returns a list of SCFs so far subscribed by the enterprise operator. The enterprise operator (or the client applications in the enterprise domain) can obtain a list of subscribed SCFs that they are allowed to access.
Direction	Application to network
Parameters	
Returns	serviceIDList The “ServiceIDList” parameter returns a list of IDs of the SCFs subscribed by the subscriber.
Errors	

6.3 Event Notification

The Event Notification framework SCF consists of a single interface class.

The event notification mechanism is used to notify the application of generic SCF related events that have occurred. Note that applications may also register to network SCF –specific notifications, using methods defined in associated interface classes.

Method	enableNotification ()
	This method is used to enable generic notifications, so that events can be sent to the application.
Direction	
Parameters	eventCriteria Specifies the event specific criteria used by the application to define the event required.
Returns	assignmentID Specifies the ID assigned by the framework for this newly enabled event notification.
Errors	

Method	disableNotification ()
	This method is used by the application to disable generic notifications from the framework.
Direction	
Parameters	assignmentID Specifies the assignment ID given by the framework when the previous <code>enableNotification()</code> was called.
Returns	
Errors	INVALID_ASSIGNMENTID

Method eventNotify()

This method notifies the application of the arrival of a generic event.

Direction Network to application.

Parameters eventInfo

Specifies specific data associated with this event.

assignmentID

Specifies the assignment id which was returned by the framework during the `enableNotification()` method. The application can use assignment id to associate events with event specific criteria and to act accordingly.

Returns**Errors****Method notificationTerminated()**

This method indicates to the application that all generic event notifications have been terminated (for example, due to faults detected).

Direction Network to application.

Parameters**Returns****Errors**

6.4 Integrity Management SCFs

6.4.1 Load Manager

The framework API should allow the load to be distributed across multiple machines and across multiple component processes, according to a load balancing policy. The separation of the load balancing mechanism and load balancing policy ensures the flexibility of the load balancing services. The load balancing policy identifies what load balancing rules the framework should follow for the specific client application. It might specify what action the framework should take as the congestion level changes. For example, some real-time critical applications will want to make sure continuous service is maintained, below a given congestion level, at all costs, whereas other applications will be satisfied with disconnecting and trying again later if the congestion level rises. Clearly, the load balancing policy is related to the QoS level to which the application is subscribed.

The Load Manager SCF consists of a single interface class. Most methods are asynchronous, in that they do not lock a thread into waiting whilst a transaction performs. In this way, the client machine can handle many more calls, than one that uses synchronous message calls.

The load management operations do not exchange callback interfaces as it is assumed that the client application has supplied its Load Management callback interface at the time it obtains the Framework's Load Manager SCF, by use of the `obtainInterfaceWithCallback` operation on the OSA Access SCF.

Method reportLoad()

The client application notifies the framework about its current load level (0,1, or 2) when the load level on the application has changed.

Errors

Method **suspendNotification()**

Suspend the notification from an application for its load status after the detection of load level change at the framework and the evaluation of the load balancing policy.

Direction Network to application

Parameters

Returns

Errors

Method **queryLoadReq ()**

The client application requests load statistic records for the framework and specified SCFs.

Direction Application to Network

Parameters **requester**

Specifies the application interface for callbacks.

serviceIds

Specifies the framework, SCFs or applications for which the load statistics shall be reported. The serviceIds is null for framework load statistics only.

timeInterval

Specifies the time interval within which the load statistics are generated.

Returns

Errors

Method **queryLoadRes ()**

Returns load statistics to the application which requested the information.

Direction Network to application

Parameters **loadStatistics**

Specifies the framework-supplied load statistics.

Returns

Errors

Method **queryLoadErr ()**

Returns an error code to the application that requested load statistics.

Direction Network to application

Parameters loadStatisticsError

Specifies the framework-supplied error code.

Returns

Errors

Method **queryAppLoadReq ()**

The framework requests for load statistic records produced by a specified application.

Direction Network to application

Parameters **serviceIds**
Specifies the SCFs or applications for which the load statistics shall be reported.

timeInterval
Specifies the time interval within which the load statistics are generated.

Returns

Errors

Method **queryAppLoadRes ()**

Report load statistics back to the framework that requested the information.

Direction Application to network

Parameters **loadStatistics**
Specifies the load statistics in the application.

Returns

Errors

Method **queryAppLoadErr ()**

Return an error response to the framework that requested the application's load statistics information.

Direction Application to network

Parameters **loadStatisticsError**
Specifies the error code associated with the failed attempt to retrieve the application's load statistics.

Returns

Errors

Method **registerLoadController ()**

Register the client application for load management under various load conditions.

Direction Application to network

- Parameters**
- requester**
Specifies the application interface for callbacks.
 - serviceIds**
Specifies the framework and SCFs to be registered for load control. To register for framework load control only, the `serviceIds` is null.
- Returns**
- Errors**
- Method** **`unregisterLoadController ()`**
- Unregister the client application for load management.
- Direction** Application to network
- Parameters**
- requester**
Specifies the application interface for callbacks.
 - serviceIds**
Specifies the framework or SCFs to be unregistered for load control.
- Returns**
- Errors**
- Method** **`resumeNotification ()`**
- Resume load management notifications to the application for the framework and specified SCFs after their load condition changes.
- Direction** Application to network
- Parameters**
- serviceId**
Specifies the framework and SCFs for which notifications are to be resumed. The `serviceIds` is null to resume notifications for the framework only.
- Returns**
- Errors**
- Method** **`suspendNotification()`**
- Suspend load management notifications to the application for the framework and specified SCFs, while the application handles a temporary load condition.
- Direction** Application to network
- Parameters**
- serviceId**
Specifies the framework and SCFs for which notifications are to be suspended. The `serviceIds` is null to suspend notifications for the framework only.
- Returns**

Errors

6.4.2 Fault Manager

This SCF is used by the application to inform the framework of events which affect the integrity of the framework and SCFs, and to request information about the integrity of the system.

It consists of a single interface class, with the following methods.

Method **activityTestReq()**

This method may be used by the application to test that the framework or an SCF is operational. On receipt of this request, the framework must carry out a test on the specified SCF or the framework itself to check that it is operating correctly and report the test result.

Direction Application to network

Parameters **activityTestID**

The identifier provided by the client application to correlate the response (when it arrives) with this request.

svcID

This parameter identifies which SCF the client application is requesting the activity test to be done for. A null value denotes that the activity test is being requested for the framework.

appID

This parameter identifies which client application is requesting the activity test, and therefore which application to send the result to.

Returns

Errors

Method **activityTestRes()**

The framework returns the result of the activity test in this method, along with a test identifier to allow correlation of result to request within the client application.

Direction Network to application

Parameters **activityTestID**

The identifier provided by the client (in the request), to correlate this response with the original request.

activityTestResult

The result of the activity test.

Returns

Errors

Method **appActivityTestReq ()**

This method is invoked by the framework to request that the client application carries out an activity test to check that is it operating correctly.

Direction Network to application

Parameters **activityTestID**

The identifier provided by the client (in the request), to correlate this response with the original request.

Returns

Errors

Method **appActivityTestRes ()**

This method is used by the client application to return the result of a previously requested activity test.

Direction Application to network

Parameters **activityTestID**

The identifier is used by the framework to correlate this response (when it arrives) with the original request.

activityTestResult

The result of the activity test.

Returns

Errors

Method **fwFaultReportInd ()**

This method is invoked by the framework to notify the client application of a failure within the framework. The client application must not continue to use the framework until it has recovered (as indicated by a fwFaultRecoveryInd).

Direction Network to application

Parameters **fault**

Specifies the fault that has been detected.

Returns

Errors

Method **fwFaultRecoveryInd ()**

This method is invoked by the framework to notify the client application that a previously reported fault has been rectified.

Direction Network to application

Parameters **fault**

Specifies the fault from which the framework has recovered.

Returns

Errors

Method **svcUnavailableInd ()**

This method is used by the client application to inform the framework that it can no longer use the indicated SCF (either due to a failure in the client application or in the SCF). On receipt of this request, the framework should take the appropriate corrective action. The framework assumes that the session between this client application and instance SCF is to be closed and updates its own records appropriately as well as attempting to inform the SCF instance and/or its administrator. If the client application then tries to continue the use of this session it should be returned an error.

Direction Application to network

Parameters **serviceID**

The identity of the SCF which can no longer be used.

appID

The identity of the application sending the indication.

Returns

Errors

Method **svcUnavailableInd ()**

This method is used by the framework to inform the client application that it can no longer use the indicated SCF due to a failure in the SCF. On receipt of this request, the client application must act to reset its use of the specified SCF (using the normal mechanisms such as the discovery and authentication interfaces to stop use of this SCF instance and begin use of a different SCF instance).

Direction Network to application

Parameters **serviceID**

The identity of the SCF which can no longer be used.

reason

The reason why the SCF is no longer available.

Returns

Errors

Method **genFaultStatsRecordReq ()**

This method is used by the application to solicit fault statistics from the framework. On receipt of this request, the framework must produce a fault statistics record, which is returned to the client application. The fault statistics record must contain information about faults relating to the SCFs specified in the *serviceIDList* parameter, during the specified period.

Direction Application to Network

Parameters **timePeriod**

The period over which the fault statistics are to be generated. A null value leaves this to the discretion of the framework.

serviceIDList

This parameter lists the SCFs that the application would like to have included in the general fault statistics record. If the application would like the framework fault statistics to be included it should include the NULL serviceID.

appID

This parameter identifies which client application is requesting the statistics record, and therefore which application to send the record to.

Returns

Errors

Method `genFaultStatsRecordRes ()`

This method is used by the framework to provide fault statistics to a client application in response to a `genFaultStatsRecordReq`.

Direction Network to application

Parameters `faultStatistics`
The fault statistics record.

serviceIDList

This parameter lists the services that have been included in the general fault statistics record. The framework is denoted by the NULL serviceID.

Returns

Errors

6.4.3 Heartbeat Management

This SCF allows the initialisation of a heartbeat supervision of the client application. In case of SCF supervision, it is the framework's responsibility to check the health status of the respective SCF.

Since the OSA API is inherently synchronous, the heartbeats themselves are synchronous for efficiency reasons.

The Heartbeat Management SCF consists of a two interface classes: Heartbeat Management and Heartbeat.

Heartbeat Management

Method `enableHeartBeat ()`

With this method, the client application registers at the framework for heartbeat supervision of itself.

Direction Application to network

Parameters **duration**

The duration in milliseconds between the heartbeats.

appInterface

This parameter refers to the callback interface.

Returns **session**

Identifies the heartbeat session. In general, the application has only one session. In case of SCF and framework supervision by the client application, the application may maintain more than one session.

Errors**Method** **disableHeartBeat()**

Allows the stop of the heartbeat supervision of the application.

Direction Application to network

Parameters **session**

Identifies the heartbeat session.

Returns**Errors****Method** **changeTimeperiod()**

Allows the administrative change of the heartbeat period.

Direction Application to network

Parameters **session**

Identifies the heartbeat session. In general, the application has only one session.

duration

The time interval in milliseconds between the heartbeats.

Returns**Errors****Method** **enableAppHeartBeat()**

With this method, the framework registers at the client application for heartbeat supervision of itself.

Direction Network to application

Parameters **duration**

The time interval in milliseconds between the heartbeats.

appInterface

This parameter refers to the callback interface.

session

Identifies the heartbeat session..

Returns

Errors

Method **disableAppHeartBeat ()**

Allows the stop of the heartbeat supervision of the application.

Direction Network to application

Parameters **session**
Identifies the heartbeat session.

Returns

Errors

Method **changeTimeperiod ()**

Allows the administrative change of the heartbeat period.

Direction Network to application

Parameters **session**
Identifies the heartbeat session.

duration
The time interval in milliseconds between the heartbeats.

Returns

Errors

Heartbeat

Method **send ()**

This is the method the client application uses in case it supervises the framework or an SCF. The sender must raise an exception if no result comes back after a certain, user-defined time.

Direction

Parameters **session**
Identifies the heartbeat session. In general, the application has only one session.

Returns

Errors

Method **send()**

This is the method the framework uses in case it supervises a client application. The sender must raise an exception if no result comes back after a certain, user-defined time.

Direction

Parameters **session**

Identifies the heartbeat session.

Returns

Errors

6.4.4 OAM

The OAM SCF is used to query the system date and time. The application and the framework can synchronise the date and time to a certain extent. Accurate time synchronisation is outside the scope of the OSA API.

The OAM SCF consists of a unique interface class.

Method **systemDateTimeQuery()**

This method is used to query the system date and time. The client application passes in its own date and time to the framework. The framework responds with the system date and time.

Direction Application to network

Parameters **clientDateAndTime**

This is the date and time of the client application.

Returns **systemDateAndTime**

This is the system date and time returned by the framework.

Errors INVALID_DATE_TIME_FORMAT

Method **systemDateTimeQuery()**

This method is used to query the system date and time. The framework passes in the system date and time to the client. The client responds with its own date and time.

Direction Network to application

Parameters **systemDateAndTime**

This is the system date and time of the framework.

Returns **clientDateAndTime**

This is the date and time returned by the client.

Errors OSA_INVALID_DATE_TIME_FORMAT

7 Network service capability features

The service capability features provided to the application by service capabilities servers to enable access to network resources.

Note: when the direction of a method in an interface class is “application to network”, this means that the method is invoked from the application to an SCS residing on the network side of the OSA interface.

7.1 Call Control

The Call control network service capability feature consists of two interface classes:

1. Call manager, containing management function for call related issues
2. Call, containing methods to control a call

A call can be controlled by one Call Manager only. A Call Manager can control several calls..

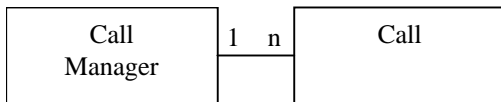


Figure 910 Call control classes usage relationship

The Call Control service capability features are described in terms of the methods in the Call Control interface classes. [Table 1](#) gives an overview of the Call Control methods and to which interface classes these methods belong.

CallManager	Call	
	routeCallToDestination_Req	
enableCallNotification	routeCallToDestination_Res	
disableCallNotification	routeCallToDestination_Err	
callNotificationTerminated	release	
callEventNotify	deassignCall	
callAborted	getCallInfo_Req	
	getCallInfo_Res	
callNotificationTerminated	getCallInfo_Err	
	superviseCall_Req	
	superviseCall_Res	
	superviseCall_Err	
	callFaultDetected	

CallManager	Call	

Table 1 Overview of Call Control interface classes and their methods

7.1.1 Call Manager

The generic call manager interface class provides the management functions to the generic call Service Capability Features. The application programmer can use this interface class to create call objects and to enable or disable call-related event notifications.

Method **enableCallNotification()**

This method is used to enable call notifications to be sent to the application.

Direction Application to network

Parameters **appInterface**
 If this parameter is set (i.e. not NULL) it specifies a reference to the application interface, which is used for callbacks. If set to NULL, the application interface defaults to the interface specified via the `setCallback()` method.

eventCriteria
 Specifies the event specific criteria used by the application to define the event required. Examples of events are “incoming call attempt reported by network”, “answer”, “no answer”, “busy”.

Returns **assignmentID**
 Specifies the ID assigned by the generic call control manager object for this newly-enabled event notification.

Errors **USER_NOT_SUBSCRIBED**
APPLICATION_NOT_ACTIVATED
USER_PRIVACY_VIOLATION

Method **disableCallNotification()**

This method is used by the application to disable call notifications.

Direction Application to network

Parameters **eventCriteria**
 Specifies the event specific criteria used by the application to define the event to be disabled. Examples of events are “incoming call attempt reported by network”, “answer”, “no answer”, “busy”.

assignmentID
 Specifies the assignment ID given by the generic call control manager object when the previous

enableNotification() was called.

Returns -

Errors INVALID_ASSIGNMENTID

Returned if the assignment ID does not correspond to one of the valid assignment Ids.

Method **callEventNotify()**

This method notifies the application of the arrival of a call-related event.

Direction Network to application

Parameters **Call Reference**

Specifies the reference to the call object to which the notification relates.

eventInfo

Specifies data associated with this event. These data include originatingAddress, originalDestinationAddress, redirectingAddress and AppInfo (see for more explanation on these data the routeCallToDestination() method).

assignmentID

Specifies the assignment id which was returned by the enableNotification() method. The application can use assignment ID to associate events with event-specific criteria and to act accordingly.

appInterface

Specifies a reference to the application object which implements the callback interface for the new call.

Returns -

Errors -

Method **callAborted()**

This method indicates to the application that the call object has aborted or terminated abnormally. No further communication will be possible between the call object and the application.

Direction Network to application

Parameters **call**

Specifies the call object that has aborted or terminated abnormally.

callSessionID

Specifies the call session ID of the call that has aborted or terminated abnormally.

Returns -

Errors -

Method **callNotificationTerminated()**

This method indicates to the application that all event notifications have been terminated (for example, due to faults detected).

Direction Network to application

Parameters -

Returns -

Errors -

7.1.2 Call

The generic call interface class provides a structure to allow simple and complex call behaviour to be used.

Method `routeCallToDestination_Req()`

This asynchronous method requests routing of the call (and inherently attached parties) to the destination party (specified in the parameter `TargetAddress`). The destination party is attached to the call via a passive leg. This means that the call is not automatically released if the destination party disconnects from the call; only the leg with which the destination party was attached to the call is released in that case. .

Direction Application to network

Parameters `callSessionID`

Specifies the call session ID of the call.

`responseRequested`

Specifies the set of observed events that will result in a `routeCallToDestination_Res()` being generated.

`targetAddress`

Specifies the destination party to which the call should be routed.

`originatingAddress`

Specifies the address of the originating (calling) party.

`originalDestinationAddress`

Specifies the original destination address of the call, i.e. the address as specified by the originating party. This parameter should be equal to the `originalDestinationAddress` as received by the application in the `eventInfo` parameter of the `callEventNotify` method.

`redirectingAddress`

Specifies the last address from which the call was redirected.

`appInfo`

Specifies application-related information pertinent to the call (such as alerting method, tele-service type, service identities and interaction indicators).

[Assignment ID](#)

[Specifies the ID assigned by the network SCS. The same ID will be returned in the `routeCallToDestinationRes` or `Err`. This allows the application to correlate the request and the result.](#)

Returns -

Errors `USER_NOT_SUBSCRIBED`

APPLICATION_NOT_ACTIVATED

USER_PRIVACY_VIOLATION

Method `routeCallToDestination_Res()`

This asynchronous method indicates that the request to route the call to the destination was successful, and indicates the response of the destination party (for example, the call was answered, not answered, refused due to busy, etc.).

Direction Network to application

Parameters `callSessionID`

Specifies the call session ID of the call.

`eventReport`

Specifies the result of the request to route the call to the destination party. It also includes the mode that the call object is in and other related information.

Returns -

Errors -

Method `routeCallToDestination_Err()`

This asynchronous method indicates that the request to route the call to the destination party was unsuccessful - the call could not be routed to the destination party (for example, the network was unable to route the call, the parameters were incorrect, the request was refused, etc.).

Direction Network to application

Parameters `callSessionID`

Specifies the call session ID of the call.

`error Indication`

Specifies the error which led to the original request failing.

Returns -

Errors -

Method `release()`

This method requests the release of the call and associated objects.

Direction Application to network

Parameters `callSessionID`

Specifies the call session ID of the call.

`cause`

Specifies the cause of the release.

Returns -

Errors -

Method **deassignCall()**

This method requests that the relationship between the application and the call and associated object be de-assigned. It leaves the call in progress, however, it purges the specified call object so that the application has no further control of call processing. If a call is de-assigned that has event reports or call information reports requested, then these reports will be disabled and any related information discarded.

Direction Application to network

Parameters **callSessionID**
Specifies the call session ID of the call.

Returns -

Errors -

Method **getCallInfo_Req()**

This asynchronous method requests information associated with the call to be provided at the appropriate time (for example, to calculate charging). This method must be invoked before the call is routed to a target address. The call object will exist after the call is ended if information is required to be sent to the application at the end of the call. The call information will be sent after any call event reports.

Note: At the end of the call, the call information must be sent before the call is deleted.

Direction Application to network

Parameters **callSessionID**
Specifies the call session ID of the call.

callInfoRequested
Specifies the call information that is requested.

Returns -

Errors -

Method **getCallInfo_Res()**

This asynchronous method reports all the necessary information requested by the application, for example to calculate charging.

Direction Network to application

Parameters **callSessionID**
Specifies the call session ID of the call.

callInfoReport
Specifies the call information requested.

Returns -

Errors -

Method **getCallInfo_Err ()**

This asynchronous method reports that the original request was erroneous, or resulted in an error condition.

Direction Network to application

Parameters **callSessionID**
 Specifies the call session ID of the call.

eError Indication
 Specifies the error which led to the original request failing.

Returns -

Errors -

Method **superviseCall_Req ()**

The application calls this method to supervise a call. The application can set a granted connection time for this call. If an application calls this function before it calls a routeCallToDestination_Req () or a user interaction function the time measurement will start as soon as the call is answered by the B-party or the user interaction system.

Direction Application to network

Parameters **CallSessionID**
 Specifies the call session ID of the call.

duration
 Specifies the granted duration of the call/session in:
 time in milliseconds for the connection, or;
 Total data transferred in ...

time
 Specifies the granted time in milliseconds for the connection. When specified as 0, volume based supervision is applied. Either bytes (volume) or time should be specified.

tariffSwitch
 Specifies an optional tariff switch indicating a change in tariff.

treatment
 Specifies how the network should react after the granted connection time expired.

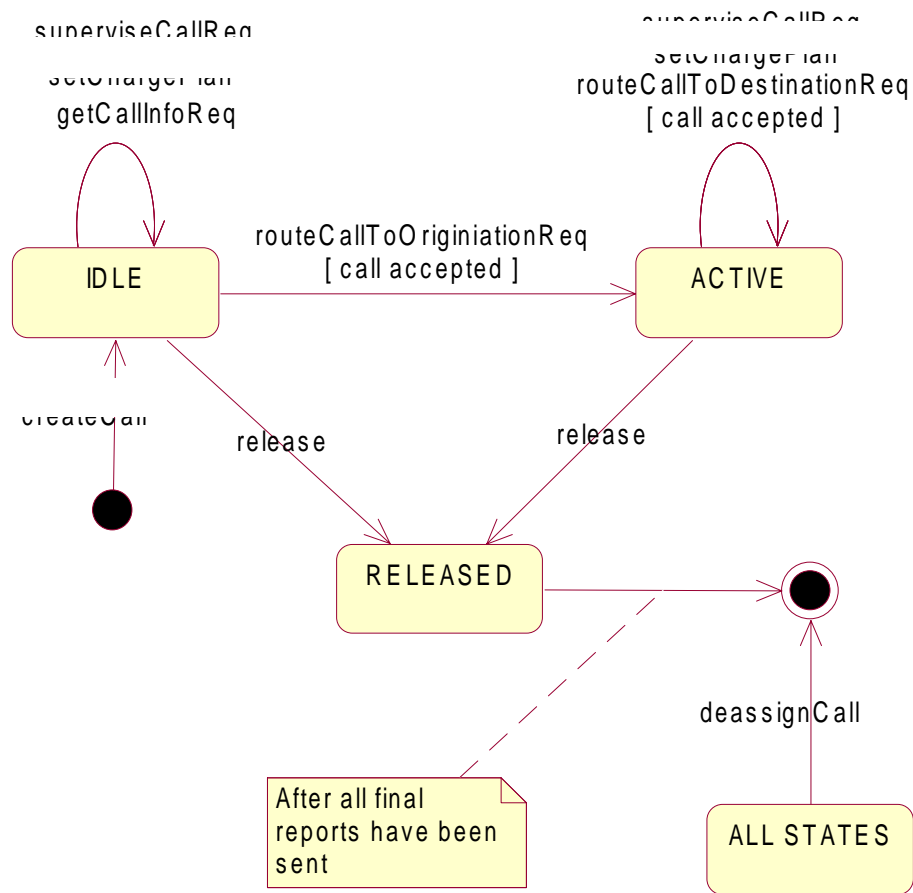
bytes
 Specifies the granted number of bytes that can be transmitted for the connection. When the quantity is specified as 0, time based supervision is applied. Either bytes (volume) or time should be specified.

Returns -

Errors -

Figure 10-11 shows the state model for the generic call interface from applications point of view. The call is created implicitly by the Generic Call Control Service when a new call event notification arrived.

It shall be noted that this state diagrams relates to the OSA interface and not to the underlying mechanism used to perform the call control.



Figure

10-11 - State diagram for the Call interface from an application point of view

Sequence Diagrams

The following section will describe some scenarios to illustrate the use of the methods described above.

Enable Call notification

The first task to perform in order to allow applications to provide call control related services to certain users is to enable call-related events for these users to trigger the application. This is done with the method `enableCallNotification()`.

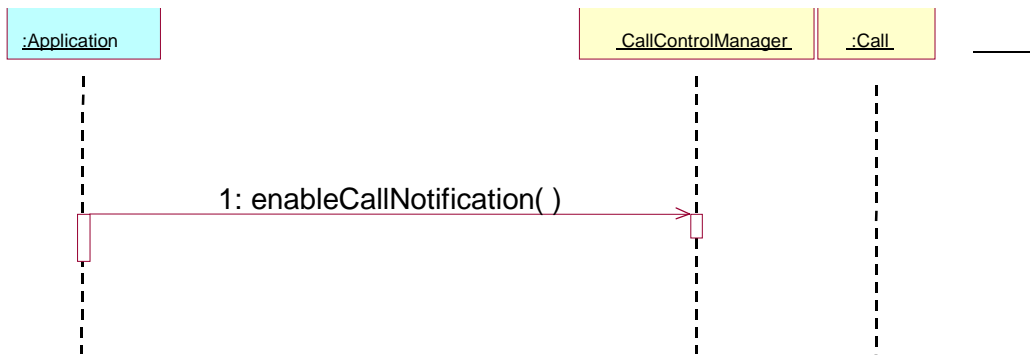


Figure 1213 Enable call notification

Number translation

The example in Figure 1314 shows a simple number translation application.

After the call is triggered (according to the criteria in a previous enableCallNotification()), the SCS notifies the application with an eventCallNotify() message. This allows the application to perform the needed actions and continue the call set-up via a routeCallToDestination_Req() message. The SCS relays the result of the call set-up (both positive and negative) to the application, which ends after that.

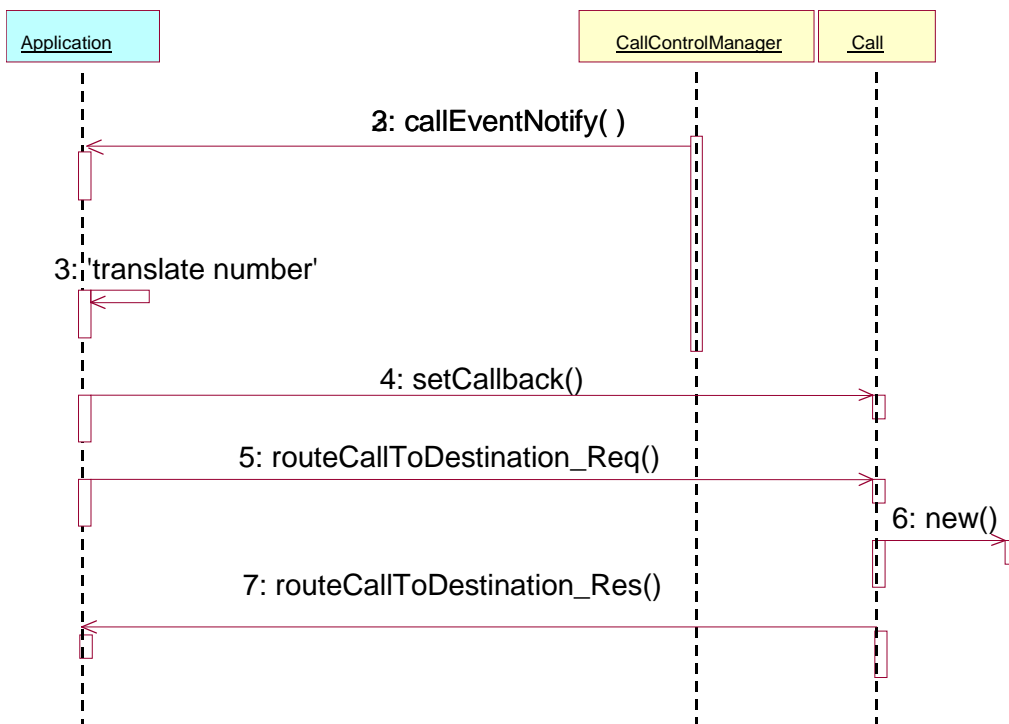


Figure 1314 Simple number translation

Call barring

The next example (Figure 1516) shows how a call barring application can be implemented:

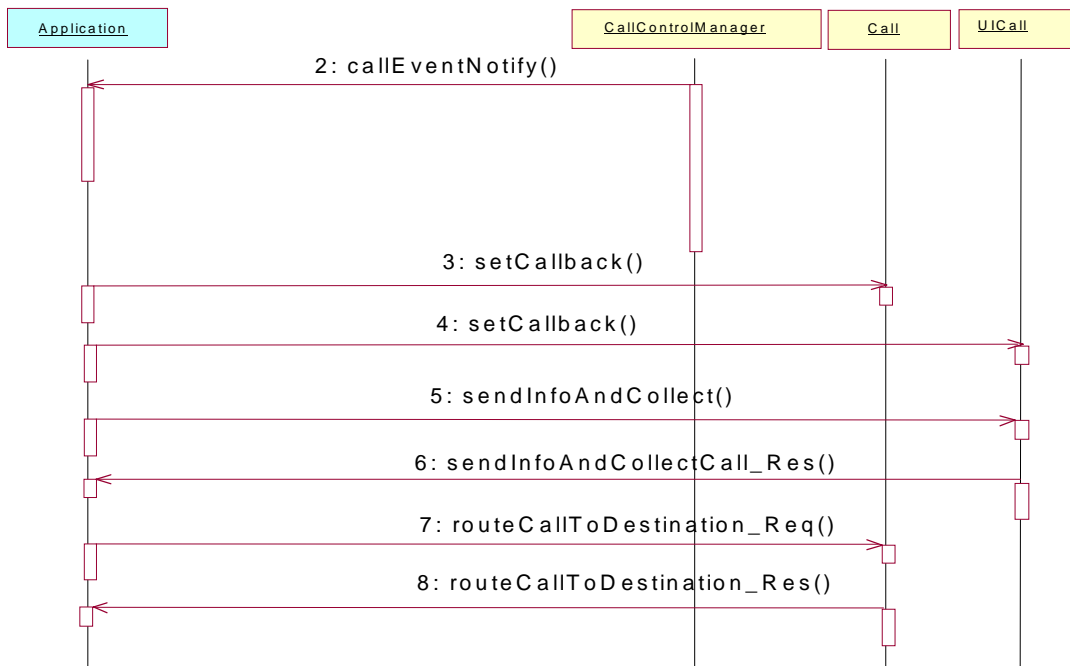


Figure 1546 Call barring application

7.2 Network User Location

The Network User Location service capability feature provides terminal location information, based on network-related information, such as a VLR Number, Location Area Identification, or Cell Global Identification. It may also provide geographical location information, if the network is able to support the corresponding capability.

It consists of a single interface class, permitting an application to perform the following:

- User location requests.
- Requests for starting (or stopping) the generation by the network of periodic user location reports.
- Requests for starting (or stopping) the generation by the network of user location reports based on location changes.

Method **locationReportReq()**

Request for mobile-related location information on one or several users.

Direction Application to network

Parameters **appLocationCamel**

If this parameter is set (i.e. not NULL) it specifies a reference to the application interface, which is used for callbacks. If set to NULL, the application interface defaults to the interface specified via the obtainInterface() method (refer to Authentication interface).

users

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

Returns

assignmentId

Specifies the assignment ID of the location-report request.

Errors

INVALID_PARAMETER_VALUE

A method parameter has an invalid value.

NO_CALLBACK_ADDRESS_SET

The requested method has been refused, because no callback address is set.

RESOURCES_UNAVAILABLE

The required resources in the network are not available.
The application may try to invoke the method at a later time.

USER_NOT_SUBSCRIBED

APPLICATION_NOT_ACTIVATED

USER_PRIVACY_VIOLATION

Method

locationReportRes ()

Delivery of a mobile location report. The report is containing mobile-related location information for one or several users.

Direction

Network to application

Parameters

assignmentId

Specifies the assignment ID of the location-report request.

locations

Specifies the location(s) of one or several users.

Returns

-

Errors

INVALID_PARAMETER_VALUE

A method parameter has an invalid value.

INVALID_ASSIGNMENT_ID

The assignment ID does not correspond to one of a valid assignment.

Method

locationReportErr ()

This method indicates that the location report request has failed.

Direction

Network to application

Parameters

assignmentId

Specifies the assignment ID of the failed location report request.

cause

Direction Application to network

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

Returns -

Errors **INVALID_ASSIGNMENT_ID**
 The assignment ID does not correspond to one of a valid assignment.

Method **periodicLocationReport ()**
 Periodic delivery of mobile location reports. The reports are containing mobile-related location information for one or several users.

Direction Network to application

Parameters **assignmentId**
 Specifies the assignment ID of the periodic location-reporting request.

locations
 Specifies the location(s) of one or several users.

Returns -

Errors **INVALID_PARAMETER_VALUE**
 A method parameter has an invalid value.

INVALID_ASSIGNMENT_ID
 The assignment ID does not correspond to one of a valid assignment.

Method **periodicLocationReportErr ()**
 This method indicates that a requested periodic location report has failed. Note that errors only concerning individual users are reported in the ordinary periodicLocationReport() message.

Direction Network to application

Parameters **assignmentId**
 Specifies the assignment ID of the failed periodic location reporting start request.

cause
 Specifies the error that led to the failure.

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

Returns -

Errors -

Errors **INVALID_ASSIGNMENT_ID**
 The assignment ID does not correspond to one of a valid assignment

Method **triggeredLocationReport ()**
 Delivery of a report that is indicating that one or several user's mobile location has changed.

Direction Network to application

Parameters **assignmentId**
 Specifies the assignment ID of the triggered location-reporting request.

location
 Specifies the location of the user.

criterion
 Specifies the criterion that triggered the report.

Returns -

Errors **INVALID_PARAMETER_VALUE**
 A method parameter has an invalid value.

INVALID_ASSIGNMENT_ID
 The assignment ID does not correspond to one of a valid assignment.

Method **triggeredLocationReportErr ()**
 This method indicates that a requested triggered location report has failed. Note that errors only concerning individual users are reported in the ordinary triggeredLocationReport() message.

Direction Network to application

Parameters **assignmentId**
 Specifies the assignment ID of the failed triggered location reporting start request.

cause
 Specifies the error that led to the failure.

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

Returns -

Errors -

7.3 User Status

The User Status service capability feature provides general user status monitoring. It allows applications to obtain the status of the user's terminal. It consists of a single interface class.

Method **statusReportReq ()**

Request for a report on the status of one or several users.

Direction Application to network

Parameters **appStatus**

If this parameter is set (i.e. not NULL) it specifies a reference to the application interface, which is used for callbacks. If set to NULL, the application interface defaults to the interface specified via the `obtainInterface ()` method (refer to Authentication interface).

users

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

Returns **assignmentId**

Specifies the assignment ID of the status-report request.

Errors **INVALID_PARAMETER_VALUE**

A method parameter has an invalid value.

NO_CALLBACK_ADDRESS_SET

The requested method has been refused, because no callback address is set.

RESOURCES_UNAVAILABLE

The required resources in the network are not available.
The application may try to invoke the method at a later time.

USER_NOT_SUBSCRIBED

APPLICATION_NOT_ACTIVATED

USER_PRIVACY_VIOLATION

Method **statusReportRes ()**

Delivery of a report, that is containing one or several user's status.

Direction Network to application

Parameters **assignmentId**

Specifies the assignment ID of the status-report request.

status

Specifies the status of one or several users.

Returns -

Errors **INVALID_PARAMETER_VALUE**

A method parameter has an invalid value.

INVALID_ASSIGNMENT_ID

The assignment ID does not correspond to one of a valid assignment.

Method **statusReportErr ()**

This method indicates that the status report request has failed.

Direction Network to application

Parameters **assignmentId**
 Specifies the assignment ID of the failed status report request.

cause
 Specifies the error that led to the failure.

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

Returns -

Errors -

Method **triggeredStatusReportingStartReq ()**

Request for triggered status reports when one or several user's status is changed. The user status service will send a report when the status changes.

Direction Application to network

Parameters **appStatus**
 If this parameter is set (i.e. not NULL) it specifies a reference to the application interface, which is used for callbacks. If set to NULL, the application interface defaults to the interface specified via the obtainInterface () method (refer to Authentication interface).

users
 Specifies the user(s) for which the status changes shall be reported.

Returns **assignmentId**
 Specifies the assignment ID of the triggered status-reporting request.

Errors **INVALID_PARAMETER_VALUE**
 A method parameter has an invalid value.

NO_CALLBACK_ADDRESS_SET
 The requested method has been refused, because no callback address is set.

RESOURCES_UNAVAILABLE
 The required resources in the network are not available.

The application may try to invoke the method at a later time.

USER_NOT_SUBSCRIBED

APPLICATION_NOT_ACTIVATED

USER_PRIVACY_VIOLATION

Method **triggeredStatusReportingStop()**

This method stops the sending of status reports for one or several users.

Direction Application to network

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

Returns -

Errors **INVALID_ASSIGNMENT_ID**
The assignment ID does not correspond to one of a valid assignment.

Method **triggeredStatusReport()**

Delivery of a report that is indicating that a user's status has changed.

Direction Network to application

Parameters **assignmentId**
Specifies the assignment ID of the triggered status-reporting request.

status
Specifies the status of the user.

Returns -

Errors **INVALID_PARAMETER_VALUE**
A method parameter has an invalid value.

INVALID_ASSIGNMENT_ID
The assignment ID does not correspond to one of a valid assignment.

Method **triggeredStatusReportErr()**

This method indicates that a requested triggered status reporting has failed. Note that errors only concerning individual users are reported in the ordinary triggeredStatusReport() message.

Direction Network to application

Parameters **assignmentId**
Specifies the assignment ID of the failed triggered status reporting start request.

cause

Specifies the error that led to the failure.

diagnostic

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

Returns -

Errors -

7.4 Terminal Capabilities

It shall be possible for an application to request Terminal Capabilities as defined by MExE [3]. The terminal capabilities are provided by a MExE compliant terminal to the MExE Service Environment either on request or by the terminal itself.

Terminal Capabilities are available only after a Capability negotiation has previously taken place between the user's MExE terminal and the MExE Service environment as specified in [3].

Note: for Release 99 only WAP MExE devices can supply terminal capabilities.

The Terminal Capabilities service capability feature is supported by a unique interface class, which consists of the following method.

The Terminal Capabilities service capability feature is supported by a unique interface class, which consists of the following method.

Method **GetTerminalCapabilities()**

This method is used by an application to get the capabilities of a user's terminal.

Direction Application to Network

Parameters **TerminalIdentity**
Identifies the terminal. It may be a logical address known by the WAP Gateway/PushProxy.

Returns **StatusCode**
Indicates whether or not the terminalCapabilities are available.

TerminalCapabilities

Specifies the latest available capabilities of the user's terminal. This information, if available, is returned as CC/PP headers as specified in W3C [12] and adopted in the WAP UAProf specification [13]. It contains URLs; terminal attributes and values, in RDF format; or a combination of both.

Errors -

7.5 Message Transfer

7.5.1 Generic User Interaction

The Generic User Interaction service capability feature is used by applications to interact with end users. It consists of two interface classes:

1. User Interaction Manager, containing management functions for User Interaction related issues
2. Generic User Interaction, containing methods to interact with an end-user

The Generic User Interaction service capability feature is described in terms of the methods in the Generic User Interaction interface classes.

The following table gives an overview of the Generic User Interaction methods and to which interface classes these methods belong.

User Interaction Manager	Generic User Interaction
createUI	sendInfoReq
createUICall	sendInfoRes
enableUINotification	sendInfoErr
disableUINotification	sendInfoAndCollectReq
userInteractionEventNotify	sendInfoAndCollectRes
userInteractionAborted	sendInfoAndCollectErr
	release
	userInteractionFaultDetected

Table 3 Overview of Generic User Interaction interface classes and their methods

User Interaction Manager

Inherits from the generic service interface.

The User Interaction Manager interface class provides the management functions to the User Interaction class interface.

Method **createUI ()**

This method is used to create a new (non call related) user interaction object.

Direction Application to network

Parameters **appUI**

Specifies the application interface for callbacks from the user interaction created.

userAddress

Indicates the end-user whom to interact with

Returns **userInteraction**
 Specifies the interface and sessionID of the user interaction created.

Errors **USER_NOT_SUBSCRIBED**
APPLICATION_NOT_ACTIVATED
USER_PRIVACY_VIOLATION

Method **createUICall()**

This method is used to create a new call related user interaction object.

The user interaction can take place to the specified party (userAddress) or to all parties in a call (callIdentifier). Only one of callIdentifier or userAddress may be defined (the other should be set to NULL).

Note that for certain implementations user interaction can only be performed towards the controlling call party, which shall be the only party in the call.

Direction Application to network

Parameters **appUI**
 Specifies the application interface for callbacks from the user interaction created.

callIdentifier
 Specifies the call interface and session ID of the call associated with the send info operation.

callLegIdentifier
 Indicates the end-user whom to interact with

Returns **userInteraction**
 Specifies the interface and sessionID of the user interaction created.

Errors

Method **enableUINotification()**

This method is used to enable the reception of user initiated user interaction.

Direction Application to network

Parameters **appInterface**
 If this parameter is set (i.e. not NULL) it specifies a reference to the application interface, which is used for callbacks. If set to NULL, the application interface defaults to the interface specified via the setCallback() method.

eventCriteria
 Specifies the event specific criteria used by the application to define the event required, like user address and service code.

Parameters

userInteractionSessionID
 Specifies the user interaction session ID of the user interaction.

assignmentID
 Specifies the ID assigned by the generic user interaction interface for a user interaction request.

response
 Specifies the type of response received from the user.

Returns

Errors

Method **sendInfoErr ()**
 This asynchronous method indicates that the request to send information was unsuccessful.

Direction Network to Application

Parameters

userInteractionSessionID
 Specifies the user interaction session ID of the user interaction.

assignmentID
 Specifies the ID assigned by the generic user interaction interface for a user interaction request.

error
 Specifies the error which led to the original request failing.

Returns

Errors

Method **sendInfoAndCollectReq ()**
 This asynchronous method plays an announcement or sends other information to the user and collects some information from the user. The announcement usually prompts for a number of characters (for example, these are digits or text strings such as "YES" if the user's terminal device is a phone).

Direction Application to Network

Parameters

userInteractionSessionID
 Specifies the user interaction session ID of the user interaction.

infoID
 Specifies the ID of the information to send to the user.

variableInfo
 Defines the variable part of the information to send to the user.

criteria
 Specifies additional properties for the collection of information, such as the maximum and minimum

number of characters, end character, first character timeout and inter-character timeout.

Returns **assignmentID**
Specifies the ID assigned by the generic user interface

Errors

Method **sendInfoAndCollectRes ()**
This asynchronous method returns the information collected to the application.

Direction Network to Application

Parameters **userInteractionSessionID**
Specifies the session ID of the user interaction.

assignmentID
Specifies the ID assigned by the generic user interaction interface for a user interaction request.

response
Specifies the type of response received from the user.

info
Specifies the information collected from the user.

Returns

Errors

Method **sendInfoAndCollectErr ()**
This asynchronous method indicates that the request to send information and collect a response was unsuccessful.

Direction Network to Application

Parameters **userInteractionSessionID**
Specifies the user interaction session ID of the user interaction.

assignmentID
Specifies the ID assigned by the generic user interaction interface for a user interaction request.

error
Specifies the error which led to the original request failing.

Returns

Errors

Method **release ()**
This method requests that the relationship between the application and the user interaction object be released. It causes the release of the used user interaction resources and interrupts any ongoing user interaction.

Direction Application to Network

Parameters **userInteractionSessionID**
Specifies the user interaction session ID of the user interaction.

Returns

Errors

Method **userInteractionFaultDetected()**
This method indicates to the application that a fault has been detected in the user interaction.

Direction Network to Application

Parameters **userInteractionIdentifier** **userInteractionSessionID**
Specifies the interface and sessionID of the user interaction service in which the fault has been detected.

fault
Specifies the fault that has been detected.

Returns

Errors

7.5.2 Call User Interaction

The Call User Interaction service capability feature is used by applications to interact with end users participating to a call. It consists of two interface classes:

1. User Interaction Manager, containing management functions for User Interaction related issues. This class is the same as the one defined in section 7.5.1.
2. Call User Interaction, extending Generic User Interaction for call-specific user interaction. It provides functions to send information to, or gather information from, a user (or call party) in a call.

The Call User Interaction service capability feature is described in terms of the methods in the Call User Interaction interface classes.

The following table gives an overview of the Call User Interaction methods and to which interface classes these methods belong.

User Interaction Manager	Call User Interaction
<i>As defined for the Generic User Interaction SCF</i>	<i>Inherits from Generic User Interaction and adds:</i>
	abortActionReq
	abortActionRes
	abortActionErr

Table 4 Overview of Call User Interaction interface classes and their methods

7.6 User Profile Management

User Profile information may be distributed between the Home Environment and the Home Environment Value-Added Services Providers. The HE-VASP may manage information specific to the services supported by their OSA applications. For this, they may use models and mechanisms, which are out of the scope of OSA release 99.

Home Environment User Profile information consists of various interface and service related information. Of particular interest in the context of release 99 is the following information:

- list of services to which the end-user is subscribed
- service status (active/inactive)
- privacy status with regards to network service capabilities (e.g. user location, user interaction)
- terminal capabilities

Home Environment user profile information may be stored centrally, or the information may be distributed over relevant physical entities.

Terminal capabilities may be accessed by OSA applications through the network Terminal Capabilities SCF.

7.8 Charging

7.8.1 Call With Charging

This class inherits from the Call interface class and adds charging specific methods.

Method **setAdviceOfCharge ()**

This method allows the application to the charging information that will be send to the end-users handset.

Direction Application to network

Parameters **callSessionID**
 Specifies the call session ID of the call.

aOCInfo
 Specifies two sets of Advice of Charge parameter according to GSM

tariffSwitch
 Specifies the tariff switch that signifies when the second set of AoC parameters becomes valid.

Returns -

Errors -

Method **setCallDetailRecordInformation ()**

Allows an application to include charging information in network generated CDR.

Direction Application to network

Parameters **callSessionID**
 Specifies the call session ID of the call.

callChargePlancallDetailRecordInfoApplication specific charging information.Free Format string containing the application specific charging information**Returns** -**Errors** -

Pre-paid with advice of charge

The next example shows how a pre-paid application can be implemented:

With a pre-paid application it is the application that will determine the charging for the call. This means that the application will hold the whole tariffing scheme needed and needs to control the whole call. For the call shown the following conditions apply:

- It is a long call
- Two tariff changes take place during the call.
- The application will inform the user about the applicable charging (the methods needed for this are described in section 7.5.2).

After the application has been triggered, it sends a superviseCall_Req() message indicating that the application will be responsible for charging the call. Before the call is be routed to the requested destination(5), the application sends the allowed time for the call (4) and informs the user about the charging applicable (using the Advice of Charge functionality in the core network) for this call (3). The sent information consists of two sets of AoC information and a tariff switch. The application will be notified via the superviseCall_Res() message if the tariff switch expired during the supervised period. This allows the application to send a new set of AoC information and a new tariff switch.

The application is notified of the expiration of the allowed time (7) and determines if the user has enough account left to continue with the call.

- 1 If there is enough account left a new time slot is allowed
- 2 Is there not enough account, the user will be notified and the call terminated after some time in order to allow the user to finish the call graciously.

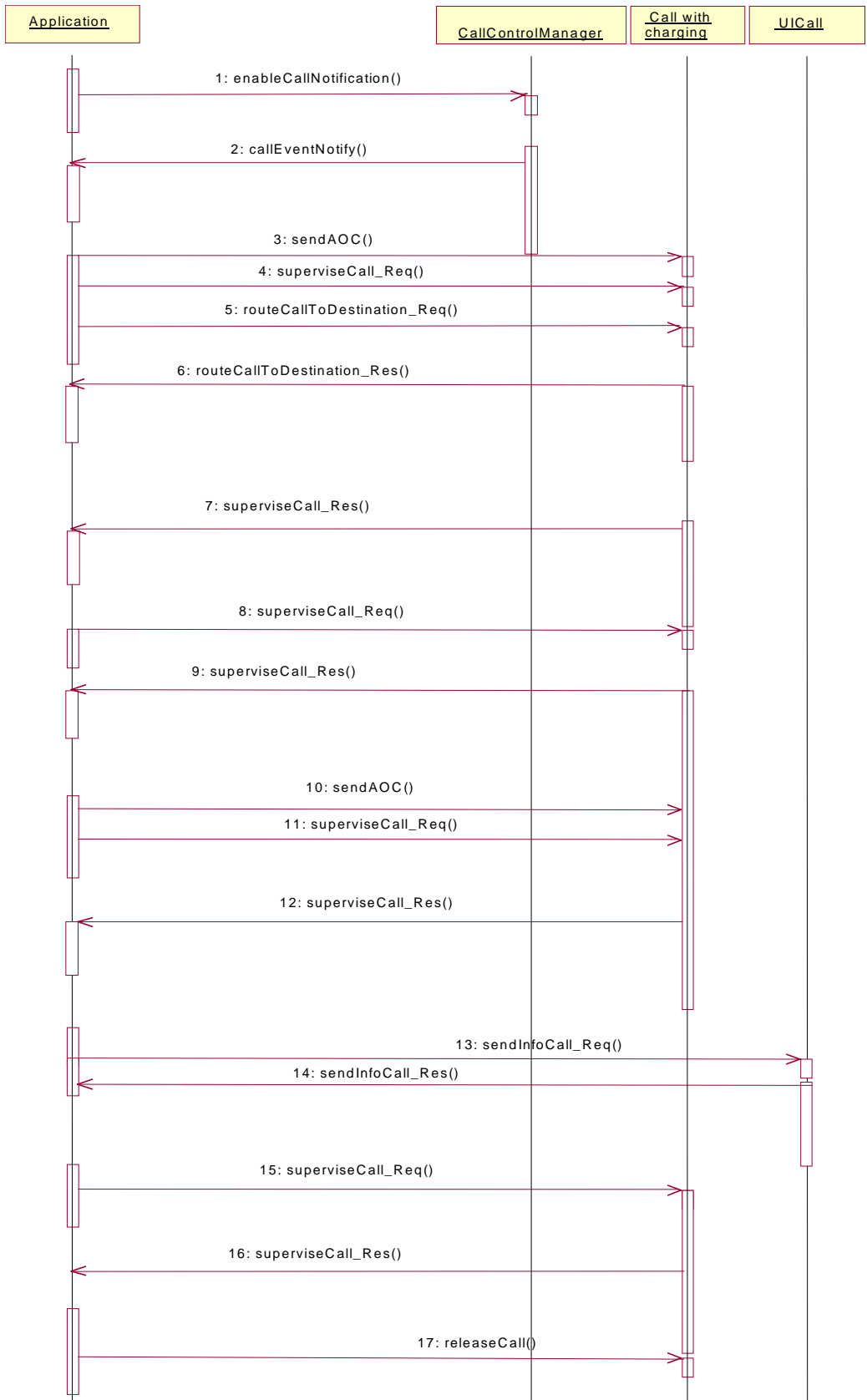


Figure 1718 Pre-paid with AoC

Annex A: Example of Framework to SCS Interface (informative)

The Service Factory interface allows the framework to get access to a manager interface of a network SCF. It is used during the signServiceAgreement, in order to return an SCF manager interface reference to the application. Each SCF has a manager interface that is the initial point of contact for the network SCF. E.g., the generic call control SCF uses the Call Manager interface.

Method **getServiceManager()**

This method returns an SCF manager interface reference for the specified application. Usually, but not necessarily, this involves the instantiation of a new SCF manager interface class.

Direction Network to network (framework to service capability server)

Parameters **application**
Specifies the application for which the SCF manager interface is requested.

Returns **serviceManager**
Specifies the SCF manager interface reference for the specified applicationID.

Errors -

History

Date	Version	Comment
July 1999	0.1.0	Initial Draft produced in Hazlet, New Jersey, USA
September 1999	0.2.0	Version presented to S2 plenary in Bonn, Germany (not including all agreed changes yet from VHE/OSA adhoc session)
September 1999	0.2.1	Output of Bonn meeting (Presented to SA for information, since V1.0.0 was not available due to 3GPP e-mail exploder problems).
October 1999	0.3.0	Version sent to S2 e-mail list and proposed to send to SA plenary
October 1999	0.3.1	Small editorial updates in Interface section (subclauses 6 and 7)
October 1999	1.0.0	Version 0.3.1 raised to V1.0.0 by S2 e-mail approval and sent to SA e-mail list for information
November 1999	1.1.0	Updated after comments in S2#9 according to S2-99C06 (S2-99B32, S2-99B33 and S2-99B36)
December 1999	1.1.1	Updated after drafting session in Munich, approved in S2#11

February 2000	1.2.0	Updated after S2#11 according to S2-000230 (S2-000081, S2-000146, S2-000148, S2-000172, S2-000187, S2-000188, S2-000189, S2-000202, S2-000203) and with various minor editorial changes
February 2000	1.3.0	Updated during OSA interim session in Stockholm (February 23-24)
<p>Rapporteur: Christophe Gourraud, Ericsson</p> <p>Email: christophe.gourraud@lmc.ericsson.se Telephone: +1 514 345 7900 (#5795)</p>		

From: TSG CN OSA Ad Hoc
To: TSG S2
Title: Request to Move the Message Sequence Charts from the OSA stage2 (TS 23.127) to the OSA stage 3 Document, Part 1 (TS 29.xxx).

TSG CN OSA Ad Hoc thanks TSG S2 for their Liaison Statement on "STATUS OF VHE/OSA WORK", with attachment TS23.127 v1.3.0.

TSG CN OSA Ad Hoc would like to propose to move the Message Sequence Charts (MSCs) from the OSA stage 2 document (TS 23.127 v1.3.0) to Part 1 of the OSA stage 3 document (TS 29.xxx). TSG CN OSA Ad Hoc has identified the following reasons in support of this proposal:

- The Message Sequence Charts (MSC), Class Diagrams, and State Transition Diagrams (STDs) are logically closely related. Therefore the OSA Ad Hoc feels it would be beneficial that these be contained in one single document.
- The fact that TSG CN OSA Ad Hoc and ETSI SPAN3 have joint meetings would better facilitate the maintenance of the Message Sequence Diagrams, and thus ensures compatibility between OSA and SPAN3.

The TSG CN OSA Ad Hoc is aware of the limited available time to achieve this proposal within the Release 99 time scales. Therefore the TSG CN OSA Ad Hoc suggests to complete this modification for Release 2000, should S2 support the proposal that the MSCs are to be moved to the stage 3.

TSG CN OSA Ad Hoc kindly requests TSG S2 to consider this proposal and to communicate back to TSG CN OSA Ad Hoc on the outcome.