

**Open Service Access (OSA);  
Application Programming Interface (API);  
Part 4: Call Control SCF  
(Parlay 3)**

---



---

Reference

RES/TISPAN-01008-04-OSA

---

Keywords

API, OSA, IDL, UML

**ETSI**

650 Route des Lucioles  
F-06921 Sophia Antipolis Cedex - FRANCE

Tel.: +33 4 92 94 42 00 Fax: +33 4 93 65 47 16

Siret N° 348 623 562 00017 - NAF 742 C  
Association à but non lucratif enregistrée à la  
Sous-Préfecture de Grasse (06) N° 7803/88

---

**Important notice**

Individual copies of the present document can be downloaded from:

<http://www.etsi.org>

The present document may be made available in more than one electronic version or in print. In any case of existing or perceived difference in contents between such versions, the reference version is the Portable Document Format (PDF). In case of dispute, the reference shall be the printing on ETSI printers of the PDF version kept on a specific network drive within ETSI Secretariat.

Users of the present document should be aware that the document may be subject to revision or change of status. Information on the current status of this and other ETSI documents is available at

<http://portal.etsi.org/tb/status/status.asp>

If you find errors in the present document, please send your comment to one of the following services:

[http://portal.etsi.org/chaicor/ETSI\\_support.asp](http://portal.etsi.org/chaicor/ETSI_support.asp)

---

**Copyright Notification**

No part may be reproduced except as authorized by written permission.  
The copyright and the foregoing restriction extend to reproduction in all media.

© European Telecommunications Standards Institute 2004.

© The Parlay Group 2004.

All rights reserved.

**DECT™**, **PLUGTESTS™** and **UMTS™** are Trade Marks of ETSI registered for the benefit of its Members.  
**TIPHON™** and the **TIPHON logo** are Trade Marks currently being registered by ETSI for the benefit of its Members.  
**3GPP™** is a Trade Mark of ETSI registered for the benefit of its Members and of the 3GPP Organizational Partners.

# Contents

Intellectual Property Rights .....	9
Foreword.....	9
1 Scope .....	10
2 References .....	10
3 Definitions and abbreviations.....	10
3.1 Definitions .....	10
3.2 Abbreviations .....	10
4 Call Control SCF .....	10
4.1 Call Model Description .....	11
4.2 General requirements on support of methods.....	12
5 The Service Interface Specifications .....	12
5.1 Interface Specification Format .....	12
5.1.1 Interface Class .....	12
5.1.2 Method descriptions.....	12
5.1.3 Parameter descriptions.....	12
5.1.4 State Model.....	12
5.2 Base Interface.....	13
5.2.1 Interface Class IpInterface .....	13
5.3 Service Interfaces .....	13
5.3.1 Overview .....	13
5.4 Generic Service Interface .....	13
5.4.1 Interface Class IpService .....	13
6 Generic Call Control Service .....	14
6.1 Sequence Diagrams .....	14
6.1.1 Additional Callbacks.....	14
6.1.2 Alarm Call .....	16
6.1.3 Application Initiated Call.....	18
6.1.4 Call Barring 1 .....	20
6.1.5 Number Translation 1 .....	22
6.1.6 Number Translation 1 (with callbacks).....	24
6.1.7 Number Translation 2 .....	26
6.1.8 Number Translation 3 .....	28
6.1.9 Number Translation 4.....	30
6.1.10 Number Translation 5 .....	32
6.1.11 Prepaid .....	33
6.1.12 Pre-Paid with Advice of Charge (AoC) .....	35
6.2 Class Diagrams.....	38
6.3 Generic Call Control Service Interface Classes.....	39
6.3.1 Interface Class IpCallControlManager .....	40
6.3.2 Interface Class IpAppCallControlManager .....	43
6.3.3 Interface Class IpCall.....	46
6.3.4 Interface Class IpAppCall.....	50
6.4 Generic Call Control Service State Transition Diagrams .....	54
6.4.1 State Transition Diagrams for IpCallControlManager.....	54
6.4.1.1 Active State .....	55
6.4.1.2 Notification terminated State .....	55
6.4.2 State Transition Diagrams for IpCall.....	55
6.4.2.1 Network Released State .....	56
6.4.2.2 Finished State.....	56
6.4.2.3 Application Released State .....	56
6.4.2.4 No Parties State .....	56
6.4.2.5 Active State .....	56
6.4.2.6 Party in Call State .....	56

6.4.2.7	Parties in Call State .....	57
6.4.2.8	Routing to Destination(s) State .....	57
6.5	Generic Call Control Service Properties .....	57
6.5.1	List of Service Properties.....	57
6.5.2	Service Property values for the CAMEL Service Environment.....	58
6.6	Generic Call Control Data Definitions .....	59
6.6.1	Generic Call Control Event Notification Data Definitions .....	60
6.6.1.1	TpCallEventName.....	60
6.6.1.2	TpCallNotificationType .....	60
6.6.1.3	TpCallEventCriteria .....	60
6.6.1.4	TpCallEventInfo.....	61
6.6.2	Generic Call Control Data Definitions.....	61
6.6.2.1	IpCall.....	61
6.6.2.2	IpCallRef.....	61
6.6.2.3	IpAppCall.....	61
6.6.2.4	IpAppCallRef.....	61
6.6.2.5	TpCallIdentifier.....	61
6.6.2.6	IpAppCallControlManager .....	61
6.6.2.7	IpAppCallControlManagerRef.....	61
6.6.2.8	IpCallControlManager .....	61
6.6.2.9	IpCallControlManagerRef.....	62
6.6.2.10	TpCallAppInfo .....	62
6.6.2.11	TpCallAppInfoType .....	62
6.6.2.12	TpCallAppInfoSet.....	62
6.6.2.13	TpCallEndedReport.....	62
6.6.2.14	TpCallFault .....	63
6.6.2.15	TpCallInfoReport.....	63
6.6.2.16	TpCallReleaseCause .....	63
6.6.2.17	TpCallReport.....	64
6.6.2.18	TpCallAdditionalReportInfo .....	64
6.6.2.19	TpCallReportRequest.....	64
6.6.2.20	TpCallAdditionalReportCriteria.....	65
6.6.2.21	TpCallReportRequestSet.....	65
6.6.2.22	TpCallReportType .....	65
6.6.2.23	TpCallTreatment .....	66
6.6.2.24	TpCallEventCriteriaResultSet.....	66
6.6.2.25	TpCallEventCriteriaResult.....	66
7	MultiParty Call Control Service.....	66
7.1	Sequence Diagrams .....	66
7.1.1	Application initiated call setup .....	66
7.1.2	Call Barring 2 .....	68
7.1.3	Call forwarding on Busy Service .....	70
7.1.4	Call Information Collect Service .....	71
7.1.5	Complex Card Service .....	74
7.1.6	Hotline Service .....	77
7.1.7	Use of the Redirected event .....	80
7.2	Class Diagrams.....	80
7.3	MultiParty Call Control Service Interface Classes.....	82
7.3.1	Interface Class IpMultiPartyCallControlManager .....	82
7.3.2	Interface Class IpAppMultiPartyCallControlManager .....	86
7.3.3	Interface Class IpMultiPartyCall .....	89
7.3.4	Interface Class IpAppMultiPartyCall.....	93
7.3.5	Interface Class IpCallLeg .....	95
7.3.6	Interface Class IpAppCallLeg.....	101
7.4	MultiParty Call Control Service State Transition Diagrams .....	105
7.4.1	State Transition Diagrams for IpMultiPartyCallControlManager.....	105
7.4.1.1	Active State .....	105
7.4.1.2	Interrupted State .....	105
7.4.1.3	Overview of allowed methods.....	105
7.4.2	State Transition Diagrams for IpMultiPartyCall.....	106
7.4.2.1	IDLE State.....	106

7.4.2.2	ACTIVE State .....	106
7.4.2.3	RELEASED State .....	107
7.4.2.4	Overview of allowed methods.....	107
7.4.3	State Transition Diagrams for IpCallLeg.....	107
7.4.3.1	Originating Call Leg .....	108
7.4.3.1.1	Initiating State .....	108
7.4.3.1.2	Analysing State.....	110
7.4.3.1.3	Active State .....	111
7.4.3.1.4	Releasing State .....	113
7.4.3.1.5	Overview of allowed methods, Originating Call Leg STD .....	115
7.4.3.2	Terminating Call Leg .....	116
7.4.3.2.1	Idle (terminating) State.....	116
7.4.3.2.2	Active (terminating) State .....	117
7.4.3.2.3	Releasing (terminating) State .....	120
7.4.3.2.4	Overview of allowed methods and trigger events, Terminating Call Leg STD.....	122
7.5	Multi-Party Call Control Service Properties .....	122
7.5.1	List of Service Properties.....	122
7.5.2	Service Property values for the CAMEL Service Environment.....	123
7.6	Multi-Party Call Control Data Definitions .....	124
7.6.1	Event Notification Data Definitions .....	124
7.6.2	Multi-Party Call Control Data Definitions.....	124
7.6.2.1	IpCallLeg .....	124
7.6.2.2	IpCallLegRef.....	124
7.6.2.3	IpAppCallLeg.....	124
7.6.2.4	IpAppCallLegRef.....	124
7.6.2.5	IpMultiPartyCall .....	124
7.6.2.6	IpMultiPartyCallRef.....	124
7.6.2.7	IpAppMultiPartyCall.....	124
7.6.2.8	IpAppMultiPartyCallRef.....	125
7.6.2.9	IpMultiPartyCallControlManager .....	125
7.6.2.10	IpMultiPartyCallControlManagerRef .....	125
7.6.2.11	IpAppMultiPartyCallControlManager .....	125
7.6.2.12	IpAppMultiPartyCallControlManagerRef.....	125
7.6.2.13	TpAppCallLegRefSet.....	125
7.6.2.14	TpMultiPartyCallIdentifier .....	125
7.6.2.15	TpAppMultiPartyCallBack .....	125
7.6.2.16	TpAppMultiPartyCallBackRefType .....	126
7.6.2.17	TpAppCallLegCallBack.....	126
7.6.2.18	TpMultiPartyCallIdentifierSet .....	126
7.6.2.19	TpCallAppInfo .....	126
7.6.2.20	TpCallAppInfoType .....	127
7.6.2.21	TpCallAppInfoSet.....	127
7.6.2.22	TpCallEventRequest .....	127
7.6.2.23	TpCallEventRequestSet .....	127
7.6.2.24	TpCallEventType .....	128
7.6.2.25	TpAdditionalCallEventCriteria .....	130
7.6.2.26	TpCallEventInfo.....	130
7.6.2.27	TpCallAdditionalEventInfo.....	131
7.6.2.28	TpCallNotificationRequest .....	131
7.6.2.29	TpCallNotificationScope.....	131
7.6.2.30	TpCallNotificationInfo.....	132
7.6.2.31	TpCallNotificationReportScope.....	132
7.6.2.32	TpNotificationRequested .....	132
7.6.2.33	TpNotificationRequestedSet .....	132
7.6.2.34	TpReleaseCause .....	132
7.6.2.35	TpReleaseCauseSet .....	132
7.6.2.36	TpCallLegIdentifier.....	133
7.6.2.37	TpCallLegIdentifierSet .....	133
7.6.2.38	TpCallLegAttachMechanism .....	133
7.6.2.39	TpCallLegConnectionProperties .....	133
7.6.2.40	TpCallLegInfoReport.....	133
7.6.2.41	TpCallLegInfoType.....	134

7.6.2.42	TpCallLegSuperviseTreatment .....	134
8	MultiMedia Call Control Service .....	134
8.1	Sequence Diagrams .....	134
8.1.1	Barring for media combined with call routing, alternative 1 .....	134
8.1.2	Barring for media combined with call routing, alternative 2 .....	136
8.1.3	Barring for media, simple .....	138
8.1.4	Call Volume charging supervision.....	139
8.2	Class Diagrams.....	141
8.3	MultiMedia Call Control Service Interface Classes .....	142
8.3.1	Interface Class IpMultiMediaCallControlManager .....	143
8.3.2	Interface Class IpAppMultiMediaCallControlManager .....	145
8.3.3	Interface Class IpMultiMediaCall.....	146
8.3.4	Interface Class IpAppMultiMediaCall.....	147
8.3.5	Interface Class IpMultiMediaCallLeg .....	148
8.3.6	Interface Class IpAppMultiMediaCallLeg.....	149
8.3.7	Interface Class IpMultiMediaStream.....	150
8.4	MultiMedia Call Control Service State Transition Diagrams .....	151
8.5	Multi-Media Call Control Data Definitions .....	151
8.5.1	Event Notification Data Definitions .....	151
8.5.1.1	TpMediaStreamRequestSet.....	151
8.5.1.2	TpMediaStreamRequest.....	151
8.5.1.3	TpMediaStreamDirection.....	152
8.5.1.4	TpMediaStreamDataTypeRequest .....	152
8.5.1.5	TpAudioCapabilitiesType .....	152
8.5.1.6	TpVideoCapabilitiesType .....	153
8.5.1.7	TpDataCapabilities.....	153
8.5.1.8	TpMediaStreamEventType .....	153
8.5.1.9	TpMediaStreamSet.....	153
8.5.1.10	TpMediaStream.....	153
8.5.1.11	TpMediaStreamDataType .....	153
8.5.2	Multi-Media Call Control Data Definitions.....	153
8.5.2.1	IpMultiMediaCall.....	153
8.5.2.2	IpMultiMediaCallRef .....	153
8.5.2.3	IpAppMultiMediaCall.....	154
8.5.2.4	IpAppMultiMediaCallRef .....	154
8.5.2.5	IpMultiMediaCallLeg .....	154
8.5.2.6	IpMultiMediaCallLegRef.....	154
8.5.2.7	IpAppMultiMediaCallLeg.....	154
8.5.2.8	IpAppMultiMediaCallLegRef.....	154
8.5.2.9	TpAppMultiMediaCallLegRefSet.....	154
8.5.2.10	TpMultiMediaCallIdentifier.....	154
8.5.2.11	TpMultiMediaCallIdentifierSet.....	154
8.5.2.12	TpMultiMediaCallLegIdentifier.....	154
8.5.2.13	TpMultiMediaCallLegIdentifierSet.....	154
8.5.2.14	IpAppMultiMediaCallControlManager.....	155
8.5.2.15	IpAppMultiMediaCallControlManagerRef.....	155
8.5.2.16	TpAppMultiMediaCallBack.....	155
8.5.2.17	TpAppMultiMediaCallBackRefType.....	155
8.5.2.18	TpAppMultiMediaCallLegCallBack.....	155
8.5.2.19	TpCallSuperviseVolume .....	156
8.5.2.20	TpNotificationMediaRequest .....	156
8.5.2.21	TpMediaNotificationRequested .....	156
8.5.2.22	TpMediaNotificationsRequestedSet.....	156
9	Conference Call Control Service.....	156
9.1	Sequence Diagrams .....	156
9.1.1	Meet-me conference without subconferencing .....	156
9.1.2	Non-add hoc add-on with subconferencing .....	158
9.1.3	Non-addhoc add-on multimedia .....	160
9.1.4	Resource Reservation .....	163
9.2	Class Diagrams.....	164

9.3	Conference Call Control Service Interface Classes.....	165
9.3.1	Interface Class IpConfCallControlManager .....	166
9.3.2	Interface Class IpAppConfCallControlManager.....	169
9.3.3	Interface Class IpConfCall.....	170
9.3.4	Interface Class IpAppConfCall.....	171
9.3.5	Interface Class IpSubConfCall .....	173
9.3.6	Interface Class IpAppSubConfCall.....	176
9.4	Conference Call Control Service State Transition Diagrams .....	177
9.5	Conference Call Control Data Definitions .....	177
9.5.1	Event Notification Data Definitions .....	178
9.5.2	Conference Call Control Data Definitions.....	178
9.5.2.1	IpConfCall.....	178
9.5.2.2	IpConfCallRef.....	178
9.5.2.3	IpAppConfCall.....	178
9.5.2.4	IpAppConfCallRef.....	178
9.5.2.5	IpSubConfCall .....	178
9.5.2.6	IpSubConfCallRef.....	178
9.5.2.7	IpAppSubConfCall.....	178
9.5.2.8	IpAppSubConfCallRef.....	178
9.5.2.9	TpSubConfCallIdentifierSet.....	178
9.5.2.10	TpConfCallIdentifier.....	179
9.5.2.11	TpSubConfCallIdentifier.....	179
9.5.2.12	IpAppConfCallControlManager.....	179
9.5.2.13	IpAppConfCallControlManagerRef.....	179
9.5.2.14	TpConfPolicyType.....	179
9.5.2.15	TpConfPolicy .....	179
9.5.2.16	TpMonoMediaConfPolicy .....	180
9.5.2.17	TpJoinEventInfo.....	180
9.5.2.18	TpConfSearchCriteria .....	180
9.5.2.19	TpConfSearchResult .....	180
9.5.2.20	TpMultiMediaConfPolicy .....	181
9.5.2.21	TpResourceReservation .....	181
9.5.2.22	TpVideoHandlingType .....	181
10	Common Call Control Data Types.....	181
10.1	TpCallAlertingMechanism .....	181
10.2	TpCallBearerService .....	182
10.3	TpCallChargePlan .....	182
10.4	TpCallPartyToChargeAdditionalInfo.....	182
10.5	TpCallPartyToChargeType .....	183
10.6	TpCallChargeOrderCategory .....	183
10.7	TpCallEndedReport.....	183
10.8	TpCallError .....	183
10.9	TpCallAdditionalErrorInfo.....	184
10.10	TpCallErrorType .....	184
10.11	TpCallInfoReport .....	184
10.12	TpCallInfoType.....	185
10.13	TpCallLoadControlMechanism.....	185
10.14	TpCallLoadControlIntervalRate.....	185
10.15	TpCallLoadControlMechanismType.....	185
10.16	TpCallMonitorMode .....	186
10.17	TpCallNetworkAccessType .....	186
10.18	TpCallPartyCategory.....	186
10.19	TpCallServiceCode .....	187
10.20	TpCallServiceCodeSet .....	187
10.21	TpCallServiceCodeType .....	187
10.22	TpCallSuperviseReport .....	187
10.23	TpCallSuperviseTreatment.....	188
10.24	TpCallTeleService.....	188
10.25	TpCallTreatment .....	189
10.26	TpCallTreatmentType .....	189
10.27	TpCallAdditionalTreatmentInfo.....	189

10.28	TpMediaType .....	189
<b>Annex A (normative): OMG IDL Description of Call Control SCF .....</b>		<b>190</b>
<b>Annex B (informative): Contents of 3GPP OSA R4 Call Control .....</b>		<b>191</b>
<b>Annex C (informative): Record of changes .....</b>		<b>192</b>
C.1	Interfaces .....	192
C.1.1	New .....	192
C.1.2	Deprecated.....	192
C.1.3	Removed.....	192
C.2	Methods .....	193
C.2.1	New .....	193
C.2.2	Deprecated.....	193
C.2.3	Modified.....	193
C.2.4	Removed.....	193
C.3	Data Definitions .....	194
C.3.1	New .....	194
C.3.2	Modified.....	194
C.3.3	Removed.....	194
C.4	Service Properties.....	194
C.4.1	New .....	194
C.4.2	Deprecated.....	195
C.4.3	Modified.....	195
C.4.4	Removed.....	195
C.5	Exceptions .....	195
C.5.1	New .....	195
C.5.2	Modified.....	196
C.5.3	Removed.....	196
C.6	Others .....	196
History .....		197

---

# Intellectual Property Rights

IPRs essential or potentially essential to the present document may have been declared to ETSI. The information pertaining to these essential IPRs, if any, is publicly available for **ETSI members and non-members**, and can be found in ETSI SR 000 314: "*Intellectual Property Rights (IPRs); Essential, or potentially Essential, IPRs notified to ETSI in respect of ETSI standards*", which is available from the ETSI Secretariat. Latest updates are available on the ETSI Web server (<http://webapp.etsi.org/IPR/home.asp>).

Pursuant to the ETSI IPR Policy, no investigation, including IPR searches, has been carried out by ETSI. No guarantee can be given as to the existence of other IPRs not referenced in ETSI SR 000 314 (or the updates on the ETSI Web server) which are, or may be, or may become, essential to the present document.

---

## Foreword

This ETSI Standard (ES) has been produced by ETSI Technical Committee Telecommunications and Internet converged Services and Protocols for Advanced Networking (TISPAN), and is now submitted for the ETSI standards Membership Approval Procedure.

The present document is part 4 of a multi-part deliverable covering Open Service Access (OSA); Application Programming Interface (API), as identified below. The API specification (ES 201 915) is structured in the following parts:

- Part 1: "Overview";
- Part 2: "Common Data Definitions";
- Part 3: "Framework";
- Part 4: "Call Control SCF";**
- Part 5: "User Interaction SCF";
- Part 6: "Mobility SCF";
- Part 7: "Terminal Capabilities SCF";
- Part 8: "Data Session Control SCF";
- Part 9: "Generic Messaging SCF";
- Part 10: "Connectivity Manager SCF";
- Part 11: "Account Management SCF";
- Part 12: "Charging SCF".

The present document has been defined jointly between ETSI, The Parlay Group (<http://www.parlay.org>) and the 3GPP, in co-operation with a number of JAIN™ Community (<http://www.java.sun.com/products/jain>) member companies.

**The present document forms part of the Parlay 3.4 set of specifications.**

**A subset of the present document is in 3GPP TS 29.198-4 4.a.0 (Release 4).**

---

# 1 Scope

The present document is part 4 of the Stage 3 specification for an Application Programming Interface (API) for Open Service Access (OSA).

The OSA specifications define an architecture that enables application developers to make use of network functionality through an open standardised interface, i.e. the OSA APIs.

The present document specifies the Call Control Service Capability Feature (SCF) aspects of the interface. All aspects of the Call Control SCF are defined here, these being:

- Sequence Diagrams.
- Class Diagrams.
- Interface specification plus detailed method descriptions.
- State Transition diagrams.
- Data Definitions.
- IDL Description of the interfaces.

The process by which this task is accomplished is through the use of object modelling techniques described by the Unified Modelling Language (UML).

---

# 2 References

The references listed in clause 2 of ES 201 915-1 contain provisions which, through reference in this text, constitute provisions of the present document.

ETSI ES 201 915-1: "Open Service Access (OSA); Application Programming Interface (API); Part 1: Overview (Parlay 3)".

---

# 3 Definitions and abbreviations

## 3.1 Definitions

For the purposes of the present document, the terms and definitions given in ES 201 915-1 apply.

## 3.2 Abbreviations

For the purposes of the present document, the abbreviations defined in ES 201 915-1 apply.

---

# 4 Call Control SCF

Two flavours of call control APIs have been included in 3GPP Release 4. These are the generic call control and the multi-party call control. The generic call control is the same API as was already present in the previous specification for 3GPP Release 99 (TS 129 198 V3.4.0) and is in principle able to satisfy the requirements on Call Control APIs for 3GPP Release 4.

However, the joint work between 3GPP CN5, ETSI and Parlay with collaboration from JAIN has been focussed on the Multi-party call control API. A number of improvements on call control functionality have been made and are reflected in this API. For this it was necessary to break the inheritance that previously existed between Generic and Multi-party call control.

The joint call control group has furthermore decided that the multi-party call control is to be considered as the future base call control family and the technical work will not be continued on Generic Call control. Errors or technical flaws will of course be corrected.

The following clauses describe each aspect of the Call Control Service Capability Feature (SCF).

The order is as follows:

- The Sequence diagrams give the reader a practical idea of how each of the SCF is implemented.
- The Class relationships clause show how each of the interfaces applicable to the SCF, relate to one another.
- The Interface specification clause describes in detail each of the interfaces shown within the Class diagram part.
- The State Transition Diagrams (STD) show the transition between states in the SCF. The states and transitions are well-defined; either methods specified in the Interface specification or events occurring in the underlying networks cause state transitions.
- The Data Definitions clause show a detailed expansion of each of the data types associated with the methods within the classes. Note that some data types are used in other methods and classes and are therefore defined within the Common Data types part of the present document.

## 4.1 Call Model Description

The adopted call model has the following objects:

- a call object. A call is a relation between a number of parties. The call object relates to the entire call view from the application. E.g. the entire call will be released when a release is called on the call. Note that different applications can have different views on the same physical call, e.g. one application for the originating side and another application for the terminating side. The applications will not be aware of each other, all "communication" between the applications will be by means of network signalling. The API currently does not specify any feature interaction mechanisms;
- a call leg object. The leg object represents a logical association between a call and an address. The relationship includes at least the signalling relation with the party. The relation with the address is only made when the leg is routed. Before that the leg object is IDLE and not yet associated with the address;
- an address. The address logically represents a party in the call;
- a terminal. A terminal is the end-point of the signalling and/or media for a party. This object type is currently not addressed.

The call object is used to establish a relation between a number of parties by creating a leg for each party within the call.

Associated with the signalling relationship represented by the call leg, there may also be a bearer connection (e.g. in the traditional voice only networks) or a number (zero or more) of media channels (in multi-media networks).

A leg can be attached to the call or detached from the call. When the leg is attached, this means that media or bearer channels related to the legs are connected to the media or bearer channels of the other legs that are attached to the same call. I.e. only legs that are attached can "speak" to each other. A leg can have a number of states, depending on the signalling received from or sent to the party associated with the leg. Usually there is a limit to the number of legs that are in being routed (i.e. the connection is being established) or connected to the call (i.e. the connection is established). Also, there usually is a limit to the number of legs that can be simultaneously attached to the same call.

Some networks distinguish between controlling and passive legs. By definition the call will be released when the controlling leg is released. All other legs are called passive legs. There can be at most one controlling leg per call. However, there is currently no way the application can influence whether a Leg is controlling or not.

There are two ways for an application to get the control of a call. The application can request to be notified of calls that meet certain criteria. When a call occurs in the network that meets these criteria, the application is notified and can control the call. Some legs will already be associated with the call in this case. Another way is to create a new call from the application.

## 4.2 General requirements on support of methods

An implementation of this API which supports or implements a method described in the present document, shall support or implement the functionality described for that method, for at least one valid set of values for the parameters of that method.

Where a method is not supported by an implementation of a Service interface, the exception `P_METHOD_NOT_SUPPORTED` shall be returned to any call of that method.

Where a method is not supported by an implementation of an Application interface, a call to that method shall be possible, and no exception shall be returned.

---

# 5 The Service Interface Specifications

## 5.1 Interface Specification Format

This clause defines the interfaces, methods and parameters that form a part of the API specification. The Unified Modelling Language (UML) is used to specify the interface classes. The general format of an interface specification is described below.

### 5.1.1 Interface Class

This shows a UML interface class description of the methods supported by that interface, and the relevant parameters and types. The Service and Framework interfaces for enterprise-based client applications are denoted by classes with name `Ip<name>`. The callback interfaces to the applications are denoted by classes with name `IpApp<name>`. For the interfaces between a Service and the Framework, the Service interfaces are typically denoted by classes with name `IpSvc<name>`, while the Framework interfaces are denoted by classes with name `IpFw<name>`.

### 5.1.2 Method descriptions

Each method (API method "call") is described. Both synchronous and asynchronous methods are used in the API. Asynchronous methods are identified by a "Req" suffix for a method request, and, if applicable, are served by asynchronous methods identified by either a "Res" or "Err" suffix for method results and errors, respectively. To handle responses and reports, the application or service developer must implement the relevant `IpApp<name>` or `IpSvc<name>` interfaces to provide the callback mechanism.

### 5.1.3 Parameter descriptions

Each method parameter and its possible values are described. Parameters described as "in" represent those that must have a value when the method is called. Those described as "out" are those that contain the return result of the method when the method returns.

### 5.1.4 State Model

If relevant, a state model is shown to illustrate the states of the objects that implement the described interface.

## 5.2 Base Interface

### 5.2.1 Interface Class IpInterface

All application, framework and service interfaces inherit from the following interface. This API Base Interface does not provide any additional methods.

<<Interface>> IpInterface

## 5.3 Service Interfaces

### 5.3.1 Overview

The Service Interfaces provide the interfaces into the capabilities of the underlying network - such as call control, user interaction, messaging, mobility and connectivity management.

The interfaces that are implemented by the services are denoted as "Service Interface". The corresponding interfaces that must be implemented by the application (e.g. for API callbacks) are denoted as "Application Interface".

## 5.4 Generic Service Interface

### 5.4.1 Interface Class IpService

Inherits from: IpInterface.

All service interfaces inherit from the following interface.

<<Interface>> IpService
setCallback (appInterface : in IpInterfaceRef) : void setCallbackWithSessionID (appInterface : in IpInterfaceRef, sessionID : in TpSessionID) : void

#### *Method*

#### **setCallback()**

This method specifies the reference address of the callback interface that a service uses to invoke methods on the application. It is not allowed to invoke this method on an interface that uses SessionIDs.

*Parameters***appInterface : in IpInterfaceRef**

Specifies a reference to the application interface, which is used for callbacks

*Raises***TpCommonExceptions, P\_INVALID\_INTERFACE\_TYPE***Method***setCallbackWithSessionID()**

This method specifies the reference address of the application's callback interface that a service uses for interactions associated with a specific session ID: e.g. a specific call, or call leg. It is not allowed to invoke this method on an interface that does not use SessionIDs.

*Parameters***appInterface : in IpInterfaceRef**

Specifies a reference to the application interface, which is used for callbacks.

**sessionID : in TpSessionID**

Specifies the session for which the service can invoke the application's callback interface.

*Raises***TpCommonExceptions, P\_INVALID\_SESSION\_ID, P\_INVALID\_INTERFACE\_TYPE**

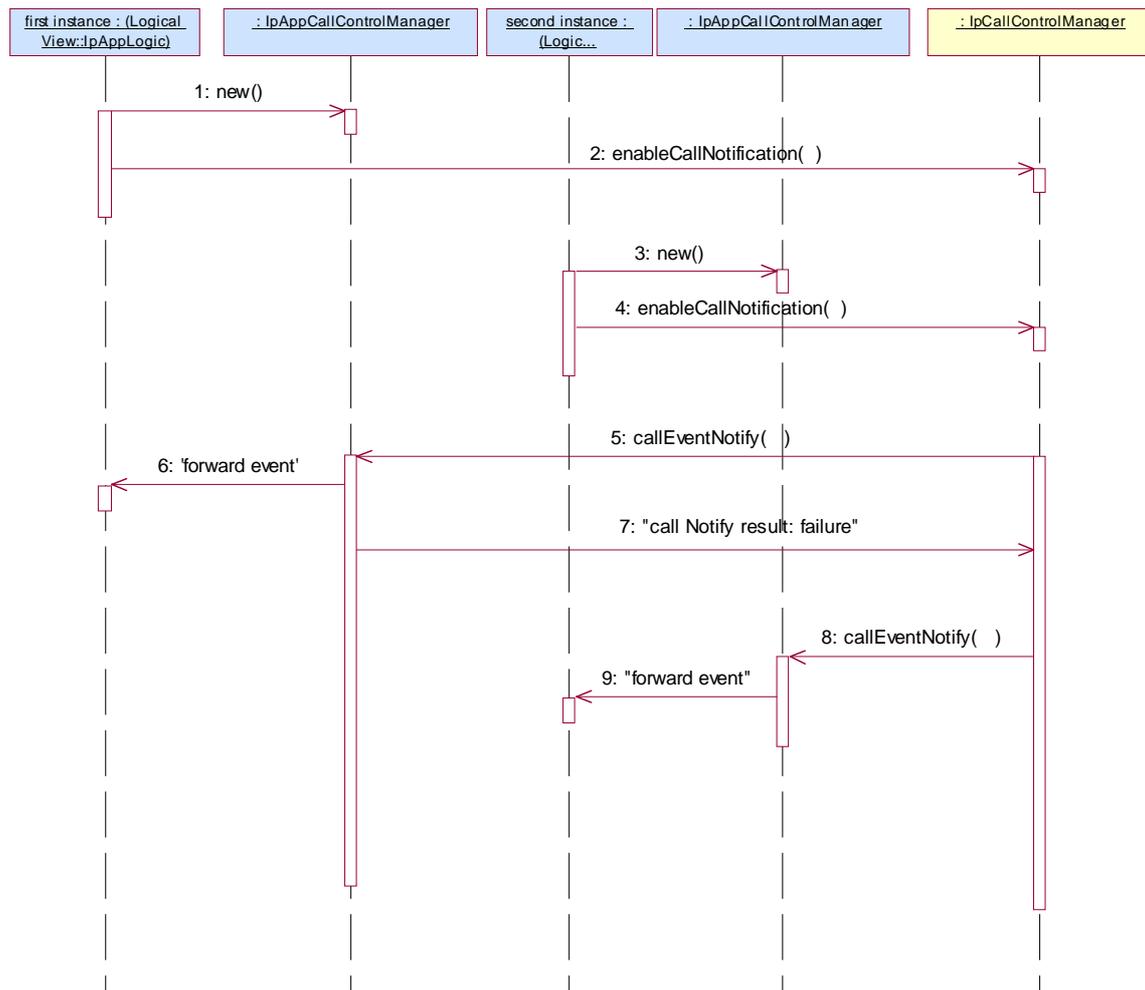
---

## 6 Generic Call Control Service

### 6.1 Sequence Diagrams

#### 6.1.1 Additional Callbacks

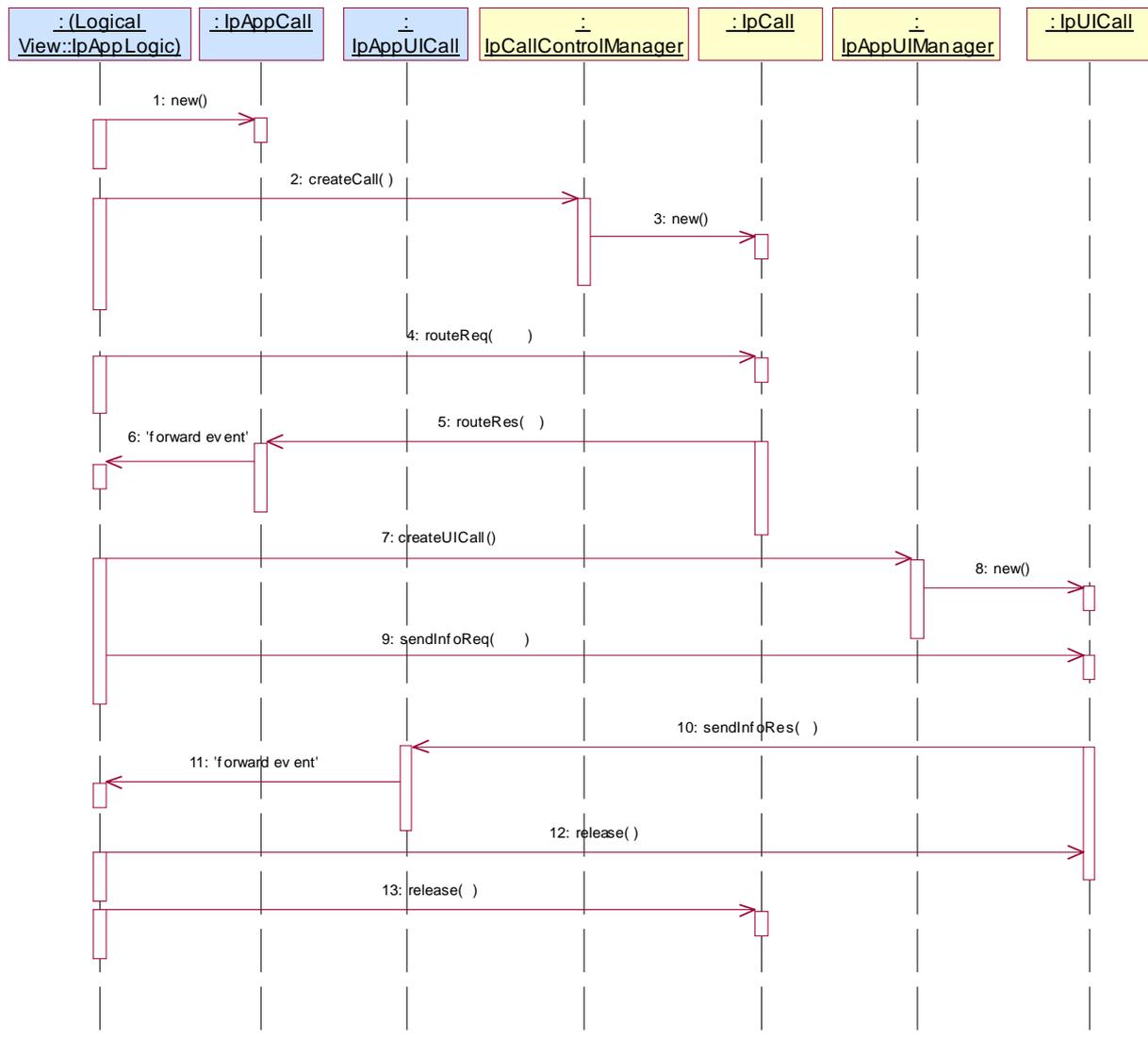
The following sequence diagram shows how an application can register two call back interfaces for the same set of events. If one of the call backs can not be used, e.g. because the application crashed, the other call back interface is used instead.



- 1: The first instance of the application is started on node 1. The application creates a new IpAppCallControlManager to handle callbacks for this first instance of the logic.
- 2: The enableCallNotification is associated with an applicationID. The call control manager uses the applicationID to decide whether this is the same application.
- 3: The second instance of the application is started on node 2. The application creates a new IpAppCallControlManager to handle callbacks for this second instance of the logic.
- 4: The same enableCallNotification request is sent as for the first instance of the logic. Because both requests are associated with the same application, the second request is not rejected, but the specified callback object is stored as an additional callback.
- 5: When the trigger occurs one of the first instance of the application is notified. The gateway may have different policies on how to handle additional callbacks, e.g. always first try the first registered or use some kind of round robin scheme.
- 6: The event is forwarded to the first instance of the logic.
- 7: When the first instance of the application is overloaded or unavailable this is communicated with an exception to the call control manager.
- 8: Based on this exception the call control manager will notify another instance of the application (if available).
- 9: The event is forwarded to the second instance of the logic.

## 6.1.2 Alarm Call

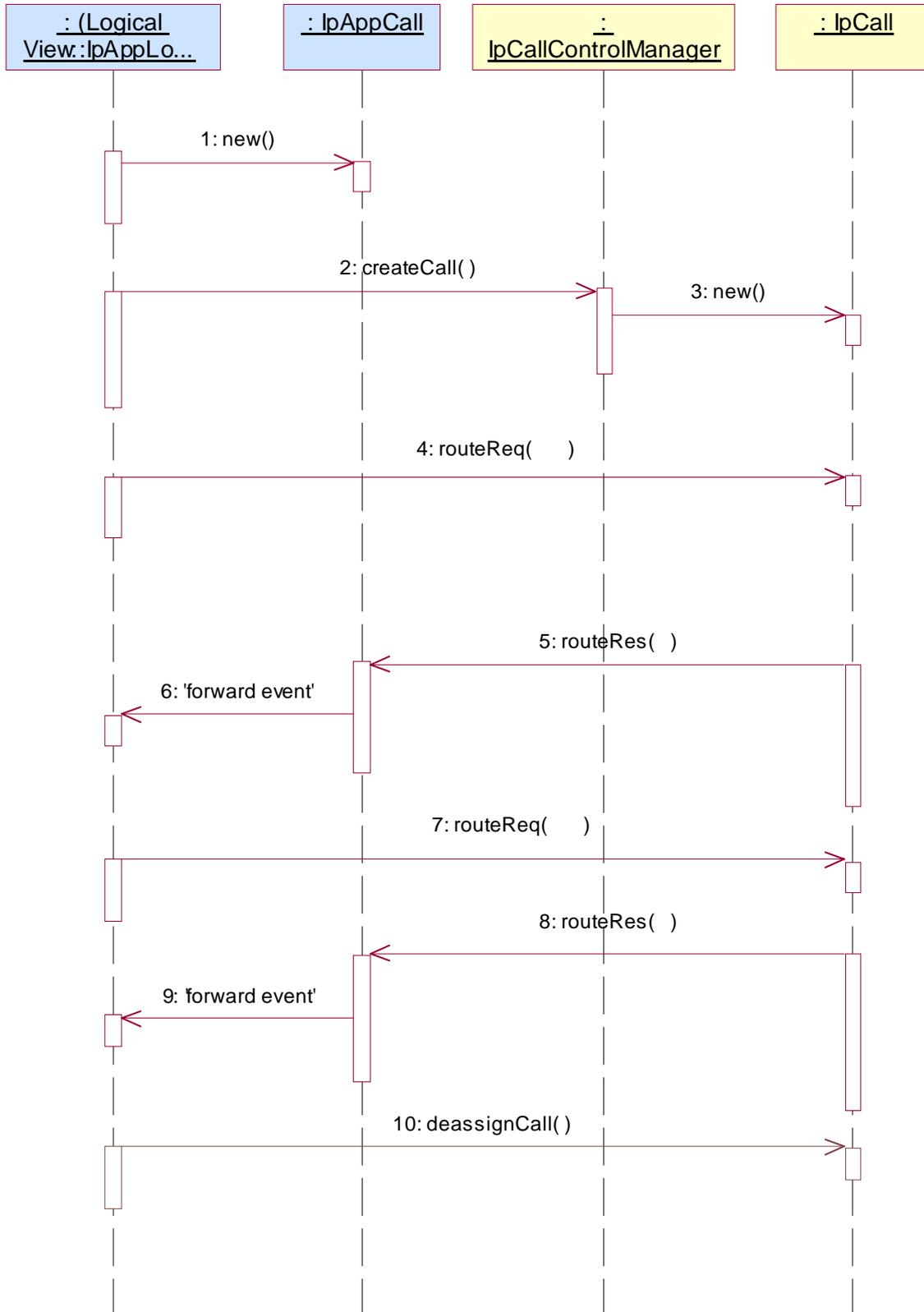
The following sequence diagram shows a "reminder message", in the form of an alarm, being delivered to a customer as a result of a trigger from an application. Typically, the application would be set to trigger at a certain time, however, the application could also trigger on events.



- 1: This message is used to create an object implementing the IpAppCall interface.
- 2: This message requests the object implementing the IpCallControlManager interface to create an object implementing the IpCall interface.
- 3: Assuming that the criteria for creating an object implementing the IpCall interface (e.g. load control values not exceeded) is met it is created.
- 4: This message instructs the object implementing the IpCall interface to route the call to the customer destined to receive the "reminder message".
- 5: This message passes the result of the call being answered to its callback object.
- 6: This message is used to forward the previous message to the IpAppLogic.
- 7: The application requests a new UICall object that is associated with the call object.
- 8: Assuming all criteria are met, a new UICall object is created by the service.
- 9: This message instructs the object implementing the IpUICall interface to send the alarm to the customer's call.
- 10: When the announcement ends this is reported to the call back interface.
- 11: The event is forwarded to the application logic.
- 12: The application releases the UICall object, since no further announcements are required. Alternatively, the application could have indicated P\_FINAL\_REQUEST in the sendInfoReq in which case the UICall object would have been implicitly released after the announcement was played.
- 13: The application releases the call and all associated parties.

### 6.1.3 Application Initiated Call

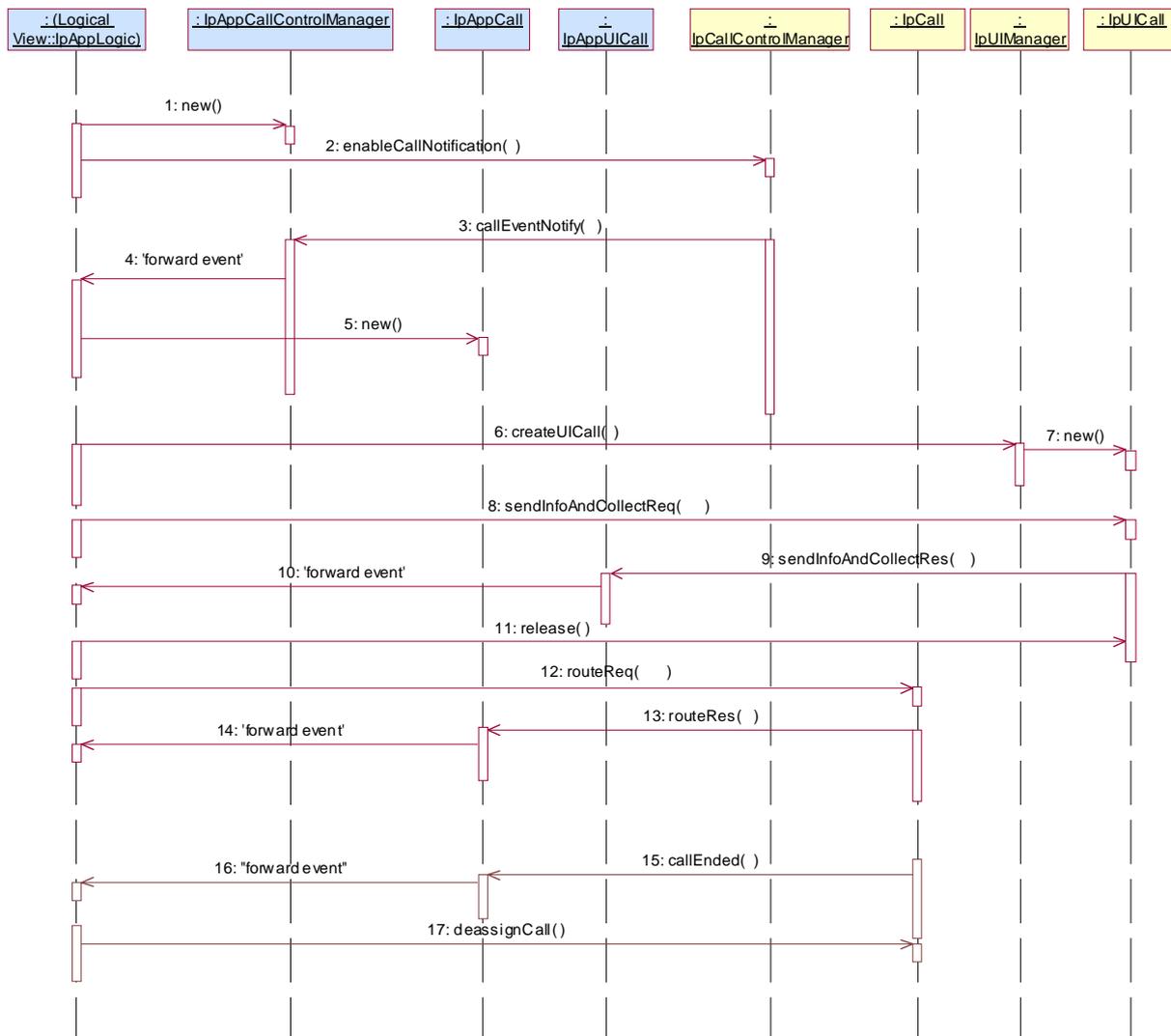
The following sequence diagram shows an application creating a call between party A and party B. This sequence could be done after a customer has accessed a Web page and selected a name on the page of a person or organisation to talk to.



- 1: This message is used to create an object implementing the IpAppCall interface.
- 2: This message requests the object implementing the IpCallControlManager interface to create an object implementing the IpCall interface.
- 3: Assuming that the criteria for creating an object implementing the IpCall interface (e.g. load control values not exceeded) is met, it is created.
- 4: This message is used to route the call to the A subscriber (origination). In the message the application request response when the A party answers.
- 5: This message indicates that the A party answered the call.
- 6: This message forwards the previous message to the application logic.
- 7: This message is used to route the call to the B-party. Also in this case a response is requested for call answer or failure.
- 8: This message indicates that the B-party answered the call. The call now has two parties and a speech connection is automatically established between them.
- 9: This message is used to forward the previous message to the IpAppLogic.
- 10: Since the application is no longer interested in controlling the call, the application deassigns the call. The call will continue in the network, but there will be no further communication between the call object and the application.

## 6.1.4 Call Barring 1

The following sequence diagram shows a call barring service, initiated as a result of a prearranged event being received by the call control service. Before the call is routed to the destination number, the calling party is asked for a PIN code. The code is accepted and the call is routed to the original called party.

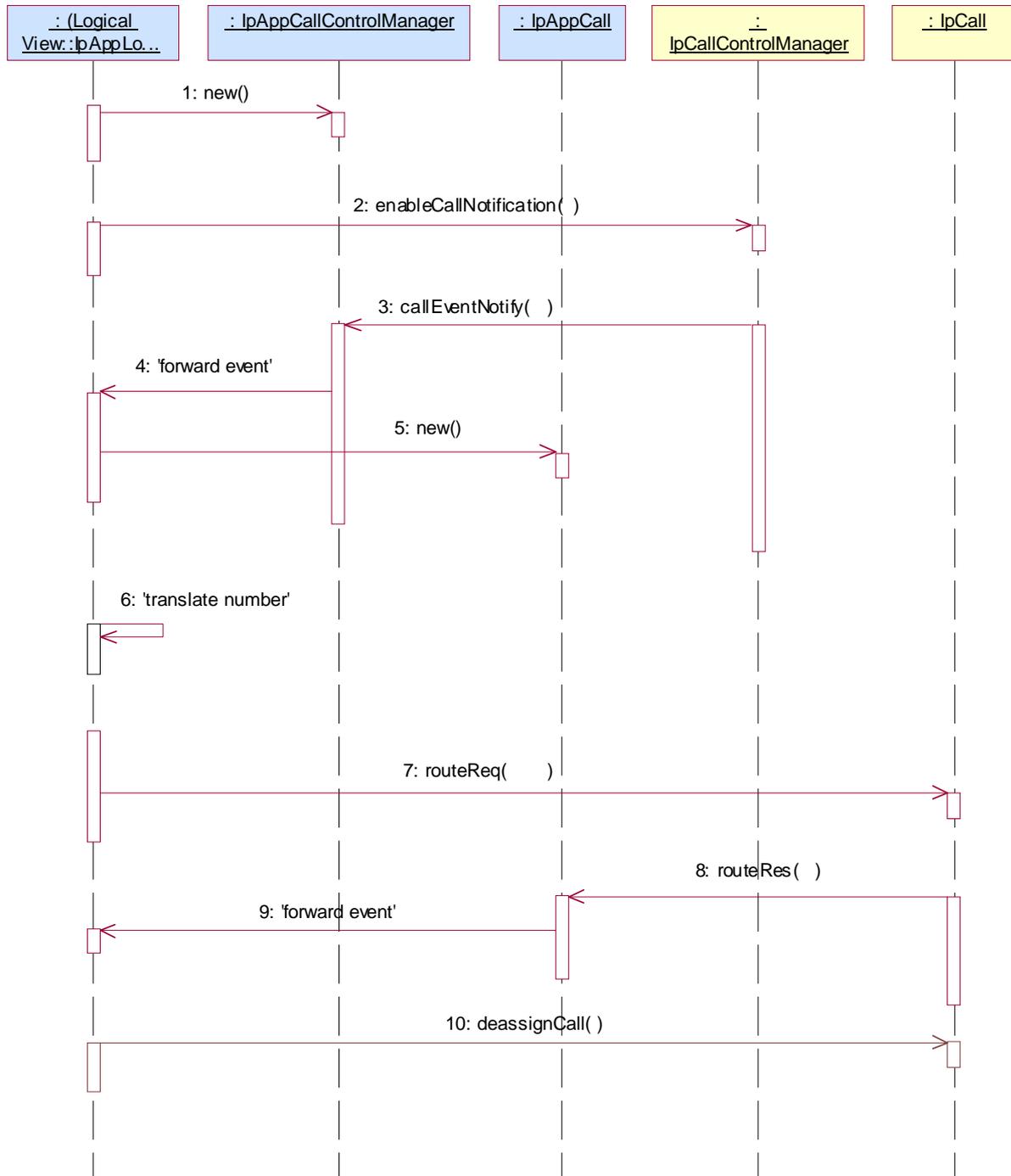


- 1: This message is used by the application to create an object implementing the IpAppCallControlManager interface.
- 2: This message is sent by the application to enable notifications on new call events. As this sequence diagram depicts a call barring service, it is likely that all new call events destined for a particular address or address range prompted for a password before the call is allowed to progress. When a new call, that matches the event criteria set, arrives a message (not shown) is directed to the object implementing the IpCallControlManager. Assuming that the criteria for creating an object implementing the IpCall interface (e.g. load control values not exceeded) is met, other messages (not shown) are used to create the call and associated call leg object.
- 3: This message is used to pass the new call event to the object implementing the IpAppCallControlManager interface.
- 4: This message is used to forward the previous message to the IpAppLogic.

- 5: This message is used by the application to create an object implementing the IpAppCall interface. The reference to this object is passed back to the object implementing the IpCallControlManager using the return parameter of the callEventNotify.
- 6: This message is used to create a new UICall object. The reference to the call object is given when creating the UICall.
- 7: Provided all the criteria are fulfilled, a new UICall object is created.
- 8: The call barring service dialogue is invoked.
- 9: The result of the dialogue, which in this case is the PIN code, is returned to its callback object.
- 10: This message is used to forward the previous message to the IpAppLogic.
- 11: This message releases the UICall object.
- 12: Assuming the correct PIN is entered, the call is forward routed to the destination party.
- 13: This message passes the result of the call being answered to its callback object.
- 14: This message is used to forward the previous message to the IpAppLogic.
- 15: When the call is terminated in the network, the application will receive a notification. This notification will always be received when the call is terminated by the network in a normal way, the application does not have to request this event explicitly.
- 16: The event is forwarded to the application.
- 17: The application must free the call related resources in the gateway by calling deassignCall.

## 6.1.5 Number Translation 1

The following sequence diagram shows a simple number translation service, initiated as a result of a prearranged event being received by the call control service.

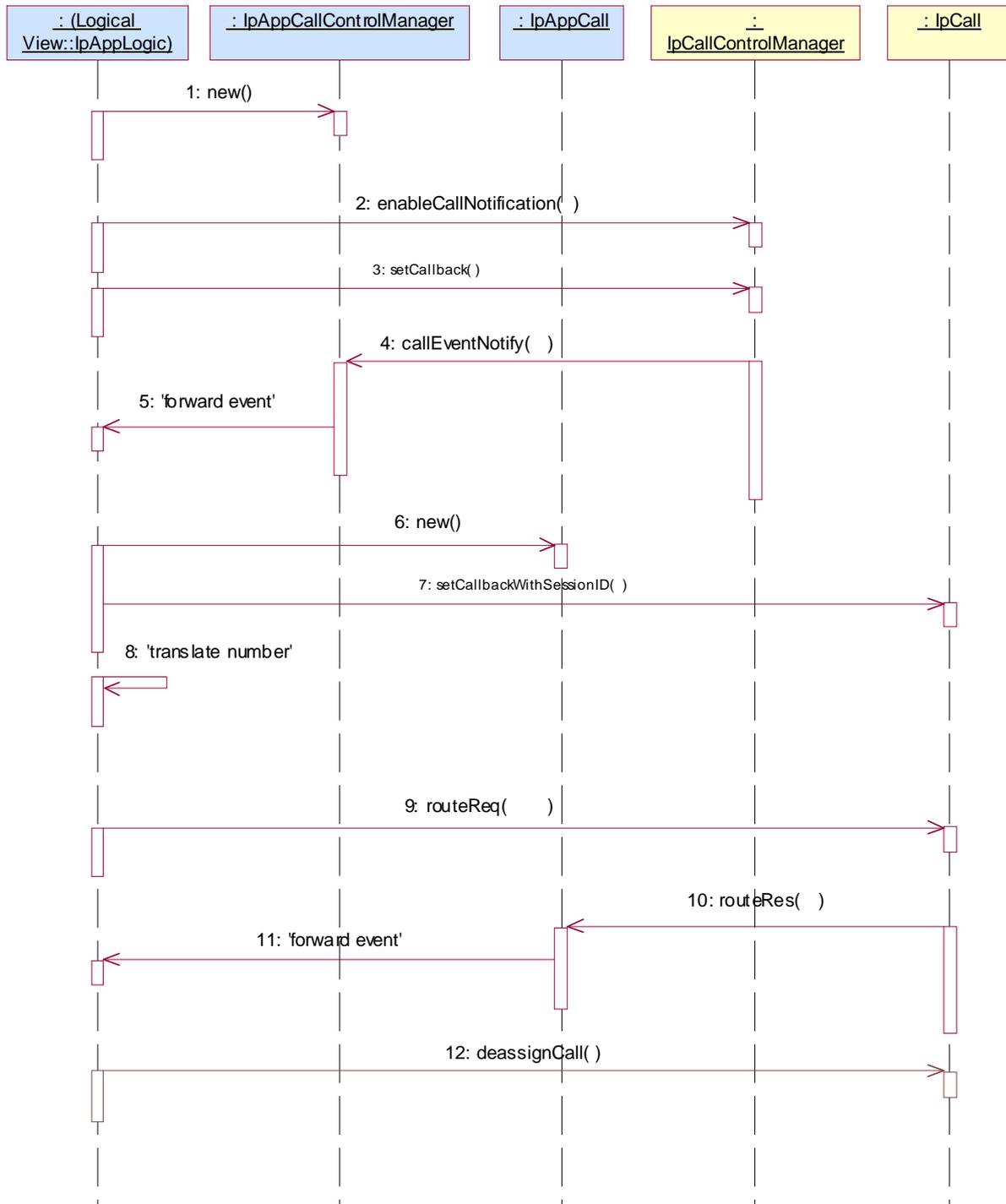


- 1: This message is used by the application to create an object implementing the IpAppCallControlManager interface.
- 2: This message is sent by the application to enable notifications on new call events. As this sequence diagram depicts a number translation service, it is likely that only new call events within a certain address range will be enabled. When a new call, that matches the event criteria set in message 2, arrives a message (not shown) is directed to the object implementing the IpCallControlManager. Assuming that the criteria for creating an object implementing the IpCall interface (e.g. load control values not exceeded) is met, other messages (not shown) are used to create the call and associated call leg object.
- 3: This message is used to pass the new call event to the object implementing the IpAppCallControlManager interface.
- 4: This message is used to forward message 3 to the IpAppLogic.
- 5: This message is used by the application to create an object implementing the IpAppCall interface. The reference to this object is passed back to the object implementing the IpCallControlManager using the return parameter of message 3.
- 6: This message invokes the number translation function.
- 7: The returned translated number is used in message 7 to route the call towards the destination.
- 8: This message passes the result of the call being answered to its callback object.
- 9: This message is used to forward the previous message to the IpAppLogic.
- 10: The application is no longer interested in controlling the call and therefore deassigns the call. The call will continue in the network, but there will be no further communication between the call object and the application.

## 6.1.6 Number Translation 1 (with callbacks)

The following sequence diagram shows a simple number translation service, initiated as a result of a prearranged event being received by the call control service.

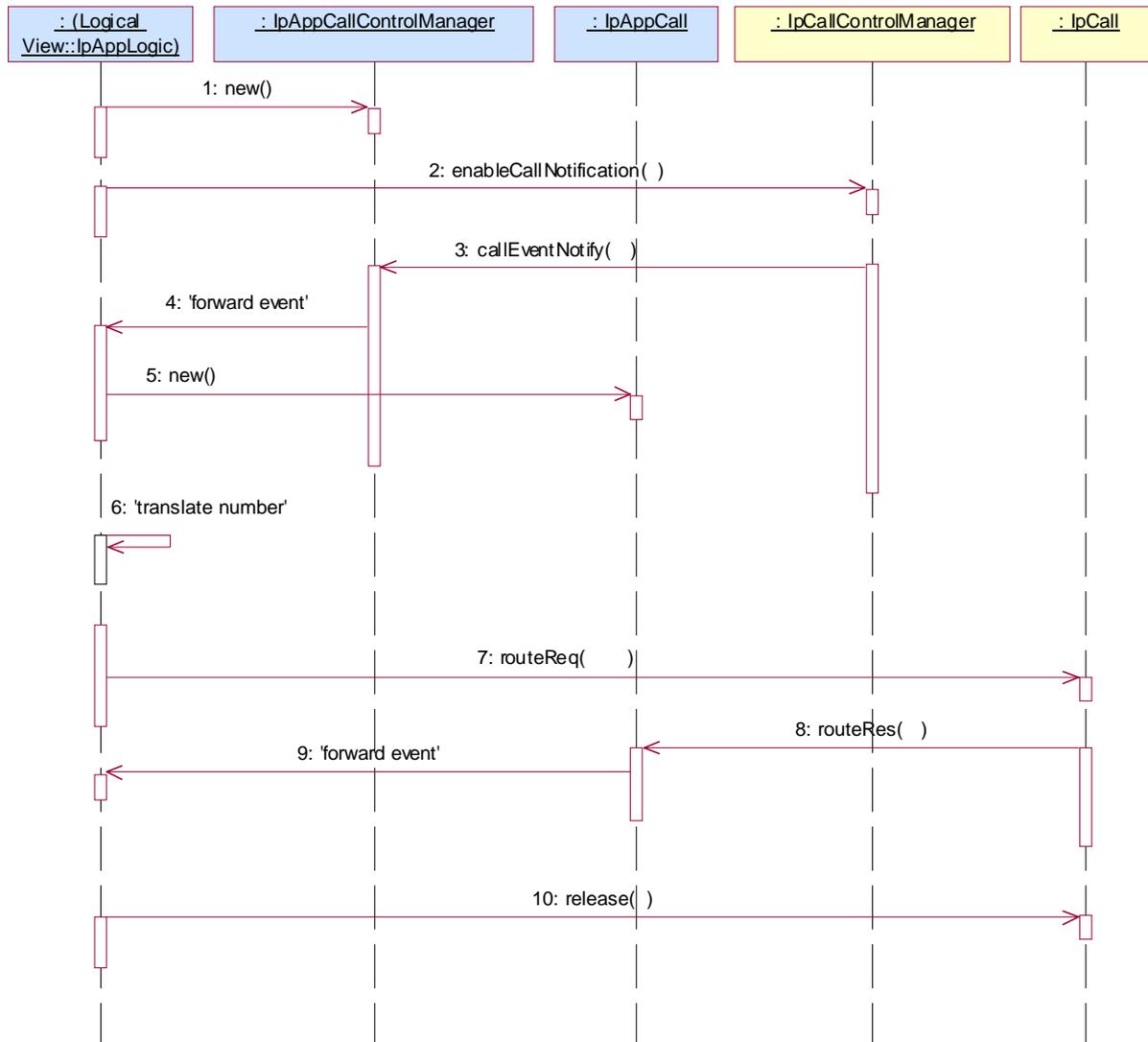
For illustration, in this sequence the callback references are set explicitly. This is optional. All the callbacks references can also be passed in other methods. From an efficiency point of view that is also the preferred method. The rest of the sequences use that mechanism.



- 1: This message is used by the application to create an object implementing the IpAppCallControlManager interface.
- 2: This message is sent by the application to enable notifications on new call events. As this sequence diagram depicts a number translation service, it is likely that only new call events within a certain address range will be enabled. When a new call, that matches the event criteria set in message 2, arrives a message (not shown) is directed to the object implementing the IpCallControlManager. Assuming that the criteria for creating an object implementing the IpCall interface (e.g. load control values not exceeded) is met, other messages (not shown) are used to create the call and associated call leg object.
- 3: This message sets the reference of the IpAppCallControlManager object in the CallControlManager. The CallControlManager reports the callEventNotify to referenced object only for enableCallNotifications that do not have an explicit IpAppCallControlManager reference specified in the enableCallNotification.
- 4: This message is used to pass the new call event to the object implementing the IpAppCallControlManager interface.
- 5: This message is used to forward message 4 to the IpAppLogic.
- 6: This message is used by the application to create an object implementing the IpAppCall interface.
- 7: This message is used to set the reference to the IpAppCall for this call.
- 8: This message invokes the number translation function.
- 9: The returned translated number is used in message 7 to route the call towards the destination.
- 10: This message passes the result of the call being answered to its callback object.
- 11: This message is used to forward the previous message to the IpAppLogic.
- 12: The application is no longer interested in controlling the call and therefore deassigns the call. The call will continue in the network, but there will be no further communication between the call object and the application.

## 6.1.7 Number Translation 2

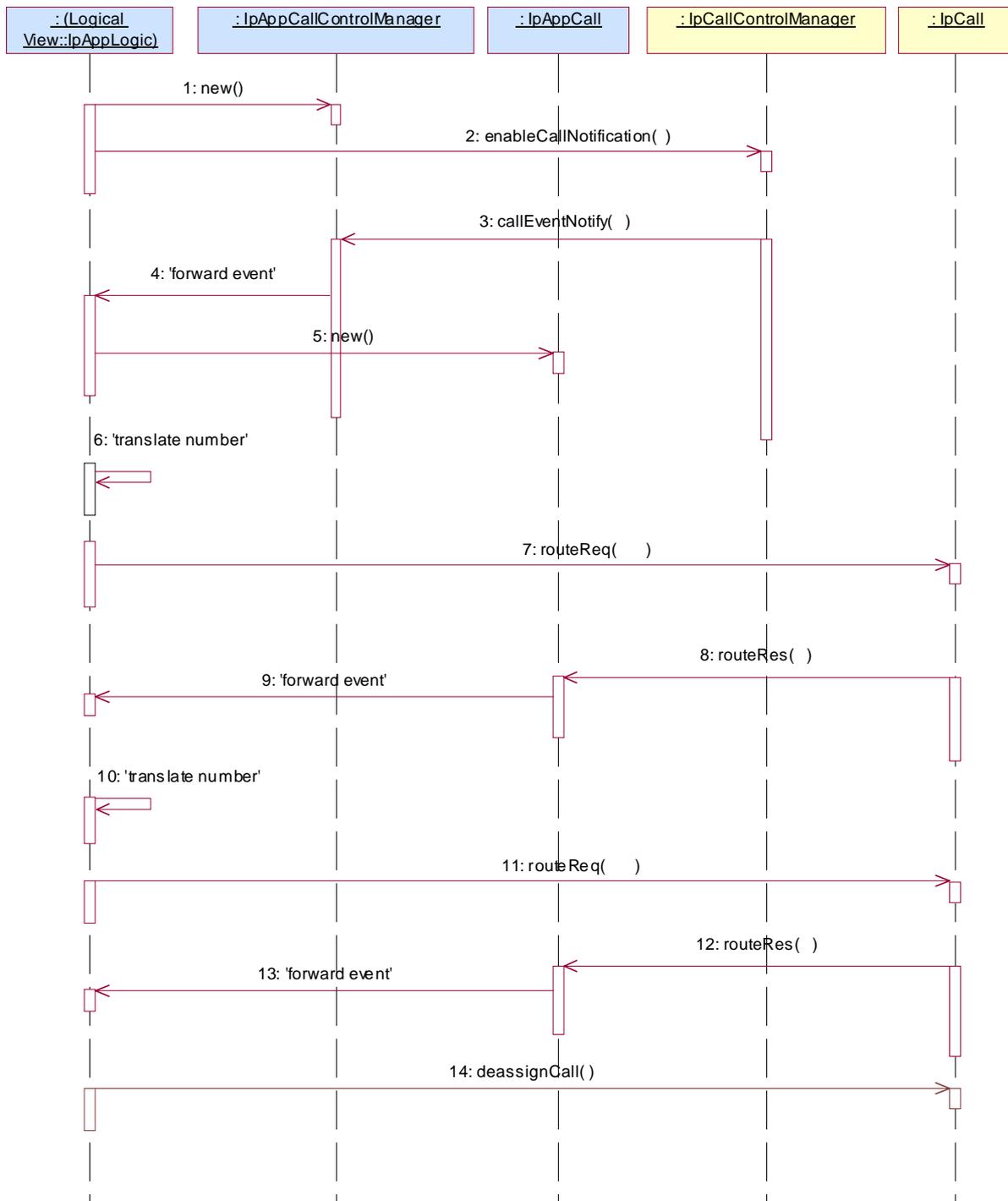
The following sequence diagram shows a number translation service, initiated as a result of a prearranged event being received by the call control service. If the translated number being routed to does not answer or is busy then the call is automatically released.



- 1: This message is used by the application to create an object implementing the IpAppCallControlManager interface.
- 2: This message is sent by the application to enable notifications on new call events. As this sequence diagram depicts a number translation service, it is likely that only new call events within a certain address range will be enabled. When a new call, that matches the event criteria, arrives a message (not shown) is directed to the object implementing the IpCallControlManager. Assuming that the criteria for creating an object implementing the IpCall interface (e.g. load control values not exceeded) is met, other messages (not shown) are used to create the call and associated call leg object.
- 3: This message is used to pass the new call event to the object implementing the IpAppCallControlManager interface.
- 4: This message is used to forward the previous message to the IpAppLogic.
- 5: This message is used by the application to create an object implementing the IpAppCall interface. The reference to this object is passed back to the object implementing the IpCallControlManager using the return parameter of the callEventNotify.
- 6: This message invokes the number translation function.
- 7: The returned translated number is used to route the call towards the destination.
- 8: Assuming the called party is busy or does not answer, the object implementing the IpCall interface sends a callback in this message, indicating the unavailability of the called party.
- 9: This message is used to forward the previous message to the IpAppLogic.
- 10: The application takes the decision to release the call.

### 6.1.8 Number Translation 3

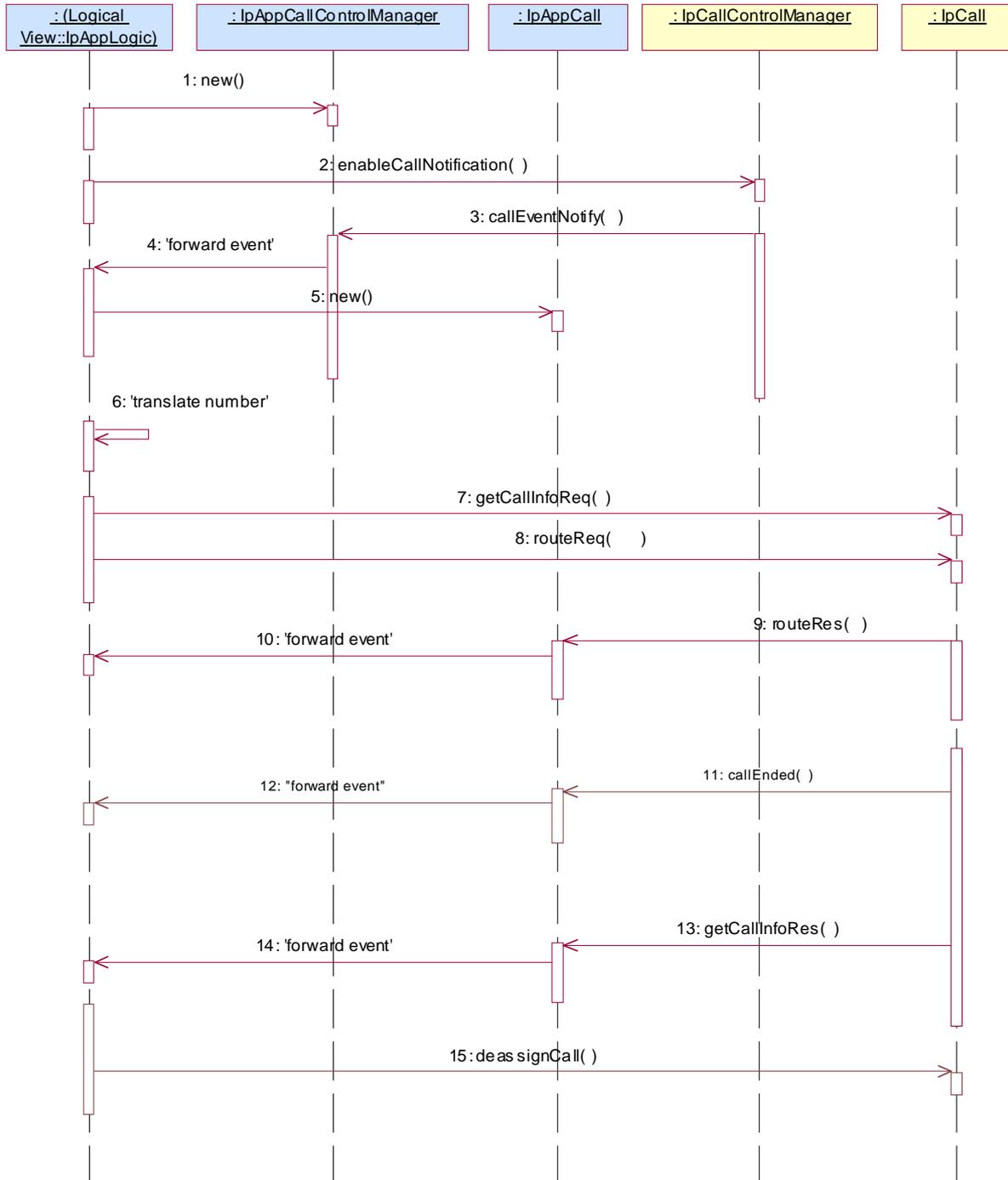
The following sequence diagram shows a number translation service, initiated as a result of a prearranged event being received by the call control service. If the translated number being routed to does not answer or is busy then the call is automatically routed to a voice mailbox.



- 1: This message is used by the application to create an object implementing the IpAppCallControlManager interface.
- 2: This message is sent by the application to enable notifications on new call events. As this sequence diagram depicts a number translation service, it is likely that only new call events within a certain address range will be enabled. When a new call, that matches the event criteria, arrives a message (not shown) is directed to the object implementing the IpCallControlManager. Assuming that the criteria for creating an object implementing the IpCall interface (e.g. load control values not exceeded) is met, other messages (not shown) are used to create the call and associated call leg object.
- 3: This message is used to pass the new call event to the object implementing the IpAppCallControlManager interface.
- 4: This message is used to forward the previous message to the IpAppLogic.
- 5: This message is used by the application to create an object implementing the IpAppCall interface. The reference to this object is passed back to the object implementing the IpCallControlManager using the return parameter of the callEventNotify.
- 6: This message invokes the number translation function.
- 7: The returned translated number is used to route the call towards the destination.
- 8: Assuming the called party is busy or does not answer, the object implementing the IpCall interface sends a callback, indicating the unavailability of the called party.
- 9: This message is used to forward the previous message to the IpAppLogic.
- 10: The application takes the decision to translate the number, but this time the number is translated to a number belonging to a voice mailbox system.
- 11: This message routes the call towards the voice mailbox.
- 12: This message passes the result of the call being answered to its callback object.
- 13: This message is used to forward the previous message to the IpAppLogic.
- 14: The application is no longer interested in controlling the call and therefore deassigns the call. The call will continue in the network, but there will be no further communication between the call object and the application.

## 6.1.9 Number Translation 4

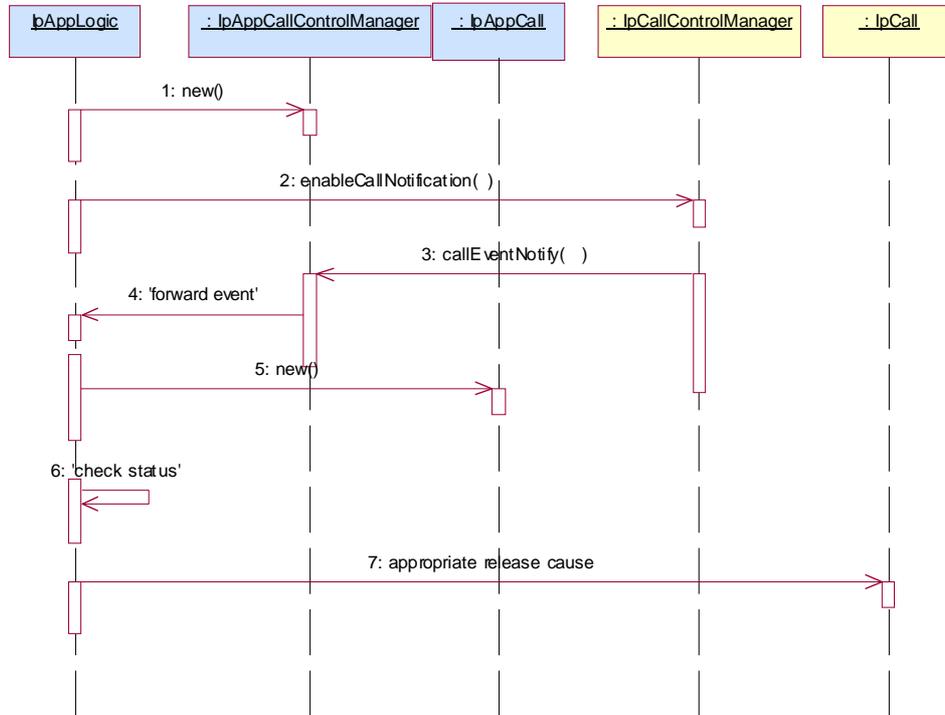
The following sequence diagram shows a number translation service, initiated as a result of a prearranged event being received by the call control service. Before the call is routed to the translated number, the application requests for all call related information to be delivered back to the application on completion of the call.



- 1: This message is used by the application to create an object implementing the IpAppCallControlManager interface.
- 2: This message is sent by the application to enable notifications on new call events. As this sequence diagram depicts a number translation service, it is likely that only new call events within a certain address range will be enabled. When a new call, that matches the event criteria, arrives a message (not shown) is directed to the object implementing the IpCallControlManager. Assuming that the criteria for creating an object implementing the IpCall interface (e.g. load control values not exceeded) is met, other messages (not shown) are used to create the call and associated call leg object.
- 3: This message is used to pass the new call event to the object implementing the IpAppCallControlManager interface.
- 4: This message is used to forward the previous message to the IpAppLogic.
- 5: This message is used by the application to create an object implementing the IpAppCall interface. The reference to this object is passed back to the object implementing the IpCallControlManager using the return parameter of the callEventNotify.
- 6: This message invokes the number translation function.
- 7: The application instructs the object implementing the IpCall interface to return all call related information once the call has been released.
- 8: The returned translated number is used to route the call towards the destination.
- 9: This message passes the result of the call being answered to its callback object.
- 10: This message is used to forward the previous message to the IpAppLogic.
- 11: Towards the end of the call, when one of the parties disconnects, a message (not shown) is directed to the object implementing the IpCall. This causes an event, to be passed to the object implementing the IpAppCall object.
- 12: This message is used to forward the previous message to the IpAppLogic.
- 13: The application now waits for the call information to be sent. Now that the call has completed, the object implementing the IpCall interface passes the call information to its callback object.
- 14: This message is used to forward the previous message to the IpAppLogic.
- 15: After the last information is received, the application deassigns the call. This will free the resources related to this call in the gateway.

## 6.1.10 Number Translation 5

The following sequence diagram shows a simple number translation service which contains a status check function, initiated as a result of a prearranged event being received. In the following sequence, when the application receives an incoming call, it checks the status of the user, and returns a busy code to the calling party.



- 1: This message is used by the application to create an object implementing the IpAppCallControlManager interface.
- 2: This message is sent by the application to enable notifications on new call events. As this sequence diagram depicts a number translation service, it is likely that only new call events within a certain address range will be enabled.

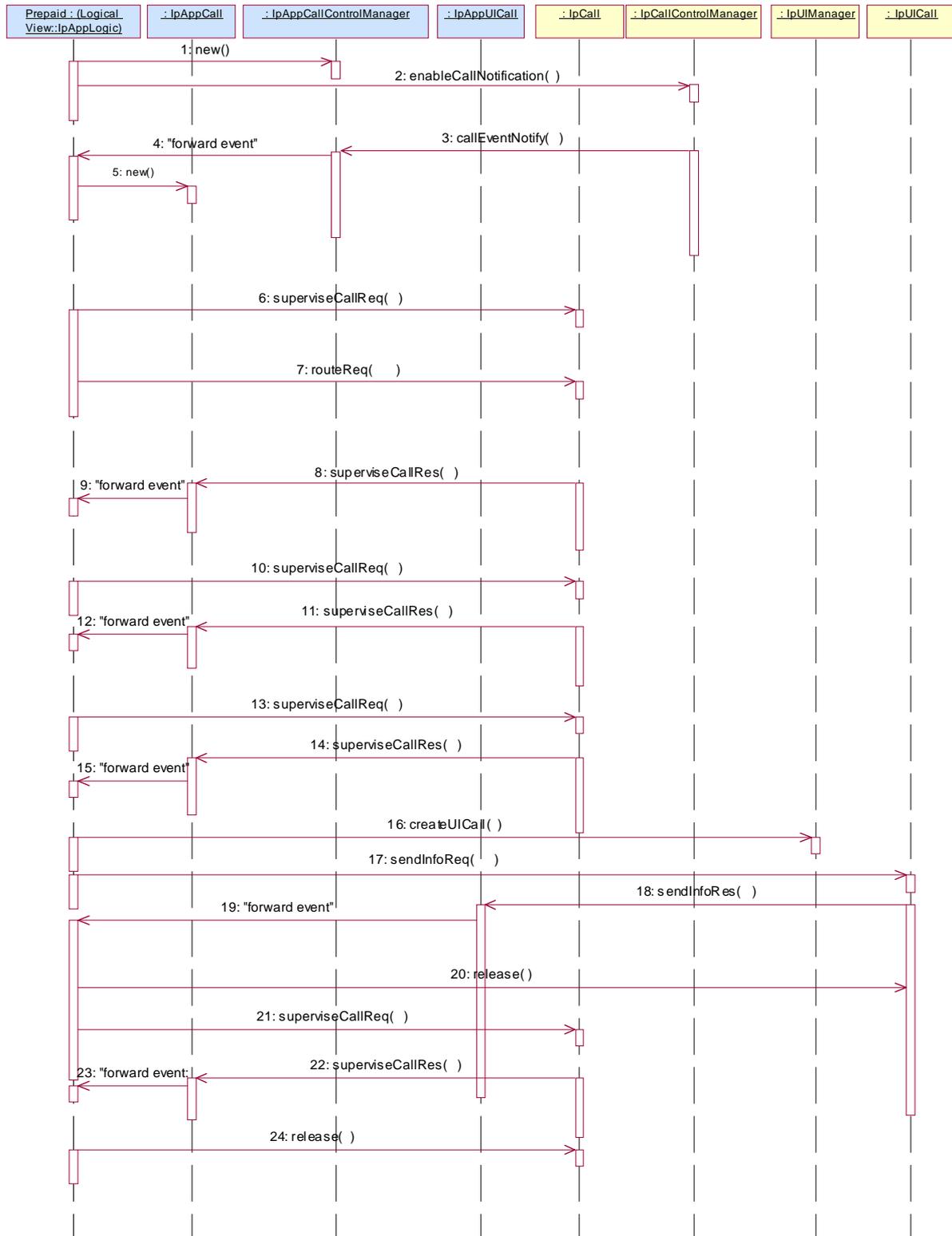
When a new call, that matches the event criteria set in message 2, arrives a message (not shown) is directed to the object implementing the IpCallControlManager. Assuming that the criteria for creating an object implementing the IpCall interface (e.g. load control values not exceeded) is met, other messages (not shown) are used to create the call and associated call leg object.

- 3: This message is used to pass the new call event to the object implementing the IpAppCallControlManager interface.
- 4: This message is used to forward message 3 to the IpAppLogic.
- 5: This message is used by the application to create an object implementing the IpAppCall interface. The reference to this object is passed back to the object implementing the IpCallControlManager using the return parameter of message 3.
- 6: This message invokes the status checking function.
- 7: The application decides to release the call, and sends a release cause to the calling party indicating that the user is busy.

## 6.1.11 Prepaid

This sequence shows a Pre-paid application.

The subscriber is using a pre-paid card or credit card to pay for the call. The application each time allows a certain timeslice for the call. After the timeslice, a new timeslice can be started or the application can terminate the call. In the following sequence the end-user will receive an announcement before his final timeslice.



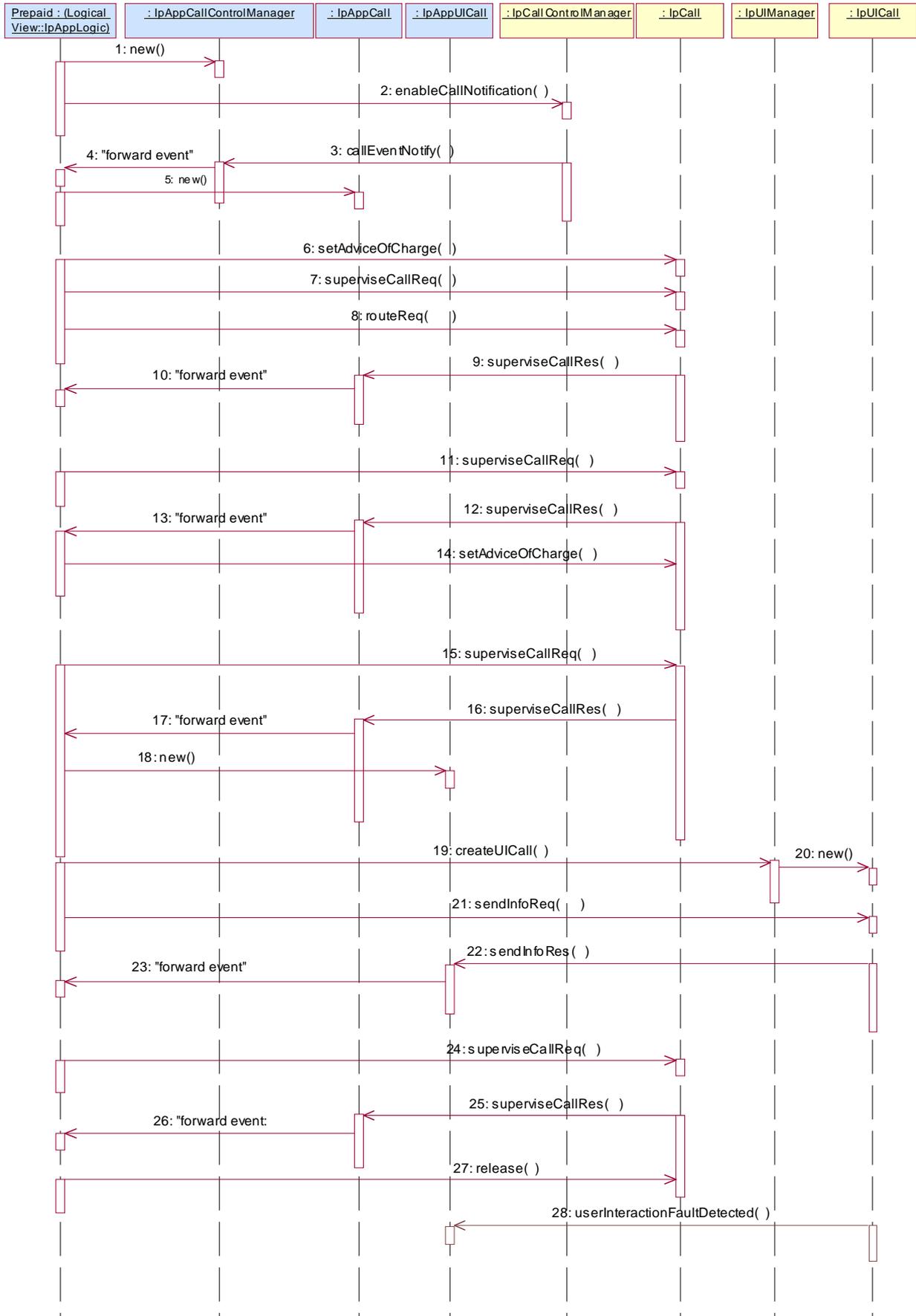
- 1: This message is used by the application to create an object implementing the IpAppCallControlManager interface.
- 2: This message is sent by the application to enable notifications on new call events. As this sequence diagram depicts a pre-paid service, it is likely that only new call events within a certain address range will be enabled. When a new call, that matches the event criteria, arrives a message (not shown) is directed to the object implementing the IpCallControlManager. Assuming that the criteria for creating an object implementing the IpCall interface (e.g. load control values not exceeded) is met, other messages (not shown) are used to create the call and associated call leg object.
- 3: The incoming call triggers the Pre-Paid Application (PPA).
- 4: The message is forwarded to the application.
- 5: A new object on the application side for the Generic Call object is created.
- 6: The Pre-Paid Application (PPA) requests to supervise the call. The application will be informed after the period indicated in the message. This period is related to the credits left on the account of the pre-paid subscriber.
- 7: Before continuation of the call, PPA sends all charging information, a possible tariff switch time and the call duration supervision period, towards the GW which forwards it to the network.
- 8: At the end of each supervision period the application is informed and a new period is started.
- 9: The message is forwarded to the application.
- 10: The Pre-Paid Application (PPA) requests to supervise the call for another call duration.
- 11: At the end of each supervision period the application is informed and a new period is started.
- 12: The message is forwarded to the application.
- 13: The Pre-Paid Application (PPA) requests to supervise the call for another call duration. When the timer expires it will indicate that the user is almost out of credit.
- 14: When the user is almost out of credit the application is informed.
- 15: The message is forwarded to the application.
- 16: The application decides to play an announcement to the parties in this call. A new UICall object is created and associated with the call.
- 17: An announcement is played informing the user about the near-expiration of his credit limit.
- 18: When the announcement is completed the application is informed.
- 19: The message is forwarded to the application.
- 20: The application releases the UICall object.
- 21: The user does not terminate so the application terminates the call after the next supervision period.
- 22: The supervision period ends.
- 23: The event is forwarded to the logic.
- 24: The application terminates the call. Since the user interaction is already explicitly terminated no userInteractionFaultDetected is sent to the application.

## 6.1.12 Pre-Paid with Advice of Charge (AoC)

This sequence shows a Pre-paid application that uses the Advice of Charge feature.

The application will send the charging information before the actual call setup and when during the call the charging changes new information is sent in order to update the end-user.

NOTE: The Advice of Charge feature requires an application in the end-user terminal to display the charges for the call, depending on the information received from the application.

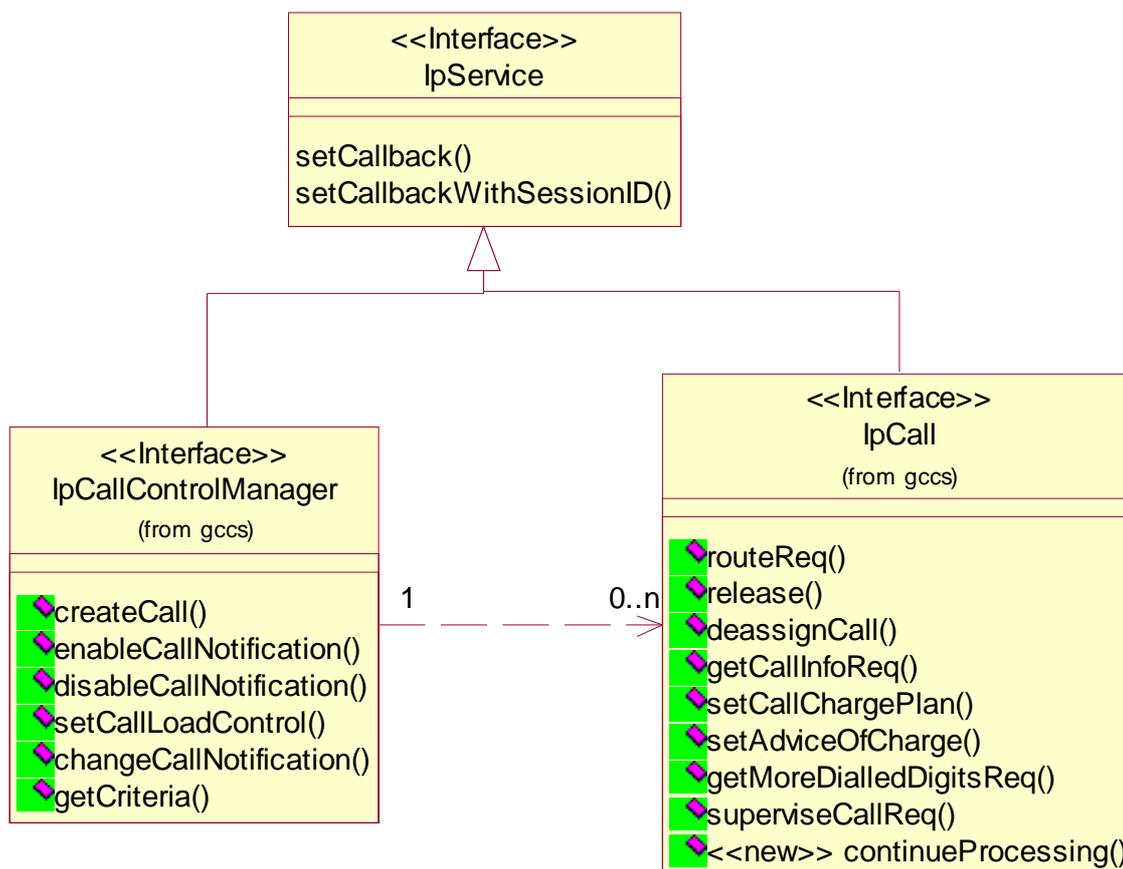


- 1: This message is used by the application to create an object implementing the IpAppCallControlManager interface.
- 2: This message is sent by the application to enable notifications on new call events. As this sequence diagram depicts a pre-paid service, it is likely that only new call events within a certain address range will be enabled. When a new call, that matches the event criteria, arrives a message (not shown) is directed to the object implementing the IpCallControlManager. Assuming that the criteria for creating an object implementing the IpCall interface (e.g. load control values not exceeded) is met, other messages (not shown) are used to create the call and associated call leg object.
- 3: The incoming call triggers the Pre-Paid Application (PPA).
- 4: The message is forwarded to the application.
- 5: A new object on the application side for the Call object is created.
- 6: The Pre-Paid Application (PPA) sends the AoC information (e.g. the tariff switch time). (it shall be noted the PPA contains ALL the tariff information and knows how to charge the user).  
  
During this call sequence 2 tariff changes take place. The call starts with tariff 1, and at the tariff switch time (e.g. 18:00 hours) switches to tariff 2. The application is not informed about this (but the end-user is!).
- 7: The Pre-Paid Application (PPA) requests to supervise the call. The application will be informed after the period indicated in the message. This period is related to the credits left on the account of the pre-paid subscriber.
- 8: The application requests to route the call to the destination address.
- 9: At the end of each supervision period the application is informed and a new period is started.
- 10: The message is forwarded to the application.
- 11: The Pre-Paid Application (PPA) requests to supervise the call for another call duration.
- 12: At the end of each supervision period the application is informed and a new period is started.
- 13: The message is forwarded to the application.
- 14: Before the next tariff switch (e.g. 19:00 hours) the application sends a new AOC with the tariff switch time. Again, at the tariff switch time, the network will send AoC information to the end-user.
- 15: The Pre-Paid Application (PPA) requests to supervise the call for another call duration. When the timer expires it will indicate that the user is almost out of credit.
- 16: When the user is almost out of credit the application is informed.
- 17: The message is forwarded to the application.
- 18: The application creates a new call back interface for the User interaction messages.
- 19: A new UI Call object that will handle playing of the announcement needs to be created.
- 20: The Gateway creates a new UI call object that will handle playing of the announcement.
- 21: With this message the announcement is played to the parties in the call.
- 22: The user indicates that the call should continue.
- 23: The message is forwarded to the application.
- 24: The user does not terminate so the application terminates the call after the next supervision period.
- 25: The user is out of credit and the application is informed.
- 26: The message is forwarded to the application.
- 27: With this message the application requests to release the call.

- 28: Terminating the call which has still a UICall object associated will result in a userInteractionFaultDetected. The UICall object is terminated in the gateway and no further communication is possible between the UICall and the application.

## 6.2 Class Diagrams

This class diagram shows the interfaces of the generic call control service package.



**Figure 1: Service Interfaces**

The generic call control service consists of two packages, one for the interfaces on the application side and one for interfaces on the service side.

The class diagrams in the following figures show the interfaces that make up the generic call control application package and the generic call control service package. Communication between these packages is indicated with the <<uses>> associations; e.g. the IpCallControlManager interface uses the IpAppCallControlManager, by means of calling callback methods.

This class diagram shows the interfaces of the generic call control application package and their relations to the interfaces of the generic call control service package.

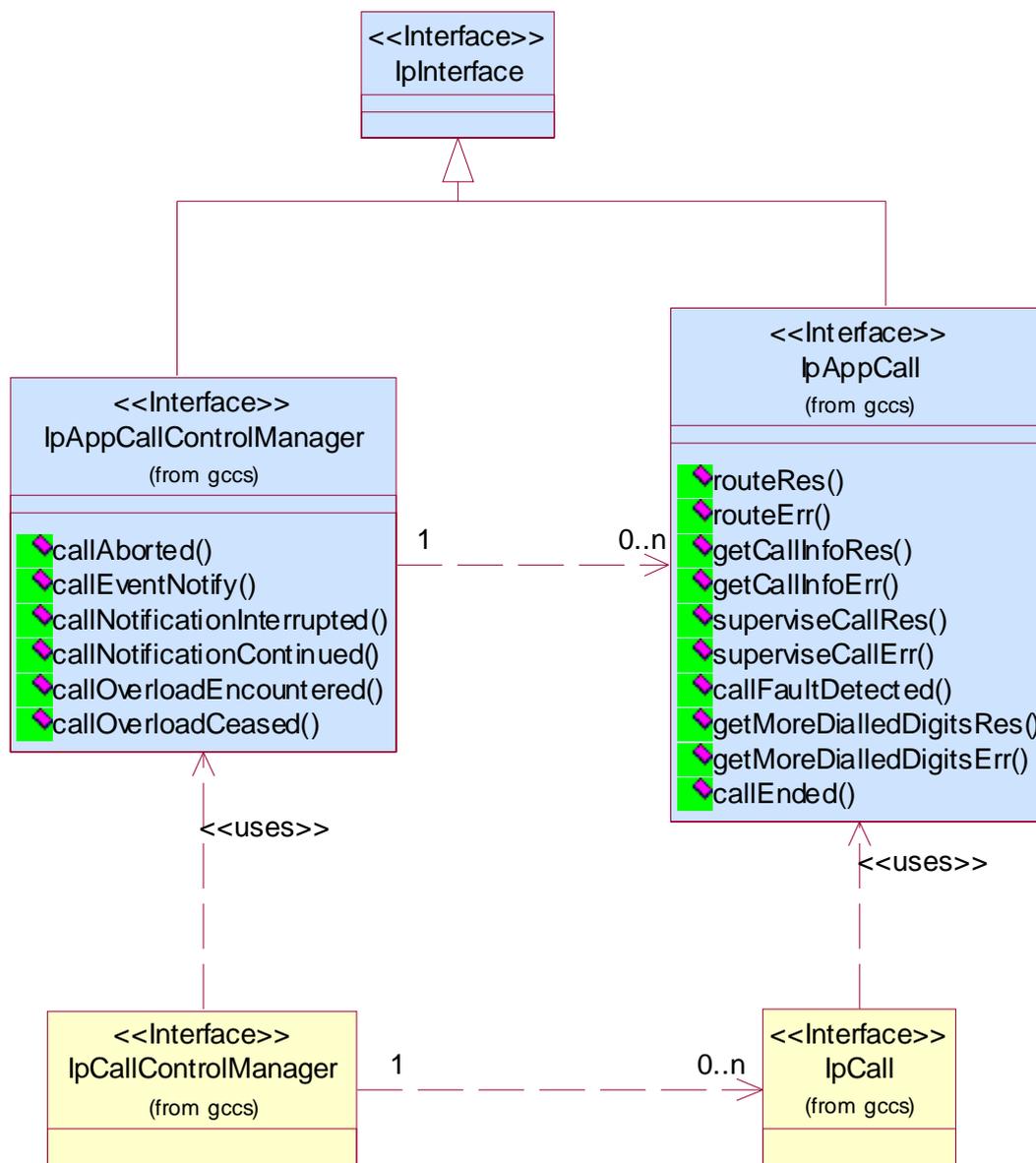


Figure 2: Application Interfaces

## 6.3 Generic Call Control Service Interface Classes

The Generic Call Control Service (GCCS) provides the basic call control service for the API. It is based around a third party model, which allows calls to be instantiated from the network and routed through the network.

The GCCS supports enough functionality to allow call routing and call management for today's Intelligent Network (IN) services in the case of a switched telephony network, or equivalent for packet based networks.

It is the intention of the GCCS that it could be readily specialised into call control specifications, for example, ITU-T Recommendations H.323, ISUP, Q.931 and Q.2931, ATM Forum specification UNI3.1 and the IETF Session Initiation Protocol, or any other call control technology.

For the generic call control service, only a subset of the call model defined in clause 4 is used; the API for generic call control does not give explicit access to the legs and the media channels. This is provided by the Multi-Party Call Control Service. Furthermore, the generic call is restricted to two party calls, i.e. only two legs are active at any given time. Active is defined here as "being routed" or connected.

The GCCS is represented by the IpCallControlManager and IpCall interfaces that interface to services provided by the network. Some 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. To handle responses and reports, the developer must implement IpAppCallControlManager and IpAppCall to provide the callback mechanism.

### 6.3.1 Interface Class IpCallControlManager

Inherits from: IpService.

This interface is the "service manager" interface for the Generic Call Control Service. The generic call control manager interface provides the management functions to the generic call control service. The application programmer can use this interface to provide overload control functionality, create call objects and to enable or disable call-related event notifications.

This interface shall be implemented by a Generic Call Control SCF. As a minimum requirement either the createCall() method shall be implemented, or the enableCallNotification() and disableCallNotification() methods shall be implemented.

<<Interface>> IpCallControlManager
createCall (appCall : in IpAppCallRef) : TpCallIdentifier enableCallNotification (appCallControlManager : in IpAppCallControlManagerRef, eventCriteria : in TpCallEventCriteria) : TpAssignmentID disableCallNotification (assignmentID : in TpAssignmentID) : void setCallLoadControl (duration : in TpDuration, mechanism : in TpCallLoadControlMechanism, treatment : in TpCallTreatment, addressRange : in TpAddressRange) : TpAssignmentID changeCallNotification (assignmentID : in TpAssignmentID, eventCriteria : in TpCallEventCriteria) : void getCriteria () : TpCallEventCriteriaResultSet

#### *Method*

#### **createCall()**

This method is used to create a new call object.

Call back reference:

An IpAppCallControlManager should already have been passed to the IpCallControlManager, otherwise the call control will not be able to report a callAborted() to the application. The application should invoke setCallback() prior to createCall if it wishes to ensure this.

Returns callReference: Specifies the interface reference and sessionID of the call created.

#### *Parameters*

**appCall : in IpAppCallRef**

Specifies the application interface for callbacks from the call created.

#### *Returns*

**TpCallIdentifier**

*Raises***TpCommonExceptions, P\_INVALID\_INTERFACE\_TYPE***Method***enableCallNotification()**

This method is used to enable call notifications so that events can be sent to the application. This is the first step an application has to do to get initial notification of calls happening in the network. When such an event happens, the application will be informed by callEventNotify(). In case the application is interested in other events during the context of a particular call session it has to use the routeReq() method on the call object. The application will get access to the call object when it receives the callEventNotify(). (Note that the enableCallNotification() is not applicable if the call is setup by the application).

The enableCallNotification method is purely intended for applications to indicate their interest to be notified when certain call events take place. It is possible to subscribe to a certain event for a whole range of addresses, e.g. the application can indicate it wishes to be informed when a call is made to any number starting with 800.

If some application already requested notifications with criteria that overlap the specified criteria, the request is refused with P\_GCCS\_INVALID\_CRITERIA. The criteria are said to overlap if both originating and terminating ranges overlap and the same number plan is used and the same CallNotificationType is used.

If a notification is requested by an application with the monitor mode set to notify, then there is no need to check the rest of the criteria for overlapping with any existing request as the notify mode does not allow control on a call to be passed over. Only one application can place an interrupt request if the criteria overlaps.

Set of the callback reference:

The call back reference can be registered either in a) enableCallNotification() or b) explicitly with a separate setCallback() method depending on how the application provides its callback reference.

Case a:

From an efficiency point of view the enableCallNotification() with explicit immediate registration (no "Null" value) of call back reference may be the preferred method.

Case b:

The enableCallNotification() with no call back reference ("Null" value) is used where (e.g. due to distributed application logic) the call back reference is provided subsequently in a setCallback().

In case the enableCallNotification() contains no callback, at the moment the application needs to be informed the gateway will use as callback the callback that has been registered by setCallback(). See example in clause 6.1.6.

Set additional callback reference:

If the same application requests two notifications with exactly the same criteria but different callback references, the second callback will be treated as an additional callback. Both notifications will share the same assignmentID. The gateway will always use the most recent callback. In case this most recent callback fails the second most recent is used. See examples in clause 6.1.1.

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

*Parameters*

**appCallControlManager : in IpAppCallControlManagerRef**

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 : in TpCallEventCriteria**

Specifies the event specific criteria used by the application to define the event required. Only events that meet these criteria are reported. Examples of events are "incoming call attempt reported by network", "answer", "no answer", "busy". Individual addresses or address ranges may be specified for destination and/or origination.

*Returns***TpAssignmentID***Raises***TpCommonExceptions, P\_INVALID\_CRITERIA, P\_INVALID\_INTERFACE\_TYPE, P\_INVALID\_EVENT\_TYPE***Method***disableCallNotification()**

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

*Parameters***assignmentID : in TpAssignmentID**

Specifies the assignment ID given by the generic call control manager interface when the previous enableCallNotification() was called. If the assignment ID does not correspond to one of the valid assignment IDs, the exception P\_INVALID\_ASSIGNMENTID will be raised. If two callbacks have been registered under this assignment ID both of them will be disabled.

*Raises***TpCommonExceptions, P\_INVALID\_ASSIGNMENT\_ID***Method***setCallLoadControl()**

This method imposes or removes load control on calls made to a particular address range within the generic call control service. The address matching mechanism is similar as defined for TpCallEventCriteria.

Returns assignmentID: Specifies the assignmentID assigned by the gateway to this request. This assignmentID can be used to correlate the callOverloadEncountered and callOverloadCeased methods with the request.

*Parameters***duration : in TpDuration**

Specifies the duration for which the load control should be set.

A duration of 0 indicates that the load control should be removed.

A duration of -1 indicates an infinite duration (i.e. until disabled by the application).

A duration of -2 indicates the network default duration.

**mechanism : in TpCallLoadControlMechanism**

Specifies the load control mechanism to use (for example, admit one call per interval), and any necessary parameters, such as the call admission rate. The contents of this parameter are ignored if the load control duration is set to zero.

**treatment : in TpCallTreatment**

Specifies the treatment of calls that are not admitted. The contents of this parameter are ignored if the load control duration is set to zero.

**addressRange : in TpAddressRange**

Specifies the address or address range to which the overload control should be applied or removed.

*Returns***TpAssignmentID**

*Raises*

**TpCommonExceptions, P\_INVALID\_ADDRESS, P\_UNSUPPORTED\_ADDRESS\_PLAN**

*Method***changeCallNotification()**

This method is used by the application to change the event criteria introduced with enableCallNotification. Any stored criteria associated with the specified assignmentID will be replaced with the specified criteria.

*Parameters*

**assignmentID : in TpAssignmentID**

Specifies the ID assigned by the generic call control manager interface for the event notification. If two call backs have been registered under this assignment ID both of them will be changed.

**eventCriteria : in TpCallEventCriteria**

Specifies the new set of event specific criteria used by the application to define the event required. Only events that meet these criteria are reported.

*Raises*

**TpCommonExceptions, P\_INVALID\_ASSIGNMENT\_ID, P\_INVALID\_CRITERIA, P\_INVALID\_EVENT\_TYPE**

*Method***getCriteria()**

This method is used by the application to query the event criteria set with enableCallNotification or changeCallNotification.

Returns eventCriteria: Specifies the event specific criteria used by the application to define the event required. Only events that meet these criteria are reported.

*Parameters*

No Parameters were identified for this method.

*Returns*

**TpCallEventCriteriaResultSet**

*Raises*

**TpCommonExceptions**

## 6.3.2 Interface Class IpAppCallControlManager

Inherits from: IpInterface.

The generic call control manager application interface provides the application call control management functions to the generic call control service.

<<Interface>> IpAppCallControlManager
callAborted (callReference : in TpSessionID) : void callEventNotify (callReference : in TpCallIdentifier, eventInfo : in TpCallEventInfo, assignmentID : in TpAssignmentID) : IpAppCallRef callNotificationInterrupted () : void callNotificationContinued () : void callOverloadEncountered (assignmentID : in TpAssignmentID) : void callOverloadCeased (assignmentID : in TpAssignmentID) : void

### *Method*

#### **callAborted()**

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

### *Parameters*

**callReference : in TpSessionID**

Specifies the sessionID of call that has aborted or terminated abnormally.

### *Method*

#### **callEventNotify()**

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

If this method is invoked with a monitor mode of P\_CALL\_MONITOR\_MODE\_INTERRUPT, then the APL has control of the call. If the APL does nothing with the call (including its associated legs) within a specified time period (the duration of which forms a part of the service level agreement), then the call in the network shall be released and callEnded() shall be invoked, giving a release cause of 102 (Recovery on timer expiry).

Set of the callback reference:

A reference to the application interface has to be passed back to the call interface to which the notification relates. However, the setting of a call back reference is only applicable if the notification is in INTERRUPT mode.

When callEventNotify() is invoked with a monitor mode of P\_CALL\_MONITOR\_MODE\_INTERRUPT, the application writer should ensure that no continue processing e.g. routeReq() is performed until an IpAppCall has been passed to the gateway, either through an explicit setCallbackWithSessionID() invocation on the supplied IpCall, or via the return of the callEventNotify() method.

The call back reference can be registered either in a) callEventNotify() or b) explicitly with a setCallbackWithSessionID() method e.g. depending on how the application provides its call reference.

Case a:

From an efficiency point of view the callEventNotify() with explicit pass of registration may be the preferred method.

Case b:

The callEventNotify() with no call back reference ("Null" value) is used where (e.g. due to distributed application logic) the callback reference is provided subsequently in a setCallbackWithSessionID().

In case the `callEventNotify()` contains no callback, at the moment the application needs to be informed the gateway will use as callback the callback that has been registered by `setCallbackWithSessionID()`. See example in clause 6.1.6.

Returns `appCall`: Specifies a reference to the application interface which implements the callback interface for the new call. If the application has previously explicitly passed a reference to the `IpAppCall` interface using a `setCallbackWithSessionID()` invocation, this parameter may be null, or if supplied must be the same as that provided during the `setCallbackWithSessionID()`.

This parameter will be null if the notification is in NOTIFY mode and in case b.

### *Parameters*

#### **callReference : in TpCallIdentifier**

Specifies the reference to the call interface to which the notification relates. If the notification is in NOTIFY mode, this parameter shall be ignored by the application client implementation, and consequently the implementation of the SCS entity invoking `callEventNotify` may populate this parameter as it chooses.

#### **eventInfo : in TpCallEventInfo**

Specifies data associated with this event.

#### **assignmentID : in TpAssignmentID**

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

### *Returns*

#### **IpAppCallRef**

### *Method*

#### **callNotificationInterrupted()**

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

Note that more permanent failures are reported via the Framework (integrity management).

### *Parameters*

No Parameters were identified for this method.

### *Method*

#### **callNotificationContinued()**

This method indicates to the application that event notifications will again be possible.

### *Parameters*

No Parameters were identified for this method.

### *Method*

#### **callOverloadEncountered()**

This method indicates that the network has detected overload and may have automatically imposed load control on calls requested to a particular address range or calls made to a particular destination within the call control service.

### *Parameters*

#### **assignmentID : in TpAssignmentID**

Specifies the `assignmentID` corresponding to the associated `setCallLoadControl`. This implies the address range for within which the overload has been encountered.

*Method***callOverloadCeased()**

This method indicates that the network has detected that the overload has ceased and has automatically removed any load controls on calls requested to a particular address range or calls made to a particular destination within the call control service.

*Parameters*

**assignmentID** : in **TpAssignmentID**

Specifies the assignmentID corresponding to the associated setCallLoadControl. This implies the address range for within which the overload has been ceased.

### 6.3.3 Interface Class IpCall

Inherits from: IpService.

The generic Call provides the possibility to control the call routing, to request information from the call, control the charging of the call, to release the call and to supervise the call. It does not give the possibility to control the legs directly and it does not allow control over the media. The first capability is provided by the multi-party call and the latter as well by the multi-media call. The call is limited to two party calls, although it is possible to provide "follow-on" calls, meaning that the call can be rerouted after the terminating party has disconnected or routing to the terminating party has failed. Basically, this means that at most two legs can be in connected or routing state at any time. This interface shall be implemented by a Generic Call Control SCF. As a minimum requirement, the routeReq (), release() and deassignCall() methods shall be implemented.

<<Interface>> IpCall
<pre> routeReq (callSessionID : in TpSessionID, responseRequested : in TpCallReportRequestSet, targetAddress   : in TpAddress, originatingAddress : in TpAddress, originalDestinationAddress : in TpAddress,   redirectingAddress : in TpAddress, applInfo : in TpCallAppInfoSet) : TpSessionID release (callSessionID : in TpSessionID, cause : in TpCallReleaseCause) : void deassignCall (callSessionID : in TpSessionID) : void getCallInfoReq (callSessionID : in TpSessionID, callInfoRequested : in TpCallInfoType) : void setCallChargePlan (callSessionID : in TpSessionID, callChargePlan : in TpCallChargePlan) : void setAdviceOfCharge (callSessionID : in TpSessionID, aOCInfo : in TpAoCInfo, tariffSwitch : in TpDuration) :   void getMoreDialledDigitsReq (callSessionID : in TpSessionID, length : in TpInt32) : void superviseCallReq (callSessionID : in TpSessionID, time : in TpDuration, treatment : in   TpCallSuperviseTreatment) : void &lt;&lt;new&gt;&gt; continueProcessing (callSessionID : in TpSessionID) : void           </pre>

*Method***routeReq()**

This asynchronous method requests routing of the call to the remote party indicated by the targetAddress.

Note that in case of routeReq() it is recommended to request for "successful" (e.g. "answer" event) and "failure" events at invocation, because those are needed for the application to keep track of the state of the call.

The extra address information such as `originatingAddress` is optional. If not present (i.e. the plan is set to `P_ADDRESS_PLAN_NOT_PRESENT`), the information provided in corresponding addresses from the route is used, otherwise the network or gateway provided numbers will be used.

If this method is invoked, and call reports have been requested, yet no `IpAppCall` interface has been provided, this method shall throw the `P_NO_CALLBACK_ADDRESS_SET` exception.

This operation continues processing of the call implicitly.

Returns `callLegSessionID`: Specifies the sessionID assigned by the gateway. This is the sessionID of the implicitly created call leg. The same ID will be returned in the `routeRes` or `Err`. This allows the application to correlate the request and the result.

This parameter is only relevant when multiple `routeReq()` calls are executed in parallel, e.g. in the multi-party call control service.

### *Parameters*

**`callSessionID` : in `TpSessionID`**

Specifies the call session ID of the call.

**`responseRequested` : in `TpCallReportRequestSet`**

Specifies the set of observed events that will result in zero or more `routeRes()` being generated.

E.g. when both answer and disconnect is monitored the result can be received two times.

If the application wants to control the call (in whatever sense) it shall enable event reports.

**`targetAddress` : in `TpAddress`**

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

**`originatingAddress` : in `TpAddress`**

Specifies the address of the originating (calling) party.

**`originalDestinationAddress` : in `TpAddress`**

Specifies the original destination address of the call.

**`redirectingAddress` : in `TpAddress`**

Specifies the address from which the call was last redirected.

**`appInfo` : in `TpCallAppInfoSet`**

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

### *Returns*

**`TpSessionID`**

### *Raises*

**`TpCommonExceptions`, `P_INVALID_SESSION_ID`, `P_INVALID_ADDRESS`,  
`P_UNSUPPORTED_ADDRESS_PLAN`, `P_INVALID_NETWORK_STATE`, `P_INVALID_CRITERIA`,  
`P_INVALID_EVENT_TYPE`**

*Method***release()**

This method requests the release of the call object and associated objects. The call will also be terminated in the network. If the application requested reports to be sent at the end of the call (e.g. by means of getCallInfoReq) these reports will still be sent to the application.

This operation continues processing of the call implicitly.

The application should always either release or deassign the call when it is finished with the call, unless a callFaultDetected is received by the application.

*Parameters*

**callSessionID** : in **TpSessionID**

Specifies the call session ID of the call.

**cause** : in **TpCallReleaseCause**

Specifies the cause of the release.

*Raises*

**TpCommonExceptions**, **P\_INVALID\_SESSION\_ID**, **P\_INVALID\_NETWORK\_STATE**

*Method***deassignCall()**

This method requests that the relationship between the application and the call and associated objects 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, call information reports or call Leg information reports requested, then these reports will be disabled and any related information discarded.

This operation continues processing of the call implicitly.

The application should always either release or deassign the call when it is finished with the call, unless callFaultDetected is received by the application.

*Parameters*

**callSessionID** : in **TpSessionID**

Specifies the call session ID of the call.

*Raises*

**TpCommonExceptions**, **P\_INVALID\_SESSION\_ID**

*Method***getCallInfoReq()**

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.

A report is received when the destination leg or party terminates or when the call ends. 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. In case the originating party is still available the application can still initiate a follow-on call using routeReq.

*Parameters*

**callSessionID** : in **TpSessionID**

Specifies the call session ID of the call.

**callInfoRequested : in TpCallInfoType**

Specifies the call information that is requested.

*Raises*

**TpCommonExceptions, P\_INVALID\_SESSION\_ID**

*Method*

**setCallChargePlan()**

Set an operator specific charge plan for the call.

*Parameters*

**callSessionID : in TpSessionID**

Specifies the call session ID of the call.

**callChargePlan : in TpCallChargePlan**

Specifies the charge plan to use.

*Raises*

**TpCommonExceptions, P\_INVALID\_SESSION\_ID**

*Method*

**setAdviceOfCharge()**

This method allows for advice of charge (AOC) information to be sent to terminals that are capable of receiving this information.

*Parameters*

**callSessionID : in TpSessionID**

Specifies the call session ID of the call.

**aOCInfo : in TpAoCInfo**

Specifies two sets of Advice of Charge parameter.

**tariffSwitch : in TpDuration**

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

*Raises*

**TpCommonExceptions, P\_INVALID\_SESSION\_ID**

*Method*

**getMoreDialledDigitsReq()**

This asynchronous method requests the call control service to collect further digits and return them to the application. Depending on the administered data, the network may indicate a new call to the gateway if a caller goes off-hook or dialled only a few digits. The application then gets a new call event which contains no digits or only the few dialled digits in the event data.

The application should use this method if it requires more dialled digits, e.g. to perform screening.

*Parameters***callSessionID : in TpSessionID**

Specifies the call session ID of the call.

**length : in TpInt32**

Specifies the maximum number of digits to collect.

*Raises***TpCommonExceptions, P\_INVALID\_SESSION\_ID***Method***superviseCallReq()**

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 routeReq() 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.

*Parameters***callSessionID : in TpSessionID**

Specifies the call session ID of the call.

**time : in TpDuration**

Specifies the granted time in milliseconds for the connection.

**treatment : in TpCallSuperviseTreatment**

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

*Raises***TpCommonExceptions, P\_INVALID\_SESSION\_ID***Method***<<new>> continueProcessing()**

This operation continues processing of the call explicitly. Applications can invoke this operation after call processing was interrupted due to detection of a notification or event the application subscribed its interest in.

In case the operation is invoked and call processing is not interrupted the exception P\_INVALID\_NETWORK\_STATE will be raised.

*Parameters***callSessionID : in TpSessionID**

Specifies the call session ID of the call.

*Raises***TpCommonExceptions, P\_INVALID\_SESSION\_ID, P\_INVALID\_NETWORK\_STATE**

### 6.3.4 Interface Class IpAppCall

Inherits from: IpInterface.

The generic call application interface is implemented by the client application developer and is used to handle call request responses and state reports.

<<Interface>> IpAppCall
<pre> routeRes (callSessionID : in TpSessionID, eventReport : in TpCallReport, callLegSessionID : in   TpSessionID) : void routeErr (callSessionID : in TpSessionID, errorIndication : in TpCallError, callLegSessionID : in   TpSessionID) : void getCallInfoRes (callSessionID : in TpSessionID, callInfoReport : in TpCallInfoReport) : void getCallInfoErr (callSessionID : in TpSessionID, errorIndication : in TpCallError) : void superviseCallRes (callSessionID : in TpSessionID, report : in TpCallSuperviseReport, usedTime : in   TpDuration) : void superviseCallErr (callSessionID : in TpSessionID, errorIndication : in TpCallError) : void callFaultDetected (callSessionID : in TpSessionID, fault : in TpCallFault) : void getMoreDialledDigitsRes (callSessionID : in TpSessionID, digits : in TpString) : void getMoreDialledDigitsErr (callSessionID : in TpSessionID, errorIndication : in TpCallError) : void callEnded (callSessionID : in TpSessionID, report : in TpCallEndedReport) : void           </pre>

### Method

#### **routeRes ( )**

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.).

If this method is invoked with a monitor mode of P\_CALL\_MONITOR\_MODE\_INTERRUPT, then the APL has control of the call. If the APL does nothing with the call (including its associated legs) within a specified time period (the duration of which forms a part of the service level agreement), then the call in the network shall be released and callEnded() shall be invoked, giving a release cause of 102 (Recovery on timer expiry).

### Parameters

**callSessionID : in TpSessionID**

Specifies the call session ID of the call.

**eventReport : in TpCallReport**

Specifies the result of the request to route the call to the destination party. It also includes the network event, date and time, monitoring mode and event specific information such as release cause.

**callLegSessionID : in TpSessionID**

Specifies the sessionID of the associated call leg. This corresponds to the sessionID returned at the routeReq() and can be used to correlate the response with the request.

### Method

#### **routeErr ( )**

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.).

*Parameters***callSessionID : in TpSessionID**

Specifies the call session ID of the call.

**errorIndication : in TpCallError**

Specifies the error which led to the original request failing.

**callLegSessionID : in TpSessionID**

Specifies the sessionID of the associated call leg. This corresponds to the sessionID returned at the routeReq() and can be used to correlate the error with the request.

*Method***getCallInfoRes( )**

This asynchronous method reports time information of the finished call or call attempt as well as release cause depending on which information has been requested by getCallInfoReq. This information may be used e.g. for charging purposes. The call information will possibly be sent after routeRes in all cases where the call or a leg of the call has been disconnected or a routing failure has been encountered.

*Parameters***callSessionID : in TpSessionID**

Specifies the call session ID of the call.

**callInfoReport : in TpCallInfoReport**

Specifies the call information requested.

*Method***getCallInfoErr( )**

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

*Parameters***callSessionID : in TpSessionID**

Specifies the call session ID of the call.

**errorIndication : in TpCallError**

Specifies the error which led to the original request failing.

*Method***superviseCallRes( )**

This asynchronous method reports a call supervision event to the application when it has indicated its interest in these kind of events.

It is also called when the connection is terminated before the supervision event occurs.

*Parameters***callSessionID : in TpSessionID**

Specifies the call session ID of the call.

**report : in TpCallSuperviseReport**

Specifies the situation which triggered the sending of the call supervision response.

**usedTime : in TpDuration**

Specifies the used time for the call supervision (in milliseconds).

*Method*

**superviseCallErr()**

This asynchronous method reports a call supervision error to the application.

*Parameters*

**callSessionID : in TpSessionID**

Specifies the call session ID of the call.

**errorIndication : in TpCallError**

Specifies the error which led to the original request failing.

*Method*

**callFaultDetected()**

This method indicates to the application that a fault in the network has been detected. The call may or may not have been terminated.

The system deletes the call object. Therefore, the application has no further control of call processing. No report will be forwarded to the application.

*Parameters*

**callSessionID : in TpSessionID**

Specifies the call session ID of the call in which the fault has been detected.

**fault : in TpCallFault**

Specifies the fault that has been detected.

*Method*

**getMoreDialledDigitsRes()**

This asynchronous method returns the collected digits to the application.

*Parameters*

**callSessionID : in TpSessionID**

Specifies the call session ID of the call.

**digits : in TpString**

Specifies the additional dialled digits if the string length is greater than zero.

*Method*

**getMoreDialledDigitsErr()**

This asynchronous method reports an error in collecting digits to the application.

*Parameters*

**callSessionID : in TpSessionID**

Specifies the call session ID of the call.

**errorIndication : in TpCallError**

Specifies the error which led to the original request failing.

*Method***callEnded()**

This method indicates to the application that the call has terminated in the network. However, the application may still receive some results (e.g. getCallInfoRes) related to the call. The application is expected to deassign the call object after having received the callEnded.

Note that the event that caused the call to end might also be received separately if the application was monitoring for it.

*Parameters***callSessionID : in TpSessionID**

Specifies the call sessionID.

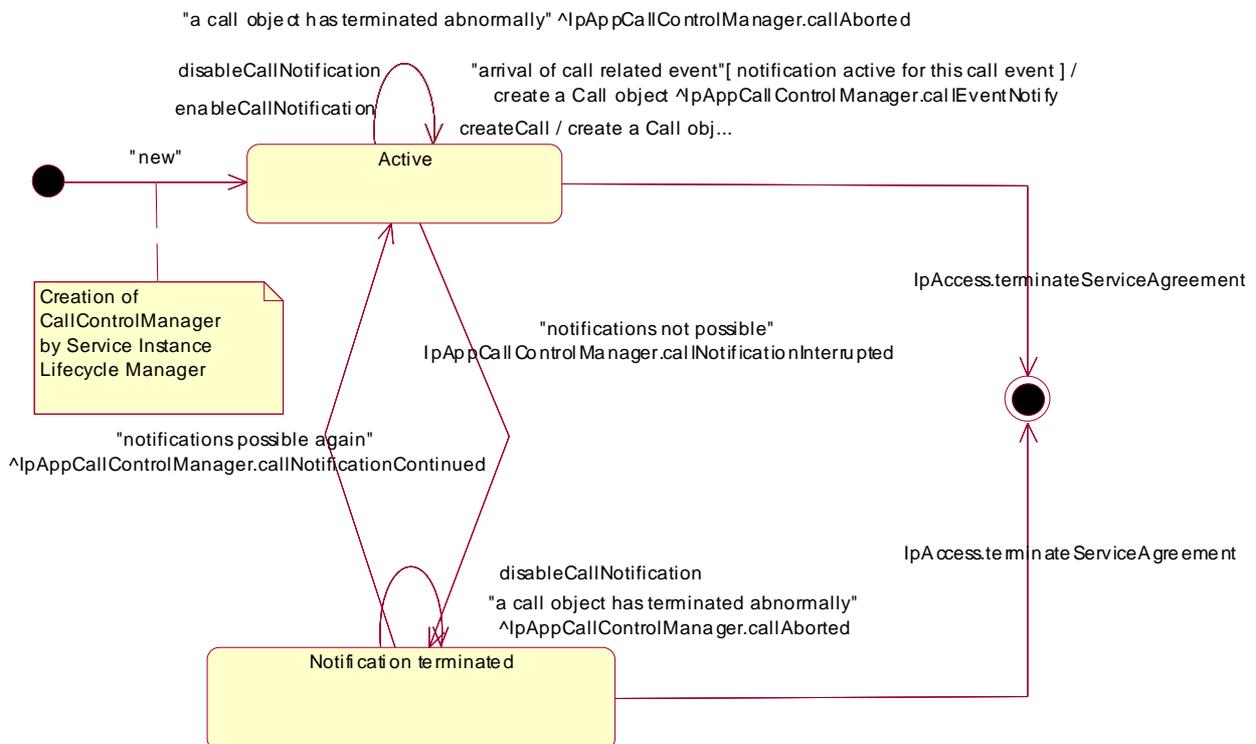
**report : in TpCallEndedReport**

Specifies the reason the call is terminated.

## 6.4 Generic Call Control Service State Transition Diagrams

### 6.4.1 State Transition Diagrams for IpCallControlManager

The state transition diagram shows the application view on the Call Control Manager object.



**Figure 3: Application view on the Call Control Manager**

### 6.4.1.1 Active State

In this state a relation between the Application and the Generic Call Control Service has been established. The state allows the application to indicate that it is interested in call related events. In case such an event occurs, the Call Control Manager will create a Call object and inform the application by invoking the operation callEventNotify() on the IpAppCallControlManager interface. The application can also indicate it is no longer interested in certain call related events by calling disableCallNotification().

### 6.4.1.2 Notification terminated State

When the Call Control Manager is in the Notification terminated state, events requested with enableCallNotification() will not be forwarded to the application. There can be multiple reasons for this: for instance it might be that the application receives more notifications from the network than defined in the Service Level Agreement. Another example is that the Service has detected it receives no notifications from the network due to e.g. a link failure. In this state no requests for new notifications will be accepted.

## 6.4.2 State Transition Diagrams for IpCall

The state transition diagram shows the application view on the Call object.

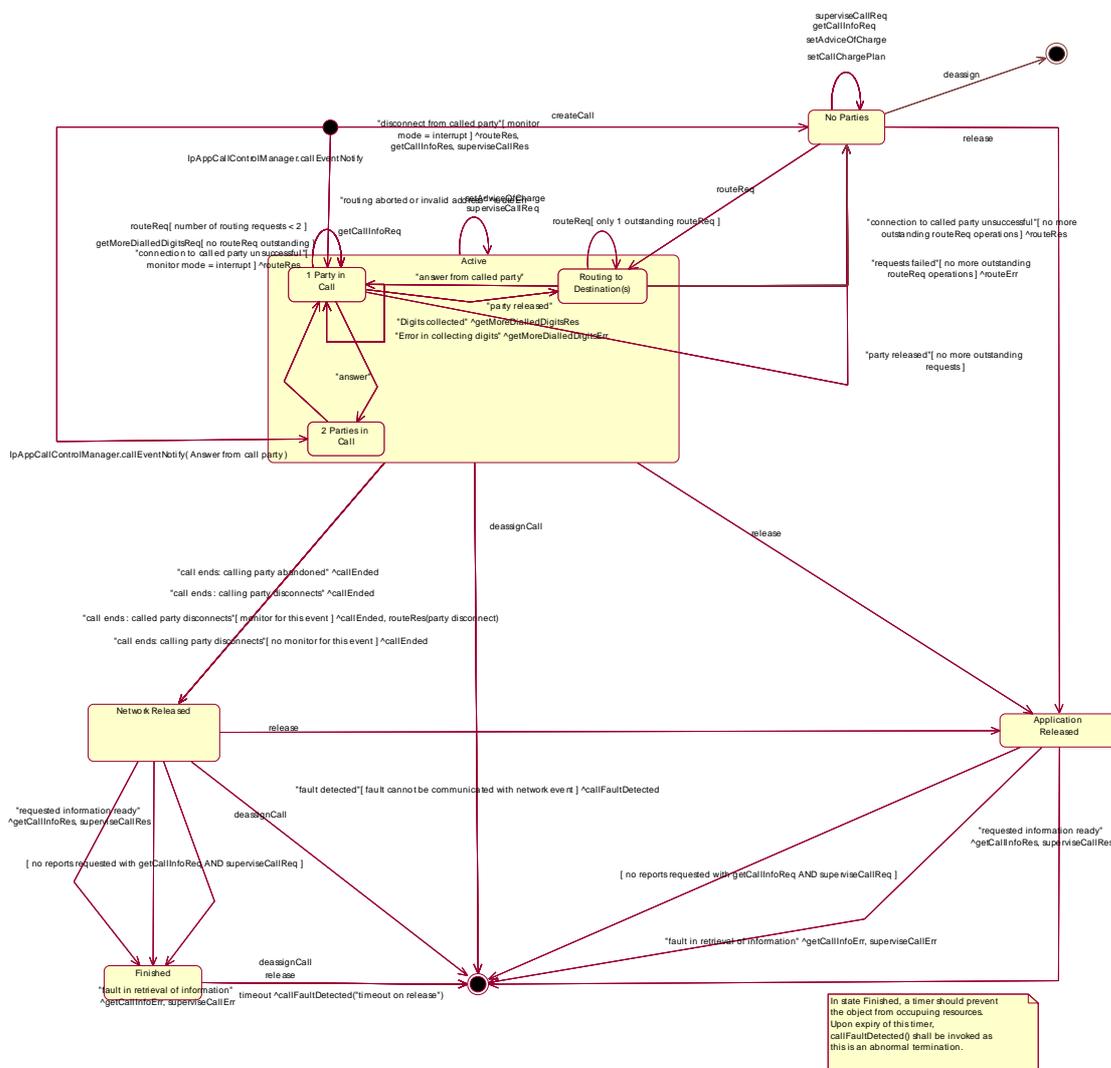


Figure 4: Application view on the IpCall object

### 6.4.2.1 Network Released State

In this state the call has ended and the Gateway collects the possible call information requested with `getCallInfoReq()` and/or `superviseCallReq()`. The information will be returned to the application by invoking the methods `getCallInfoRes()` and/or `superviseCallRes()` on the application. Also when a call was unsuccessful these methods are used. In case the application has not requested additional call related information immediately a transition is made to state Finished.

### 6.4.2.2 Finished State

In this state the call has ended and no call related information is to be send to the application. The application can only release the call object. Calling the `deassignCall()` operation has the same effect. Note that the application has to release the object itself as good OO practice requires that when an object was created on behalf of a certain entity, this entity is also responsible for destroying it when the object is no longer needed.

### 6.4.2.3 Application Released State

In this state the application has requested to release the Call object and the Gateway collects the possible call information requested with `getCallInfoReq()` and/or `superviseCallReq()`. In case the application has not requested additional call related information the Call object is destroyed immediately.

### 6.4.2.4 No Parties State

In this state the Call object has been created. The application can request the gateway for a certain type of charging of the call by calling `setCallChargePlan()`. The application can request for charging related information by calling `getCallInfoReq()`. Furthermore the application can request supervision of the call by calling `superviseCallReq()`. It is also allowed to request Advice of Charge information to be sent by calling `setAdviceOfCharge()`.

### 6.4.2.5 Active State

In this state a call between two parties is being setup or present. Refer to the substates for more details. The application can request supervision of the call by calling `superviseCallReq()`. It is also allowed to send Advice of Charge information by calling `setAdviceOfCharge()` as well as to define the charging by invoking `setCallChargePlan()`.

Call processing is suspended when a network event is met for the call, which was requested to be monitored in the `P_CALL_MONITOR_MODE_INTERRUPT`. In order to resume of the suspended call processing, the application invokes `continueProcessing()`, `routeReq()`, `release()` or `deassignCall()` method.

### 6.4.2.6 Party in Call State

In this state there is one party in the call.

In this state the application can request the gateway for a certain type of charging of the call by calling `setCallChargePlan()`. The application can also request for charging related information by calling `getCallInfoReq()`. The `setCallChargePlan()` and `getCallInfoReq()` should be issued before requesting a connection to a second party in the call by means of `routeReq()`.

Two cases apply: network initiated calls and application initiated calls.

In case the call originated from the network the application can now request for more digits in case more digits are needed. When the calling party abandons the call before the application has invoked the `routeReq()` operation, the application is informed with `callEnded()`. When the calling party abandons the call after the application has invoked `routeReq()` but before the call has actually been established, the gateway informs the application by invoking `callEnded()`.

In case the call was setup by the application and the called party was reached by issuing a `routeReq()` the application can request a connection to a second call party by calling the operation `routeReq()` again.

Otherwise, it depends on the actual number of invoked (and still outstanding or successful) routing requests whether the application can still call the routeReq() operation in order to setup a connection to a called party. Also in this case the called party can disconnect before another party is reached. In this case depending on the actual configuration, the call is ended or a transition is made back to the Routing to Destinations substate. When the second party answers the call, a transition will be made to the 2 Parties in Call state.

In this state user interaction is possible.

#### 6.4.2.7 Parties in Call State

A connection between two parties has been established.

In case the calling party disconnects, the gateway informs the application by invoking callEnded().

When the called party disconnects different situations apply:

1. the application is monitoring for this event in interrupt mode: a transition is made to the 1 Party in Call state, the application is informed with routeRes with indication that the called party has disconnected and all requested reports are sent to the application. The application now again has control of the call;
2. the application is monitoring for this event but not in interrupt mode. In this case a transition is made to the Network Released state and the gateway informs the application by invoking the operation routeRes() and callEnded();
3. the application is not monitoring for this event. In this case the application is informed by the gateway invoking the callEnded() operation and a transition is made to the Network Released state.

In this state user interaction is possible, depending on the underlying network.

#### 6.4.2.8 Routing to Destination(s) State

In this state there is at least one outstanding routeReq.

## 6.5 Generic Call Control Service Properties

### 6.5.1 List of Service Properties

The following table lists properties relevant for the GCC API.

Property	Type	Description/Interpretation
P_TRIGGERING_EVENT_TYPES	INTEGER_SET	Indicates the static event types supported by the SCS. Static events are the events by which applications are initiated.
P_DYNAMIC_EVENT_TYPES	INTEGER_SET	Indicates the dynamic event types supported by the SCS. Dynamic events are the events the application can request for during the context of a call.
P_ADDRESSPLAN	INTEGER_SET	Indicates the supported address plans (defined in TpAddressPlan.) e.g. {P_ADDRESS_PLAN_E164, P_ADDRESS_PLAN_IP}). Note that more than one address plan may be supported.
P_UI_CALL_BASED	BOOLEAN_SET	Value = TRUE : User interaction can be performed on call level and a reference to a Call object can be used in the IpUIManager.createUICall() operation. Value = FALSE: No User interaction on call level is supported.
P_UI_AT_ALL_STAGES	BOOLEAN_SET	Value = TRUE: User Interaction can be performed at any stage during a call. Value = FALSE: User Interaction can be performed in case there is only one party in the call.
P_MEDIA_TYPE	INTEGER_SET	Specifies the media type used by the Service. Values are defined by data-type TpMediaType : P_AUDIO, P_VIDEO, P_DATA

The previous table lists properties related to capabilities of the SCS itself. The following table lists properties that are used in the context of the Service Level Agreement, e.g. to restrict the access of applications to the capabilities of the SCS.

Property	Type	Description
P_TRIGGERING_ADDRESSES (Deprecated)	ADDRESSRANGE_SET	Indicates for which numbers the notification may be set. For terminating notifications it applies to the terminating number, for originating notifications it applies only to the originating number.
P_NOTIFICATION_ADDRESS_RANGES	XML_ADDRESS_RANGE_SET	Indicates for which numbers notifications may be set. More than one range may be present. For terminating notifications they apply to the terminating number, for originating notifications they apply only to the originating number.
P_NOTIFICATION_TYPES	INTEGER_SET	Indicates whether the application is allowed to set originating and/or terminating triggers in the ECN. Set is: P_ORIGINATING P_TERMINATING
P_MONITOR_MODE	INTEGER_SET	Indicates whether the application is allowed to monitor in interrupt and/or notify mode. Set is: P_INTERRUPT P_NOTIFY
P_NUMBERS_TO_BE_CHANGED	INTEGER_SET	Indicates which numbers the application is allowed to change or fill for legs in an incoming call. Allowed value set: {P_ORIGINAL_CALLED_PARTY_NUMBER, P_REDIRECTING_NUMBER, P_TARGET_NUMBER, P_CALLING_PARTY_NUMBER}.
P_CHARGEPLAN_ALLOWED	INTEGER_SET	Indicates which charging is allowed in the setCallChargePlan indicator. Allowed values: {P_TRANSPARENT_CHARGING, P_CHARGE_PLAN}
P_CHARGEPLAN_MAPPING	INTEGER_INTEGER_MAP	Indicates the mapping of chargeplans (we assume they can be indicated with integers) to a logical network chargeplan indicator. When the chargeplan supports indicates P_CHARGE_PLAN then only chargeplans in this mapping are allowed.

## 6.5.2 Service Property values for the CAMEL Service Environment

Implementations of the Generic Call Control API relying on the CSE of CAMEL phase 3 shall have the Service Properties outlined above set to the indicated values:

```
P_OPERATION_SET = {
'IpCallControlManager.enableCallNotification',
'IpCallControlManager.disableCallNotification',
'IpCallControlManager.changeCallNotification',
'IpCallControlManager.getCriteria',
'IpCallControlManager.setCallLoadControl',
'IpCall.routeReq',
'IpCall.release',
'IpCall.deassignCall',
'IpCall.getCallInfoReq',
'IpCall.setCallChargePlan',
'IpCall.setAdviceOfCharge',
'IpCall.superviseCallReq'
}
```

```
P_TRIGGERING_EVENT_TYPES = {
P_EVENT_GCCS_ADDRESS_COLLECTED_EVENT,
P_EVENT_GCCS_ADDRESS_ANALYSED_EVENT,
P_EVENT_GCCS_CALLED_PARTY_BUSY,
P_EVENT_GCCS_CALLED_PARTY_UNREACHABLE,
P_EVENT_GCCS_NO_ANSWER_FROM_CALLED_PARTY,
P_EVENT_GCCS_ROUTE_SELECT_FAILURE
}
```

```
P_DYNAMIC_EVENT_TYPES = {
P_CALL_REPORT_ANSWER,
P_CALL_REPORT_BUSY,
P_CALL_REPORT_NO_ANSWER,
P_CALL_REPORT_DISCONNECT,
P_CALL_REPORT_ROUTING_FAILURE,
P_CALL_REPORT_NOT_REACHABLE
}
```

```
P_ADDRESS_PLAN = {  
P_ADDRESS_PLAN_E164  
}  
  
P_UI_CALL_BASED = {  
TRUE  
}  
  
P_UI_AT_ALL_STAGES = {  
FALSE  
}  
  
P_MEDIA_TYPE = {  
P_AUDIO  
}
```

## 6.6 Generic Call Control Data Definitions

This clause provides the GCC data definitions necessary to support the API specification.

The general format of a Data Definition specification is described below.

- Data Type

This shows the name of the data type.

- Description

This describes the data type.

- Tabular Specification

This specifies the data types and values of the data type.

- Example

If relevant, an example is shown to illustrate the data type.

All data types referenced but not defined in this clause are either in the common call control data definitions clause of the present document (clause 10) or in the common data definitions which may be found in ES 201 915-2.

## 6.6.1 Generic Call Control Event Notification Data Definitions

### 6.6.1.1 TpCallEventName

Defines the names of event being notified. The following events are supported. The values may be combined by a logical "OR" function when requesting the notifications. Additional events that can be requested/received during the call process are found in the TpCallReportType data-type.

Name	Value	Description
P_EVENT_NAME_UNDEFINED	0	Undefined
P_EVENT_GCCS_OFFHOOK_EVENT	1	GCCS - Offhook event This can be used for hot-line features. In case this event is set in the TpCallEventCriteria, only the originating address(es) may be specified in the criteria.
P_EVENT_GCCS_ADDRESS_COLLECTED_EVENT	2	GCCS - Address information collected The network has collected the information from the A-party, but not yet analysed the information. The number can still be incomplete. Applications might set notifications for this event when part of the number analysis needs to be done in the application (see also the getMoreDialledDigitsReq method on the call class).
P_EVENT_GCCS_ADDRESS_ANALYSED_EVENT	4	GCCS - Address information is analysed The dialled number is a valid and complete number in the network.
P_EVENT_GCCS_CALLED_PARTY_BUSY	8	GCCS - Called party is busy
P_EVENT_GCCS_CALLED_PARTY_UNREACHABLE	16	GCCS - Called party is unreachable (e.g. the called party has a mobile telephone that is currently switched off).
P_EVENT_GCCS_NO_ANSWER_FROM_CALLED_PARTY	32	GCCS - No answer from called party
P_EVENT_GCCS_ROUTE_SELECT_FAILURE	64	GCCS - Failure in routing the call
P_EVENT_GCCS_ANSWER_FROM_CALL_PARTY	128	GCCS - Party answered call.

### 6.6.1.2 TpCallNotificationType

Defines the type of notification. Indicates whether it is related to the originating of the terminating user in the call.

Name	Value	Description
P_ORIGINATING	0	Indicates that the notification is related to the originating user in the call.
P_TERMINATING	1	Indicates that the notification is related to the terminating user in the call.

### 6.6.1.3 TpCallEventCriteria

Defines the Sequence of Data Elements that specify the criteria for an event notification.

Of the addresses only the Plan and the AddrString are used for the purpose of matching the notifications against the criteria.

Sequence Element Name	Sequence Element Type	Description
DestinationAddress	TpAddressRange	Defines the destination address or address range for which the notification is requested.
OriginatingAddress	TpAddressRange	Defines the origination address or a address range for which the notification is requested.
CallEventName	TpCallEventName	Name of the event(s)
CallNotificationType	TpCallNotificationType	Indicates whether it is related to the originating or the terminating user in the call.
MonitorMode	TpCallMonitorMode	Defines the mode that the call is in following the notification. Monitor mode P_CALL_MONITOR_MODE_DO_NOT_MONITOR is not a legal value here.

### 6.6.1.4 TpCallEventInfo

Defines the Sequence of Data Elements that specify the information returned to the application in a Call event notification.

Sequence Element Name	Sequence Element Type
DestinationAddress	TpAddress
OriginatingAddress	TpAddress
OriginalDestinationAddress	TpAddress
RedirectingAddress	TpAddress
CallAppInfo	TpCallAppInfoSet
CallEventName	TpCallEventName
CallNotificationType	TpCallNotificationType
MonitorMode	TpCallMonitorMode

## 6.6.2 Generic Call Control Data Definitions

### 6.6.2.1 IpCall

Defines the address of an IpCall Interface.

### 6.6.2.2 IpCallRef

Defines a Reference to type IpCall.

### 6.6.2.3 IpAppCall

Defines the address of an IpAppCall Interface.

### 6.6.2.4 IpAppCallRef

Defines a Reference to type IpAppCall.

### 6.6.2.5 TpCallIdentifier

Defines the Sequence of Data Elements that unambiguously specify the Generic Call object.

Sequence Element Name	Sequence Element Type	Sequence Element Description
CallReference	IpCallRef	This element specifies the interface reference for the call object.
CallSessionID	TpSessionID	This element specifies the call session ID of the call.

### 6.6.2.6 IpAppCallControlManager

Defines the address of an IpAppCallControlManager Interface.

### 6.6.2.7 IpAppCallControlManagerRef

Defines a Reference to type IpAppCallControlManager.

### 6.6.2.8 IpCallControlManager

Defines the address of an IpCallControlManager Interface.

### 6.6.2.9 IpCallControlManagerRef

Defines a Reference to type IpCallControlManager.

### 6.6.2.10 TpCallAppInfo

Defines the Tagged Choice of Data Elements that specify application-related call information.

	Tag Element Type	
	TpCallAppInfoType	

Tag Element Value	Choice Element Type	Choice Element Name
P_CALL_APP_ALERTING_MECHANISM	TpCallAlertingMechanism	CallAppAlertingMechanism
P_CALL_APP_NETWORK_ACCESS_TYPE	TpCallNetworkAccessType	CallAppNetworkAccessType
P_CALL_APP_TELE_SERVICE	TpCallTeleService	CallAppTeleService
P_CALL_APP_BEARER_SERVICE	TpCallBearerService	CallAppBearerService
P_CALL_APP_PARTY_CATEGORY	TpCallPartyCategory	CallAppPartyCategory
P_CALL_APP_PRESENTATION_ADDRESS	TpAddress	CallAppPresentationAddress
P_CALL_APP_GENERIC_INFO	TpString	CallAppGenericInfo
P_CALL_APP_ADDITIONAL_ADDRESS	TpAddress	CallAppAdditionalAddress

### 6.6.2.11 TpCallAppInfoType

Defines the type of call application-related specific information.

Name	Value	Description
P_CALL_APP_UNDEFINED	0	Undefined
P_CALL_APP_ALERTING_MECHANISM	1	The alerting mechanism or pattern to use
P_CALL_APP_NETWORK_ACCESS_TYPE	2	The network access type (e.g. ISDN)
P_CALL_APP_TELE_SERVICE	3	Indicates the tele-service (e.g. telephony)
P_CALL_APP_BEARER_SERVICE	4	Indicates the bearer service (e.g. 64 kbit/s unrestricted data).
P_CALL_APP_PARTY_CATEGORY	5	The category of the calling party
P_CALL_APP_PRESENTATION_ADDRESS	6	The address to be presented to other call parties
P_CALL_APP_GENERIC_INFO	7	Carries unspecified service-service information
P_CALL_APP_ADDITIONAL_ADDRESS	8	Indicates an additional address

### 6.6.2.12 TpCallAppInfoSet

Defines a Numbered Set of Data Elements of TpCallAppInfo.

### 6.6.2.13 TpCallEndedReport

Defines the Sequence of Data Elements that specify the reason for the call ending.

Sequence Element Name	Sequence Element Type	Description
CallLegSessionID	TpSessionID	The leg that initiated the release of the call. If the call release was not initiated by the leg, then this value is set to -1.
Cause	TpCallReleaseCause	The cause of the call ending.

### 6.6.2.14 TpCallFault

Defines the cause of the call fault detected.

Name	Value	Description
P_CALL_FAULT_UNDEFINED	0	Undefined
P_CALL_TIMEOUT_ON_RELEASE	1	This fault occurs when the final report has been sent to the application, but the application did not explicitly release or deassign the call object, within a specified time. The timer value is operator specific.
P_CALL_TIMEOUT_ON_INTERRUPT	2	This fault occurs when the application did not instruct the gateway how to handle the call within a specified time, after the gateway reported an event that was requested by the application in interrupt mode. The timer value is operator specific.

### 6.6.2.15 TpCallInfoReport

Defines the Sequence of Data Elements that specify the call information requested. Information that was not requested is invalid.

Sequence Element Name	Sequence Element Type	Description
CallInfoType	TpCallInfoType	The type of call report.
CallInitiationStartTime	TpDateAndTime	The time and date when the call, or follow-on call, was started as a result of a routeReq.
CallConnectedToResourceTime	TpDateAndTime	The date and time when the call was connected to the resource. This data element is only valid when information on user interaction is reported.
CallConnectedToDestinationTime	TpDateAndTime	The date and time when the call was connected to the destination (i.e. when the destination answered the call). If the destination did not answer, the time is set to an empty string. This data element is invalid when information on user interaction is reported.
CallEndTime	TpDateAndTime	The date and time when the call or follow-on call or user interaction was terminated.
Cause	TpCallReleaseCause	The cause of the termination.

A callInfoReport will be generated at the end of user interaction and at the end of the connection with the associated address. This means that either the destination related information is present or the resource related information, but not both.

### 6.6.2.16 TpCallReleaseCause

Defines the Sequence of Data Elements that specify the cause of the release of a call.

Sequence Element Name	Sequence Element Type
Value	TpInt32
Location	TpInt32

NOTE: The Value and Location are specified as in ITU-T Recommendation Q.850.

The following example was taken from Q.850 to aid understanding:

Equivalent Call Report	Cause Value Set by Application	Cause Value from Network
P_CALL_REPORT_BUSY	17	17
P_CALL_REPORT_NO_ANSWER	19	18,19,21
P_CALL_REPORT_DISCONNECT	16	16
P_CALL_REPORT_REDIRECTED	23	23
P_CALL_REPORT_SERVICE_CODE	31	NA
P_CALL_REPORT_NOT_REACHABLE	20	20
P_CALL_REPORT_ROUTING_FAILURE	3	Any other value

### 6.6.2.17 TpCallReport

Defines the Sequence of Data Elements that specify the call report and call leg report specific information.

Sequence Element Name	Sequence Element Type
MonitorMode	TpCallMonitorMode
CallEventTime	TpDateAndTime
CallReportType	TpCallReportType
AdditionalReportInfo	TpCallAdditionalReportInfo

### 6.6.2.18 TpCallAdditionalReportInfo

Defines the Tagged Choice of Data Elements that specify additional call report information for certain types of reports.

Tag Element Type
TpCallReportType

Tag Element Value	Choice Element Type	Choice Element Name
P_CALL_REPORT_UNDEFINED	NULL	Undefined
P_CALL_REPORT_PROGRESS	NULL	Undefined
P_CALL_REPORT_ALERTING	NULL	Undefined
P_CALL_REPORT_ANSWER	NULL	Undefined
P_CALL_REPORT_BUSY	TpCallReleaseCause	Busy
P_CALL_REPORT_NO_ANSWER	NULL	Undefined
P_CALL_REPORT_DISCONNECT	TpCallReleaseCause	CallDisconnect
P_CALL_REPORT_REDIRECTED	TpAddress	ForwardAddress
P_CALL_REPORT_SERVICE_CODE	TpCallServiceCode	ServiceCode
P_CALL_REPORT_ROUTING_FAILURE	TpCallReleaseCause	RoutingFailure
P_CALL_REPORT_QUEUED	TpString	QueueStatus
P_CALL_REPORT_NOT_REACHABLE	TpCallReleaseCause	NotReachable

### 6.6.2.19 TpCallReportRequest

Defines the Sequence of Data Elements that specify the criteria relating to call report requests.

Sequence Element Name	Sequence Element Type
MonitorMode	TpCallMonitorMode
CallReportType	TpCallReportType
AdditionalReportCriteria	TpCallAdditionalReportCriteria

### 6.6.2.20 TpCallAdditionalReportCriteria

Defines the Tagged Choice of Data Elements that specify specific criteria.

	Tag Element Type	
	TpCallReportType	

Tag Element Value	Choice Element Type	Choice Element Name
P_CALL_REPORT_UNDEFINED	NULL	Undefined
P_CALL_REPORT_PROGRESS	NULL	Undefined
P_CALL_REPORT_ALERTING	NULL	Undefined
P_CALL_REPORT_ANSWER	NULL	Undefined
P_CALL_REPORT_BUSY	NULL	Undefined
P_CALL_REPORT_NO_ANSWER	TpDuration	NoAnswerDuration
P_CALL_REPORT_DISCONNECT	NULL	Undefined
P_CALL_REPORT_REDIRECTED	NULL	Undefined
P_CALL_REPORT_SERVICE_CODE	TpCallServiceCode	ServiceCode
P_CALL_REPORT_ROUTING_FAILURE	NULL	Undefined
P_CALL_REPORT_QUEUED	NULL	Undefined
P_CALL_REPORT_NOT_REACHABLE	NULL	Undefined

### 6.6.2.21 TpCallReportRequestSet

Defines a Numbered Set of Data Elements of TpCallReportRequest.

### 6.6.2.22 TpCallReportType

Defines a specific call event report type.

Name	Value	Description
P_CALL_REPORT_UNDEFINED	0	Undefined.
P_CALL_REPORT_PROGRESS	1	Call routing progress event: an indication from the network that progress has been made in routing the call to the requested call party. This message may be sent more than once, or may not be sent at all by the gateway with respect to routing a given call leg to a given address.
P_CALL_REPORT_ALERTING	2	Call is alerting at the call party.
P_CALL_REPORT_ANSWER	3	Call answered at address.
P_CALL_REPORT_BUSY	4	Called address refused call due to busy.
P_CALL_REPORT_NO_ANSWER	5	No answer at called address.
P_CALL_REPORT_DISCONNECT	6	The media stream of the called party has disconnected. This does not imply that the call has ended. When the call is ended, the callEnded method is called. This event can occur both when the called party hangs up, or when the application explicitly releases the leg using IpCallLeg.release() This cannot occur when the app explicitly releases the call leg and the call.
P_CALL_REPORT_REDIRECTED	7	Call redirected to new address: an indication from the network that the call has been redirected to a new address.
P_CALL_REPORT_SERVICE_CODE	8	Mid-call service code received.
P_CALL_REPORT_ROUTING_FAILURE	9	Call routing failed - re-routing is possible.
P_CALL_REPORT_QUEUED	10	The call is being held in a queue. This event may be sent more than once during the routing of a call.
P_CALL_REPORT_NOT_REACHABLE	11	The called address is not reachable; e.g. the phone has been switched off or the phone is outside the coverage area of the network.

### 6.6.2.23 TpCallTreatment

Defines the Sequence of Data Elements that specify the treatment for calls that will be handled only by the network (for example, call which are not admitted by the call load control mechanism).

Sequence Element Name	Sequence Element Type
CallTreatmentType	TpCallTreatmentType
ReleaseCause	TpCallReleaseCause
AdditionalTreatmentInfo	TpCallAdditionalTreatmentInfo

### 6.6.2.24 TpCallEventCriteriaResultSet

Defines a set of TpCallEventCriteriaResult.

### 6.6.2.25 TpCallEventCriteriaResult

Defines a sequence of data elements that specify a requested call event notification criteria with the associated assignmentID.

Sequence Element Name	Sequence Element Type	Sequence Element Description
CallEventCriteria	TpCallEventCriteria	The event criteria that were specified by the application.
AssignmentID	TpInt32	The associated assignmentID. This can be used to disable the notification.

---

## 7 MultiParty Call Control Service

### 7.1 Sequence Diagrams

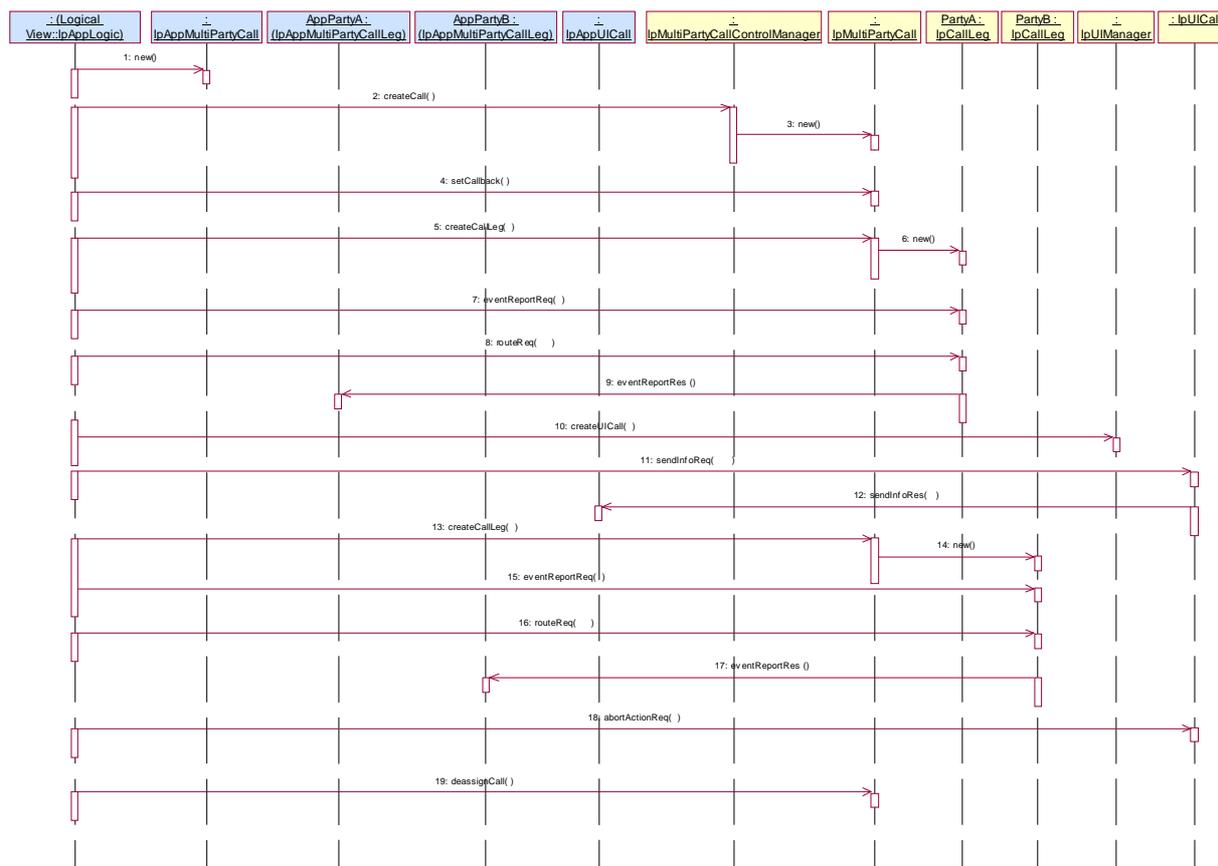
#### 7.1.1 Application initiated call setup

The following sequence diagram shows an application creating a call between party A and party B. Here, a call is created first. Then party A's call leg is created before events are requested on it for answer and then routed to the call. On answer from Party A, an announcement is played indicating that the call is being set up to party B. While the announcement is being played, party B's call leg is created and then events are requested on it for answer. On answer from Party B the announcement is cancelled and party B is routed to the call.

The service may as a variation be extended to include 3 parties (or more). After the two party call is established, the application can create a new leg and request to route it to a new destination address in order to establish a 3 party call.

The event that causes this to happen could for example be the report of answer event from B-party or controlled by the A-party by entering a service code (mid-call event).

The procedure for call setup to party C is exactly the same as for the set up of the connection to party B (sequence 13 to 17 in the sequence diagram).

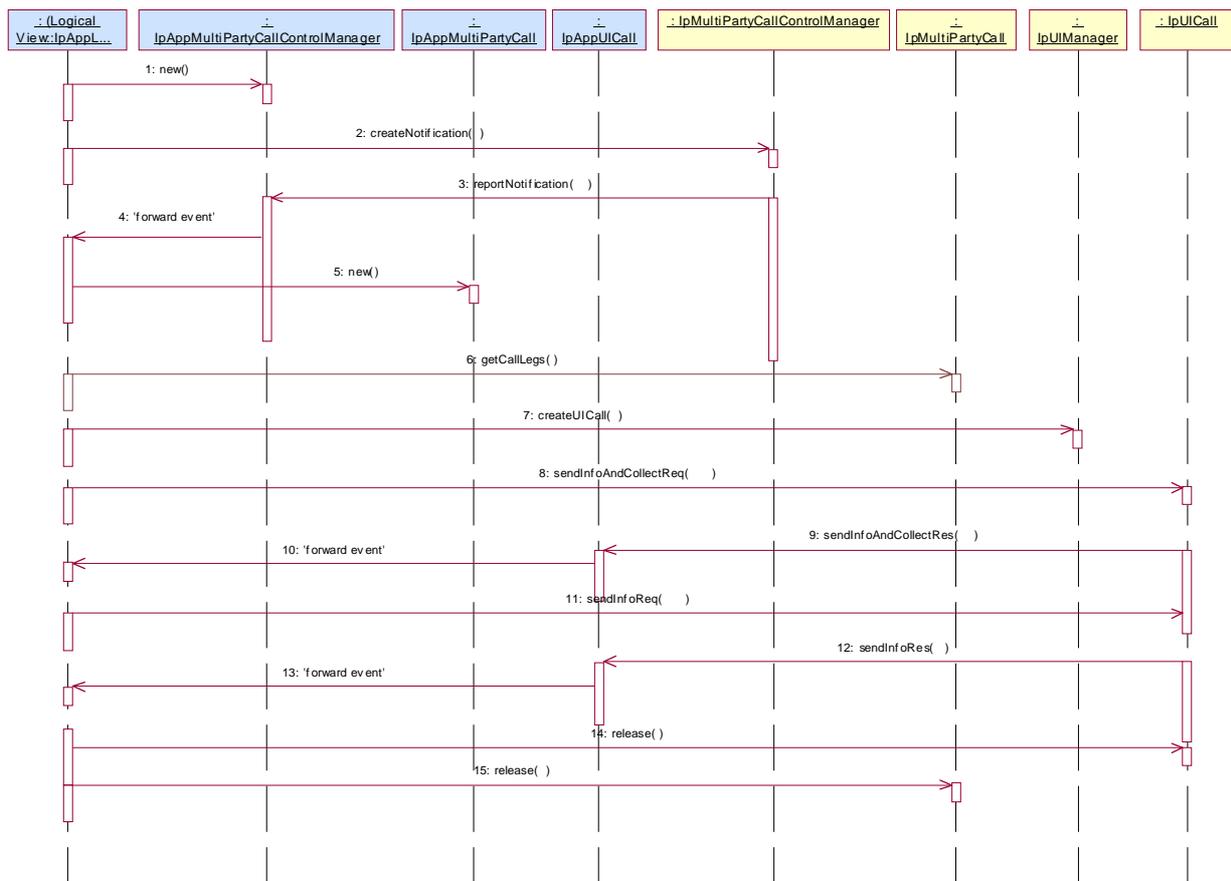


- 1: This message is used to create an object implementing the IpAppMultiPartyCall interface.
- 2: This message requests the object implementing the IpMultiPartyCallControlManager interface to create an object implementing the IpMultiPartyCall interface.
- 3: Assuming that the criteria for creating an object implementing the IpMultiPartyCall interface (e.g. load control values not exceeded) is met it is created.
- 4: Once the object implementing the IpMultiPartyCall interface is created it is used to pass the reference of the object implementing the IpAppMultiPartyCall interface as the callback reference to the object implementing the IpMultiPartyCall interface. Note that the reference to the callback interface could already have been passed in the createCall.
- 5: This message instructs the object implementing the IpMultiPartyCall interface to create a call leg for customer A.
- 6: Assuming that the criteria for creating an object implementing the IpCallLeg interface is met, message 6 is used to create it.
- 7: This message requests the call leg for customer A to inform the application when the call leg answers the call.
- 8: The call is then routed to the originating call leg.
- 9: Assuming the call is answered, the object implementing party A's IpCallLeg interface passes the result of the call being answered back to its callback object. This message is then forwarded via another message (not shown) to the object implementing the IpAppLogic interface.
- 10: A UICall object is created and associated with the just created call leg.
- 11: This message is used to inform party A that the call is being routed to party B.
- 12: An indication that the dialogue with party A has commenced is returned via message 13 and eventually forwarded via another message (not shown) to the object implementing the IpAppLogic interface.

- 13: This message instructs the object implementing the IpMultiPartyCall interface to create a call leg for customer B.
- 14: Assuming that the criteria for creating a second object implementing the IpCallLeg interface is met, it is created.
- 15: This message requests the call leg for customer B to inform the application when the call leg answers the call.
- 16: The call is then routed to the call leg.
- 17: Assuming the call is answered, the object implementing party B's IpCallLeg interface passes the result of the call being answered back to its callback object. This message is then forwarded via another message (not shown) to the object implementing the IpAppLogic interface.
- 18: This message then instructs the object implementing the IpUICall interface to stop sending announcements to party A.
- 19: The application deassigns the call. This will also deassign the associated user interaction.

## 7.1.2 Call Barring 2

The following sequence diagram shows a call barring service, initiated as a result of a prearranged event being received by the call control service. Before the call is routed to the destination number, the calling party is asked for a PIN code. The code is rejected and the call is cleared.

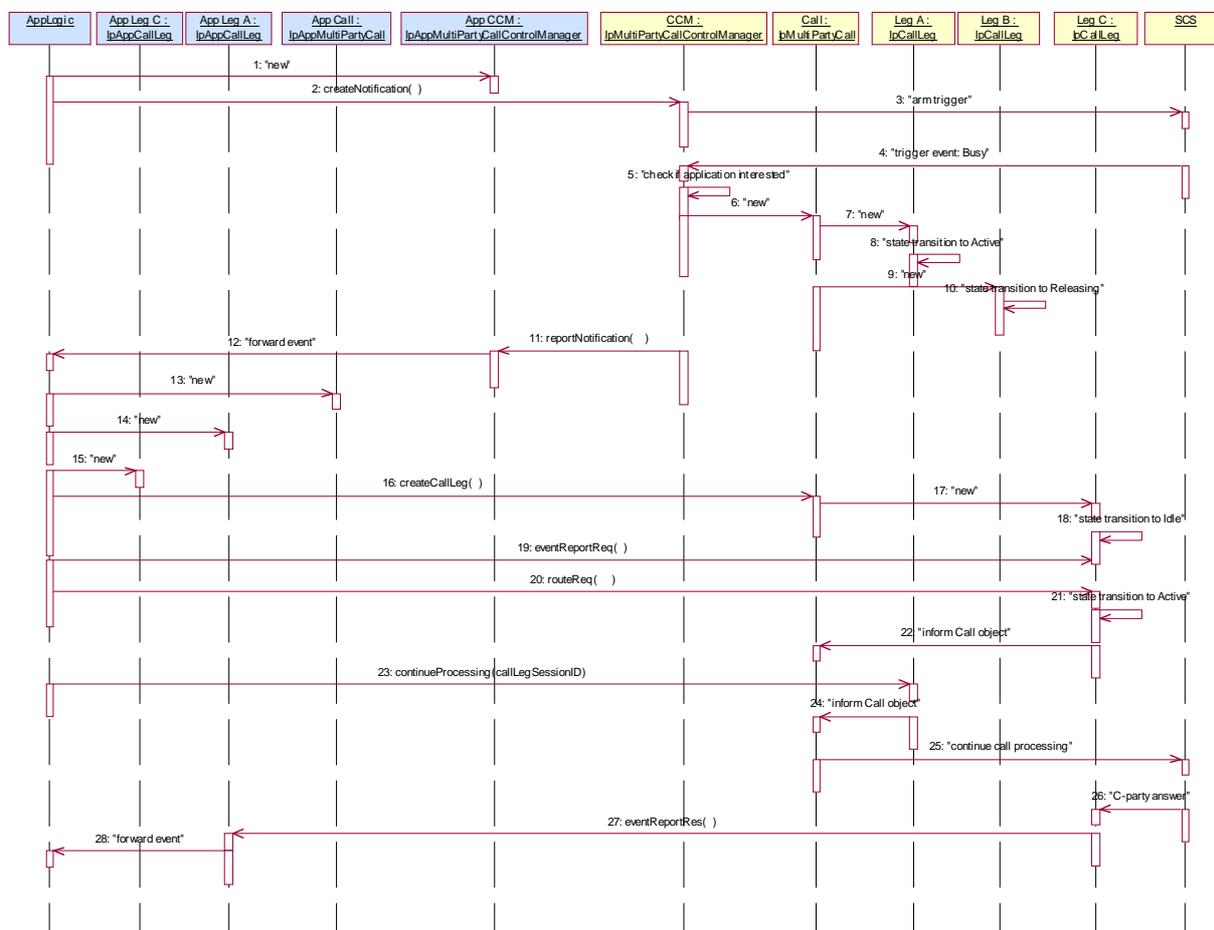


- 1: This message is used by the application to create an object implementing the IpAppMultiPartyCallControlManager interface.
- 2: This message is sent by the application to enable notifications on new call events. As this sequence diagram depicts a call barring service, it is likely that all new call events destined for a particular address or address range prompted for a password before the call is allowed to progress. When a new call, that matches the event criteria, arrives a message (not shown) is directed to the object implementing the IpMultiPartyCallControlManager. Assuming that the criteria for creating an object implementing the IpMultiPartyCall interface (e.g. load control values not exceeded) is met, other messages (not shown) are used to create the call and associated call leg object.
- 3: This message is used to pass the new call event to the object implementing the IpAppMultiPartyCallControlManager interface.
- 4: This message is used to forward message 3 to the IpAppLogic.
- 5: This message is used by the application to create an object implementing the IpAppMultiPartyCall interface. The reference to this object is passed back to the object implementing the IpMultiPartyCallControlManager using the return parameter of the callEventNotify.
- 6: The application requests an list of all the legs currently in the call.
- 7: This message is used to create a UICall object that is associated with the incoming leg of the call.
- 8: The call barring service dialogue is invoked.
- 9: The result of the dialogue, which in this case is the PIN code, is returned to its callback object.
- 10: This message is used to forward the previous message to the IpAppLogic.
- 11: Assuming an incorrect PIN is entered, the calling party is informed using additional dialogue of the reason why the call cannot be completed.
- 12: This message passes the indication that the additional dialogue has been sent.
- 13: This message is used to forward the previous message to the IpAppLogic.
- 14: No more UI is required, so the UICall object is released.
- 15: This message is used by the application to clear the call.

### 7.1.3 Call forwarding on Busy Service

The following sequence diagram shows an application establishing a call forwarding on busy.

When a call is made from A to B but the B-party is detected to be busy, then the application is informed of this and sets up a connection towards a C party. The C party can for instance be a voicemail system.



- 1: This message is used by the application to create an object implementing the IpAppMultiPartyCallControlManager interface.
- 2: This message is sent by the application to enable notifications on new call events.
- 4: When a new call, that matches the event criteria, arrives a message ("busy") is directed to the object implementing the IpMultiPartyCallControlManager. Assuming that the criteria for creating an object implementing the IpMultiPartyCall interface is met, other messages are used to create the call and associated call leg objects.
- 6: A new MultiPartyCall object is created to handle this particular call.
- 7: A new CallLeg object corresponding to Party A is created.
- 8: The new Call Leg instance transits to state Initiating.
- 11: This message is used to pass the new call event to the object implementing the IpAppMultiPartyCallControlManager interface. Applied monitor mode is "interrupt".
- 12: This message is used to forward the message to the IpAppLogic.

- 13: This message is used by the application to create an object implementing the IpAppMultiPartyCall interface. The reference to this object is passed back to the object implementing the IpMultiPartyCallControlManager using the return parameter of the reportNotification.
- 14: A new AppCallLeg is created to receive callbacks for the Leg corresponding to party A.
- 15: A new AppCallLeg C is created to receive callbacks for another leg.
- 16: This message is used to create a new call leg object. The object is created in the idle state and not yet routed in the network.
- 19: The application requests to be notified (monitor mode "INTERRUPT") when party C answers the call.
- 20: The application requests to route the terminating leg to reach the associated party C.  
  
The application may request information about the original destination address be sent by setting up the field P\_CALL\_APP\_ORIGINAL\_DESTINATION\_ADDRESS of TpCallAppInfo in the request to route the call leg to the remote party C.
- 23: The application requests to resume call processing for the terminating call leg to party B to terminate the leg. Alternative the application could request to deassign the leg to party B for example if it is not interested in possible requested call leg information (getInfoRes, superviseRes).  
  
When the terminating call leg is destroyed, the AppLeg B is notified and the event is forwarded to the application logic (not shown).
- 25: The application requests to resume call processing for the originating call leg.  
  
As a result call processing is resumed in the network that will try to reach the associated party B.
- 26: When the party C answers the call, the termination call leg is notified.
- 27: Assuming the call is answered, the object implementing party C's IpCallLeg interface passes the result of the call being answered back to its callback object.
- 28: This answer message is then forwarded to the object implementing the IpAppLogic interface.

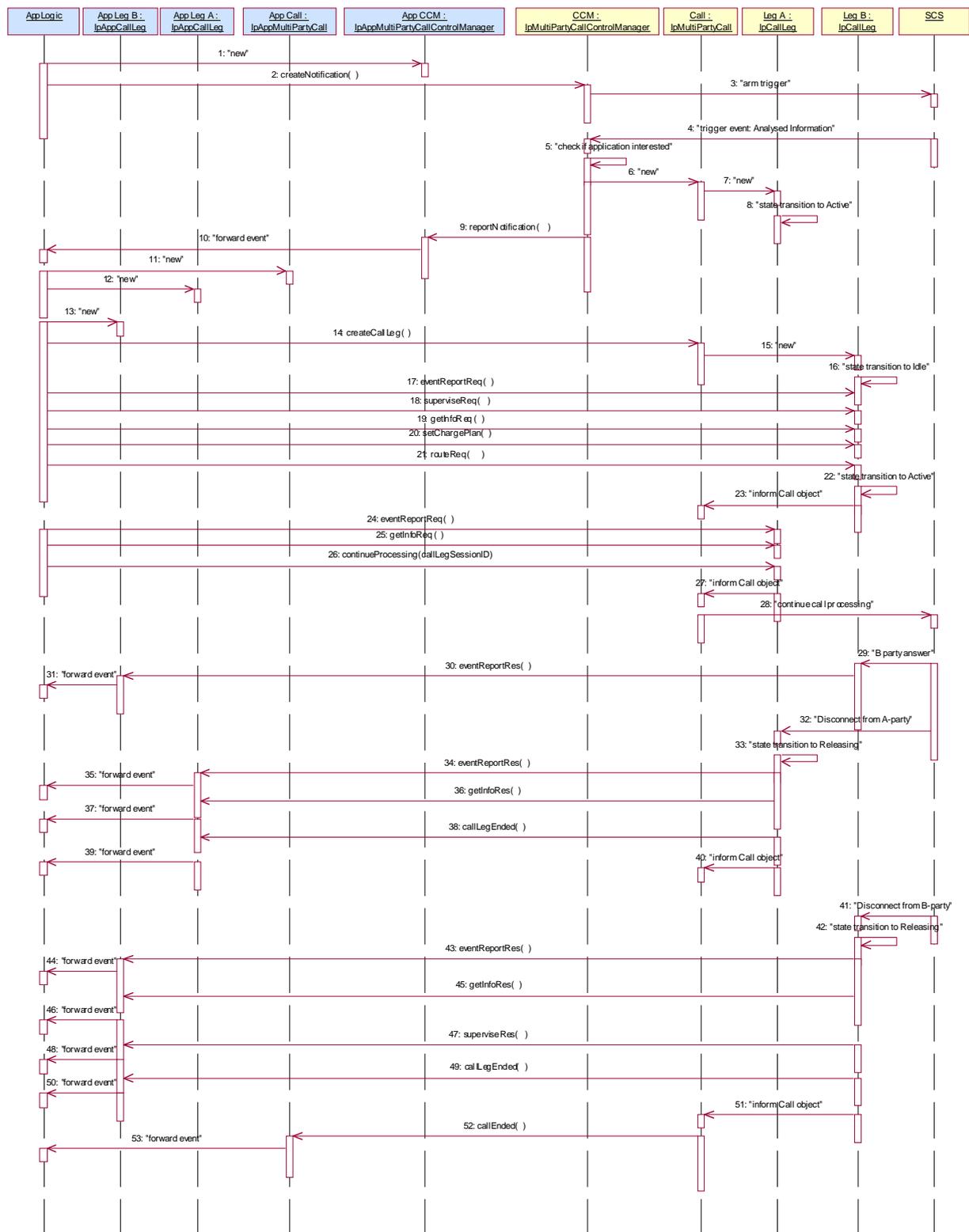
#### 7.1.4 Call Information Collect Service

The following sequence diagram shows an application monitoring a call between party A and a party B in order to collect call information at the end of the call for e.g. charging and/or statistic information collection purposes. The service may apply to ordinary two-party calls, but could also include a number translation of the dialled number and special charging (e.g. a premium rate service).

Additional call leg related information is requested with the getInfoReq and superviseReq methods.

The answer and call release events are in this service example requested to be reported in notify mode and additional call leg related information is requested with the getInfoReq and superviseReq methods in order to illustrate the information that can be collected and sent to the application at the end of the call.

Furthermore is shows the order in which information is sent to the application: network release event followed by possible requested call leg information, then the destroy of the call leg object (callLegEnded) and finally the destroy of the call object (callEnded).



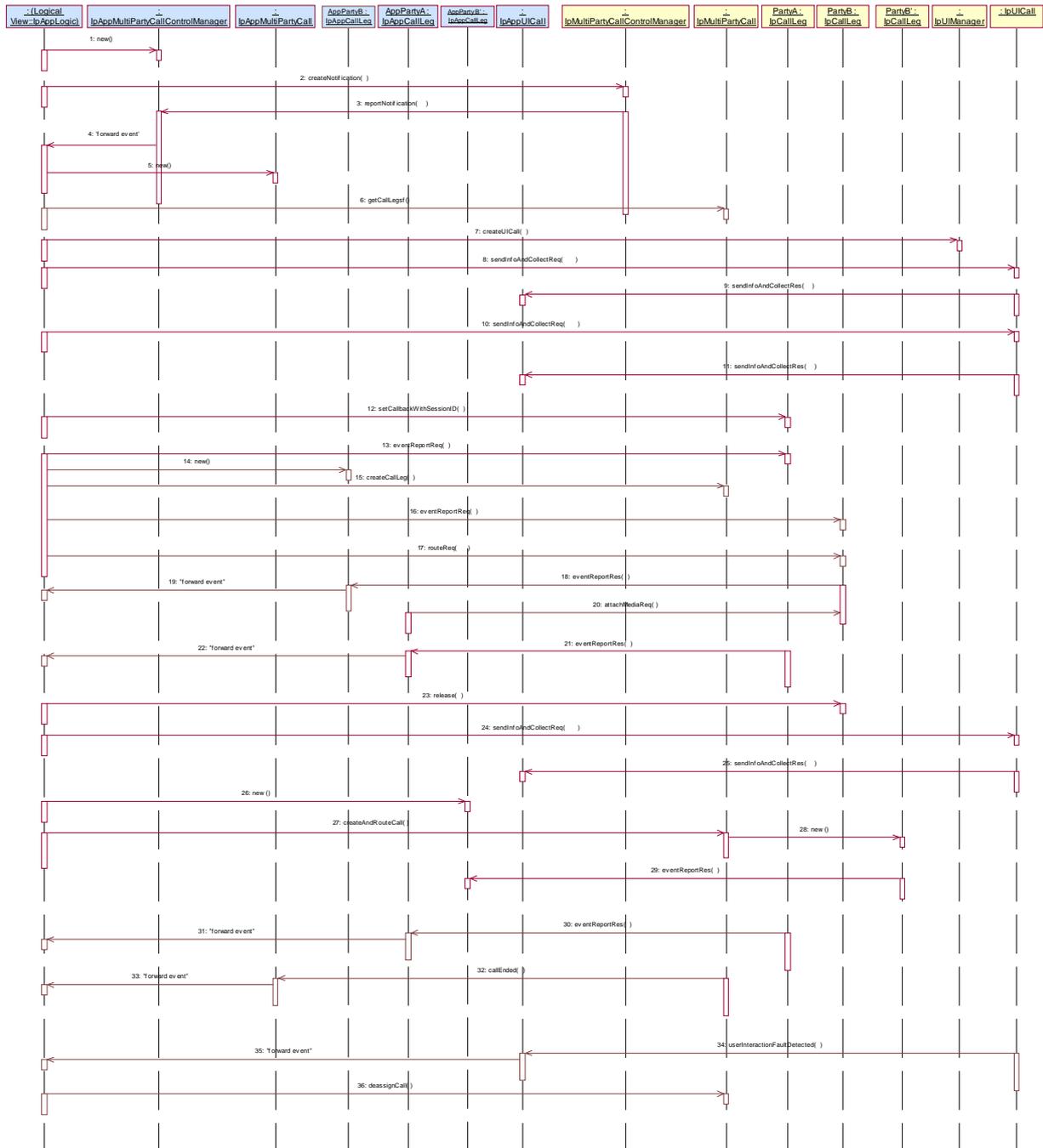
- 1: This message is used by the application to create an object implementing the IpAppMultiPartyCallControlManager interface.
- 2: This message is sent by the application to enable notifications on new call events.

- 4: When a new call that matches the event criteria arrives, a message ("analysed information") is directed to the object implementing the IpMultiPartyCallControlManager. Assuming that the criteria for creating an object implementing the IpMultiPartyCall interface is met, other messages are used to create the call and associated call leg object.
- 6: A new MultiPartyCall object is created to handle this particular call.
- 7: A new CallLeg object corresponding to Party A is created.
- 8: The new Call Leg instance transits to state Active.
- 9: This message is used to pass the new call event to the object implementing the IpAppMultiPartyCallControlManager interface. Applied monitor mode is "interrupt".
- 10: This message is used to forward message 9 to the IpAppLogic.
- 11: This message is used by the application to create an object implementing the IpAppMultiPartyCall interface. The reference to this object is passed back to the object implementing the IpMultiPartyCallControlManager using the return parameter of the reportNotification.
- 12: A new AppCallLeg is created to receive callbacks for the Leg corresponding to party A.
- 13: A new AppCallLeg is created to receive callbacks for another leg.
- 14: This message is used to create a new call leg object. The object is created in the idle state and not yet routed in the network.
- 15: A new CallLeg corresponding to party B is created.
- 16: A transition to state Idle is made after the Call leg has been created.
- 17: The application requests to be notified (monitor mode "NOTIFY") when party B answers the call and when the leg to B-party is released.
- 18: The application requests to supervise the call leg to party B.
- 19: The application requests information associated with the call leg to party b for example to calculate charging.
- 20: The application requests a specific charge plan to be set for the call leg to party B.
- 21: The application requests to route the terminating leg to reach the associated party B.
- 22: The Call Leg instance transits to state Active.
- 24: The application requests to be notified (monitor mode "Notify") when the leg to A-party is released.
- 25: The application requests information associated with the call leg to party A for example to calculate charging.
- 26: The application requests to resume call processing for the originating call leg.  
As a result call processing is resumed in the network that will try to reach the associated party B.
- 29: When the B-party answers the call, the termination call leg is notified.
- 30: Assuming the call is answered, the object implementing party B's IpCallLeg interface passes the result of the call being answered back to its callback object (monitor mode "NOTIFY").
- 31: This answer message is then forwarded.
- 32: When the A-party releases the call, the originating call leg is notified (monitor mode "NOTIFY") and makes a transition to "releasing state".
- 34: The application IpAppLeg A is notified, as the release event has been requested to be reported in Notify mode.
- 35: The event is forwarded to the application logic.
- 36: The call leg information is reported.

- 37: The event is forwarded to the application logic.
- 38: The origination call leg is destroyed, the AppLeg A is notified.
- 39: The event is forwarded to the application logic.
- 41: When the B-party releases the call or the call is released as a result of the release request from party A, i.e. a "originating release" indication, the terminating call leg is notified and makes a transition to "releasing state".
- 43: If a network release event is received being a "terminating release" indication from called party B, the application IpAppLeg B is notified, as the release event from party B has been requested to be reported in NOTIFY mode.  
  
Note that no report is sent if the release is caused by propagation of network release event being a "originating release" indication coming from calling party A..
- 44: The event is forwarded to the application logic.
- 45: The call leg information is reported.
- 46: The event is forwarded to the application logic.
- 47: The supervised call leg information is reported.
- 48: The event is forwarded to the application logic.
- 49: The terminating call leg is destroyed, the AppLeg B is notified.
- 50: The event is forwarded to the application logic.
- 52: Assuming the IpCall object has been informed that the legs have been destroyed, the IpAppMultiPartyCall is notified that the call is ended.
- 53: The event is forwarded to the application logic.

### 7.1.5 Complex Card Service

The following sequence diagram shows an advanced card service, initiated as a result of a prearranged event being received by the call control service. Before the call is made, the calling party is asked for an ID and PIN code. If the ID and PIN code are accepted, the calling party is prompted to enter the address of the destination party. A trigger of '#5' is then set on the controlling leg (the calling party's leg) such that if the calling party enters a '#5' an event will be sent to the application. The call is then routed to the destination party. Sometime during the call the calling party enters '#5' which causes the called leg to be released. The calling party is now prompted to enter the address of a new destination party, to which it is then routed.



- 1: This message is used by the application to create an object implementing the IpAppMultiPartyCallControlManager interface.
- 2: This message is sent by the application to enable notifications on new call events. As this sequence diagram depicts a call barring service, it is likely that all new call events destined for a particular address or address range result in the caller being prompted for a password before the call is allowed to progress. When a new call, that matches the event criteria set in message 2, arrives a message (not shown) is directed to the object implementing the IpMultiPartyCallControlManager interface. Assuming that the criteria for creating an object implementing the IpMultiPartyCall interface (e.g. load control values not exceeded) is met, other messages (not shown) are used to create the call and associated call leg object.
- 3: This message is used to pass the new call event to the object implementing the IpAppMultiPartyCallControlManager interface.

- 4: This message is used to forward message 3 to the IpAppLogic.
- 5: This message is used by the application to create an object implementing the IpAppMultiPartyCall interface. The reference to this object is passed back to the object implementing the IpMultiPartyCallControlManager using the return parameter of message 3.
- 6: This message returns the call legs currently in the call. In principle a reference to the call leg of the calling party is already obtained by the application when it was notified of the new call event.
- 7: This message is used to associate a user interaction object with the calling party.
- 8: The initial card service dialogue is invoked using this message.
- 9: The result of the dialogue, which in this case is the ID and PIN code, is returned to its callback object using this message and eventually forwarded via another message (not shown) to the IpAppLogic.
- 10: Assuming the correct ID and PIN are entered, the final dialogue is invoked.
- 11: The result of the dialogue, which in this case is the destination address, is returned and eventually forwarded via another message (not shown) to the IpAppLogic.
- 12: This message is used to forward the address of the callback object.
- 13: The trigger for follow-on calls is set (on service code).
- 14: A new AppCallLeg is created to receive callbacks for another leg. Alternatively, the already existing AppCallLeg object could be passed in the subsequent createCallLeg(). In that case the application has to use the sessionIDs of the legs to distinguish between callbacks destined for the A-leg and callbacks destined for the B-leg.
- 15: This message is used to create a new call leg object. The object is created in the idle state and not yet routed in the network.
- 16: The application requests to be notified when the leg is answered.
- 17: The application routes the leg. As a result the network will try to reach the associated party.
- 18: When the B-party answers the call, the application is notified.
- 19: The event is forwarded to the application logic.
- 20: Legs that are created and routed explicitly are by default in state detached. This means that the media is not connected to the other parties in the call. In order to allow inband communication between the new party and the other parties in the call the media have to be explicitly attached.
- 21: At some time during the call the calling party enters '#5'. This causes this message to be sent to the object implementing the IpAppCallLeg interface, which forwards this event as a message (not shown) to the IpAppLogic.
- 22: The event is forwarded to the application.
- 23: This message releases the called party.
- 24: Another user interaction dialogue is invoked.
- 25: The result of the dialogue, which in this case is the new destination address is returned and eventually forwarded via another message (not shown) to the IpAppLogic.
- 26: A new AppCallLeg is created to receive callbacks for another leg.
- 27: The call is then forward routed to the new destination party.
- 28: As a result a new Callleg object is created.
- 29: This message passes the result of the call being answered to its callback object and is eventually forwarded via another message (not shown) to the IpAppLogic.

- 30: When the A-party terminates the application is informed.
- 31: The event is forwarded to the application logic.
- 32: Since the release of the A-party will in this case terminate the entire call, the application is also notified with this message.
- 33: The event is forwarded to the application logic.
- 34: Since the user interaction object were not released at the moment that the call terminated, the application receives this message to indicate that the UI resources are released in the gateway and no further communication is possible.
- 35: The event is forwarded to the application logic.
- 36: The application deassigns the call object.

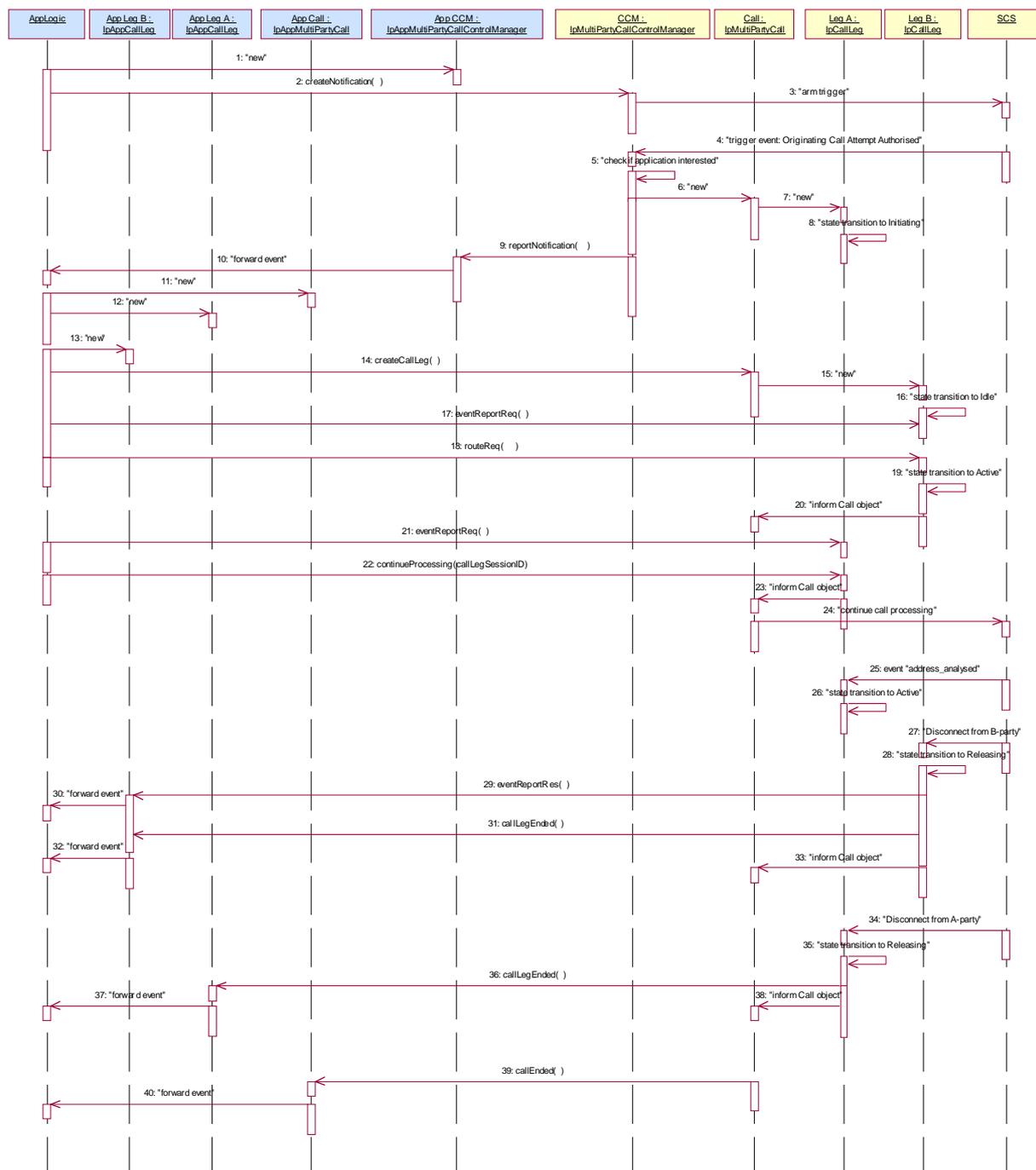
### 7.1.6 Hotline Service

The following sequence diagram shows an application establishing a call between party A and pre-arranged party B defined to constitute a hot-line address. The address of the destination party is provided by the application as the calling party makes a call attempt (goes off-hook) and do not dial any number within a predefined time. In this case a pre-defined number (hot-line number) is provided by the application. The call is then routed to the pre-defined destination party.

The call release is monitored to enable the sending of information to the application at call release, e.g. for charging purposes.

Note that this service could be extended as follows:

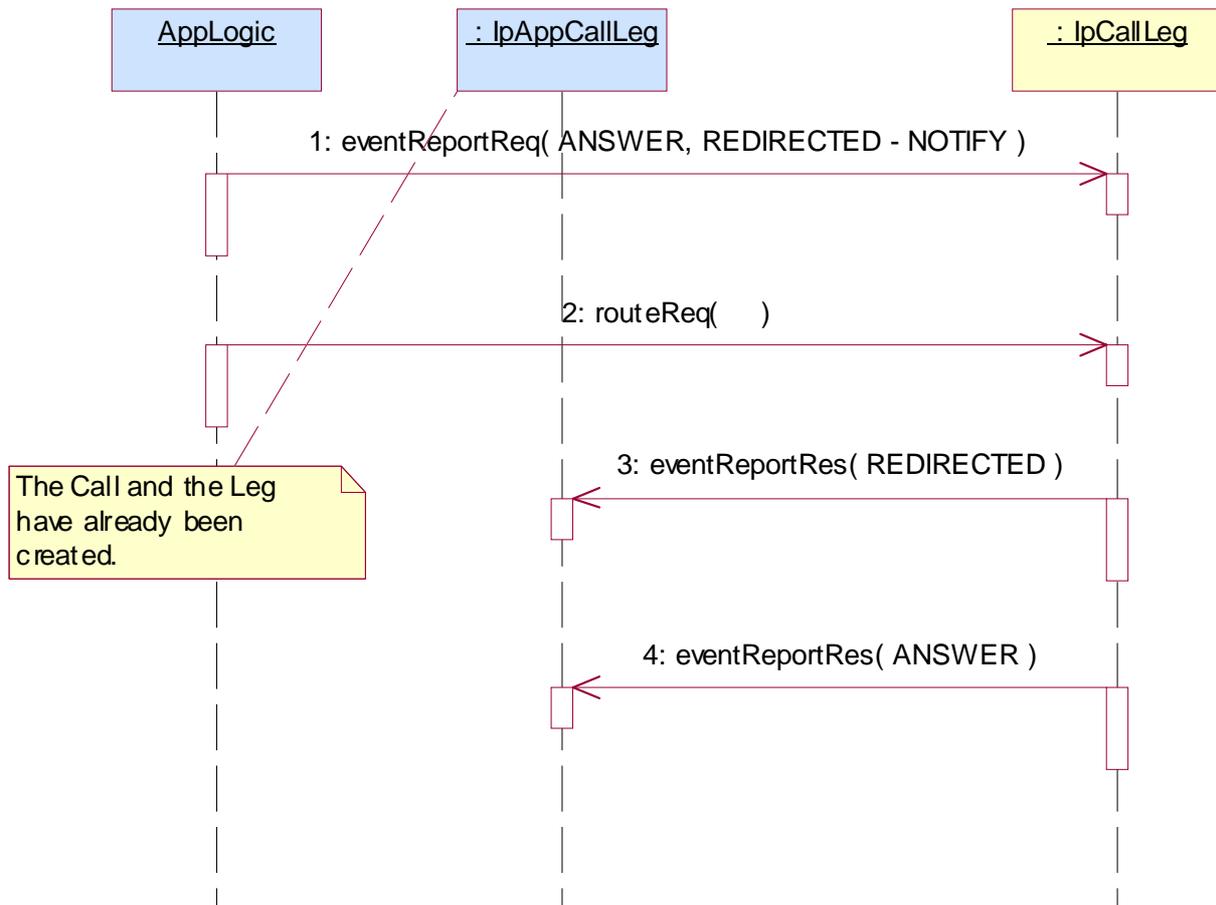
Sometime during the call the calling party enters '#5' which causes the called leg to be released. The calling party is now prompted to enter the address of a new destination party, to which it is then routed.



- 1: This message is used by the application to create an object implementing the IpAppMultiPartyCallControlManager interface.
- 2: This message is sent by the application to enable notifications on new call events.
- 4: When a new call that matches the event criteria arrives, a message ("analysed information") is directed to the object implementing the IpMultiPartyCallControlManager. Assuming that the criteria for creating an object implementing the IpMultiPartyCall interface is met, other messages are used to create the call and associated call leg object.
- 6: A new MultiPartyCall object is created to handle this particular call.
- 7: A new CallLeg object corresponding to Party A is created.

- 8: The new Call Leg instance transits to state Initiating.
- 9: This message is used to pass the new call event to the object implementing the IpAppMultiPartyCallControlManager interface. Applied monitor mode is "interrupt".
- 10: This message is used to forward message 9 to the IpAppLogic.
- 11: This message is used by the application to create an object implementing the IpAppMultiPartyCall interface. The reference to this object is passed back to the object implementing the IpMultiPartyCallControlManager using the return parameter of the reportNotification.
- 12: A new AppCallLeg is created to receive callbacks for the Leg corresponding to party A.
- 13: A new AppCallLeg is created to receive callbacks for another leg.
- 14: This message is used to create a new call leg object. The object is created in the idle state and not yet routed in the network.
- 15: A new CallLeg corresponding to party B is created.
- 16: A transition to state Idle is made after the Call leg has been created.
- 17: The application requests to be notified (monitor mode "NOTIFY") when the leg to party B is released.
- 18: The application requests to route the terminating leg to reach the associated party as specified by the application ("hot-line number").
- 19: The Call Leg instance transits to state Active.
- 21: The application requests to be notified (monitor mode "Notify") when the leg to A-party is released.
- 22: The application requests to resume call processing for the originating call leg.  
  
As a result call processing is resumed in the network that will try to reach the associated party as specified by the application (E.164 number provided by application).
- 25: The originating call leg is notified that the number (provided by application) has been analysed by the network and the originating call leg STD makes a transition to "active" state. The application is not notified as it has not requested this event to be reported.
- 27: When the B-party releases the call, the terminating call leg is notified (monitor mode "NOTIFY") and makes a transition to "Releasing state".
- 29: The application is notified, as the release event has been requested to be reported in Notify mode.
- 30: The event is forwarded to the application logic.
- 31: The terminating call leg is destroyed, the AppLeg B is notified.
- 32: This answer message is then forwarded.
- 34: When the call release ("terminating release" indication) is propagated in the network toward the party A, the originating call leg is notified and makes a transition to "releasing state". This release event (being propagated from party B) is not reported to the application.
- 36: When the originating call leg is destroyed, the AppLeg A is notified.
- 37: The event is forwarded to the application logic.
- 39: When all legs have been destroyed, the IpAppMultiPartyCall is notified that the call is ended.
- 40: The event is forwarded to the application logic.

### 7.1.7 Use of the Redirected event

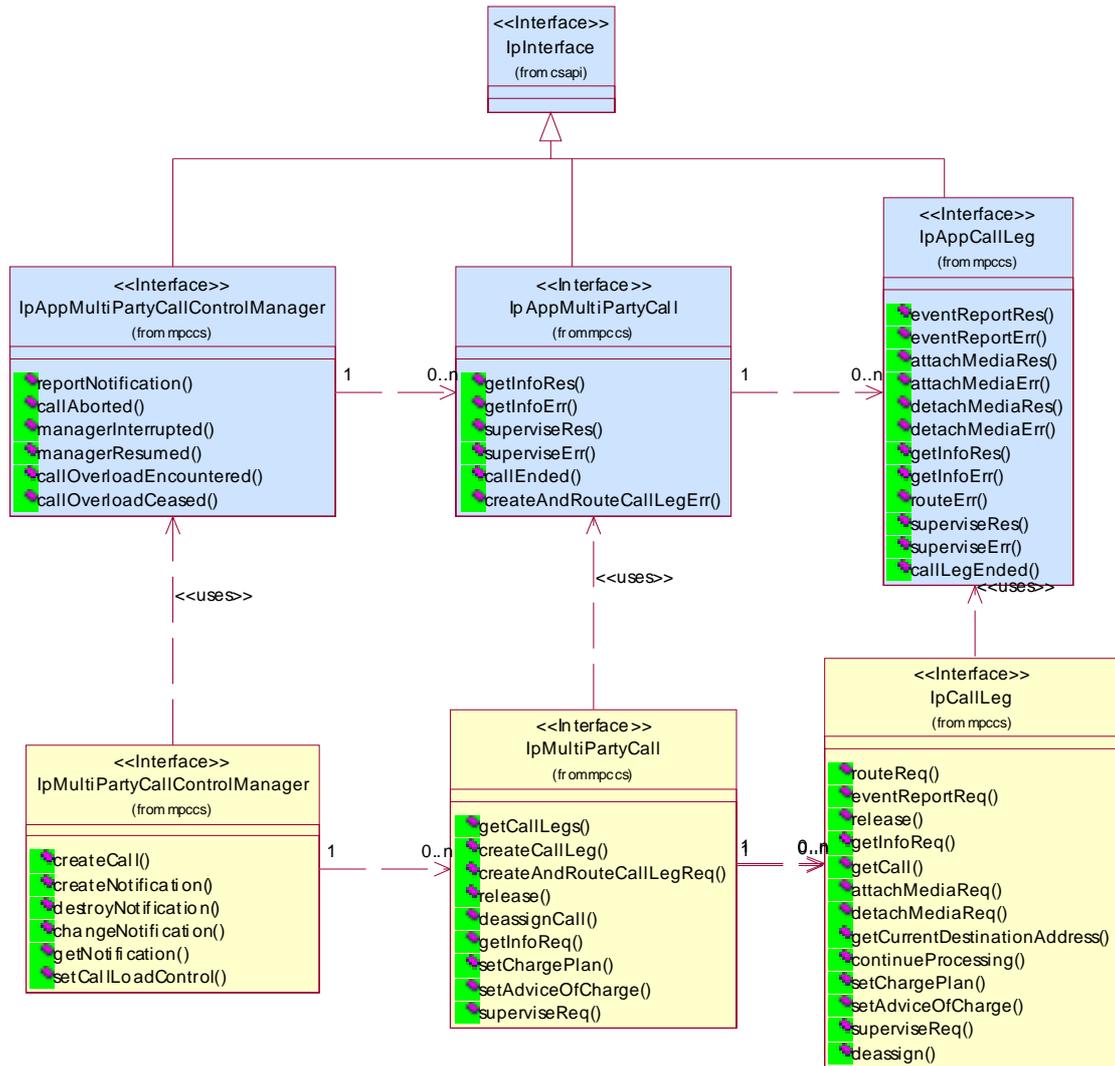


- 1: The application has already created the call and a call leg. It places an event report request for the ANSWER and REDIRECTED events in NOTIFY mode.
- 2: The application routes the call leg.
- 3: The call is redirected within the network and the application is informed. The new destination address is passed within the event. The event is not disarmed, so subsequent redirections will also be reported. Also, the same call leg is used so the application does not have to create a new one.
- 4: The call is answered at its new destination.

## 7.2 Class Diagrams

The multiparty call control service consists of two packages, one for the interfaces on the application side and one for interfaces on the service side.

The class diagrams in the following figures show the interfaces that make up the multi party call control application package and the multi party call control service package. This class diagram shows the interfaces of the multi-party call control application package and their relations to the interfaces of the multi-party call control service package.



**Figure 5: Application Interfaces**

This class diagram shows the interfaces of the multi-party call control service package.

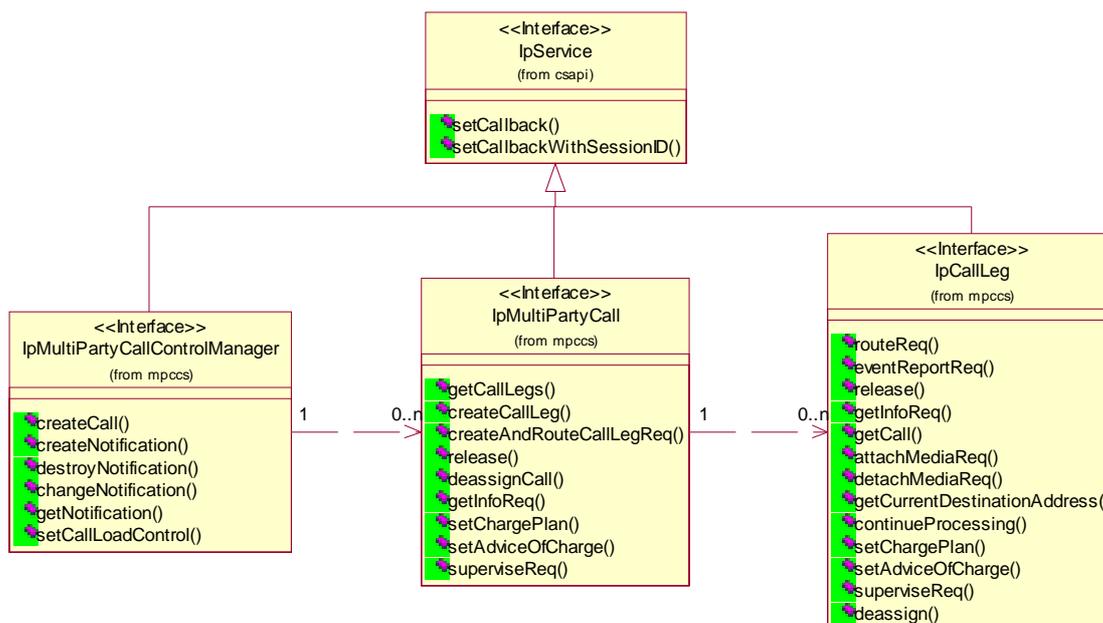


Figure 6: Service Interfaces

## 7.3 MultiParty Call Control Service Interface Classes

The Multi-party Call Control service enhances the functionality of the Generic Call Control Service with leg management. It also allows for multi-party calls to be established, i.e. up to a service specific number of legs can be connected simultaneously to the same call.

The Multi-party Call Control Service is represented by the `IpMultiPartyCallControlManager`, `IpMultiPartyCall`, `IpCallLeg` interfaces that interface to services provided by the network. Some 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. To handle responses and reports, the developer must implement `IpAppMultiPartyCallControlManager`, `IpAppMultiPartyCall` and `IpAppCallLeg` to provide the callback mechanism.

### 7.3.1 Interface Class `IpMultiPartyCallControlManager`

Inherits from: `IpService`.

This interface is the "service manager" interface for the Multi-party Call Control Service. The multi-party call control manager interface provides the management functions to the multi-party call control service. The application programmer can use this interface to provide overload control functionality, create call objects and to enable or disable call-related event notifications. The action table associated with the STD shows in what state the `IpMultiPartyCallControlManager` must be if a method can successfully complete. In other words, if the `IpMultiPartyCallControlManager` is in another state the method will throw an exception immediately. This interface shall be implemented by a Multi Party Call Control SCF. As a minimum requirement either the `createCall()` method shall be implemented, or the `createNotification()` and `destroyNotification()` methods shall be implemented.

<<Interface>> <b>IpMultiPartyCallControlManager</b>
<pre> createCall (appCall : in IpAppMultiPartyCallRef) : TpMultiPartyCallIdentifier createNotification (appCallControlManager : in IpAppMultiPartyCallControlManagerRef, notificationRequest :   in TpCallNotificationRequest) : TpAssignmentID destroyNotification (assignmentID : in TpAssignmentID) : void changeNotification (assignmentID : in TpAssignmentID, notificationRequest : in TpCallNotificationRequest) :   void getNotification () : TpNotificationRequestedSet setCallLoadControl (duration : in TpDuration, mechanism : in TpCallLoadControlMechanism, treatment : in   TpCallTreatment, addressRange : in TpAddressRange) : TpAssignmentID         </pre>

*Method***createCall()**

This method is used to create a new call object. An IpAppMultiPartyCallControlManager should already have been passed to the IpMultiPartyCallControlManager, otherwise the call control will not be able to report a callAborted() to the application. The application should invoke setCallback() prior to createCall() if it wishes to ensure this.

Returns callReference: Specifies the interface reference and sessionID of the call created.

*Parameters*

**appCall : in IpAppMultiPartyCallRef**

Specifies the application interface for callbacks from the call created.

*Returns*

**TpMultiPartyCallIdentifier**

*Raises*

**TpCommonExceptions, P\_INVALID\_INTERFACE\_TYPE**

*Method***createNotification()**

This method is used to enable call notifications so that events can be sent to the application. This is the first step an application has to do to get initial notifications of calls happening in the network. When such an event happens, the application will be informed by reportNotification(). In case the application is interested in other events during the context of a particular call session it has to use the createAndRouteCallLegReq() method on the call object or the eventReportReq() method on the call leg object. The application will get access to the call object when it receives the reportNotification(). (Note that createNotification() is not applicable if the call is setup by the application).

The createNotification method is purely intended for applications to indicate their interest to be notified when certain call events take place. It is possible to subscribe to a certain event for a whole range of addresses, e.g. the application can indicate it wishes to be informed when a call is made to any number starting with 800.

If some application already requested notifications with criteria that overlap the specified criteria, the request is refused with P\_INVALID\_CRITERIA. The criteria are said to overlap if both originating and terminating ranges overlap and the same number plan is used.

If a notification is requested by an application with monitor mode set to notify, then there is no need to check the rest of the criteria for overlapping with any existing request as the notify mode does not allow control on a call to be passed over. Only one application can place an interrupt request if the criteria overlaps.

Set of the callback reference:

The call back reference can be registered either in a) createNotification() or b) explicitly with a setCallback() method e.g. depending on how the application provides its callback reference.

Case a:

From an efficiency point of view the createNotification() with explicit registration may be the preferred method.

Case b:

The createNotification() with no call back reference ("Null" value) is used where (e.g. due to distributed application logic) the call back reference is provided subsequently in a setCallback().

In case the createNotification() contains no callback, at the moment the application needs to be informed the gateway will use as callback the callback that has been registered by setCallback().

Set additional Call back:

If the same application requests two notifications with exactly the same criteria but different callback references, the second callback will be treated as an additional callback. Both notifications will share the same assignmentID. The gateway will always use the most recent callback. In case this most recent callback fails the second most recent is used. In case the createNotification contains no callback, at the moment the application needs to be informed the gateway will use as callback the callback that has been registered by setCallback().

Returns assignmentID: Specifies the ID assigned by the call control manager interface for this newly-enabled event notification.

### *Parameters*

**appCallControlManager : in IpAppMultiPartyCallControlManagerRef**

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.

**notificationRequest : in TpCallNotificationRequest**

Specifies the event specific criteria used by the application to define the event required. Only events that meet these criteria are reported. Examples of events are "incoming call attempt reported by network", "answer", "no answer", "busy". Individual addresses or address ranges may be specified for destination and/or origination.

### *Returns*

**TpAssignmentID**

### *Raises*

**TpCommonExceptions, P\_INVALID\_CRITERIA, P\_INVALID\_INTERFACE\_TYPE, P\_INVALID\_EVENT\_TYPE**

### *Method*

**destroyNotification( )**

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

*Parameters***assignmentID : in TpAssignmentID**

Specifies the assignment ID given by the generic call control manager interface when the previous enableNotification() was called. If the assignment ID does not correspond to one of the valid assignment IDs, the exception P\_INVALID\_ASSIGNMENTID will be raised. If two callbacks have been registered under this assignment ID both of them will be disabled.

*Raises***TpCommonExceptions, P\_INVALID\_ASSIGNMENT\_ID***Method***changeNotification()**

This method is used by the application to change the event criteria introduced with createNotification. Any stored criteria associated with the specified assignmentID will be replaced with the specified criteria.

*Parameters***assignmentID : in TpAssignmentID**

Specifies the ID assigned by the generic call control manager interface for the event notification. If two callbacks have been registered under this assignment ID both of them will be changed.

**notificationRequest : in TpCallNotificationRequest**

Specifies the new set of event specific criteria used by the application to define the event required. Only events that meet these criteria are reported.

*Raises***TpCommonExceptions, P\_INVALID\_ASSIGNMENT\_ID, P\_INVALID\_CRITERIA, P\_INVALID\_EVENT\_TYPE***Method***getNotification()**

This method is used by the application to query the event criteria set with createNotification or changeNotification.

Returns notificationsRequested: Specifies the notifications that have been requested by the application.

*Parameters*

No Parameters were identified for this method.

*Returns***TpNotificationRequestedSet***Raises***TpCommonExceptions***Method***setCallLoadControl()**

This method imposes or removes load control on calls made to a particular address range within the call control service. The address matching mechanism is similar as defined for TpCallEventCriteria.

Returns assignmentID: Specifies the assignmentID assigned by the gateway to this request. This assignmentID can be used to correlate the callOverloadEncountered and callOverloadCeased methods with the request.

*Parameters***duration : in TpDuration**

Specifies the duration for which the load control should be set.

A duration of 0 indicates that the load control should be removed.

A duration of -1 indicates an infinite duration (i.e. until disabled by the application).

A duration of -2 indicates the network default duration.

**mechanism : in TpCallLoadControlMechanism**

Specifies the load control mechanism to use (for example, admit one call per interval), and any necessary parameters, such as the call admission rate. The contents of this parameter are ignored if the load control duration is set to zero.

**treatment : in TpCallTreatment**

Specifies the treatment of calls that are not admitted. The contents of this parameter are ignored if the load control duration is set to zero.

**addressRange : in TpAddressRange**

Specifies the address or address range to which the overload control should be applied or removed.

*Returns*

**TpAssignmentID**

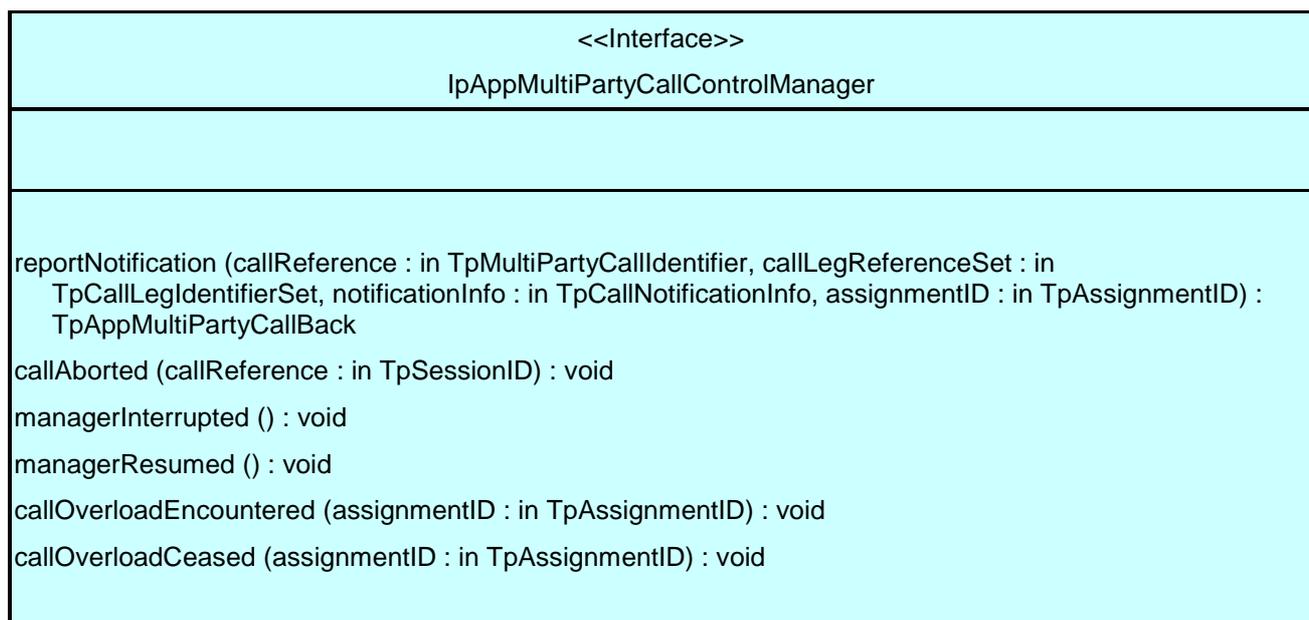
*Raises*

**TpCommonExceptions, P\_INVALID\_ADDRESS, P\_UNSUPPORTED\_ADDRESS\_PLAN**

### 7.3.2 Interface Class IpAppMultiPartyCallControlManager

Inherits from: IpInterface.

The Multi-Party call control manager application interface provides the application call control management functions to the Multi-Party call control service.



*Method***reportNotification()**

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

If this method is invoked with a monitor mode of P\_CALL\_MONITOR\_MODE\_INTERRUPT, then the APL has control of the call. If the APL does nothing with the call (including its associated legs) within a specified time period (the duration of which forms a part of the service level agreement), then the call in the network shall be released and callEnded() shall be invoked, giving a release cause of P\_TIMER\_EXPIRY.

Set of the callback reference:

A reference to the application interface has to be passed back to the call interface to which the notification relates. However, the setting of a call back reference is only applicable if the notification is in INTERRUPT mode.

The call back reference can be registered either in a) reportNotification() or b) explicitly with a setCallbackWithSessionID() method depending on how the application provides its callback reference.

Case a:

From an efficiency point of view the reportNotification() with explicit pass of registration may be the preferred method.

Case b:

The reportNotification() with no call back reference ("Null" value) is used where (e.g. due to distributed application logic) the call back reference is provided subsequently in a setCallbackWithSessionID().

In case reportNotification() contains no callback, at the moment the application needs to be informed the gateway will use as callback the callback that has been registered by setCallbackWithSessionID().

Returns appCallBack: Specifies references to the application interface which implements the callback interface for the new call and/or new call leg. If the application has previously explicitly passed a reference to the callback interface using a setCallbackWithSessionID() invocation, this parameter may be set to P\_APP\_CALLBACK\_UNDEFINED, or if supplied must be the same as that provided during the setCallbackWithSessionID().

This parameter will be set to P\_APP\_CALLBACK\_UNDEFINED if the notification is in NOTIFY mode and in case b.

*Parameters***callReference : in TpMultiPartyCallIdentifier**

Specifies the reference to the call interface to which the notification relates. If the notification is being given in NOTIFY mode, this parameter shall be ignored by the application client implementation, and consequently the implementation of the SCS entity invoking reportNotification may populate this parameter as it chooses.

**callLegReferenceSet : in TpCallLegIdentifierSet**

Specifies the set of all call leg references. First in the set is the reference to the originating callLeg. It indicates the call leg related to the originating party. In case there is a destination call leg this will be the second leg in the set. from the notificationInfo can be found on whose behalf the notification was sent.

However, if the notification is being given in NOTIFY mode, this parameter shall be ignored by the application client implementation, and consequently the implementation of the SCS entity invoking reportNotification may populate this parameter as it chooses.

**notificationInfo : in TpCallNotificationInfo**

Specifies data associated with this event (e.g. the originating or terminating leg which reports the notification).

**assignmentID : in TpAssignmentID**

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

*Returns***TpAppMultiPartyCallBack***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 and application.

*Parameters***callReference : in TpSessionID**

Specifies the sessionID of call that has aborted or terminated abnormally.

*Method***managerInterrupted()**

This method indicates to the application that event notifications and method invocations have been temporarily interrupted (for example, due to network resources unavailable).

Note that more permanent failures are reported via the Framework (integrity management).

*Parameters*

No Parameters were identified for this method.

*Method***managerResumed()**

This method indicates to the application that event notifications are possible and method invocations are enabled.

*Parameters*

No Parameters were identified for this method.

*Method***callOverloadEncountered()**

This method indicates that the network has detected overload and may have automatically imposed load control on calls requested to a particular address range or calls made to a particular destination within the call control service.

*Parameters***assignmentID : in TpAssignmentID**

Specifies the assignmentID corresponding to the associated setCallLoadControl. This implies the addressrange for within which the overload has been encountered.

*Method***callOverloadCeased()**

This method indicates that the network has detected that the overload has ceased and has automatically removed any load controls on calls requested to a particular address range or calls made to a particular destination within the call control service.

*Parameters***assignmentID : in TpAssignmentID**

Specifies the assignmentID corresponding to the associated setCallLoadControl. This implies the addressrange for within which the overload has been ceased.

### 7.3.3 Interface Class IpMultiPartyCall

Inherits from: IpService.

The Multi-Party Call provides the possibility to control the call routing, to request information from the call, control the charging of the call, to release the call and to supervise the call. It also gives the possibility to manage call legs explicitly. An application may create more than one call leg.

This interface shall be implemented by a Multi Party Call Control SCF. The release() and deassignCall() methods, and either the createCallLeg() or the createAndRouteCallLegReq(), shall be implemented as a minimum requirement.

<<Interface>> IpMultiPartyCall
<pre> getCallLegs (callSessionID : in TpSessionID) : TpCallLegIdentifierSet createCallLeg (callSessionID : in TpSessionID, appCallLeg : in IpAppCallLegRef) : TpCallLegIdentifier createAndRouteCallLegReq (callSessionID : in TpSessionID, eventsRequested : in TpCallEventRequestSet,   targetAddress : in TpAddress, originatingAddress : in TpAddress, appInfo : in TpCallAppInfoSet,   appLegInterface : in IpAppCallLegRef) : TpCallLegIdentifier release (callSessionID : in TpSessionID, cause : in TpReleaseCause) : void deassignCall (callSessionID : in TpSessionID) : void getInfoReq (callSessionID : in TpSessionID, callInfoRequested : in TpCallInfoType) : void setChargePlan (callSessionID : in TpSessionID, callChargePlan : in TpCallChargePlan) : void setAdviceOfCharge (callSessionID : in TpSessionID, aOCInfo : in TpAoCInfo, tariffSwitch : in TpDuration) :   void superviseReq (callSessionID : in TpSessionID, time : in TpDuration, treatment : in   TpCallSuperviseTreatment) : void </pre>

#### Method

#### **getCallLegs()**

This method requests the identification of the call leg objects associated with the call object. Returns the legs in the order of creation.

Returns callLegList: Specifies the call legs associated with the call. The set contains both the sessionIDs and the interface references.

#### Parameters

**callSessionID : in TpSessionID**

Specifies the call session ID of the call.

#### Returns

**TpCallLegIdentifierSet**

#### Raises

**TpCommonExceptions, P\_INVALID\_SESSION\_ID**

*Method***createCallLeg()**

This method requests the creation of a new call leg object.

Returns callLeg: Specifies the interface and sessionID of the call leg created.

*Parameters*

**callSessionID : in TpSessionID**

Specifies the call session ID of the call.

**appCallLeg : in IpAppCallLegRef**

Specifies the application interface for callbacks from the call leg created.

*Returns*

**TpCallLegIdentifier**

*Raises*

**TpCommonExceptions, P\_INVALID\_SESSION\_ID, P\_INVALID\_INTERFACE\_TYPE**

*Method***createAndRouteCallLegReq()**

This asynchronous operation requests creation and routing of a new callLeg. In case the connection to the destination party is established successfully the CallLeg is attached to the call, i.e. no explicit attachMediaReq() operation is needed. Requested events will be reported on the IpAppCallLeg interface. This interface the application must provide through the appLegInterface parameter.

The extra address information such as originatingAddress is optional. If not present (i.e. the plan is set to P\_ADDRESS\_PLAN\_NOT\_PRESENT), the information provided in corresponding addresses from the route is used, otherwise the network or gateway provided numbers will be used.

If the application wishes that the call leg should be represented in the network as being a redirection it should include a value for the field P\_CALL\_APP\_ORIGINAL\_DESTINATION\_ADDRESS of TpCallAppInfo.

If this method is invoked, and call reports have been requested, yet the IpAppCallLeg interface parameter is NULL, this method shall throw the P\_NO\_CALLBACK\_ADDRESS\_SET exception.

Returns callLegReference: Specifies the reference to the CallLeg interface that was created.

*Parameters*

**callSessionID : in TpSessionID**

Specifies the call session ID of the call.

**eventsRequested : in TpCallEventRequestSet**

Specifies the event specific criteria used by the application to define the events required. Only events that meet these criteria are reported. Examples of events are "address analysed", "answer" and "release".

**targetAddress : in TpAddress**

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

**originatingAddress : in TpAddress**

Specifies the address of the originating (calling) party.

**appInfo : in TpCallAppInfoSet**

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

**appLegInterface : in IpAppCallLegRef**

Specifies a reference to the application interface that implements the callback interface for the new call leg. Requested events will be reported by the eventReportRes() operation on this interface.

*Returns*

**TpCallLegIdentifier**

*Raises*

**TpCommonExceptions, P\_INVALID\_SESSION\_ID, P\_INVALID\_INTERFACE\_TYPE, P\_INVALID\_ADDRESS, P\_UNSUPPORTED\_ADDRESS\_PLAN, P\_INVALID\_NETWORK\_STATE, P\_INVALID\_EVENT\_TYPE, P\_INVALID\_CRITERIA**

*Method***release()**

This method requests the release of the call object and associated objects. The call will also be terminated in the network. If the application requested reports to be sent at the end of the call (e.g. by means of getInfoReq) these reports will still be sent to the application.

*Parameters*

**callSessionID : in TpSessionID**

Specifies the call session ID of the call.

**cause : in TpReleaseCause**

Specifies the cause of the release.

*Raises*

**TpCommonExceptions, P\_INVALID\_SESSION\_ID, P\_INVALID\_NETWORK\_STATE**

*Method***deassignCall()**

This method requests that the relationship between the application and the call and associated objects 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 call information reports, call leg event reports or call Leg information reports requested, then these reports will be disabled and any related information discarded.

*Parameters*

**callSessionID : in TpSessionID**

Specifies the call session ID of the call.

*Raises*

**TpCommonExceptions, P\_INVALID\_SESSION\_ID**

*Method***getInfoReq()**

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.

A report is received when the destination leg or party terminates or when the call ends. 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. In case the originating party is still available the application can still initiate a follow-on call using routeReq.

*Parameters*

**callSessionID** : in TpSessionID

Specifies the call session ID of the call.

**callInfoRequested** : in TpCallInfoType

Specifies the call information that is requested.

*Raises*

**TpCommonExceptions, P\_INVALID\_SESSION\_ID**

*Method***setChargePlan()**

Set an operator specific charge plan for the call.

*Parameters*

**callSessionID** : in TpSessionID

Specifies the call session ID of the call.

**callChargePlan** : in TpCallChargePlan

Specifies the charge plan to use.

*Raises*

**TpCommonExceptions, P\_INVALID\_SESSION\_ID**

*Method***setAdviceOfCharge()**

This method allows for advice of charge (AOC) information to be sent to terminals that are capable of receiving this information.

*Parameters*

**callSessionID** : in TpSessionID

Specifies the call session ID of the call.

**aOCInfo** : in TpAoCInfo

Specifies two sets of Advice of Charge parameter.

**tariffSwitch** : in TpDuration

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

*Raises*

**TpCommonExceptions**, **P\_INVALID\_SESSION\_ID**, **P\_INVALID\_CURRENCY**,  
**P\_INVALID\_AMOUNT**

*Method***superviseReq()**

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 operation before it routes a call or a user interaction operation the time measurement will start as soon as the call is answered by the B-party or the user interaction system.

*Parameters*

**callSessionID** : in **TpSessionID**

Specifies the call session ID of the call.

**time** : in **TpDuration**

Specifies the granted time in milliseconds for the connection.

**treatment** : in **TpCallSuperviseTreatment**

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

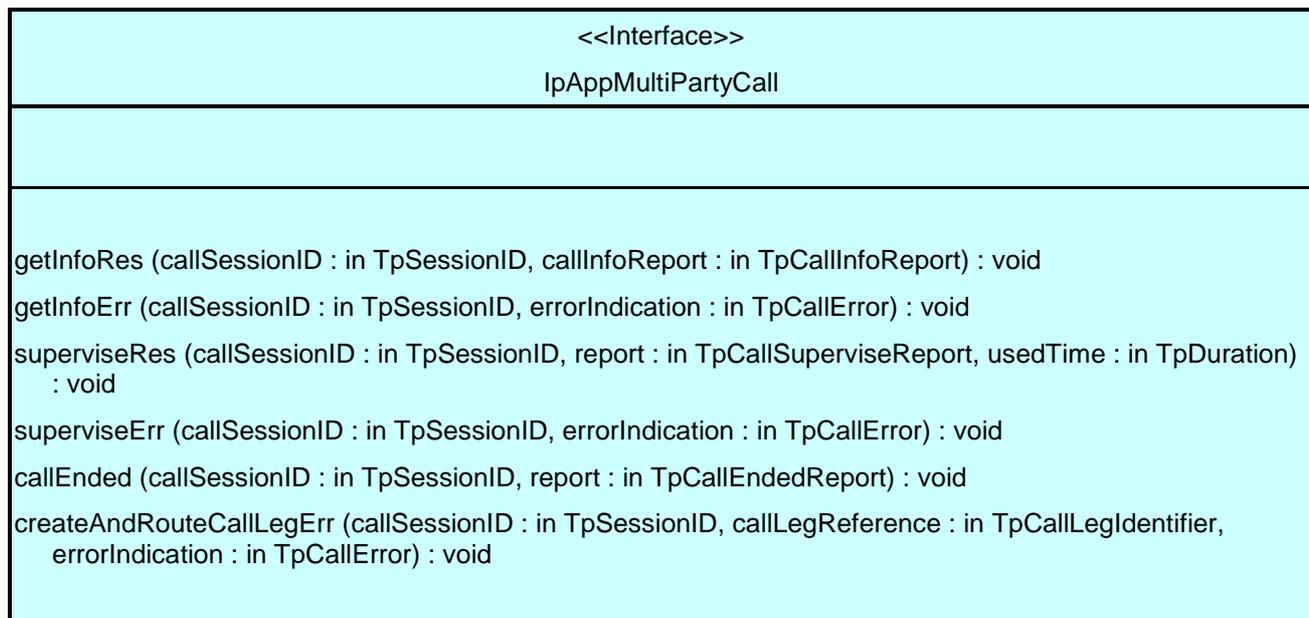
*Raises*

**TpCommonExceptions**, **P\_INVALID\_SESSION\_ID**

### 7.3.4 Interface Class IpAppMultiPartyCall

Inherits from: IpInterface.

The Multi-Party call application interface is implemented by the client application developer and is used to handle call request responses and state reports.



*Method***getInfoRes()**

This asynchronous method reports time information of the finished call or call attempt as well as release cause depending on which information has been requested by getInfoReq. This information may be used e.g. for charging purposes. The call information will possibly be sent after reporting of all cases where the call or a leg of the call has been disconnected or a routing failure has been encountered.

*Parameters*

**callSessionID : in TpSessionID**

Specifies the call session ID of the call.

**callInfoReport : in TpCallInfoReport**

Specifies the call information requested.

*Method***getInfoErr()**

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

*Parameters*

**callSessionID : in TpSessionID**

Specifies the call session ID of the call.

**errorIndication : in TpCallError**

Specifies the error which led to the original request failing.

*Method***superviseRes()**

This asynchronous method reports a call supervision event to the application when it has indicated its interest in these kind of events.

It is also called when the connection is terminated before the supervision event occurs.

*Parameters*

**callSessionID : in TpSessionID**

Specifies the call session ID of the call

**report : in TpCallSuperviseReport**

Specifies the situation which triggered the sending of the call supervision response.

**usedTime : in TpDuration**

Specifies the used time for the call supervision (in milliseconds).

*Method***superviseErr()**

This asynchronous method reports a call supervision error to the application.

*Parameters***callSessionID : in TpSessionID**

Specifies the call session ID of the call.

**errorIndication : in TpCallError**

Specifies the error which led to the original request failing.

*Method***callEnded( )**

This method indicates to the application that the call has terminated in the network.

Note that the event that caused the call to end might have been received separately if the application was monitoring for it.

*Parameters***callSessionID : in TpSessionID**

Specifies the call sessionID.

**report : in TpCallEndedReport**

Specifies the reason the call is terminated.

*Method***createAndRouteCallLegErr( )**

This asynchronous method indicates that the request to route the call leg to the destination party was unsuccessful - the call leg could not be routed to the destination party (for example, the network was unable to route the call leg, the parameters were incorrect, the request was refused, etc.). Note that the event cases that can be monitored and correspond to an unsuccessful setup of a connection (e.g. busy, no\_answer) will be reported by eventReportRes() and not by this operation.

*Parameters***callSessionID : in TpSessionID**

Specifies the call session ID of the call.

**callLegReference : in TpCallLegIdentifier**

Specifies the reference to the CallLeg interface that was created.

**errorIndication : in TpCallError**

Specifies the error which led to the original request failing.

### 7.3.5 Interface Class IpCallLeg

Inherits from: IpService.

The call leg interface represents the logical call leg associating a call with an address. The call leg tracks its own states and allows charging summaries to be accessed. The leg represents the signalling relationship between the call and an address. An application that uses the IpCallLeg interface to set up connections has good control, e.g. by defining leg specific event request and can obtain call leg specific report and events.

This interface shall be implemented by a Multi Party Call Control SCF. The routeReq(), eventReportReq(), release(), continueProcessing() and deassign() methods shall be implemented as a minimum requirement.

<<Interface>> IpCallLeg
<pre> routeReq (callLegSessionID : in TpSessionID, targetAddress : in TpAddress, originatingAddress : in   TpAddress, applInfo : in TpCallAppInfoSet, connectionProperties : in TpCallLegConnectionProperties) :   void eventReportReq (callLegSessionID : in TpSessionID, eventsRequested : in TpCallEventRequestSet) : void release (callLegSessionID : in TpSessionID, cause : in TpReleaseCause) : void getInfoReq (callLegSessionID : in TpSessionID, callLegInfoRequested : in TpCallLegInfoType) : void getCall (callLegSessionID : in TpSessionID) : TpMultiPartyCallIdentifier attachMediaReq (callLegSessionID : in TpSessionID) : void detachMediaReq (callLegSessionID : in TpSessionID) : void getCurrentDestinationAddress (callLegSessionID : in TpSessionID) : TpAddress continueProcessing (callLegSessionID : in TpSessionID) : void setChargePlan (callLegSessionID : in TpSessionID, callChargePlan : in TpCallChargePlan) : void setAdviceOfCharge (callLegSessionID : in TpSessionID, aOCInfo : in TpAoCInfo, tariffSwitch : in TpDuration)   : void superviseReq (callLegSessionID : in TpSessionID, time : in TpDuration, treatment : in   TpCallLegSuperviseTreatment) : void deassign (callLegSessionID : in TpSessionID) : void           </pre>

### *Method*

#### **routeReq( )**

This asynchronous method requests routing of the call leg to the remote party indicated by the targetAddress.

In case the connection to the destination party is established successfully the CallLeg will be either detached or attached to the call based on the attach Mechanism values specified in the connectionProperties parameter.

The extra address information such as originatingAddress is optional. If not present (i.e. the plan is set to P\_ADDRESS\_PLAN\_NOT\_PRESENT), the information provided in the corresponding addresses from the route is used, otherwise network or gateway provided addresses will be used.

If the application wishes that the call leg should be represented in the network as being a redirection it should include a value for the field P\_CALL\_APP\_ORIGINAL\_DESTINATION\_ADDRESS of TpCallAppInfo.

This operation continues processing of the call leg.

### *Parameters*

#### **callLegSessionID : in TpSessionID**

Specifies the call leg session ID of the call leg.

#### **targetAddress : in TpAddress**

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

**originatingAddress : in TpAddress**

Specifies the address of the originating (calling) party.

**appInfo : in TpCallAppInfoSet**

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

**connectionProperties : in TpCallLegConnectionProperties**

Specifies the properties of the connection.

*Raises*

**TpCommonExceptions, P\_INVALID\_SESSION\_ID, P\_INVALID\_NETWORK\_STATE, P\_INVALID\_ADDRESS, P\_UNSUPPORTED\_ADDRESS\_PLAN**

*Method***eventReportReq( )**

This asynchronous method sets, clears or changes the criteria for the events that the call leg object will be set to observe.

*Parameters***callLegSessionID : in TpSessionID**

Specifies the call leg session ID of the call leg.

**eventsRequested : in TpCallEventRequestSet**

Specifies the event specific criteria used by the application to define the events required. Only events that meet these criteria are reported. Examples of events are "address analysed", "answer" and "release".

*Raises*

**TpCommonExceptions, P\_INVALID\_SESSION\_ID, P\_INVALID\_EVENT\_TYPE, P\_INVALID\_CRITERIA**

*Method***release( )**

This method requests the release of the call leg. If successful, the associated address (party) will be released from the call, and the call leg deleted. Note that in some cases releasing the party may lead to release of the complete call in the network. The application will be informed of this with callEnded().

This operation continues processing of the call leg.

*Parameters***callLegSessionID : in TpSessionID**

Specifies the call leg session ID of the call leg.

**cause : in TpReleaseCause**

Specifies the cause of the release.

*Raises*

**TpCommonExceptions, P\_INVALID\_SESSION\_ID, P\_INVALID\_NETWORK\_STATE**

*Method***getInfoReq()**

This asynchronous method requests information associated with the call leg to be provided at the appropriate time (for example, to calculate charging). Note that in the call leg information must be accessible before the objects of concern are deleted.

*Parameters*

**callLegSessionID : in TpSessionID**

Specifies the call leg session ID of the call leg.

**callLegInfoRequested : in TpCallLegInfoType**

Specifies the call leg information that is requested.

*Raises*

**TpCommonExceptions, P\_INVALID\_SESSION\_ID**

*Method***getCall()**

This method requests the call associated with this call leg.

Returns callReference: Specifies the interface and sessionID of the call associated with this call leg.

*Parameters*

**callLegSessionID : in TpSessionID**

Specifies the call leg session ID of the call leg.

*Returns*

**TpMultiPartyCallIdentifier**

*Raises*

**TpCommonExceptions, P\_INVALID\_SESSION\_ID**

*Method***attachMediaReq()**

This method requests that the call leg be attached to its call object. This will allow transmission on all associated bearer connections or media streams to and from other parties in the call. The call leg must be in the connected state for this method to complete successfully.

*Parameters*

**callLegSessionID : in TpSessionID**

Specifies the sessionID of the call leg to attach to the call.

*Raises*

**TpCommonExceptions, P\_INVALID\_SESSION\_ID, P\_INVALID\_NETWORK\_STATE**

*Method***detachMediaReq( )**

This method will detach the call leg from its call, i.e. this will prevent transmission on any associated bearer connections or media streams to and from other parties in the call. The call leg must be in the connected state for this method to complete successfully.

*Parameters*

**callLegSessionID : in TpSessionID**

Specifies the sessionID of the call leg to detach from the call.

*Raises*

**TpCommonExceptions, P\_INVALID\_SESSION\_ID, P\_INVALID\_NETWORK\_STATE**

*Method***getCurrentDestinationAddress( )**

Queries the current address of the destination the leg has been directed to.

Returns the address of the destination point towards which the call leg has been routed.

If this method is invoked on the Originating Call Leg, exception P\_INVALID\_STATE will be thrown.

*Parameters*

**callLegSessionID : in TpSessionID**

Specifies the call session ID of the call leg.

*Returns*

**TpAddress**

*Raises*

**TpCommonExceptions, P\_INVALID\_SESSION\_ID**

*Method***continueProcessing( )**

This operation continues processing of the call leg. Applications can invoke this operation after call leg processing was interrupted due to detection of a notification or event the application subscribed its interest in.

*Parameters*

**callLegSessionID : in TpSessionID**

Specifies the call leg session ID of the call leg.

*Raises*

**TpCommonExceptions, P\_INVALID\_SESSION\_ID, P\_INVALID\_NETWORK\_STATE**

*Method***setChargePlan( )**

Set an operator specific charge plan for the call leg.

*Parameters***callLegSessionID : in TpSessionID**

Specifies the call leg session ID of the call party.

**callChargePlan : in TpCallChargePlan**

Specifies the charge plan to use.

*Raises***TpCommonExceptions, P\_INVALID\_SESSION\_ID***Method***setAdviceOfCharge()**

This method allows for advice of charge (AOC) information to be sent to terminals that are capable of receiving this information.

*Parameters***callLegSessionID : in TpSessionID**

Specifies the call leg session ID of the call party.

**aOCInfo : in TpAoCInfo**

Specifies two sets of Advice of Charge parameter.

**tariffSwitch : in TpDuration**

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

*Raises***TpCommonExceptions, P\_INVALID\_SESSION\_ID, P\_INVALID\_CURRENCY,  
P\_INVALID\_AMOUNT***Method***superviseReq()**

The application calls this method to supervise a call leg. The application can set a granted connection time for this call. If an application calls this function before it calls a routeReq() 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.

*Parameters***callLegSessionID : in TpSessionID**

Specifies the call leg session ID of the call party.

**time : in TpDuration**

Specifies the granted time in milliseconds for the connection.

**treatment : in TpCallLegSuperviseTreatment**

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

*Raises***TpCommonExceptions, P\_INVALID\_SESSION\_ID**

*Method***deassign()**

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

The application should not release or deassign the call leg when received a callLegEnded() or callEnded(). This operation continues processing of the call leg.

*Parameters*

**callLegSessionID : in TpSessionID**

Specifies the call leg session ID of the call leg.

*Raises*

**TpCommonExceptions, P\_INVALID\_SESSION\_ID**

### 7.3.6 Interface Class IpAppCallLeg

Inherits from: IpInterface.

The application call leg interface is implemented by the client application developer and is used to handle responses and errors associated with requests on the call leg in order to be able to receive leg specific information and events.

<<Interface>> IpAppCallLeg
<pre> eventReportRes (callLegSessionID : in TpSessionID, eventInfo : in TpCallEventInfo) : void eventReportErr (callLegSessionID : in TpSessionID, errorIndication : in TpCallError) : void attachMediaRes (callLegSessionID : in TpSessionID) : void attachMediaErr (callLegSessionID : in TpSessionID, errorIndication : in TpCallError) : void detachMediaRes (callLegSessionID : in TpSessionID) : void detachMediaErr (callLegSessionID : in TpSessionID, errorIndication : in TpCallError) : void getInfoRes (callLegSessionID : in TpSessionID, callLegInfoReport : in TpCallLegInfoReport) : void getInfoErr (callLegSessionID : in TpSessionID, errorIndication : in TpCallError) : void routeErr (callLegSessionID : in TpSessionID, errorIndication : in TpCallError) : void superviseRes (callLegSessionID : in TpSessionID, report : in TpCallSuperviseReport, usedTime : in   TpDuration) : void superviseErr (callLegSessionID : in TpSessionID, errorIndication : in TpCallError) : void callLegEnded (callLegSessionID : in TpSessionID, cause : in TpReleaseCause) : void           </pre>

*Method***eventReportRes ( )**

This asynchronous method reports that an event has occurred that was requested to be reported (for example, a mid-call event, the party has requested to disconnect, etc.).

Depending on the type of event received, outstanding requests for events are discarded. The exact details of these so-called disarming rules are captured in the data definition of the event type.

If this method is invoked for a report with a monitor mode of P\_CALL\_MONITOR\_MODE\_INTERRUPT, then the application has control of the call leg. If the application does nothing with the call leg within a specified time period (the duration which forms a part of the service level agreement), then the connection in the network shall be released and callLegEnded() shall be invoked, giving a release cause of P\_TIMER\_EXPIRY.

*Parameters*

**callLegSessionID : in TpSessionID**

Specifies the call leg session ID of the call leg on which the event was detected.

**eventInfo : in TpCallEventInfo**

Specifies data associated with this event.

*Method***eventReportErr ( )**

This asynchronous method indicates that the request to manage call leg event reports was unsuccessful, and the reason (for example, the parameters were incorrect, the request was refused, etc.).

*Parameters*

**callLegSessionID : in TpSessionID**

Specifies the call leg session ID of the call leg.

**errorIndication : in TpCallError**

Specifies the error which led to the original request failing.

*Method***attachMediaRes ( )**

This asynchronous method reports the attachment of a call leg to a call has succeeded. The media channels or bearer connections to this leg is now available.

*Parameters*

**callLegSessionID : in TpSessionID**

Specifies the call leg session ID of the call leg to which the information relates.

*Method***attachMediaErr ( )**

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

*Parameters*

**callLegSessionID : in TpSessionID**

Specifies the call leg session ID of the call leg.

**errorIndication : in TpCallError**

Specifies the error which led to the original request failing.

*Method*

**detachMediaRes()**

This asynchronous method reports the detachment of a call leg from a call has succeeded. The media channels or bearer connections to this leg is no longer available.

*Parameters*

**callLegSessionID : in TpSessionID**

Specifies the call leg session ID of the call leg to which the information relates.

*Method*

**detachMediaErr()**

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

*Parameters*

**callLegSessionID : in TpSessionID**

Specifies the call leg session ID of the call leg.

**errorIndication : in TpCallError**

Specifies the error which led to the original request failing.

*Method*

**getInfoRes()**

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

*Parameters*

**callLegSessionID : in TpSessionID**

Specifies the call leg session ID of the call leg to which the information relates.

**callLegInfoReport : in TpCallLegInfoReport**

Specifies the call leg information requested.

*Method*

**getInfoErr()**

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

*Parameters*

**callLegSessionID : in TpSessionID**

Specifies the call leg session ID of the call leg.

**errorIndication : in TpCallError**

Specifies the error which led to the original request failing.

*Method***routeErr()**

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

*Parameters*

**callLegSessionID : in TpSessionID**

Specifies the call leg session ID of the call leg.

**errorIndication : in TpCallError**

Specifies the error which led to the original request failing.

*Method***superviseRes()**

This asynchronous method reports a call leg supervision event to the application when it has indicated its interest in these kind of events.

It is also called when the connection to a party is terminated before the supervision event occurs.

*Parameters*

**callLegSessionID : in TpSessionID**

Specifies the call leg session ID of the call leg.

**report : in TpCallSuperviseReport**

Specifies the situation which triggered the sending of the call leg supervision response.

**usedTime : in TpDuration**

Specifies the used time for the call leg supervision (in milliseconds).

*Method***superviseErr()***Parameters*

**callLegSessionID : in TpSessionID**

Specifies the call leg session ID of the call leg.

**errorIndication : in TpCallError**

Specifies the error which led to the original request failing.

*Method***callLegEnded()**

This method indicates to the application that the leg has terminated in the network. The application has received all requested results (e.g. getInfoRes) related to the call leg. The call leg will be destroyed after returning from this method.

*Parameters*

**callLegSessionID : in TpSessionID**

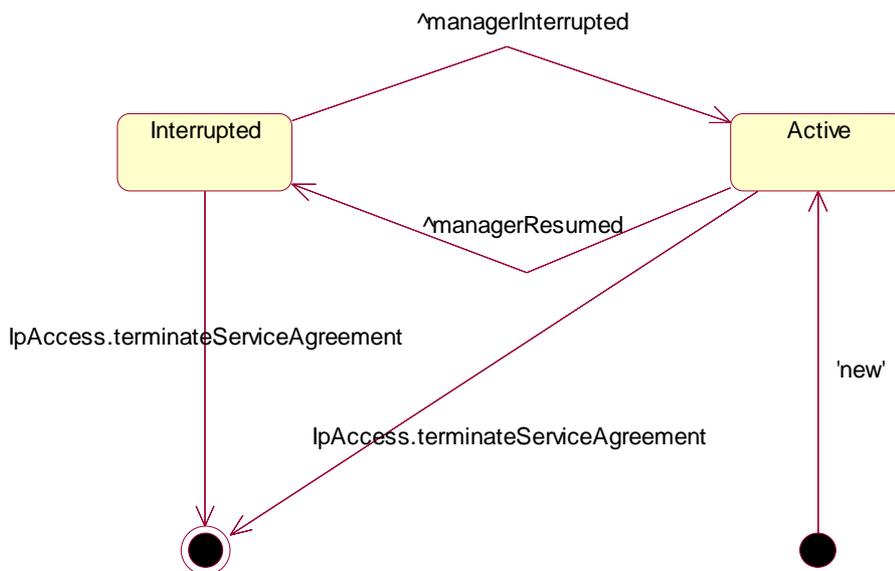
Specifies the call leg session ID of the call leg.

**cause : in TpReleaseCause**

Specifies the reason the connection is terminated.

## 7.4 MultiParty Call Control Service State Transition Diagrams

### 7.4.1 State Transition Diagrams for IpMultiPartyCallControlManager



**Figure 7: Application view and the Multi-Party Call Control Manager**

#### 7.4.1.1 Active State

In this state a relation between the Application and the Service has been established. The state allows the application to indicate that it is interested in call related events. In case such an event occurs, the Manager will create a Call object with the appropriate number of Call Leg objects and inform the application. The application can also indicate it is no longer interested in certain call related events by calling `destroyNotification()`.

#### 7.4.1.2 Interrupted State

When the Manager is in the Interrupted state it is temporarily unavailable for use. Events requested cannot be forwarded to the application and methods in the API cannot successfully be executed. A number of reasons can cause this: for instance the application receives more notifications from the network than defined in the Service Agreement. Another example is that the Service has detected it receives no notifications from the network due to e.g. a link failure.

#### 7.4.1.3 Overview of allowed methods

Call Control Manager State	Methods applicable
Active	createCall, createNotification, destroyNotification, changeNotification, getNotification, setCallLoadControl
Interrupted	getNotification



Furthermore, the application can request for call supervision. The Application can request charging related information reports, set the charge plan and set Advice Of Charge indicators in this state prior to call establishment.

### 7.4.2.3 RELEASED State

In this state the last Call leg object has released or the call itself was released. While the call is in this state, the requested call information will be collected and returned through getInfoRes() and/or superviseRes(). As soon as all information is returned, the application will be informed that the call has ended and Call object transition to the end state.

### 7.4.2.4 Overview of allowed methods

Methods applicable	Call Control Call State	Call Control Manager State
getCallLegs,	Idle, Active, Released	-
createCallLeg, createAndRouteCallLegReq, setAdviceOfCharge, superviseReq, release	Idle, Active	Active
deassignCall	Active	Active
setChargePlan, getInfoReq	Idle, Active	-
		Active

## 7.4.3 State Transition Diagrams for IpCallLeg

The IpCallLeg State Transition Diagram is divided in two State Transition Diagrams, one for the originating call leg and one for the terminating call leg.

Call Leg State Model General Objectives:

- 1) Events in backwards direction (upstream), coming from terminating leg, are not visible in originating leg model.
- 2) Events in forwards direction (downstream), coming from originating leg, are not visible in terminating leg model.
- 3) States are as seen from the application: if there is no change in the method an application is permitted to apply on the IpCallLeg object, then there is no state change. Therefore receipt of e.g. answer or alerting events on terminating leg do not change state (see note 2).
- 4) The application is to send a request to continue processing (using an appropriate method like continueProcessing) for each leg and event reported in monitor mode "interrupt".
- 5) In case on a leg more than one network event (for example mid-call event "service\_code") is to be reported to the application at quasi the same time, then the events are to be reported one by one to the application in the order received from the network. When for a leg an event is reported in interrupt mode, a next pending event is not to be reported to the application until a request to resume call processing for the current reported event has been received on the leg.

NOTE 1: Call processing is suspended if for a leg a network event is met, which was requested to be monitored in the P\_CALL\_MONITOR\_MODE\_INTERRUPT.

NOTE 2: Even though there in the Originating Call Leg STD is no change in the methods the application is permitted to apply to the IpCallLeg object for the states Analysing and Active, separate states are maintained. The states may therefore from an application viewpoint appear as just one state that may be have substates like Analysing and Active. The digit collection task in state Analysing state may be viewed as a specialised task that may not at all be applicable in some networks and therefore here described as being a state on its own.

### 7.4.3.1 Originating Call Leg

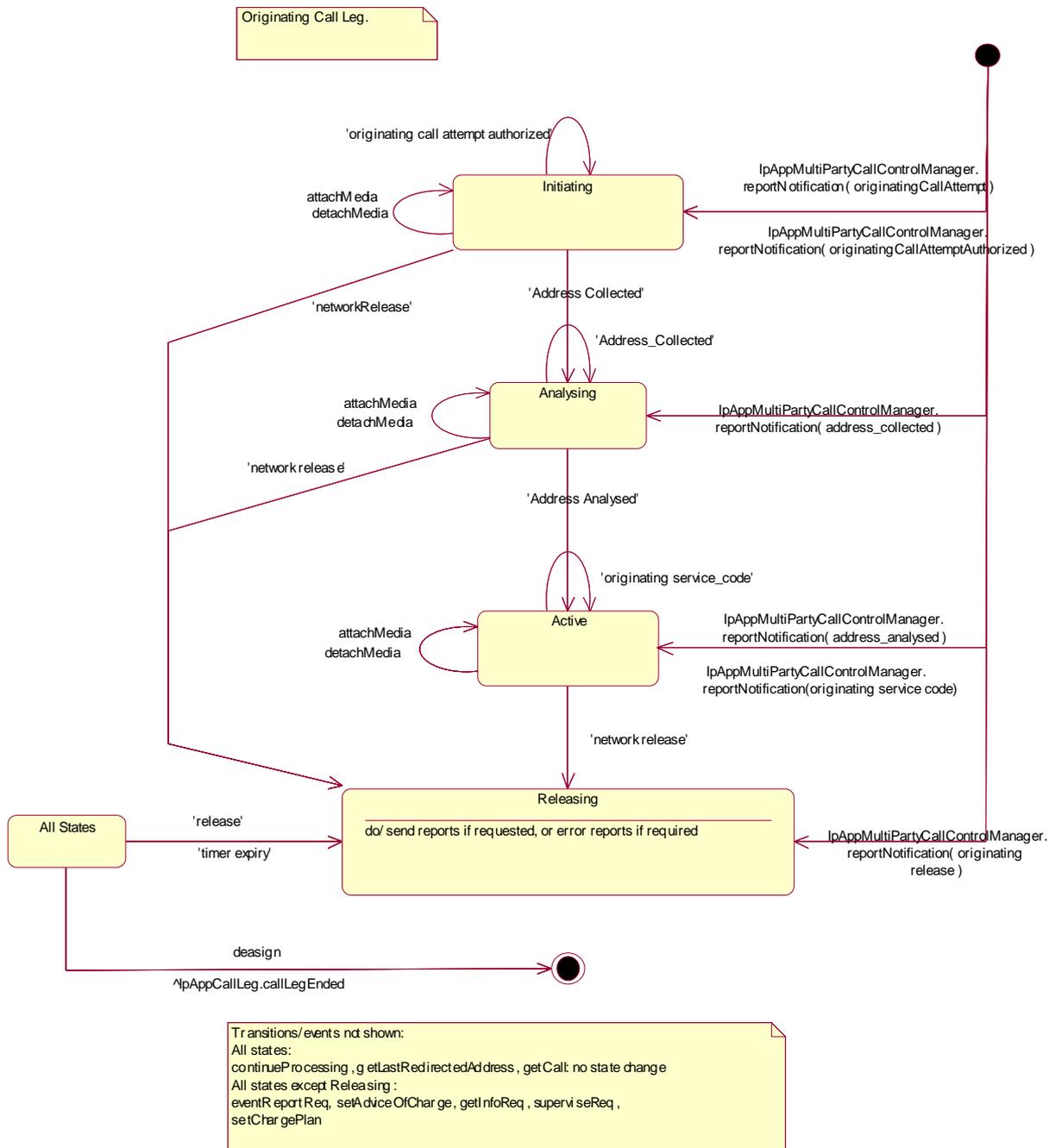


Figure 9: Originating Leg

#### 7.4.3.1.1 Initiating State

**Entry events:**

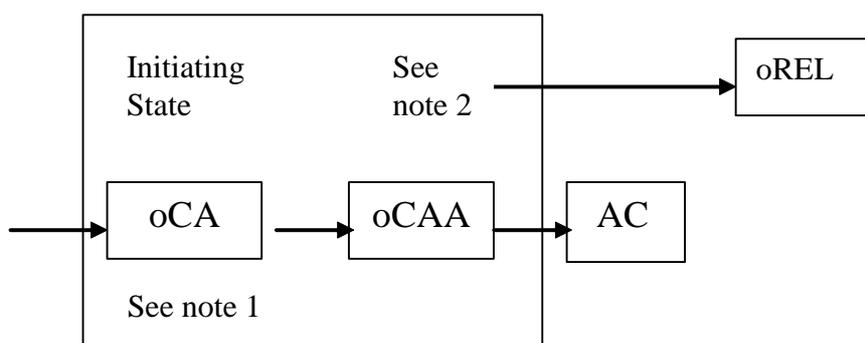
- Sending of a reportNotification() method by the IpMultiPartyCallControlManager for an "Originating\_Call\_Attempt" initial notification criterion.
- iii) Sending of a reportNotification() method by the IpMultiPartyCallControlManager for an "Originating\_Call\_Attempt\_Authorised" initial notification criterion.

**Functions:**

In this state the network checks the authority/ability of the party to place the connection to the remote (destination) party with the given properties, e.g. based on the originating party's identity and service profile.

The setup of the connection for the party has been initiated and the application activity timer is being provided.

The figure below shows the order in which network events may be detected in the Initiating state and depending on the monitor mode be reported to the application.



NOTE 1: Event oCA only applicable as an initial notification.

NOTE 2: The release event (oREL) can occur in any state resulting in a transition to Releasing state.

Abbreviations used for the events:

oCA originating Call Attempt;

oCAA originating Call Attempt Authorized;

AC Address Collected;

oREL originating RElease.

**Figure 10: Application view on event reporting order in Initiating State**

In this state the following functions are applicable:

- The detection of an "Originating\_Call\_Attempt" initial notification criterion.
- The detection of an "Originating\_Call\_Attempt\_Authorised" initial notification criterion as a result that the call attempt authorisation is successful.
- The report of the "Originating\_Call\_Attempt\_Authorised" event indication whereby the following functions are performed:
  - i) When the P\_CALL\_MONITOR\_MODE\_INTERRUPT is requested for the call leg event P\_CALL\_EVENT\_CALL\_ATTEMPT\_AUTHORISED then the event is reported and call leg processing is suspended.
  - ii) When the P\_CALL\_MONITOR\_MODE\_NOTIFY is requested for the call leg event P\_CALL\_EVENT\_CALL\_ATTEMPT\_AUTHORISED then the event is notified and call leg processing continues.
  - iii) When the P\_CALL\_MONITOR\_MODE\_DO\_NOT\_MONITOR is requested for the call leg event P\_CALL\_EVENT\_CALL\_ATTEMPT\_AUTHORISED then no monitoring is performed.
- The receipt of destination address information, i.e. initial information package/dialling string as received from calling party.
- Resumption of suspended call leg processing occurs on receipt of a continueProcessing() method.

**Exit events:**

- Availability of destination address information, i.e. the initial information package/dialling string received from the calling party.
- Application activity timer expiry indicating that no requests from the application have been received during a certain period.
- Receipt of a deassign() method.
- Receipt of a release() method.
- Detection of a "originating release" indication as a result of a premature disconnect from the calling party.

**7.4.3.1.2 Analysing State****Entry events:**

- Availability of an "Address\_Collected" event indication as a result of the receipt of the (complete) initial information package/dialling string from the calling party.
- Sending of a reportNotification() method by the IpMultiPartyCallControlManager for an "Address\_Collected" initial notification criterion.

**Functions:**

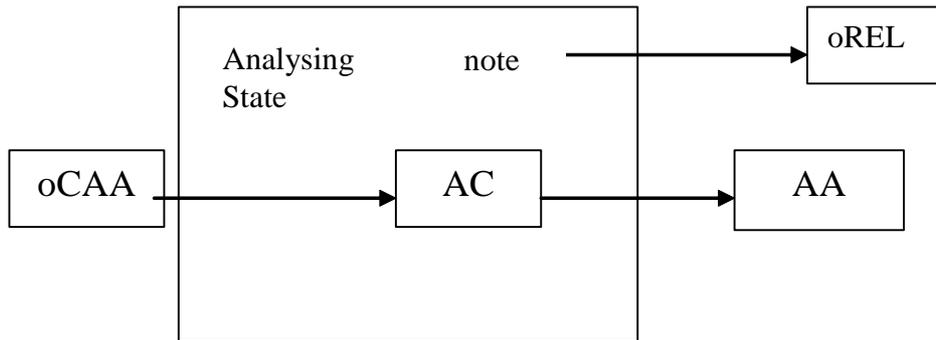
In this state the destination address provided by the calling party is collected and analysed.

The received information (dialled address string from the calling party) is being collected and examined in accordance to the dialling plan in order to determine end of address information (digit) collection. Additional address digits can be collected. Upon completion of address collection the address is analysed.

The address analysis is being made according to the dialling plan in force to determine the routing address of the call leg connection and the connection type (e.g. local, transit, gateway).

The request (with eventReportReq method) to collect a variable number of more address digits and report them to the application (within eventReportRes method) is handled within this state. The collection of more digits as requested and the reporting of received digits to the application (when the digit collect criteria is met) is done in this state. This action is recursive, e.g. the application could ask for 3 digits to be collected and when report request can be done repeatedly, e.g. the application may for example request first for 3 digits to be collected and when reported request further digits.

The figure below shows the order in which network events may be detected in the Analysing state and depending on the monitor mode be reported to the application.



NOTE: The release event (oREL) can occur in any state resulting in a transition to Releasing state.

Abbreviations used for the events:

oCAA: originating Call Attempt Authorized;  
 AC: Address Collected;  
 AA: Address Analysed;  
 oREL: originating RElease.

**Figure 11: Application view on event reporting order in Analysing State**

In this state the following functions are applicable:

- The detection of a "Address\_Collected" initial notification criterion.
- On receipt of the "Address\_Collected" indication the following functions are performed:
  - i) When the P\_CALL\_MONITOR\_MODE\_INTERRUPT is requested for the call leg event P\_CALL\_EVENT\_ADDRESS\_COLLECTED then the event is reported and call leg processing is suspended.
  - ii) When the P\_CALL\_MONITOR\_MODE\_NOTIFY is requested for the call leg event P\_CALL\_EVENT\_ADDRESS\_COLLECTED then the event is notified and call leg processing continues.
  - iii) When the P\_CALL\_MONITOR\_MODE\_DO\_NOT\_MONITOR is requested for the call leg event P\_CALL\_EVENT\_ADDRESS\_COLLECTED then no monitoring is performed.
- Receipt of a eventReportReq() method defining the criteria for the events the call leg object is to observe.
- Resumption of suspended call leg processing occurs on receipt of a continueProcessing() or a routeReq() method.

**Exit events:**

- Detection of an "Address\_Analysed" indication as a result of the availability of the routing address and nature of address.
- Receipt of a deassign() method.
- Receipt of a release() method.

Detection of a "originating release" indication as a result of a premature disconnect from the calling party.

#### 7.4.3.1.3 Active State

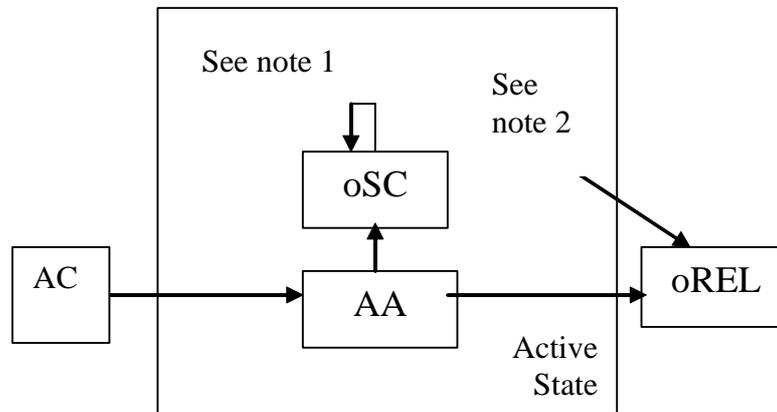
**Entry events:**

- Receipt of an "Address\_Analysed" indication as a result of the availability of the routing address and nature of address.
- Sending of a reportNotification() method by the IpMultiPartyCallControlManager for an "Address\_Analysed" initial indication criterion.

**Functions:**

In this state the call leg connection to the calling party exists and originating mid call events can be received.

The figure below shows the order in which network events may be detected in the Active state and depending on the monitor mode be reported to the application.



NOTE 1: Only the detected service code or the range to which the service code belongs is disarmed as the service code is reported to the application.

NOTE 2: The release event (oREL) can occur in any state resulting in a transition to Releasing state.

Abbreviations used for the events:

AC Address Collected;  
 AA Address Analysed;  
 oSC originating Service Code;  
 oREL originating RELease.

**Figure 12: Application view on event reporting order Active State**

In this state the following functions are applicable:

- The detection of a Address\_Analysed initial indication criterion.
- On receipt of the "Address\_Analysed" indication the following functions are performed:
  - i) When the P\_CALL\_MONITOR\_MODE\_INTERRUPT is requested for the call leg event P\_CALL\_EVENT\_ADDRESS\_ANALYSED then the event is reported and call leg processing is suspended.
  - ii) When the P\_CALL\_MONITOR\_MODE\_NOTIFY is requested for the call leg event P\_CALL\_EVENT\_ADDRESS\_ANALYSED then the event is notified and call leg processing continues.
  - iii) When the P\_CALL\_MONITOR\_MODE\_DO\_NOT\_MONITOR is requested for the call leg event P\_CALL\_EVENT\_ADDRESS\_ANALYSED then no monitoring is performed.
- Resumption of suspended call leg processing occurs on receipt of a continueProcessing() method.
- In this state the routing information is interpreted, the authority of the calling party to establish this connection is verified and the call leg connection is set up to the remote party.
- In this state a connection to the call party is established.
- Detection of a "terminating release" indication (not visible to the application) from remote party caused by a network release event propagated from a terminating party, possibly resulting in an "originating release" indication and causing the originating call leg STD to transit to Releasing state:
- Detection of a disconnect from the calling party.

- Receipt of a deassign() method.
- Receipt of a release() method.
- On receipt of the "originating\_service code" indication the following functions are performed:
  - i) When the P\_CALL\_MONITOR\_MODE\_INTERRUPT is requested for the call leg event P\_CALL\_EVENT\_ORIGINATING\_SERVICE\_CODE then the event is reported and call leg processing is suspended.
  - ii) When the P\_CALL\_MONITOR\_MODE\_NOTIFY is requested for the call leg event P\_CALL\_EVENT\_ORIGINATING\_SERVICE\_CODED then the event is notified and call leg processing continues.
  - iii) When the P\_CALL\_MONITOR\_MODE\_DO\_NOT\_MONITOR is requested for the call leg event P\_CALL\_EVENT\_ORIGINATING\_SERVICE\_CODE then no monitoring is performed.
- Resumption of suspended call leg processing occurs on receipt of a continueProcessing() method.

**Exit events:**

- Detection of an "originating release" indication as a result of a disconnect from the calling party and a "terminating release" indication as a result of a disconnect from called party.
- Receipt of a deassign() method.
- Receipt of a release() method from the application.

#### 7.4.3.1.4 Releasing State

**Entry events:**

- Detection of an "Originating\_Release" indication as a result of the network release initiated by calling party or called party.
- Reception of the release() method from the application.
- A transition due to fault detection to this state is made when the Call leg object is in a state and no requests from the application have been received during a certain time period (timer expiry).

**Functions:**

In this state the connection to the call party is released as requested by the network or by the application and the reports are processed and sent to the application if requested.

When the Releasing state is entered the order of actions to be performed is as follows:

- i) the network release event handling is performed.
- ii) the possible call leg information requested with getInfoReq() and/or superviseReq() is collected and send to the application.
- iii) the callLegEnded() method is sent to the application to inform that the call leg object is destroyed.

In this state the following functions are applicable:

- The detection of a "originating\_release" initial indication criterion..
- On receipt of the "originating\_release" indication the following functions are performed:
  - The network release event handling is performed as follows:
    - i) When the P\_CALL\_MONITOR\_MODE\_INTERRUPT is requested for the call leg event P\_CALL\_EVENT\_RELEASE then the event is reported and call leg processing is suspended.
    - ii) When the P\_CALL\_MONITOR\_MODE\_NOTIFY is requested for the call leg event P\_CALL\_EVENT\_RELEASE then the event is notified and call leg processing continues.

- iii) When the P\_CALL\_MONITOR\_MODE\_DO\_NOT\_MONITOR is requested for the call leg event P\_CALL\_EVENT\_RELEASE then no monitoring is performed.
- Resumption of suspended call leg processing occurs on receipt of a continueProcessing() method.
  - The possible call leg information requested with the getInfoReq() and/or superviseReq() is collected and sent to the application with respectively the getInfoRes() and/or superviseRes() methods.
  - The callLegEnded() method is sent to the application after all information has been sent. In case that the application has not requested additional call leg related information the call leg object is destroyed immediately and additionally the application will also be informed that the connection has ended
  - In case of abnormal termination due to a fault and the application requested for call leg related information previously, the application will be informed that this information is not available and additionally the application is informed that the call leg object is destroyed (callLegEnded).

NOTE: The call in the network may continue or be released, depending e.g. on the call state.

- In case the release() method is received in Releasing state it will be discarded. The request from the application to release the leg is ignored in this case because release of the leg is already ongoing.

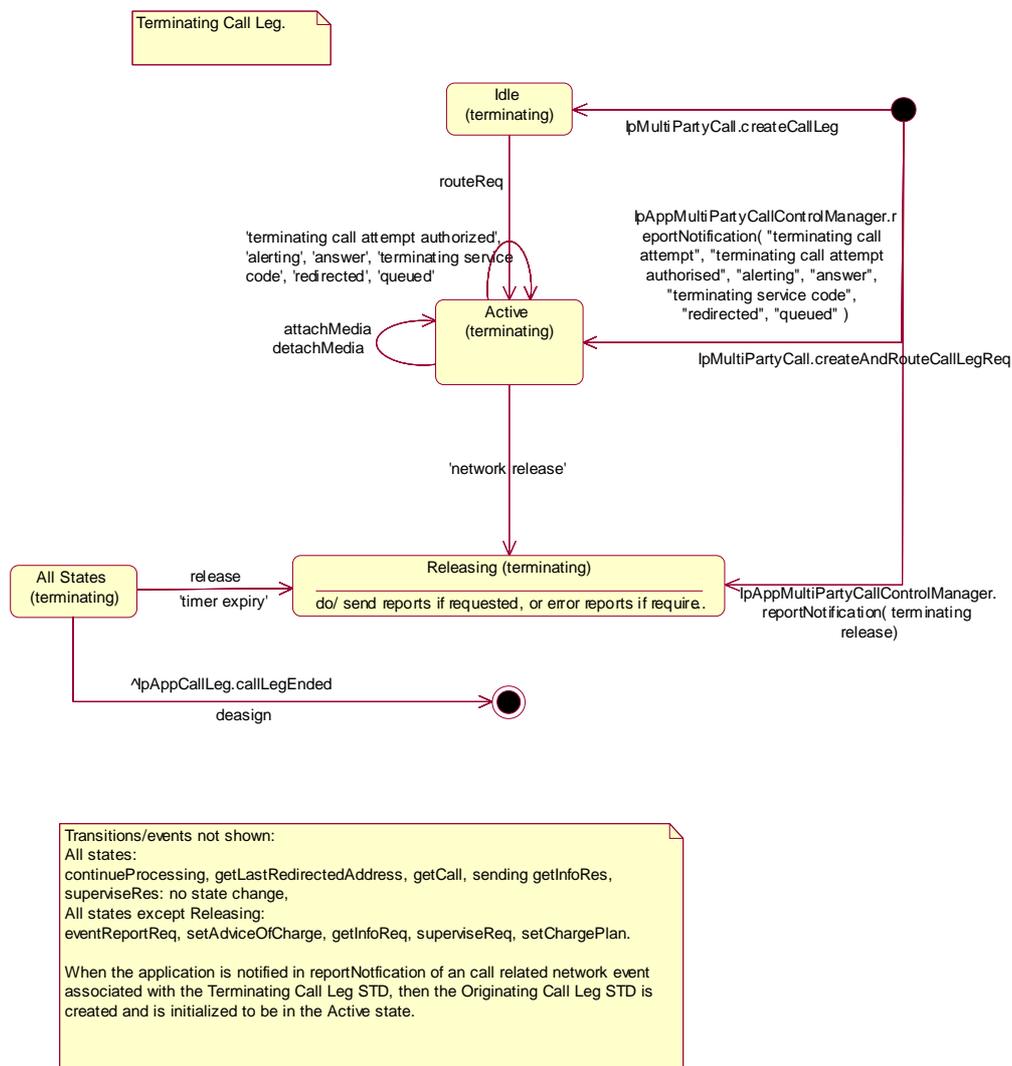
***Exit events:***

- In case that the application has not requested additional call leg related information the call leg object is destroyed immediately and additionally the application is informed that the call leg connection has ended, by sending the callLegEnded() method.
- After the sending of the last call leg information to the application the Call Leg object is destroyed and additionally the application is informed that the call leg connection has ended, by sending the callLegEnded() method.

## 7.4.3.1.5 Overview of allowed methods, Originating Call Leg STD

State	Methods allowed
Initiating	attachMediaReq (as a request), detachMediaReq, (as a request) getCall, continueProcessing, release (call leg), deassign eventReportReq, getInfoReq, setChargePlan, setAdviceOfCharge, superviseReq
Analysing	attachMediaReq (as a request), detachMediaReq, (as a request) getCall, continueProcessing, release (call leg), deassign eventReportReq, getInfoReq, setChargePlan, setAdviceOfCharge, superviseReq
Active	attachMediaReq, detachMediaReq, getCall, continueProcessing, release deassign eventReportReq, getInfoReq, setChargePlan, setAdviceOfCharge, superviseReq
Releasing	getCall, continueProcessing, release deassign

### 7.4.3.2 Terminating Call Leg



**Figure 13: Terminating Leg**

#### 7.4.3.2.1 Idle (terminating) State

**Entry events:**

- Receipt of a createCallLeg() method to start an application initiated call leg connection.

**Functions:**

In this state the call leg object is created and the interface connection is idled.

The application activity timer is being provided.

In this state the following functions are applicable:

- Invoking routeReq will result in a request to actually route the call leg object.
- Resumption of call leg processing occurs on receipt of a routeReq() method.

**Exit events:**

- Receipt of a routeReq() method from the application.
- Application activity timer expiry indicating that no requests from the application have been received during a certain period to continue processing.
- Receipt of a deassign() method.
- Receipt of a release() method.
- Detection of a network release event being an "originating release" indication as a result of a premature disconnect from the calling party.

**7.4.3.2.2 Active (terminating) State****Entry events:**

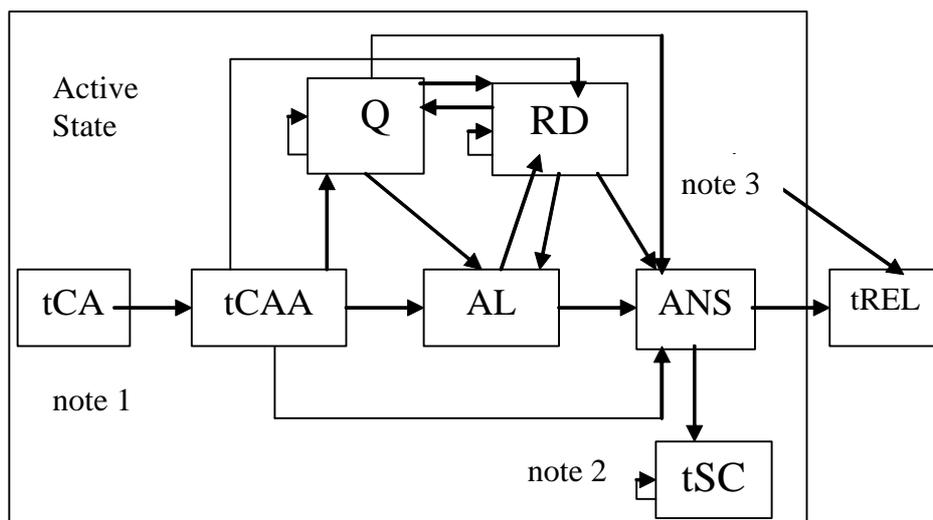
- Receipt of an routeReq will result in actually routing the call leg object.
- Receipt of a createAndRouteCallLegReq() method to start an application initiated call leg connection.
- Sending of a reportNotification() method by the IpMultiPartyCallControlManager for the following trigger criteria: "Terminating\_Call\_Attempt", "Terminating\_Call\_Attempt\_Authorised", "Alerting", "Answer", "Terminating service code", "Redirected" and "Queued".

**Functions:**

In this state the routing information is interpreted, the authority of the called party to establish this connection is verified for the call leg connection. In this state a connection to the call party is established whereby events from the network may indicate to the application when the party is alerted (acknowledge connection setup) and when the party answer (confirmation of connection setup).

Furthermore, in this state terminating service code events can be received.

The figure below shows the order in which network events may be detected in the Active state and depending on the monitor mode be reported to the application.



NOTE 1: Event tCA applicable as initial notification.

NOTE 2: Only the detected service code or the range to which the service code belongs is disarmed as the service code is reported to the application.

NOTE 3: The release event (tREL) can occur in any state resulting in a transition to Releasing state.

Abbreviations used for the events:

tCA terminating Call Attempt;  
 tCAA terminating Call Attempt Authorized;  
 AL Alerting;  
 ANS Answer;  
 tREL terminating RELEASE;  
 Q Queued;  
 RD ReDirected;  
 tSC terminating Service Code.

**Figure 14: Application view on event reporting order in Active State**

In this state the following functions are applicable:

- The detection and report of the "Terminating\_Call\_Attempt\_Authorised" event indication whereby the following functions are performed:
  - i) When the P\_CALL\_MONITOR\_MODE\_INTERRUPT is requested for the call leg event P\_CALL\_EVENT\_TERMINATING\_CALL\_ATTEMPT\_AUTHORISED then the event is reported and call leg processing is suspended.
  - ii) When the P\_CALL\_MONITOR\_MODE\_NOTIFY is requested for the call leg event P\_CALL\_EVENT\_TERMINATING\_CALL\_ATTEMPT\_AUTHORISED then the event is notified and call leg processing continues.
  - iii) When the P\_CALL\_MONITOR\_MODE\_DO\_NOT\_MONITOR is requested for the call leg event P\_CALL\_EVENT\_CALL\_TERMINATING\_ATTEMPT\_AUTHORISED then no monitoring is performed.
- Detection of an "Queued" indication as a result of the terminating call being queued.
- On receipt of the "Queued" indication the following functions are performed:
  - i) When the P\_CALL\_MONITOR\_MODE\_INTERRUPT is requested for the call leg event P\_CALL\_EVENT\_QUEUED then the event is reported and call leg processing is suspended.
  - ii) When the P\_CALL\_MONITOR\_MODE\_NOTIFY is requested for the call leg event P\_CALL\_EVENT\_QUEUED then the event is notified and call leg processing continues.
  - iii) When the P\_CALL\_MONITOR\_MODE\_DO\_NOT\_MONITOR is requested for the call leg event P\_CALL\_EVENT\_QUEUED then no monitoring is performed.

- On receipt of the "Alerting" indication the following functions are performed:
  - i) When the P\_CALL\_MONITOR\_MODE\_INTERRUPT is requested for the call leg event P\_CALL\_EVENT\_ALERTING then the event is reported and call leg processing is suspended.
  - ii) When the P\_CALL\_MONITOR\_MODE\_NOTIFY is requested for the call leg event P\_CALL\_EVENT\_ALERTING then the event is notified and call leg processing continues.
  - iii) When the P\_CALL\_MONITOR\_MODE\_DO\_NOT\_MONITOR is requested for the call leg event P\_CALL\_EVENT\_ALERTING then no monitoring is performed.
- Detection of an "Answer" indication as a result of the remote party being connected (answered).
- On receipt of the "Answer" indication the following functions are performed:
  - i) When the P\_CALL\_MONITOR\_MODE\_INTERRUPT is requested for the call leg event P\_CALL\_EVENT\_ANSWER then the event is reported and call leg processing is suspended.
  - ii) When the P\_CALL\_MONITOR\_MODE\_NOTIFY is requested for the call leg event P\_CALL\_EVENT\_ANSWER then the event is notified and call leg processing continues.
  - iii) When the P\_CALL\_MONITOR\_MODE\_DO\_NOT\_MONITOR is requested for the call leg event P\_CALL\_EVENT\_ANSWER then no monitoring is performed.
- The detection of a "service\_code" trigger criterion suspends call leg processing.
- On receipt of the "service code" indication the following functions are performed:
  - i) When the P\_CALL\_MONITOR\_MODE\_INTERRUPT is requested for the call leg event P\_CALL\_EVENT\_TERMINATING\_SERVICE\_CODE then the event is reported and call leg processing is suspended.
  - ii) When the P\_CALL\_MONITOR\_MODE\_NOTIFY is requested for the call leg event P\_CALL\_EVENT\_TERMINATING\_SERVICE\_CODE then this is not a valid event (that event is not notified) and call leg processing continues.
  - iii) When the P\_CALL\_MONITOR\_MODE\_DO\_NOT\_MONITOR is requested for the call leg event P\_CALL\_EVENT\_TERMINATING\_SERVICE\_CODE then no monitoring is performed.
- On receipt of the "redirected" indication the following functions are performed:
  - i) When the P\_CALL\_MONITOR\_MODE\_INTERRUPT is requested for the call leg event P\_CALL\_EVENT\_REDIRECTED then the event is reported and call leg processing is suspended.
  - ii) When the P\_CALL\_MONITOR\_MODE\_NOTIFY is requested for the call leg event P\_CALL\_EVENT\_REDIRECTED then the event is notified and call leg processing continues.
  - iii) When the P\_CALL\_MONITOR\_MODE\_DO\_NOT\_MONITOR is requested for the call leg event P\_CALL\_EVENT\_REDIRECTED then no monitoring is performed.
- Resumption of call leg processing occurs on receipt of a continueProcessing() method.

**Exit events:**

- Detection of a network release event being an "terminating release" indication as a result of the following events:
  - i) Unable to select a route or indication from the remote party of the call leg connection cannot be presented (this is the network determined busy condition)
  - ii) Occurrence of an authorisation failure when the authority to place the call leg connection was denied (e.g. business group restriction mismatch).
  - iii) Detection of a route busy condition received from the remote call leg connection portion.
  - iv) Detection of a no-answer condition received from the remote call leg connection portion.

- v) Detection that the remote party was not reachable.
- Detection of a network release event being an "originating release" indication as a result of the following events:
  - vi) Detection of a premature disconnect from the calling party.
- Receipt of a deassign() method.
- Receipt of a release() method from the application.
- Detection of a network release event being an "originating release" indication as a result of a disconnect from the calling party or a "terminating release" indication as a result of a disconnect from the called party.

#### 7.4.3.2.3 Releasing (terminating) State

##### **Entry events:**

- Detection of a network release event being an "originating release" indication as a result of the network release initiated by calling party or a "terminating release" indication as a result of the network release initiated by called party.
- Sending of the release() method by the application.
- A transition due to fault detection to this state is made when the Call leg object awaits a request from the application and this is not received within a certain time period.
- Detection of a network event being a "terminating release" indication as a result of the following events:
  - i) Unable to select a route or indication from the remote party of the call leg connection cannot be presented (this is the network determined busy condition).
  - ii) Occurrence of an authorisation failure when the authority to place the call leg connection was denied (e.g. business group restriction mismatch).
  - iii) Detection of a route busy condition received from the remote call leg connection portion.
  - iv) Detection of a no-answer condition received from the remote call leg connection portion.
- v) Detection that the remote party was not reachable.
- Detection of a network release event being an "originating release" indication as a result of the following events:
  - vi) Detection of a premature disconnect from the calling party.

##### **Functions:**

In this state the connection to the call party is released as requested by the network or by the application and the reports are processed and sent to the application if requested.

When the Releasing state is entered the order of actions to be performed is as follows:

- i) the release event handling is performed;
- ii) the possible call leg information requested with getInfoReq() and/or superviseReq() is collected and sent to the application;
- iii) the callLegEnded() method is sent to the application to inform that the call leg object is destroyed.

Where the entry to this state is caused by the application, for example because the application has requested the leg to be released or deassigned or a fault (e.g. timer expiry, no response from application) has been detected, then i) is not applicable. In the fault case for action ii) error report methods are sent to the application for any possible requested reports.

In this state the following functions are applicable:

- The detection of a "Terminating Release" trigger criterion.
- On receipt of the network release event being a "Terminating Release" indication the following functions are performed:
  - The network release event handling is performed as follows:
    - i) When the P\_CALL\_MONITOR\_MODE\_INTERRUPT is requested for the call leg event P\_CALL\_EVENT\_TERMINATING\_RELEASE then the event is reported and call leg processing is suspended.
    - ii) When the P\_CALL\_MONITOR\_MODE\_NOTIFY is requested for the call leg event P\_CALL\_EVENT\_TERMINATING\_RELEASE then the event is notified and call leg processing continues.
    - iii) When the P\_CALL\_MONITOR\_MODE\_DO\_NOT\_MONITOR is requested for the call leg event P\_CALL\_EVENT\_TERMINATING\_RELEASE then no monitoring is performed.
- Resumption of suspended call leg processing occurs on receipt of a continueProcessing() method.
- The possible call leg information requested with the getInfoReq() and/or superviseReq() is collected and sent to the application with respectively the getInfoRes() and/or superviseRes() methods.
- The callLegEnded() method is sent to the application after all information has been sent. In case that the application has not requested additional call leg related information the call leg object is destroyed immediately and additionally the application will also be informed that the connection has ended.
- In case of abnormal termination due to a fault and the application requested for call leg related information previously, the application will be informed that this information is not available and additionally the application is informed that the call leg object is destroyed (callLegEnded).

NOTE: The call in the network may continue or be released, depending e.g. on the call state.

- In case the release() method is received in Releasing state it will be discarded. The request from the application to release the leg is ignored in this case because release of the leg is already ongoing.

**Exit events:**

- In case that the application has not requested additional call leg related information the call leg object is destroyed immediately and additionally the application is informed that the call leg connection has ended, by sending the callLegEnded() method.
- After the sending of the last call leg information to the application the Call Leg object is destroyed and additionally the application is informed that the call leg connection has ended, by sending the callLegEnded() method.

## 7.4.3.2.4 Overview of allowed methods and trigger events, Terminating Call Leg STD

State	Methods allowed
Idle	routeReq, getCall, getCurrentDestinationAddress, release, deassign eventReportReq, getInfoReq, setChargePlan, setAdviceOfCharge, superviseReq
Active	attachMediaReq detachMediaReq getCall, getCurrentDestinationAddress, continueProcessing, release, deassign eventReportReq, getInfoReq, setChargePlan, setAdviceOfCharge, superviseReq
Releasing	getCall, getCurrentDestinationAddress, continueProcessing, release, deassign

## 7.5 Multi-Party Call Control Service Properties

## 7.5.1 List of Service Properties

The following table lists properties relevant for the MPCC API. These properties are additional to the properties of the GCC, from which the MPCC is an extension.

Property	Type	Description
P_MAX_CALLEGS_PER_CALL	INTEGER_SET	Indicates how many parties can be in one call.
P_UI_CALLLEG_BASED	BOOLEAN_SET	Value = TRUE : User interaction can be performed on leg level and a reference to a CallLeg object can be used in the IpUIManager.createUICall() operation. Value = FALSE : No user interaction on leg level is supported.
P_ROUTING_WITH_CALLLEG_OPERATIONS	BOOLEAN_SET	Value = TRUE : the atomic operations for routing a CallLeg are supported {IpMultiPartyCall.createCallLeg(), IpCallLeg.eventReportReq(), IpCallLeg.routeReq(), IpCallLeg.attachMediaReq()} Value = FALSE : the convenience function has to be used for routing a CallLeg.
P_MEDIA_ATTACH_EXPLICIT	BOOLEAN_SET	Value = TRUE : the CallLeg shall be explicitly attached to a Call. Value = FALSE : the CallLeg is automatically attached to a Call, no IpCallLeg.attachMediaReq() is needed when a party answers.

## 7.5.2 Service Property values for the CAMEL Service Environment

Implementations of the MultiParty Call Control API relying on the CSE of CAMEL phase 3 shall have the Service Properties outlined above set to the indicated values:

```
P_OPERATION_SET = {
'IpMultiPartyCallControlManager.createNotification',
'IpMultiPartyCallControlManager.destroyNotification',
'IpMultiPartyCallControlManager.changeNotification',
'IpMultiPartyCallControlManager.getNotification',
'IpMultiPartyCallControlManager.setCallLoadControl'
'IpMultiPartyCall.getCallLegs',
'IpMultiPartyCall.createCallLeg',
'IpMultiPartyCall.createAndRouteCallLegReq',
'IpMultiPartyCall.release',
'IpMultiPartyCall.deassignCall',
'IpMultiPartyCall.getInfoReq',
'IpMultiPartyCall.setChargePlan',
'IpMultiPartyCall.setAdviceOfCharge',
'IpMultiPartyCall.superviseReq',
'IpCallLeg.routeReq',
'IpCallLeg.eventReportReq',
'IpCallLeg.release',
'IpCallLeg.getInfoReq',
'IpCallLeg.getCall',
'IpCallLeg.continueProcessing'
}
```

```
P_TRIGGERING_EVENT_TYPES = {
P_CALL_EVENT_ADDRESS_COLLECTED,
P_CALL_EVENT_ADDRESS_ANALYSED,
P_CALL_EVENT_ORIGINATING_RELEASE,
P_CALL_EVENT_TERMINATING_CALL_ATTEMPT_AUTHORISED,
P_CALL_EVENT_TERMINATING_RELEASE
}
```

NOTE: P\_CALL\_EVENT\_ORIGINATING\_RELEASE only for the routing failure case,  
TpReleaseCause = P\_ROUTING\_FAILURE.

```
P_DYNAMIC_EVENT_TYPES = {
P_CALL_EVENT_ANSWER,
P_CALL_EVENT_ORIGINATING_RELEASE,
P_CALL_EVENT_TERMINATING_RELEASE
}
```

```
P_ADDRESS_PLAN = {
P_ADDRESS_PLAN_E164
}
```

```
P_UI_CALL_BASED = {
TRUE
}
```

```
P_UI_AT_ALL_STAGES = {
FALSE
}
```

```
P_MEDIA_TYPE = {
P_AUDIO
}
```

```
P_MAX_CALLLEGS_PER_CALL = {
0,
2
}
```

```
P_UI_CALLLEG_BASED = {
FALSE
}
```

```
P_MEDIA_ATTACH_EXPLICIT = {
FALSE
}
```

## 7.6 Multi-Party Call Control Data Definitions

This clause provides the MPCC data definitions necessary to support the API specification.

The general format of a data definition specification is described below.

- Data Type

This shows the name of the data type.

- Description

This describes the data type.

- Tabular Specification

This specifies the data types and values of the data type.

- Example

If relevant, an example is shown to illustrate the data type.

All data types referenced but not defined in this clause are either in the common call control data definitions clause of the present document (clause 10) or in the common data definitions which may be found in ES 201 915-2.

### 7.6.1 Event Notification Data Definitions

No specific event notification data defined.

### 7.6.2 Multi-Party Call Control Data Definitions

#### 7.6.2.1 IpCallLeg

Defines the address of an IpCallLeg Interface.

#### 7.6.2.2 IpCallLegRef

Defines a Reference to type IpCallLeg.

#### 7.6.2.3 IpAppCallLeg

Defines the address of an IpAppCallLeg Interface.

#### 7.6.2.4 IpAppCallLegRef

Defines a Reference to type IpAppCallLeg.

#### 7.6.2.5 IpMultiPartyCall

Defines the address of an IpMultiPartyCall Interface.

#### 7.6.2.6 IpMultiPartyCallRef

Defines a Reference to type IpMultiPartyCall.

#### 7.6.2.7 IpAppMultiPartyCall

Defines the address of an IpAppMultiPartyCall Interface.

### 7.6.2.8 IpAppMultiPartyCallRef

Defines a Reference to type IpAppMultiPartyCall.

### 7.6.2.9 IpMultiPartyCallControlManager

Defines the address of an IpMultiPartyCallControlManager Interface.

### 7.6.2.10 IpMultiPartyCallControlManagerRef

Defines a Reference to type IpMultiPartyCallControlManager.

### 7.6.2.11 IpAppMultiPartyCallControlManager

Defines the address of an IpAppMultiPartyCallControlManager Interface.

### 7.6.2.12 IpAppMultiPartyCallControlManagerRef

Defines a Reference to type IpAppMultiPartyCallControlManager.

### 7.6.2.13 TpAppCallLegRefSet

Defines a Numbered Set of Data Elements of IpAppCallLegRef.

### 7.6.2.14 TpMultiPartyCallIdentifier

Defines the Sequence of Data Elements that unambiguously specify the Call object.

Sequence Element Name	Sequence Element Type	Sequence Element Description
CallReference	IpMultiPartyCallRef	This element specifies the interface reference for the Multi-party call object.
CallSessionID	TpSessionID	This element specifies the call session ID.

### 7.6.2.15 TpAppMultiPartyCallBack

Defines the Tagged Choice of Data Elements that references the application callback interfaces.

Tag Element Type		
	TpAppMultiPartyCallBackRefType	

Tag Element Value	Choice Element Type	Choice Element Name
P_APP_CALLBACK_UNDEFINED	NULL	Undefined
P_APP_MULTIPARTY_CALL_CALLBACK	IpAppMultiPartyCallRef	AppMultiPartyCall
P_APP_CALL_LEG_CALLBACK	IpAppCallLegRef	AppCallLeg
P_APP_CALL_AND_CALL_LEG_CALLBACK	TpAppCallLegCallBack	AppMultiPartyCallAndCallLeg

### 7.6.2.16 TpAppMultiPartyCallBackRefType

Defines the type application call back interface.

Name	Value	Description
P_APP_CALLBACK_UNDEFINED	0	Application Call back interface undefined
P_APP_MULTIPARTY_CALL_CALLBACK	1	Application Multi-Party Call interface referenced
P_APP_CALL_LEG_CALLBACK	2	Application CallLeg interface referenced
P_APP_CALL_AND_CALL_LEG_CALLBACK	3	Application Multi-Party Call and CallLeg interface referenced

### 7.6.2.17 TpAppCallLegCallBack

Defines the Sequence of Data Elements that references a call and a call leg application interface.

Sequence Element Name	Sequence Element Type	
AppMultiPartyCall	IpAppMultiPartyCallRef	
AppCallLegSet	TpAppCallLegRefSet	Specifies the set of all call leg call back references. First in the set is the reference to the call back of the originating callLeg. In case there is a call back to a destination call leg this will be second in the set.

### 7.6.2.18 TpMultiPartyCallIdentifierSet

Defines a Numbered Set of Data Elements of TpMultiPartyCallIdentifier.

### 7.6.2.19 TpCallAppInfo

Defines the Tagged Choice of Data Elements that specify application-related call information.

Tag Element Type
TpCallAppInfoType

Tag Element Value	Choice Element Type	Choice Element Name
P_CALL_APP_ALERTING_MECHANISM	TpCallAlertingMechanism	CallAppAlertingMechanism
P_CALL_APP_NETWORK_ACCESS_TYPE	TpCallNetworkAccessType	CallAppNetworkAccessType
P_CALL_APP_TELE_SERVICE	TpCallTeleService	CallAppTeleService
P_CALL_APP_BEARER_SERVICE	TpCallBearerService	CallAppBearerService
P_CALL_APP_PARTY_CATEGORY	TpCallPartyCategory	CallAppPartyCategory
P_CALL_APP_PRESENTATION_ADDRESS	TpAddress	CallAppPresentationAddress
P_CALL_APP_GENERIC_INFO	TpString	CallAppGenericInfo
P_CALL_APP_ADDITIONAL_ADDRESS	TpAddress	CallAppAdditionalAddress
P_CALL_APP_ORIGINAL_DESTINATION_ADDRESS	TpAddress	CallAppOriginalDestinationAddress
P_CALL_APP_REDIRECTING_ADDRESS	TpAddress	CallAppRedirectingAddress

### 7.6.2.20 TpCallAppInfoType

Defines the type of call application-related specific information.

Name	Value	Description
P_CALL_APP_UNDEFINED	0	Undefined
P_CALL_APP_ALERTING_MECHANISM	1	The alerting mechanism or pattern to use
P_CALL_APP_NETWORK_ACCESS_TYPE	2	The network access type (e.g. ISDN)
P_CALL_APP_TELE_SERVICE	3	Indicates the tele-service (e.g. telephony)
P_CALL_APP_BEARER_SERVICE	4	Indicates the bearer service (e.g. 64 kbit/s unrestricted data).
P_CALL_APP_PARTY_CATEGORY	5	The category of the calling party
P_CALL_APP_PRESENTATION_ADDRESS	6	The address to be presented to other call parties
P_CALL_APP_GENERIC_INFO	7	Carries unspecified service-service information
P_CALL_APP_ADDITIONAL_ADDRESS	8	Indicates an additional address
P_CALL_APP_ORIGINAL_DESTINATION_ADDRESS	9	Contains the original address specified by the originating user when launching the call.
P_CALL_APP_REDIRECTING_ADDRESS	10	Contains the address of the user from which the call is diverting.

### 7.6.2.21 TpCallAppInfoSet

Defines a Numbered Set of Data Elements of TpCallAppInfo.

### 7.6.2.22 TpCallEventRequest

Defines the Sequence of Data Elements that specify the criteria relating to call report requests.

Sequence Element Name	Sequence Element Type
CallEventType	TpCallEventType
AdditionalCallEventCriteria	TpAdditionalCallEventCriteria
CallMonitorMode	TpCallMonitorMode

### 7.6.2.23 TpCallEventRequestSet

Defines a Numbered Set of Data Elements of TpCallEventRequest.

### 7.6.2.24 TpCallEventType

Defines a specific call event report type.

Name	Value	Description
P_CALL_EVENT_UNDEFINED	0	Undefined
P_CALL_EVENT_ORIGINATING_CALL_ATTEMPT	1	An originating call attempt takes place (e.g. Off-hook event).
P_CALL_EVENT_ORIGINATING_CALL_ATTEMPT_AUTHORISED	2	An originating call attempt is authorised
P_CALL_EVENT_ADDRESS_COLLECTED	3	The destination address has been collected.
P_CALL_EVENT_ADDRESS_ANALYSED	4	The destination address has been analysed.
P_CALL_EVENT_ORIGINATING_SERVICE_CODE	5	Mid-call originating service code received.
P_CALL_EVENT_ORIGINATING_RELEASE	6	A originating call/call leg is released
P_CALL_EVENT_TERMINATING_CALL_ATTEMPT	7	A terminating call attempt takes place
P_CALL_EVENT_TERMINATING_CALL_ATTEMPT_AUTHORISED	8	A terminating call is authorized
P_CALL_EVENT_ALERTING	9	Call is alerting at the call party.
P_CALL_EVENT_ANSWER	10	Call answered at address.
P_CALL_EVENT_TERMINATING_RELEASE	11	A terminating call leg has been released or the call could not be routed.
P_CALL_EVENT_REDIRECTED	12	Call redirected to new address: an indication from the network that the call has been redirected to a new address (no events disarmed as a result of this).
P_CALL_EVENT_TERMINATING_SERVICE_CODE	13	Mid call terminating service code received.
P_CALL_EVENT_QUEUED	14	The Call Event has been queued. (no events are disarmed as a result of this)

#### EVENT HANDLING RULES:

The following general event handling rules apply to dynamically armed events:

When requesting events for one leg;

- When the monitor mode is set to P\_CALL\_MONITOR\_MODE\_DO\_NOT\_MONITOR all events armed for that eventtype are disarmed. The additionalEventCriteria are not taken into account.
- When requesting two events for the same event type with different criteria and/or different monitor mode the last used criteria and monitor mode apply.
- Events that are not applicable to a leg are refused with exception P\_INVALID\_EVENT\_TYPE. The same exception is used when criteria are used that are not applicable to the leg, E.g. requesting P\_CALL\_EVENT\_TERMINATING\_SERVICE\_CODE on an originating leg is refused with exception P\_INVALID\_CRITERIA.  
When P\_CALL\_EVENT\_ORIGINATING\_RELEASE is requested with P\_BUSY in the criteria the request is refused with the same exception.

When receiving events:

- If an armed event is met, then it is disarmed, unless explicit stated that it will not to be disarmed.
- If an event is met that causes the release of the related leg, then all events related to that leg are disarmed.
- When an event is met on a call leg irrespective of the event monitor mode, then only events belonging to that call leg may become disarmed (see table below).
- If a call is released, then all events related to that call are disarmed.

NOTE: Event disarmed means monitor mode is set to DO\_NOT\_MONITOR. and event armed means monitor mode is set to INTERRUPT or NOTIFY.

The table below defines the disarming rules for dynamic events. In case such an event occurs on a call leg the table shows which events are disarmed (are not monitored anymore) on that call leg and should be re-armed by eventReportReq() in case the application is still interested in these events.

Event Occurred	Events Disarmed
P_CALL_EVENT_UNDEFINED	Not Applicable
P_CALL_EVENT_ORIGINATING_CALL_ATTEMPT	Not applicable, can only be armed as trigger
P_CALL_EVENT_ORIGINATING_CALL_ATTEMPT_AUTHORISED	P_CALL_EVENT_ORIGINATING_CALL_ATTEMPT_AUTHORISED
P_CALL_EVENT_ADDRESS_COLLECTED	P_CALL_EVENT_ADDRESS_COLLECTED
P_CALL_EVENT_ADDRESS_ANALYSED	P_CALL_EVENT_ADDRESS_COLLECTED P_CALL_EVENT_ADDRESS_ANALYSED
P_CALL_EVENT_ALERTING	P_CALL_EVENT_ALERTING P_CALL_EVENT__TERMINATING_RELEASE with criteria: P_USER_NOT_AVAILABLE P_BUSY P_NOT_REACHABLE P_ROUTING_FAILURE P_CALL_RESTRICTED P_UNAVAILABLE_RESOURCES
P_CALL_EVENT_ANSWER	P_CALL_EVENT_ALERTING P_CALL_EVENT_ANSWER P_CALL_EVENT_TERMINATING_RELEASE with criteria: P_USER_NOT_AVAILABLE P_BUSY P_NOT_REACHABLE P_ROUTING_FAILURE P_CALL_RESTRICTED P_UNAVAILABLE_RESOURCES P_NO_ANSWER
P_CALL_EVENT_ORIGINATING_RELEASE	All pending network events for the call leg are disarmed
P_CALL_EVENT_TERMINATING_RELEASE	All pending network events for the call leg are disarmed
P_CALL_EVENT_ORIGINATING_SERVICE_CODE	P_CALL_EVENT_ORIGINATING_SERVICE_CODE *) see note
P_CALL_EVENT_TERMINATING_SERVICE_CODE	P_CALL_EVENT_TERMINATING_SERVICE_CODE *) see note
<b>NOTE:</b> Only the detected service code or the range to which the service code belongs is disarmed.	

### 7.6.2.25 TpAdditionalCallEventCriteria

Defines the Tagged Choice of Data Elements that specify specific criteria.

	Tag Element Type	
	TpCallEventType	

Tag Element Value	Choice Element Type	Choice Element Name
P_CALL_EVENT_UNDEFINED	NULL	Undefined
P_CALL_EVENT_ORIGINATING_CALL_ATTEMPT	NULL	Undefined
P_CALL_EVENT_ORIGINATING_CALL_ATTEMPT_AUTHORISED	NULL	Undefined
P_CALL_EVENT_ADDRESS_COLLECTED	TpInt32	MinAddressLength
P_CALL_EVENT_ADDRESS_ANALYSED	NULL	Undefined
P_CALL_EVENT_ORIGINATING_SERVICE_CODE	TpCallServiceCodeSet	OriginatingServiceCode
P_CALL_EVENT_ORIGINATING_RELEASE	TpReleaseCauseSet	OriginatingReleaseCauseSet
P_CALL_EVENT_TERMINATING_CALL_ATTEMPT	NULL	Undefined
P_CALL_EVENT_TERMINATING_CALL_ATTEMPT_AUTHORISED	NULL	Undefined
P_CALL_EVENT_ALERTING	NULL	Undefined
P_CALL_EVENT_ANSWER	NULL	Undefined
P_CALL_EVENT_TERMINATING_RELEASE	TpReleaseCauseSet	TerminatingReleaseCauseSet
P_CALL_EVENT_REDIRECTED	NULL	Undefined
P_CALL_EVENT_TERMINATING_SERVICE_CODE	TpCallServiceCodeSet	TerminatingServiceCode
P_CALL_EVENT_QUEUED	NULL	Undefined

### 7.6.2.26 TpCallEventInfo

Defines the Sequence of Data Elements that specify the event report specific information.

Sequence Element Name	Sequence Element Type
CallEventType	TpCallEventType
AdditionalCallEventInfo	TpCallAdditionalEventInfo
CallMonitorMode	TpCallMonitorMode
CallEventTime	TpDateAndTime

### 7.6.2.27 TpCallAdditionalEventInfo

Defines the Tagged Choice of Data Elements that specify additional call event information for certain types of events.

	Tag Element Type	
	TpCallEventType	

Tag Element Value	Choice Element Type	Choice Element Name
P_CALL_EVENT_UNDEFINED	NULL	Undefined
P_CALL_EVENT_ORIGINATING_CALL_ATTEMPT	NULL	Undefined
P_CALL_EVENT_ORIGINATING_CALL_ATTEMPT_AUTHORISED	NULL	Undefined
P_CALL_EVENT_ADDRESS_COLLECTED	TpAddress	CollectedAddress
P_CALL_EVENT_ADDRESS_ANALYSED	TpAddress	CalledAddress
P_CALL_EVENT_ORIGINATING_SERVICE_CODE	TpCallServiceCode	OriginatingServiceCode
P_CALL_EVENT_ORIGINATING_RELEASE	TpReleaseCause	OriginatingReleaseCause
P_CALL_EVENT_TERMINATING_CALL_ATTEMPT	NULL	Undefined
P_CALL_EVENT_TERMINATING_CALL_ATTEMPT_AUTHORISED	NULL	Undefined
P_CALL_EVENT_ALERTING	NULL	Undefined
P_CALL_EVENT_ANSWER	NULL	Undefined
P_CALL_EVENT_TERMINATING_RELEASE	TpReleaseCause	TerminatingReleaseCause
P_CALL_EVENT_REDIRECTED	TpAddress	ForwardAddress
P_CALL_EVENT_TERMINATING_SERVICE_CODE	TpCallServiceCode	TerminatingServiceCode
P_CALL_EVENT_QUEUED	NULL	Undefined

### 7.6.2.28 TpCallNotificationRequest

Defines the Sequence of Data Elements that specify the criteria for an event notification.

Sequence Element Name	Sequence Element Type	Description
CallNotificationScope	TpCallNotificationScope	Defines the scope of the notification request.
CallEventsRequested	TpCallEventRequestSet	Defines the events which are requested

### 7.6.2.29 TpCallNotificationScope

Defines a the sequence of Data elements that specify the scope of a notification request.

Of the addresses only the Plan and the AddrString are used for the purpose of matching the notifications against the criteria.

Sequence Element Name	Sequence Element Type	Description
DestinationAddress	TpAddressRange	Defines the destination address or address range for which the notification is requested.
OriginatingAddress	TpAddressRange	Defines the origination address or address range for which the notification is requested.

### 7.6.2.30 TpCallNotificationInfo

Defines the Sequence of Data Elements that specify the information returned to the application in a Call notification report.

Sequence Element Name	Sequence Element Type	Description
CallNotificationReportScope	TpCallNotificationReportScope	Defines the scope of the notification report.
CallAppInfo	TpCallAppInfoSet	Contains additional call info.
CallEventInfo	TpCallEventInfo	Contains the event which is reported.

### 7.6.2.31 TpCallNotificationReportScope

Defines the Sequence of Data Elements that specify the scope for which a notification report was sent.

Sequence Element Name	Sequence Element Type	Description
DestinationAddress	TpAddress	Contains the destination address of the call.
OriginatingAddress	TpAddress	Contains the origination address of the call

### 7.6.2.32 TpNotificationRequested

Defines the Sequence of Data Elements that specify the criteria relating to event requests.

Sequence Element Name	Sequence Element Type
AppCallNotificationRequest	TpCallNotificationRequest
AssignmentID	TpInt32

### 7.6.2.33 TpNotificationRequestedSet

Defines a numbered Set of Data Elements of TpNotificationRequested.

### 7.6.2.34 TpReleaseCause

Defines the reason for a release.

Name	Value	Description
P_UNDEFINED	0	The reason of release is not known, because no info was received from the network.
P_USER_NOT_AVAILABLE	1	The user is not available in the network. This means that the number is not allocated or that the user is not registered.
P_BUSY	2	The user is busy.
P_NO_ANSWER	3	No answer was received
P_NOT_REACHABLE	4	The user terminal is not reachable
P_ROUTING_FAILURE	5	A routing failure occurred. For example an invalid address was received
P_PREMATURE_DISCONNECT	6	The user disconnected the call/call leg during the setup phase.
P_DISCONNECTED	7	A disconnect was received.
P_CALL_RESTRICTED	8	The call was subject of restrictions
P_UNAVAILABLE_RESOURCE	9	The request could not be carried out as no resources were available.
P_GENERAL_FAILURE	10	A general network failure occurred.
P_TIMER_EXPIRY	11	The call/call leg was released because an activity timer expired.

### 7.6.2.35 TpReleaseCauseSet

Defines a Numbered Set of Data Elements of TpReleaseCause.

### 7.6.2.36 TpCallLegIdentifier

Defines the Sequence of Data Elements that unambiguously specify the Call Leg object.

Sequence Element Name	Sequence Element Type	Sequence Element Description
CallLegReference	IpCallLegRef	This element specifies the interface reference for the callLeg object.
CallLegSessionID	TpSessionID	This element specifies the callLeg session ID.

### 7.6.2.37 TpCallLegIdentifierSet

Defines a Numbered Set of Data Elements of TpCallLegIdentifier.

### 7.6.2.38 TpCallLegAttachMechanism

Defines how a CallLeg should be attached to the call.

Name	Value	Description
P_CALLLEG_ATTACH_IMPLICITLY	0	CallLeg should be attached implicitly to the call.
P_CALLLEG_ATTACH_EXPLICITLY	1	CallLeg should be attached explicitly to the call by using the attachMediaReq() operation. This allows e.g. the application to do first user interaction to the party before he/she is placed in the call.

### 7.6.2.39 TpCallLegConnectionProperties

Defines the Sequence of Data Elements that specify the connection properties of the Call Leg object.

Sequence Element Name	Sequence Element Type	Sequence Element Description
AttachMechanism	TpCallLegAttachMechanism	Defines how a CallLeg should be attached to the call.

### 7.6.2.40 TpCallLegInfoReport

Defines the Sequence of Data Elements that specify the call leg information requested.

Sequence Element Name	Sequence Element Type	Description
CallLegInfoType	TpCallLegInfoType	The type of call leg information.
CallLegStartTime	TpDateAndTime	The time and date when the call leg was started (i.e. the leg was routed).
CallLegConnectedToResourceTime	TpDateAndTime	The date and time when the call leg was connected to the resource. If no resource was connected the time is set to an empty string. Either this element is valid or the CallLegConnectedToAddressTime is valid, depending on whether the report is sent as a result of user interaction.
CallLegConnectedToAddressTime	TpDateAndTime	The date and time when the call leg was connected to the destination (i.e. when the destination answered the call). If the destination did not answer, the time is set to an empty string. Either this element is valid or the CallConnectedToResourceTime is valid, depending on whether the report is sent as a result of user interaction.
CallLegEndTime	TpDateAndTime	The date and time when the call leg was released.
ConnectedAddress	TpAddress	The address of the party associated with the leg. If during the call the connected address was received from the party then this is returned, otherwise the destination address (for legs connected to a destination) or the originating address (for legs connected to the origination) is returned.
CallLegReleaseCause	TpReleaseCause	The cause of the termination. May be present with P_CALL_LEG_INFO_RELEASE_CAUSE was specified.
CallAppInfo	TpCallAppInfoSet	Additional information for the leg. May be present with P_CALL_LEG_INFO_APPINFO was specified.

### 7.6.2.41 TpCallLegInfoType

Defines the type of call leg information requested and reported. The values may be combined by a logical "OR" function.

Name	Value	Description
P_CALL_LEG_INFO_UNDEFINED	00h	Undefined
P_CALL_LEG_INFO_TIMES	01h	Relevant call times
P_CALL_LEG_INFO_RELEASE_CAUSE	02h	Call leg release cause
P_CALL_LEG_INFO_ADDRESS	04h	Call leg connected address
P_CALL_LEG_INFO_APPINFO	08h	Call leg application related information

### 7.6.2.42 TpCallLegSuperviseTreatment

Defines the treatment of the call leg by the call control service when the call leg supervision timer expires. The values may be combined by a logical "OR" function.

Name	Value	Description
P_CALL_LEG_SUPERVISE_RELEASE	01h	Release the call leg when the call leg supervision timer expires
P_CALL_LEG_SUPERVISE_RESPOND	02h	Notify the application when the call leg supervision timer expires
P_CALL_LEG_SUPERVISE_APPLY_TONE	04h	Send a warning tone on the call leg when the call leg supervision timer expires. If call leg release is requested, then the call leg will be released following the tone after an administered time period

---

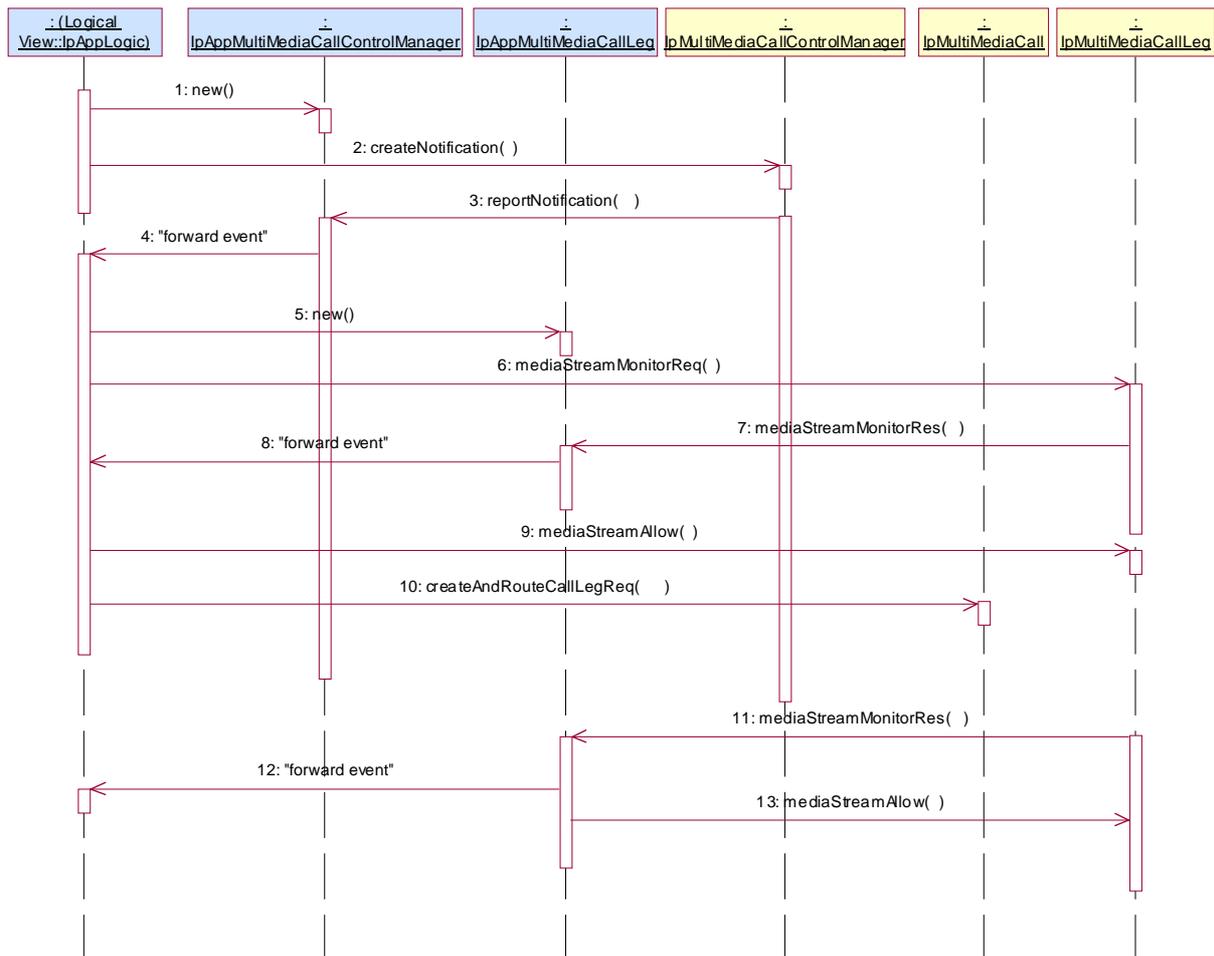
## 8 MultiMedia Call Control Service

### 8.1 Sequence Diagrams

#### 8.1.1 Barring for media combined with call routing, alternative 1

This sequence illustrates how one application can influence both the call routing and the media stream establishment of one call.

In this sequence there is one application handling both the media barring and the routing of the call.



- 1: The application creates a AppMultiMediaCallControlManager interface in order to handle callback methods.
- 2: The application expresses interest in all calls from subscriber A. Since createNotification is used and not createMediaNotification all calls are reported regardless of the media used.
- 3: A makes a call with the SIP INVITE with SDP media stream indicating video. The application is notified.
- 4: The event is forwarded to the application.
- 5: The application creates a new AppMultiMediaCallLeg interface to receive callbacks.
- 6: The application sets a monitor on video media streams to be established (added) for the indicated leg.
- 7: Since the video media stream was included in the SIP invite, the media streams monitored will be returned in the monitor result.
- 8: The event is forwarded to the application.
- 9: The application denies the video media stream, i.e. it is not included in the allowed media streams. This corresponds to removing the media stream from the setup.
- 10: The application requests to reroute the call to a different destination (or the same one, etc.).
- 11: Later in the call the A party tries to establish a lower bandwidth video media stream. This is again reported with MediaStreamMonitorRes.
- 12: The event is forwarded.

- 13: This time the application allows the establishment of the media stream by including the media stream in the allowed list.

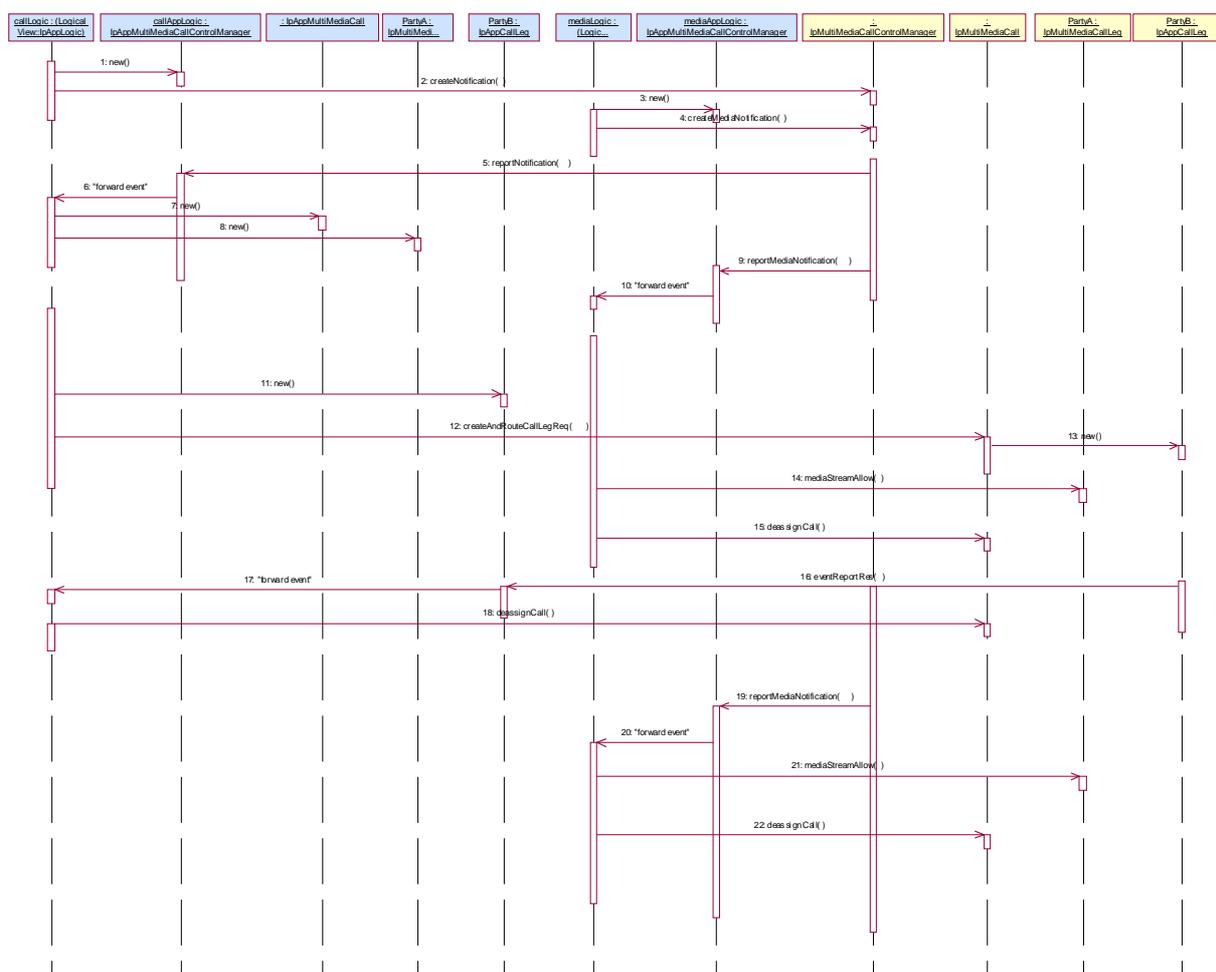
## 8.1.2 Barring for media combined with call routing, alternative 2

This sequence illustrates how one application can influence both the call routing and the media establishment of one call.

Media establishment and call establishment are regarded separately by the application.

From the gateway point of view it can actually be regarded as two separately triggered applications, one for media control and one for routing. This is also the way that it is shown here, for clarity.

However, an implementation of the application could combine the media logic and call logic in one object.



- 1: The application creates a new AppMultiMediaCallControlManager interface.
- 2: The application expresses interest in all calls from subscriber A for rerouting purposes.
- 3: The application creates a new AppMultiMediaCallControlManager interface. This is to be used for the media control only.
- 4: Separately the application expresses interest in some media streams for calls from and to A. The request indicates interrupt mode.

- 5: Subscriber A makes a call with the SIP INVITE with SDP media stream indicating video. Since the media establishment is combined with the SIP INVITE message, both applications are triggered (not necessarily in the order shown).

Here the call application is notified about the call setup.

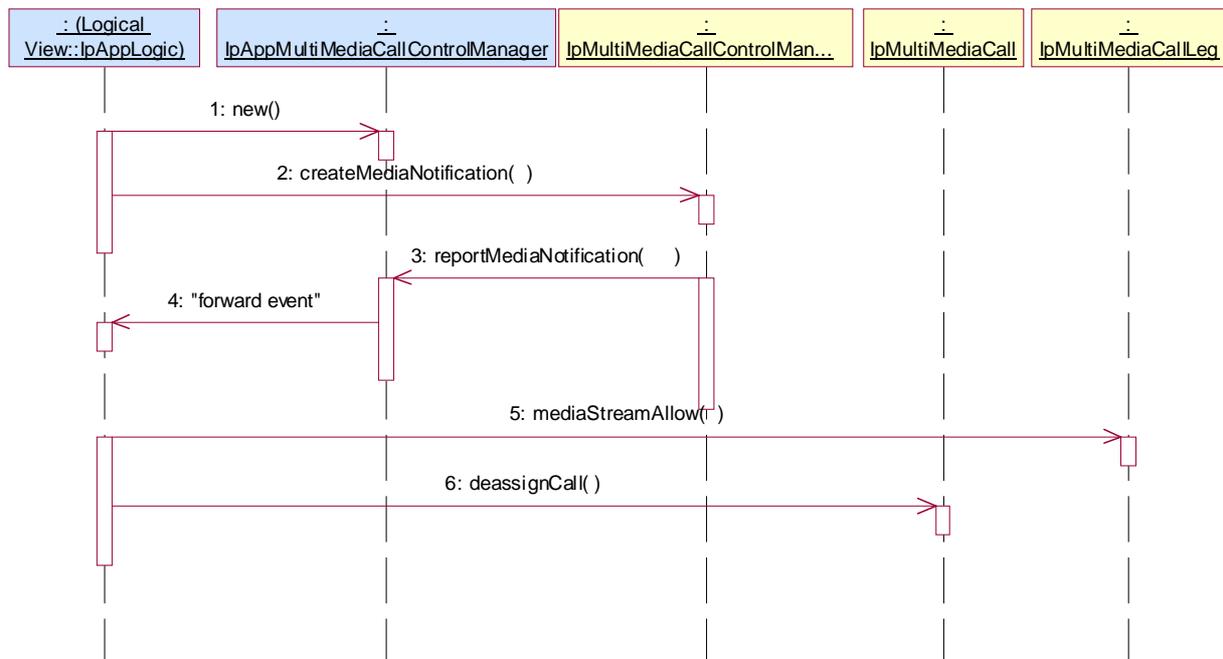
- 6: The event is forwarded to the call control application.
- 7: The call control application creates a new AppMultiMediaCall interface.
- 8: The call control application creates a new AppMultiMediaCallLeg interface.
- 9: The media application is notified about the call setup. All media streams from the setup will be indicated.
- 10: The event is forwarded to the media application.
- 11: The call control application creates a new AppMultiMediaCallLeg interface.
- 12: The call application decides to reroute the call to another address. Included in the request are monitors on answer and call end.

However, since the media was also triggered in mode interrupt the call will not proceed until the media streams are confirmed or rejected.

- 14: The application allows the audio media stream, but refuses the high bandwidth video, by excluding it from the allowed list. Since both call processing and media handling is now acknowledged, the call routing can continue (with a changed SDP parameter reflecting the manipulated media).
- 15: The Media application is no longer interested in the call.
- 16: When the B subscriber answers the call application is notified.
- 17: The event is forwarded to the call application.
- 19: When later in the call A tries to establish a lower bandwidth video stream the media application is triggered.
- 20: The triggering is forwarded to the media application.
- 21: The application now allows the establishment of the media stream by including the media stream in the mediaStreamAllow list.
- 22: The media application is no longer interested in the call.

### 8.1.3 Barring for media, simple

This sequence illustrates how an application can block the establishment of video streams for a certain user.



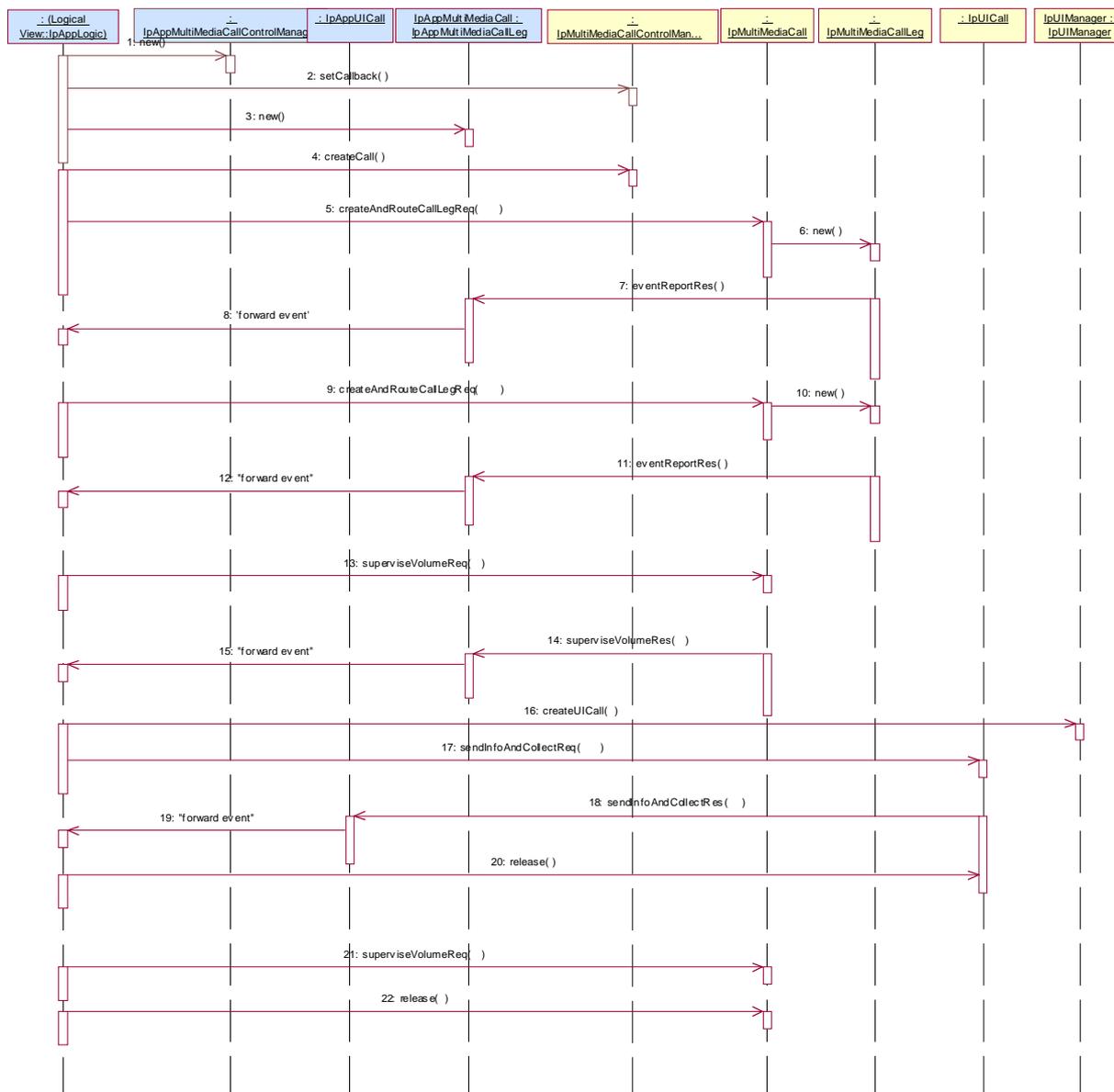
- 1: The application starts a new AppMultiMediaCallControlManager interface for reception of callbacks.
- 2: The application expresses interest in all calls from or to subscriber A that use video. The just created App interface is given as the callback interface.
- 3: Subscriber A makes a call with the SIP INVITE with SDP media stream indicating video.
- 4: The message is forwarded to the application.
- 5: The application indicates that the setup of the media stream is not allowed by not including the media stream in the allowed list. This has the effect of suppressing the video capabilities in the setup.
- 6: The application is no longer interested in the call.

New attempts to open video streams will again be indicated with a `reportMediaNotification`.

## 8.1.4 Call Volume charging supervision

This sequence illustrates how an application may supervise a call based on the number of bytes that are exchanged.

NOTE: In the sequence diagram below, a single box represents both an IpAppCall and an IpAppCallLeg for space reasons.



- 1: The application creates a new interface to receive callbacks on the call control manager.
- 2: The created interface is set as the callback interface for the call control manager.
- 3: The application creates a new interface to receive callback on the call.
- 4: The application requests the creation of a call.
- 5: The application initiates the call by routing to the origination. This will implicitly create a call leg. The application requests a notification when the party answers.
- 7: When the A party answers the application is notified.

- 8: The message is forwarded to the logic.
- 9: The application also routes the call to the destination. This implicitly creates a call leg. The application requests to be notified on answer of the B-party.
- 11: When the B-party answers the application is notified.
- 12: The message is forwarded to the logic.
- 13: The application requests to supervise the call. In the request the application specifies a limit on the amount of bytes that may be transferred. The application specifies that if the limit is reached the application should be notified.
- 14: When the limit is reached a notification is send to the application.
- 15: The message is forwarded to the logic.
- 17: The application plays an announcement to the user, asking whether the user wants to end the call or continue the call.
- 18: When the user answers whether the call should continue.
- 19: The message is forwarded to the logic.
- 20: The Uicall is released, since no further announcements are needed.
- 21: In case the user answers that the call should continue, the supervision is reset with a new maximum number of allowed bytes. (Note that this might have charging consequences, not shown).
- 22: If the user answered that the call should not continue, the call is released.

## 8.2 Class Diagrams

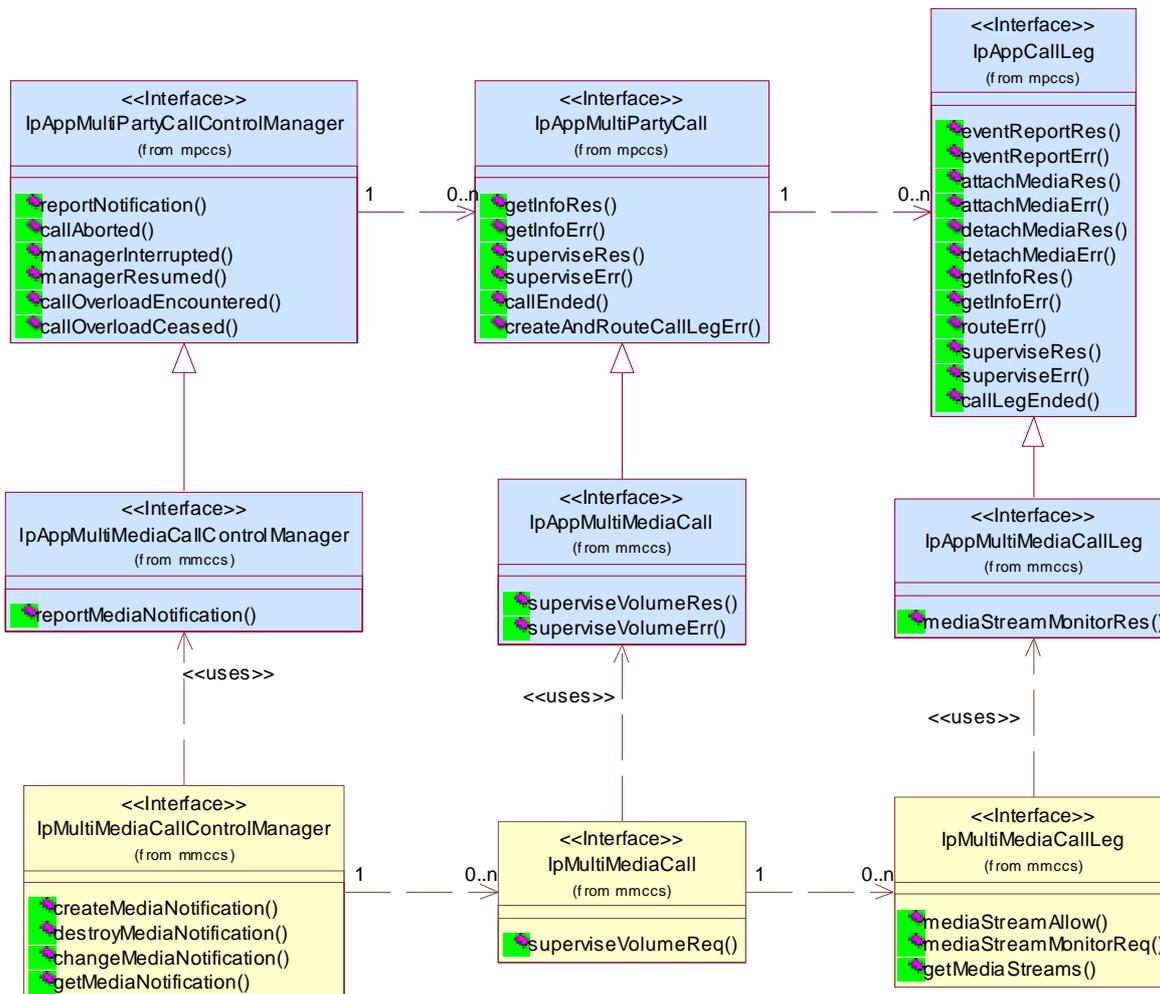


Figure 15: Application Interfaces

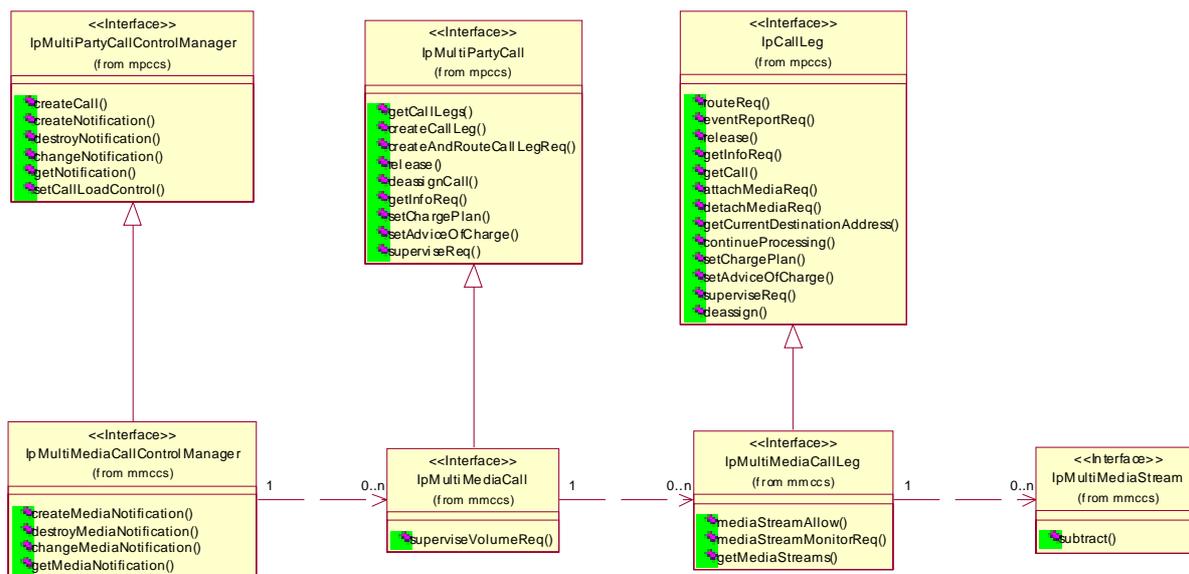


Figure 16: Service Interfaces

### 8.3 MultiMedia Call Control Service Interface Classes

The MultiMedia Call Control service enhances the functionality of the MultiParty Call Control Service with multi-media capabilities.

The MultiMedia Call Control Service is represented by the IpMultiMediaCallControlManager, IpMultiMediaCall, IpMultiMediaCallLeg and IpMultiMediaStream interfaces that interface to services provided by the network. Some 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. To handle responses and reports, the developer must implement IpAppMultiMediaCallControlManager, IpAppMultiMediaCall and IpAppMultiMediaCallLeg to provide the callback mechanism.

To handle the multi-media aspects of a call the concept of media stream is introduced. A media stream is bi-directional media stream and is associated with a call leg. These media streams are usually negotiated between the terminals in the call. The multi-party Call Service gives the application control over the media streams associated with the legs in a multi-media call in the following way:

- the application can be triggered on the establishment of a media stream that meets the application defined characteristics;
- the application can monitor on the establishment (addition) or release (subtraction) of media streams of an ongoing call;
- the application can allow or deny the establishment of media streams (provided the stream establishment was monitored/notified in interrupt mode);
- the application can explicitly subtract already established media streams;
- the application can request the media streams associated with a specific leg.

### 8.3.1 Interface Class IpMultiMediaCallControlManager

Inherits from: IpMultiPartyCallControlManager.

The Multi Media Call Control Manager is the factory interface for creating multimedia calls. The multi-media call control manager interface provides the management functions to the multi-media call control service. The application programmer can use this interface to create, destroy, change and get media stream related notifications. This interface shall be implemented by a Multi Media Call Control SCF. As a minimum requirement the createMediaNotification() and destroyMediaNotification() methods shall be implemented. The minimum required methods from IpMultiPartyCallControlManager are also required.

<<Interface>> IpMultiMediaCallControlManager
<pre> createMediaNotification (appInterface : in IpAppMultiMediaCallControlManagerRef, notificationMediaRequest     : in TpNotificationMediaRequest) : TpAssignmentID destroyMediaNotification (assignmentID : in TpAssignmentID) : void changeMediaNotification (assignmentID : in TpAssignmentID, notificationMediaRequest : in     TpNotificationMediaRequest) : void getMediaNotification () : TpMediaNotificationRequestedSet           </pre>

#### Method

#### **createMediaNotification()**

This method is used to create media stream notifications so that events can be sent to the application.

This applies both to callsetup media (e.g. SIP initial INVITE or H.323 with faststart) and for media setup during the call.

This is the first step an application has to do to get initial notifications of media streams happening in the network. When such an event happens, the application will be informed by reportMediaNotification(). In case the application is interested in other events during the context of a particular call session it has to use the mediaStreamMonitorReq() method on the Multi-Media call leg object.

The createMediaNotification method is purely intended for applications to indicate their interest to be notified when certain media stream events take place. It is possible to subscribe to a certain media stream event for a whole range of addresses, e.g. the application can indicate it wishes to be informed when a call is made to any number starting with 800.

If some application already requested notifications with criteria that overlap the specified criteria, the request is refused with P\_INVALID\_CRITERIA. The criteria are said to overlap if both originating and terminating ranges overlap and the same number plan is used.

If the same application requests two notifications with exactly the same criteria but different callback references, the second callback will be treated as an additional callback. Both notifications will share the same assignmentID. The gateway will always use the most recent callback. In case this most recent callback fails the second most recent is used. In case the createMediaNotification contains no callback, at the moment the application needs to be informed the gateway will use as callback the one that has been registered by setCallback().

Returns assignmentID: Specifies the ID assigned by the multi-media call control manager interface for this newly-created notification.

*Parameters*

**appInterface : in IpAppMultiMediaCallControlManagerRef**

Specifies a reference to the application interface, which is used for callbacks.

**notificationMediaRequest : in TpNotificationMediaRequest**

The mediaMonitorMode is a parameter of TpMediaStreamRequest and can be in interrupt or in notify mode. If in interrupt mode the application has to specify which media streams are allowed by calling mediaStreamAllow on the callLeg.

The notificationMediaRequest parameter specifies the event specific criteria used by the application to define the event required. This is the media portion of the criteria. Only events that meet the notificationMediaRequest are reported.

Individual addresses or address ranges may be specified for the destination and/or origination.

*Returns*

**TpAssignmentID**

*Raises*

**TpCommonExceptions, P\_INVALID\_CRITERIA, P\_INVALID\_INTERFACE\_TYPE, P\_INVALID\_EVENT\_TYPE**

*Method*

**destroyMediaNotification()**

This method is used by the application to disable Multi Media Channel notifications.

*Parameters*

**assignmentID : in TpAssignmentID**

Specifies the assignment ID given by the Multi Media call control manager interface when the previous enableMediaNotification was called. If the assignment ID does not correspond to one of the valid assignment IDs, the exception P\_INVALID\_ASSIGNMENTID will be raised.

*Raises*

**TpCommonExceptions**

*Method*

**changeMediaNotification()**

This method is used by the application to change the event criteria introduced with createMediaNotification. Any stored criteria associated with the specified assignmentID will be replaced with the specified criteria.

*Parameters*

**assignmentID : in TpAssignmentID**

Specifies the ID assigned by the multi-media call control manager interface for the media stream notification. If two callbacks have been registered under this assignment ID both of them will be changed.

**notificationMediaRequest : in TpNotificationMediaRequest**

Specifies the new set of event specific criteria used by the application to define the event required. Only events that meet these criteria are reported.

*Raises*

**TpCommonExceptions, P\_INVALID\_ASSIGNMENT\_ID, P\_INVALID\_CRITERIA, P\_INVALID\_EVENT\_TYPE**

*Method***getMediaNotification()**

This method is used by the application to query the event criteria set with createMediaNotification or changeMediaNotification.

Returns notificationsMediaRequested: Specifies the notifications that have been requested by the application.

*Parameters*

No Parameters were identified for this method.

*Returns*

**TpMediaNotificationRequestedSet**

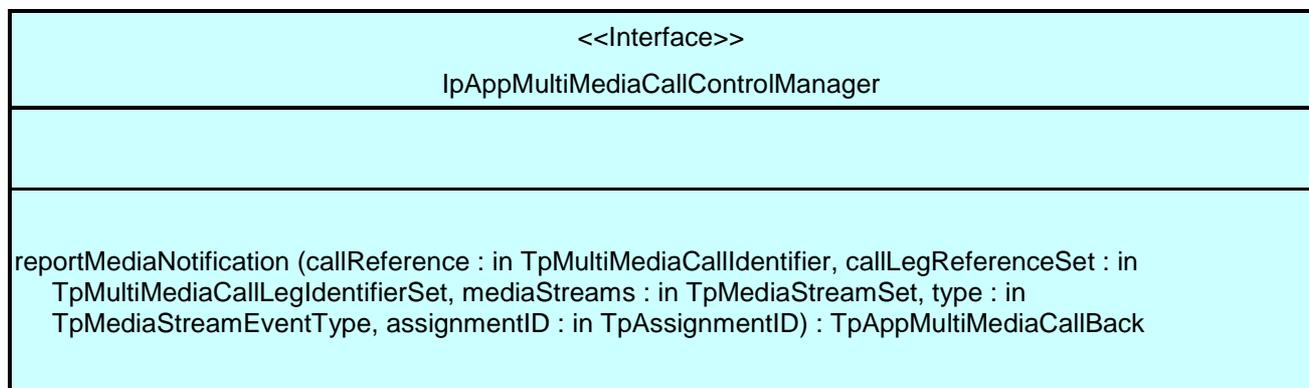
*Raises*

**TpCommonExceptions**

### 8.3.2 Interface Class IpAppMultiMediaCallControlManager

Inherits from: IpAppMultiPartyCallControlManager.

The Multi Media call control manager application interface provides the application call control management functions to the multi media call control service.

*Method***reportMediaNotification()**

This method is used to inform the application about the establishment of media streams.

If the corresponding monitor was in interrupt mode, then the application has to allow or deny the streams using mediaStreamAllow() method.

Returns appMultiMediaCallBack: Specifies references to the application interface which implements the callback interface for the new multi-media call and/or new call leg. This parameter may be null if the notification is being given in NOTIFY mode. If the application has previously explicitly passed a reference to the callback interface using a setCallbackWithSessionID() invocation, this parameter may be set to null, or if supplied must be the same as that provided during the setCallbackWithSessionID().

*Parameters***callReference : in TpMultiMediaCallIdentifier**

Specifies the call interface on which the media streams were added or subtracted. It also gives the corresponding sessionID.

**callLegReferenceSet : in TpMultiMediaCallLegIdentifierSet**

Specifies set of all callLeg references (interface and sessionID) for which the media streams were established or subtracted.

First in the set is the reference to the originating callLeg. It indicates the call leg related to the originating party. In case there is a destination call leg this will be the second leg in the set. from the notificationInfo can be found on whose behalf the notification was sent.

However, this parameter will be null if the notification is being given in NOTIFY mode.

**mediaStreams : in TpMediaStreamSet**

Specifies all the media streams that are established. Note that this can be more media streams than requested in the createMediaNotification, e.g. when faststart is used in H.323 or in SIP when an INVITE method with SDP media stream parameters is used.

**type : in TpMediaStreamEventType**

Refers to the type of event on the media stream, i.e. added or subtracted.

**assignmentID : in TpAssignmentID**

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

*Returns***TpAppMultiMediaCallBack**

### 8.3.3 Interface Class IpMultiMediaCall

Inherits from: IpMultiPartyCall.

This interface shall be implemented by a Multi Media Call Control SCF. Implementation of the superviseVolumeReq() method is optional. The minimum required methods from IpMultiPartyCall are required.

<<Interface>> IpMultiMediaCall
superviseVolumeReq (callSessionID : in TpSessionID, volume : in TpCallSuperviseVolume, treatment : in TpCallSuperviseTreatment) : void

*Method***superviseVolumeReq( )**

The application calls this method to supervise a call. The application can set a granted data volume this call.

*Parameters***callSessionID : in TpSessionID**

Specifies the call session ID of the call.

**volume : in TpCallSuperviseVolume**

Specifies the granted time in milliseconds for the connection.

**treatment : in TpCallSuperviseTreatment**

Specifies how the network should react after the granted volume expired.

*Raises***TpCommonExceptions, P\_INVALID\_SESSION\_ID**

### 8.3.4 Interface Class IpAppMultiMediaCall

Inherits from: IpAppMultiPartyCall.

The application multi-media call interface contains the callbacks that will be used from the multi-media call interface for asynchronous results to requests performed by the application. The application should implement this interface.

<<Interface>> IpAppMultiMediaCall
superviseVolumeRes (callSessionID : in TpSessionID, report : in TpCallSuperviseReport, usedVolume : in TpCallSuperviseVolume) : void superviseVolumeErr (callSessionID : in TpSessionID, errorIndication : in TpCallError) : void

*Method***superviseVolumeRes ( )**

This asynchronous method reports a call supervision event to the application when it has indicated its interest in these kind of events.

It is also called when the connection is terminated before the supervision event occurs.

*Parameters***callSessionID : in TpSessionID**

Specifies the call session ID of the call.

**report : in TpCallSuperviseReport**

Specifies the situation which triggered the sending of the call supervision response.

**usedVolume : in TpCallSuperviseVolume**

Specifies the used time for the call supervision (in milliseconds).

*Method***superviseVolumeErr()**

This asynchronous method reports a call supervision error to the application.

*Parameters*

**callSessionID** : in TpSessionID

Specifies the call session ID of the call.

**errorIndication** : in TpCallError

Specifies the error which led to the original request failing.

### 8.3.5 Interface Class IpMultiMediaCallLeg

Inherits from: IpCallLeg.

The Multi-Media call leg represents the signalling relationship between the call and an address. Associated with the signalling relationship there can be multiple media channels. Media channels can be started and stopped by the terminals themselves. The application can monitor on these changes and influence them.

This interface shall be implemented by a Multi Media Call Control SCF. The `mediaStreamAllow()` and `mediaStreamMonitorReq()` methods shall be implemented as a minimum requirement. The minimum required methods from IpCallLeg are also required.

<<Interface>> IpMultiMediaCallLeg
<pre> mediaStreamAllow (callLegSessionID : in TpSessionID, mediaStreamList : in TpSessionIDSet) : void mediaStreamMonitorReq (callLegSessionID : in TpSessionID, mediaStreamEventCriteria : in   TpMediaStreamRequestSet) : void getMediaStreams (callLegSessionID : in TpSessionID) : TpMediaStreamSet </pre>

*Method***mediaStreamAllow()**

This method can be used to allow setup of a media stream that was reported by a `mediaStreamMonitorRes` method.

*Parameters*

**callLegSessionID** : in TpSessionID

Specifies the call leg session ID of the call leg.

**mediaStreamList** : in TpSessionIDSet

Refers to the media streams (sessionIDs) as received in the `mediaStreamMonitorRes()` or in the `reportMediaNotification()` that is allowed to be established.

*Raises***TpCommonExceptions, P\_INVALID\_SESSION\_ID***Method***mediaStreamMonitorReq( )**

With this method the application can set monitors on the addition and subtraction of media streams. The monitors can either be general or restricted to certain types of codecs.

Monitoring on addition of media streams can be done in either interrupt or notify mode. In the first case the application has to allow or deny the establishment of the stream with `mediaStreamAllow`.

Monitoring on subtraction of media streams is only allowed in notify mode.

*Parameters***callLegSessionID : in TpSessionID**

Specifies the session ID of the call leg.

**mediaStreamEventCriteria : in TpMediaStreamRequestSet**

Specifies the event specific criteria used by the application to define the event required. The `mediaMonitorMode` is a parameter of `TpMediaStreamRequest` and can be in interrupt or in notify mode. If in interrupt mode the application has to respond with `mediaStreamAllow()`.

*Raises***TpCommonExceptions, P\_INVALID\_SESSION\_ID, P\_INVALID\_CRITERIA, P\_INVALID\_EVENT\_TYPE***Method***getMediaStreams( )**

This method is used to return all currently established media streams for the leg.

*Parameters***callLegSessionID : in TpSessionID**

This method is used to return all currently open media channels for the leg.

*Returns***TpMediaStreamSet***Raises***TpCommonExceptions, P\_INVALID\_SESSION\_ID**

### 8.3.6 Interface Class IpAppMultiMediaCallLeg

Inherits from: `IpAppCallLeg`.

The application multi-media call leg interface contains the callbacks that will be called from the multi-media call leg for asynchronous results to requests performed by the application. The application should implement this interface.

<<Interface>> <b>IpAppMultiMediaCallLeg</b>
<b>mediaStreamMonitorRes</b> (callLegSessionID : in TpSessionID, streams : in TpMediaStreamSet, type : in TpMediaStreamEventType) : void

### Method

#### **mediaStreamMonitorRes ( )**

This method is used to inform the application about the media streams that are being established (added) or subtracted.

If the corresponding request was done in interrupt mode, the application has to allow or deny the media streams using `mediaStreamAllow()`.

### Parameters

#### **callLegSessionID : in TpSessionID**

Specifies the session ID of the call leg for which the media channels are opened or closed.

#### **streams : in TpMediaStreamSet**

Specifies all the media streams that are added. Note that this can be more media streams than requested in the `createMediaNotification`, e.g. when faststart is used in H.323 or SIP INVITE with SDP media stream parameters is used.

#### **type : in TpMediaStreamEventType**

Refers to the type of event on the media stream, i.e. added or subtracted.

## 8.3.7 Interface Class IpMultiMediaStream

Inherits from: IpService.

The Multi Media Stream Interface represents a bi-directional information stream associated with a call leg. Currently, the only available method is to subtract the media stream. This interface and the `subtract()` method shall be implemented by a Multi Media Call Control SCF.

<<Interface>> <b>IpMultiMediaStream</b>
<b>subtract</b> (mediaStreamSessionID : in TpSessionID) : void

### Method

#### **subtract ( )**

This method can be used to subtract the multi-media stream.

*Parameters***mediaStreamSessionID** : in **TpSessionID**

Specifies the sessionID for the media stream.

*Raises***TpCommonExceptions**, **P\_INVALID\_SESSION\_ID**

## 8.4 MultiMedia Call Control Service State Transition Diagrams

There are no State Transition Diagrams for the MultiMedia Call Control Service package.

## 8.5 Multi-Media Call Control Data Definitions

This clause provides the Multi-Media call control data definitions necessary to support the API specification.

The general format of a data definition specification is described below.

- Data Type

This shows the name of the data type.

- Description

This describes the data type.

- Tabular Specification

This specifies the data types and values of the data type.

- Example

If relevant, an example is shown to illustrate the data type.

All data types referenced but not defined in this clause are either in the common call control data definitions clause of the present document (clause 10) or in the common data definitions which may be found in ES 201 915-2.

### 8.5.1 Event Notification Data Definitions

#### 8.5.1.1 TpMediaStreamRequestSet

Defines a **Numbered Set of Data Elements** of **TpMediaStreamRequest**.

#### 8.5.1.2 TpMediaStreamRequest

Defines the **Sequence of Data Elements** that specify the type of media stream.

Sequence Element Name	Sequence Element Type
Direction	TpMediaStreamDirection
DataTypeRequest	TpMediaStreamDataTypeRequest
MediaMonitorMode	TpCallMonitorMode

### 8.5.1.3 TpMediaStreamDirection

Defines the direction in which the media stream is established (as seen from the leg).

Name	Value	Description
P_SEND_ONLY	0	Indicates that the offerer is only willing to send this media stream
P_RECEIVE_ONLY	1	Indicates that the offerer is only willing to receive this media stream
P_SEND_RECEIVE	2	Indicates that the offerer is willing to send and receive this media stream

### 8.5.1.4 TpMediaStreamDataTypeRequest

Defines the Tagged Choice of Data Elements that specify the media type and associated codecs that are of interest.

Tag Element Type
TpMediaType

Tag Element Value	Choice Element Type	Choice Element Name
P_AUDIO	TpAudioCapabilitiesType	Audio
P_VIDEO	TpVideoCapabilitiesType	Video
P_DATA	TpDataCapabilities	Data

### 8.5.1.5 TpAudioCapabilitiesType

Defines the audio codec. The requested capabilities can be indicated by adding the values together (i.e. a logical OR function). e.g. 28 indicates interest in all G.722 codes (4+8+16).

Name	Value	Description
P_G711_64K	1	g.711 on 64k, both alaw and ulaw
P_G711_56K	2	g.711 on 56k, both alaw and ulaw
P_G722_64K	4	
P_G722_56K	8	
P_G722_48K	16	
P_G7231	32	
P_G728	64	
P_G729	128	
P_G729_ANNEX_A	256	
P_IS1172	512	
P_IS1318	1024	
P_G729_ANNEXB	2048	
P_G729_ANNEX_A_AND_B	4096	
P_G7231_ANNEX_C	8192	
P_GSM_FULLRATE	16384	
P_GSM_HALFRATE	32768	
P_GSM_ENHANCED	65536	

### 8.5.1.6 TpVideoCapabilitiesType

Defines the video codec. The requested capabilities can be indicated by adding the values together (i.e. a logical OR function). e.g. 3 indicates both H.261 and H.262 codecs.

Name	Value	Description
P_H261	1	
P_H262	2	
P_H263	4	
P_IS11172	8	

### 8.5.1.7 TpDataCapabilities

A TpInt32 defining the minimum maxBitRate in bit/s. I.e. all data media streams whose maxBitRate exceeds this number are reported.

### 8.5.1.8 TpMediaStreamEventType

Defines the action performed on the media stream.

Name	Value	Description
P_MEDIA_STREAM_ADDED	0	The media stream is added
P_MEDIA_STREAM_SUBTRACTED	1	The media stream is subtracted.

### 8.5.1.9 TpMediaStreamSet

Defines a Numbered Set of Data Elements of TpMediaStream.

### 8.5.1.10 TpMediaStream

Defines the Sequence of Data Elements that specify the type of media stream.

Sequence Element Name	Sequence Element Type
Direction	TpMediaStreamDirection
DataType	TpMediaStreamDataType
ChannelSessionID	TpSessionID
MediaStream	IpMultiMediaStream

### 8.5.1.11 TpMediaStreamDataType

Defines the type of the reported media stream. It is identical to TpMediaStreamDataTypeRequest, only now the values are not used as a mask, but as the actual codec should be indicated for audio and video. For data the actual maximum bit rate is indicated.

## 8.5.2 Multi-Media Call Control Data Definitions

### 8.5.2.1 IpMultiMediaCall

Defines the address of an IpMultiMediaCall Interface.

### 8.5.2.2 IpMultiMediaCallRef

Defines a Reference to type IpMultiMediaCall.

### 8.5.2.3 IpAppMultiMediaCall

Defines the address of an IpAppMultiMediaCall Interface.

### 8.5.2.4 IpAppMultiMediaCallRef

Defines a Reference to type IpAppMultiMediaCall.

### 8.5.2.5 IpMultiMediaCallLeg

Defines the address of an IpMultiMediaCallLeg Interface.

### 8.5.2.6 IpMultiMediaCallLegRef

Defines a Reference to type IpMultiMediaCallLeg.

### 8.5.2.7 IpAppMultiMediaCallLeg

Defines the address of an IpAppMultiMediaCallLeg Interface.

### 8.5.2.8 IpAppMultiMediaCallLegRef

Defines a Reference to type IpAppMultiMediaCallLeg.

### 8.5.2.9 TpAppMultiMediaCallLegRefSet

Defines a Numbered Set of Data Elements of IpAppMultiMediaCallLegRef.

### 8.5.2.10 TpMultiMediaCallIdentifier

Defines the Sequence of Data Elements that unambiguously specify the MultiMediaCall object.

Sequence Element Name	Sequence Element Type	Sequence Element Description
MMCallReference	IpMultiMediaCallRef	This element specifies the interface reference for the call object.
MMCallSessionID	TpSessionID	This element specifies the call session ID of the call created.

### 8.5.2.11 TpMultiMediaCallIdentifierSet

Defines a Numbered Set of Data Elements of TpMultiMediaCallIdentifier.

### 8.5.2.12 TpMultiMediaCallLegIdentifier

Defines the Sequence of Data Elements that unambiguously specify the Call Leg object.

Sequence Element Name	Sequence Element Type	Sequence Element Description
MMCallLegReference	IpMultiMediaCallLegRef	This element specifies the interface reference for the callLeg object.
MMCallLegSessionID	TpSessionID	This element specifies the callLeg session ID of the call created.

### 8.5.2.13 TpMultiMediaCallLegIdentifierSet

Defines a Numbered Set of Data Elements of TpMultiMediaCallLegIdentifier.

### 8.5.2.14 IpAppMultiMediaCallControlManager

Defines the address of an IpAppMultiMediaCallControlManager Interface.

### 8.5.2.15 IpAppMultiMediaCallControlManagerRef

Defines a Reference to type IpAppMultiMediaCallControlManager.

### 8.5.2.16 TpAppMultiMediaCallBack

Defines the Tagged Choice of Data Elements that references the application callback interfaces.

Tag Element Type	
	TpAppMultiMediaCallBackRefType

Tag Element Value	Choice Element Type	Choice Element Name
P_APP_CALLBACK_UNDEFINED	NULL	Undefined
P_APP_MULTIMEDIA_CALL_CALLBACK	IpAppMultiMediaCallRef	AppMultiMediaCall
P_APP_CALL_LEG_CALLBACK	IpAppMultiMediaCallLegRef	AppMultiMediaCallLeg
P_APP_CALL_AND_CALL_LEG_CALLBACK	TpAppMultiMediaCallLegCallBack	AppMultiMediaCallAndCallLeg

### 8.5.2.17 TpAppMultiMediaCallBackRefType

Defines the type application call back interface.

Name	Value	Description
P_APP_CALLBACK_UNDEFINED	0	Application Call back interface undefined
P_APP_MULTIMEDIA_CALL_CALLBACK	1	Application Multi-Media Call interface referenced
P_APP_CALL_LEG_CALLBACK	2	Application Multi-Media CallLeg interface referenced
P_APP_CALL_AND_CALL_LEG_CALLBACK	3	Application Multi-Media Call and CallLeg interface referenced

### 8.5.2.18 TpAppMultiMediaCallLegCallBack

Defines the Sequence of Data Elements that references a call and a call leg application interface.

Sequence Element Name	Sequence Element Type	
AppMultiMediaCall	IpAppMultiMediaCallRef	
AppCallLegSet	TpAppMultiMediaCallLegRefSet	Specifies the set of all call leg call back references. First in the set is the reference to the call back of the originating callLeg. In case there is a call back to a destination call leg this will be second in the set.

### 8.5.2.19 TpCallSuperviseVolume

Defines the Sequence of Data Elements that specify the amount of volume that is allowed to be transmitted for the specific connection.

Sequence Element Name	Sequence Element Type	Sequence Element Description
VolumeQuantity	TpInt32	This data type is identical to a TpInt32, and defines the quantity of the granted volume that can be transmitted for the specific connection.
VolumeUnit	TpInt32	This data type is identical to a TpInt32, and defines the unit of the granted volume that can be transmitted for the specific connection.  Unit must be specified as $10^n$ number of bytes, where $n$ denotes the power.  When the unit is for example in kilobytes, VolumeUnit must be set to 3.

### 8.5.2.20 TpNotificationMediaRequest

Defines the Sequence of Data Elements that specify the criteria for a media stream notification.

Sequence Element Name	Sequence Element Type	Description
MediaNotificationScope	TpCallNotificationScope	Defines the scope of the notification request.
MediaStreamsRequested	TpMediaStreamRequestSet	Defines the media stream events which are requested

### 8.5.2.21 TpMediaNotificationRequested

Defines the Sequence of Data Elements that specify the criteria relating to event requests.

Sequence Element Name	Sequence Element Type
AppNotificationMediaRequest	TpNotificationMediaRequest
AssignmentID	TpInt32

### 8.5.2.22 TpMediaNotificationsRequestedSet

Defines a numbered Set of Data Elements of TpMediaNotificationRequested.

## 9 Conference Call Control Service

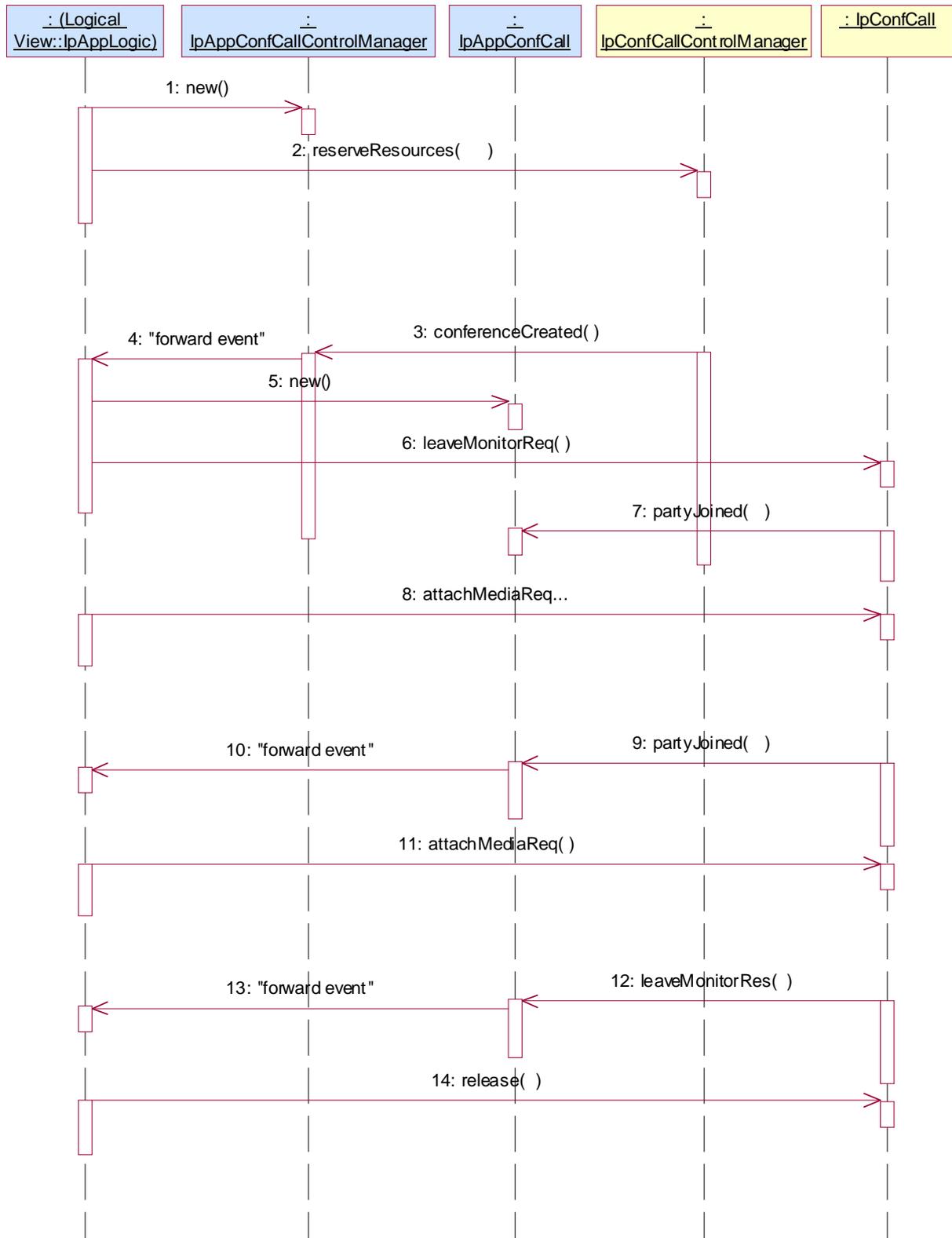
### 9.1 Sequence Diagrams

#### 9.1.1 Meet-me conference without subconferencing

This sequence illustrates a pre-arranged meet-me conference for a specified time period. During this timeslot parties can "call in to" the meet-me conference by dialling a special number.

For each participant joining the conference, the application can decide to accept the participant in to the conference.

The application can also be notified when parties are leaving the conference.



- 1: The application creates a new object to receive the callbacks from the conference call control manager.
- 2: The application reserves resources for some time in the future.

With this same method the application registers interest in the creation of the conference (e.g. when the first party to joins the conference or at the specified start time, this is implementation dependant).

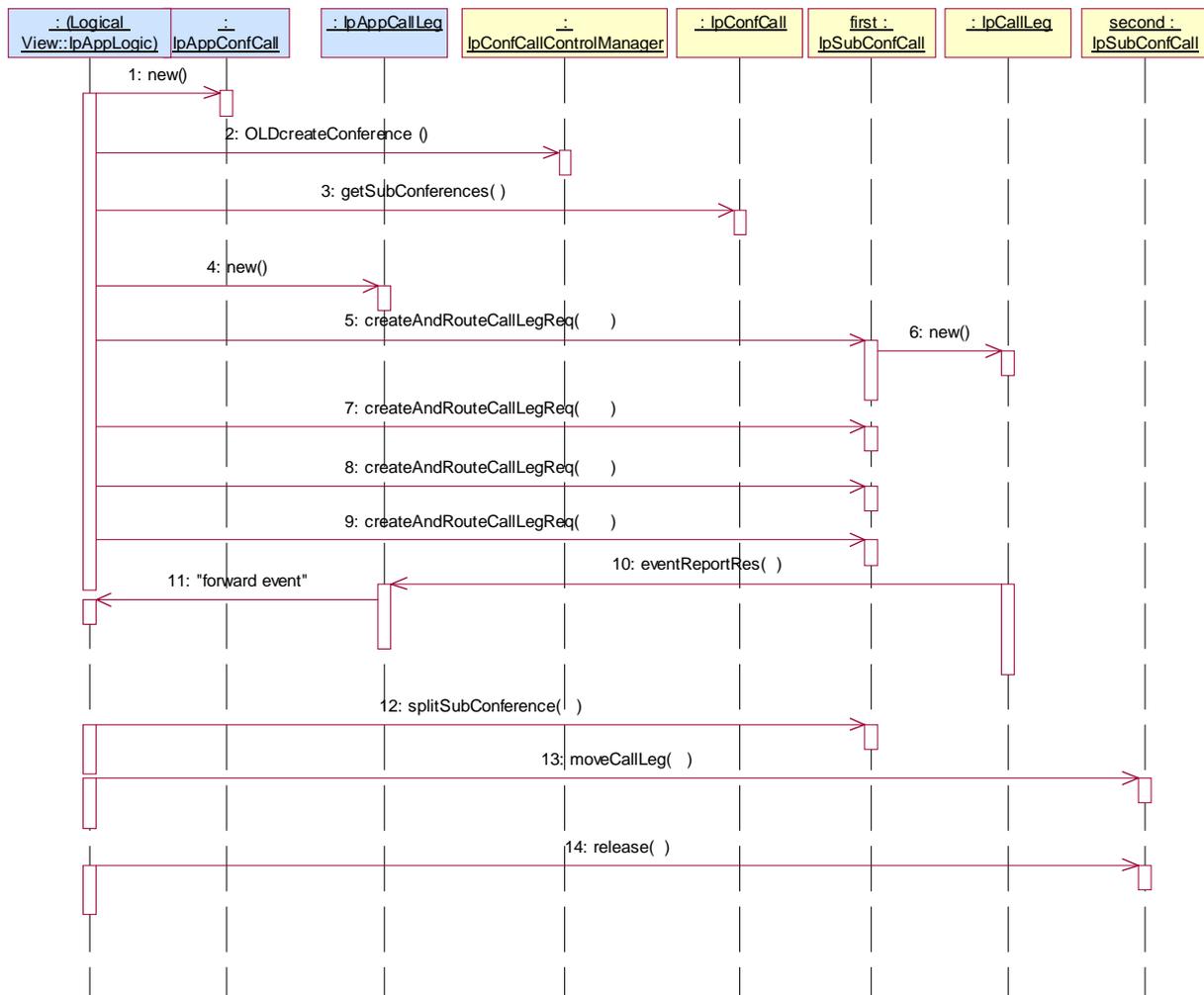
The reservation also includes the conference policy. One of the elements is whether joined parties must be explicitly attached. If so, this is treated as an implicit joinMonitorReq.

- 3: The conference is created.
- 4: The message is forwarded to the application.
- 5: The application creates an object to receive the call back messages from the conference call.
- 6: The application also requests to be notified when parties leave the conference.
- 7: The application is notified of the first party that joined the conference.
- 8: When the party is allowed to join the conference, the party is added.  
Alternatively, the party could have been rejected with a releaseCallLeg.
- 9: A new party joins the conference and the application is notified.
- 10: The message is forwarded to the application.
- 11: This party also is allowed into the conference by attaching the leg.
- 12: A party leaves the conference.
- 13: The message is forwarded to the application.
- 14: The application decides to release the entire conference.

### 9.1.2 Non-add hoc add-on with subconferencing

This sequence illustrates a prearranged add-on conference. The end user that initiates the call, communicates with the conference application via a web interface (not shown). By dragging and dropping names from the addressbook, the end-users adds parties to the conference.

Also via the web-interface, the end-user can group parties in subconferences. Only parties in the same subconference can talk to each other.



- 1: The application creates a new interface to receive the callbacks from the conference call.
- 2: The application initiates the conference. There has been no prior resource reservation, so there is a chance that no resources are available when parties are added to the conference.

The conferenceCall interface object is returned.

- 3: Together with the conference a subconference is implicitly created.

However, the subconference is not returned as a result of the createConference, therefore the application uses this method to get the subconference.

- 4: The application creates a new IpAppCallLeg interface.
- 5: The application adds the first party to the subconference. This process is repeated for all 4 parties. Note that in the following not all steps are shown.
- 6: The gateway creates a new IpCallLeg interface.
- 7: The application adds parties to the subconference.
- 8: The application adds parties to the subconference.
- 9: The application adds parties to the subconference.
- 10: When a party A answers the application is notified.

We assume that all parties answer. This happens in the same way as for party A and is not shown in the following.

11: The message is forwarded to the application.

12: The application decides to split the conference. Party C & D are indicated in the message.

The gateway will create a new subconference and move party C and D to the new subconference.

The configuration is A & B are in speech, C & D are in speech. There is no bearer connection between the two subconferences.

13: The application moves one of the legs from the second subconference back to the first. The configuration now is A, B & C are in speech configuration. D is alone in its own subconference.

14: The second subconference is released. Since party D was in this subconference, this calleg is also released.

This leaves one subconference with A, B & C.

### 9.1.3 Non-adhoc add-on multimedia

This sequence illustrates a prearranged add-on multi-media conference. The end user that initiates the call, communicates with the conference application via a web interface (not shown). By dragging and dropping names from the addressbook, the end-users adds parties to the conference.

Also via the web-interface, the end-user can do things that normally the chair would be able to do, e.g. determine who has the floor (e.g. whose video is being broadcast to the other participants) or inspect the video of participants who do not have the floor (e.g. to see how they react to the current speaker).



- 1: The application creates a new object for receiving callbacks from the MMSubConference.
- 2: When the user selects the appropriate option in the web interface, the application will create a conference without resource reservation. The policy for video is set to "chairperson" switched.
- 3: The application requests the subconference that was implicitly created together with the conference.
- 4: The application creates a new IpAppCallLeg interface.
- 5: The application adds the first party to the subconference. This process is repeated for all 4 parties. Note that in the following not all steps are shown.
- 6: The gateway creates a new IpCallLeg interface.
- 7: The application creates a new IpAppCallLeg interface.
- 8: The application add parties to the conference and monitors on success.
- 9: The gateway creates a new IpCallLeg interface.
- 10: The application add parties to the conference and monitors on success.
- 11: The application add parties to the conference and monitors on success.

12: When a party A answers the application is notified.

We assume that all parties answer.

14: We assume that A was the initiating party.

The initiating end-user is assigned the chairpersonship.

This message is needed to synchronise the chairpersonship in the application with the MCU chairpersonship, since the chair can also use H.323 messages to control the conference.

15: When a party B answers the application is notified. We assume the other parties answer as well and this is not shown below in the sequence.

16: Chairperson (A) decides via WWW interface that party B is the speaker. This means that the video of B is broadcast to the rest.

17: The chairperson select the video of C in order to judge their reactions on B's proposal.

18: The chairperson select the video of D in order to judge their reactions on B's proposal.

19: The chairperson goes back to receiving the broadcasted videostream (B).

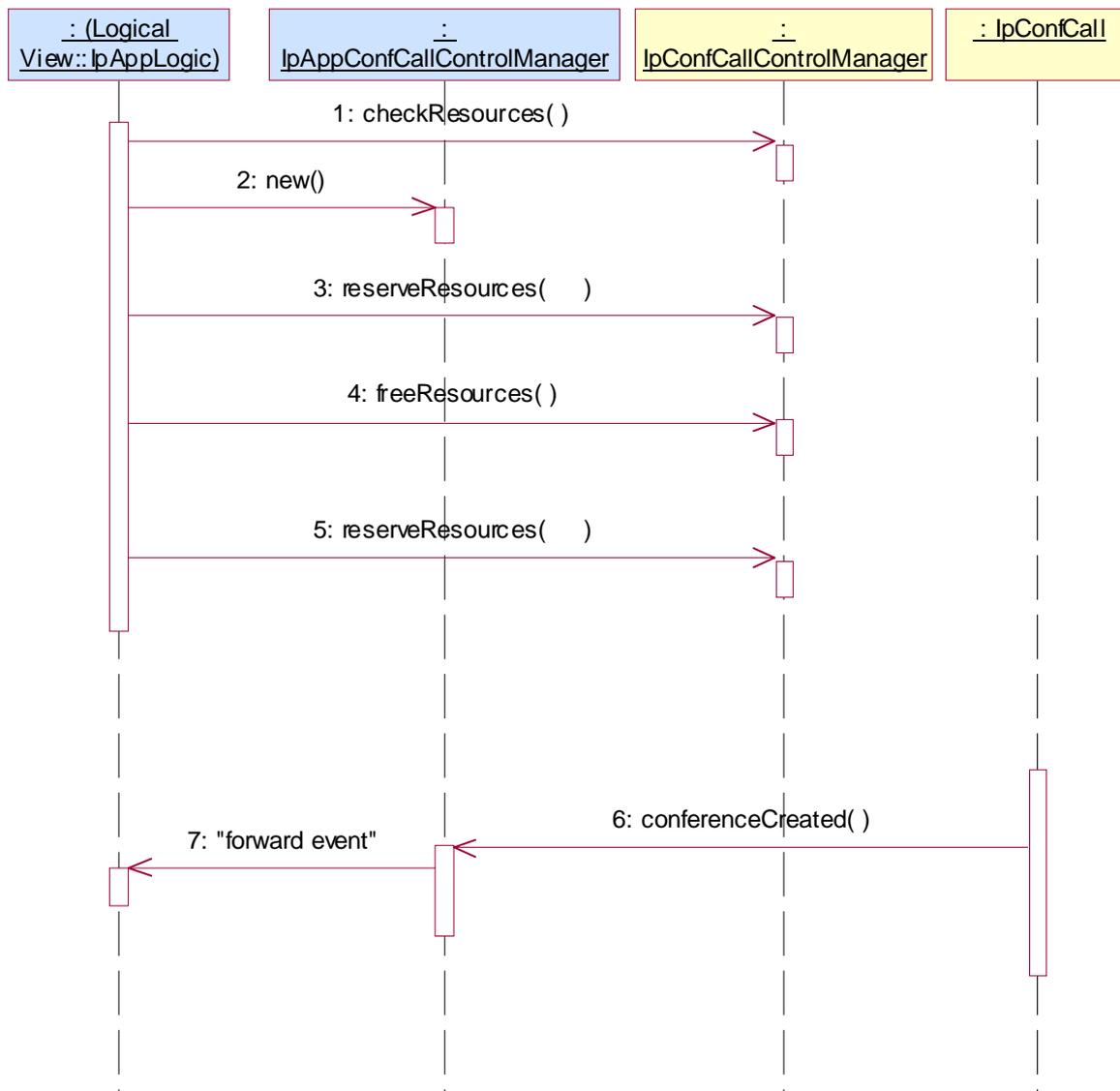
20: User C requests the floor via the H.323 signals. The application is notified of this.

21: The message is forwarded to the application logic.

22: The chairperson (via the WWW interface) grants the request by appointing C as the speaker.

## 9.1.4 Resource Reservation

This sequence illustrates how an application can check and reserve resources for a meet-me conference.



- 1: The application checks if enough conference resources are available in a given time period.
- 2: The application creates a object to receive callback messages.
- 3: The application reserves resources for the time period. The callback object is in order to receive a notification when the conference is started.
- 4: Because the time was wrong by accident, the application cancels the earlier reservation.
- 5: The application makes a new reservation.
- 6: At the specified time, or when the first party joins the conference the application is notified.
- 7: The event is forwarded to the application.

## 9.2 Class Diagrams

The conference call control service consists of two packages, one for the interfaces on the application side and one for interfaces on the service side. The class diagrams in the following figures show the interfaces that make up the conference call control application package and the conference call control service package.

This class diagram shows the interfaces that make up the application conference call control service package and the relation to the interfaces in the conference call control service package.

The diagram also shows the inheritance relation between the multi-party call application interfaces and the conference call application interfaces; the conference interfaces are specialisations of the corresponding multi-party call interfaces.

Communication between the application and service packages is done via the <<uses>> relations; the interfaces can communicate with callback methods in the corresponding application interfaces.

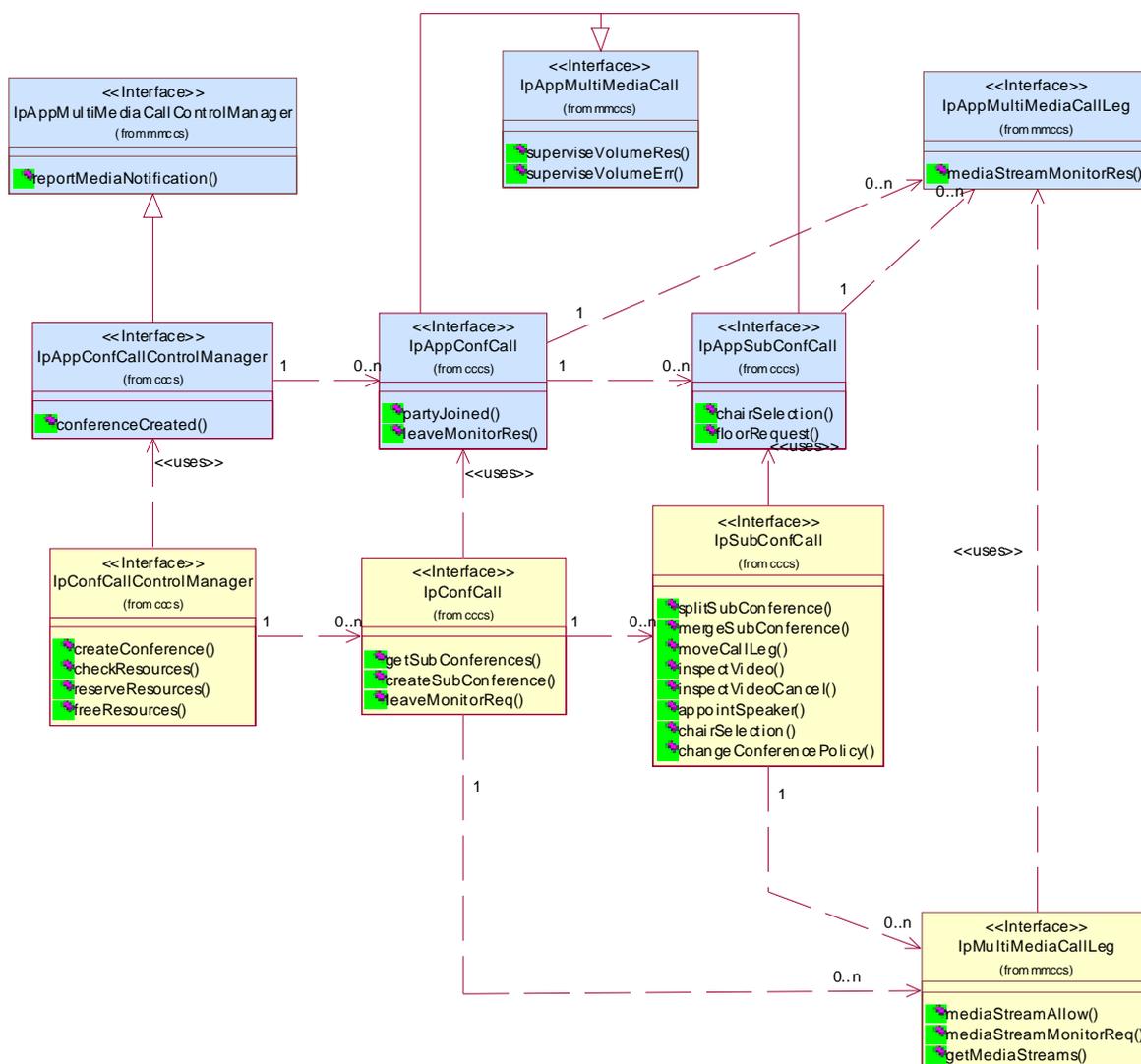


Figure 17: Application Interfaces

This class diagram shows the interfaces that make up the conference call control service package.

The diagram also shows the inheritance relation between the multi-party call interfaces and the conference call interfaces; the conference interfaces are specialisations of the corresponding multi-party call interfaces.

Furthermore, the class diagram illustrates that the conference call control manager can instantiate or be associated with zero or more conference calls. Each conference call can have one or more subconferences associated with it. Each subconference contains zero or more call legs associated with it. Detached legs are not associated with any specific subconference, instead they are associated with the conference call itself.

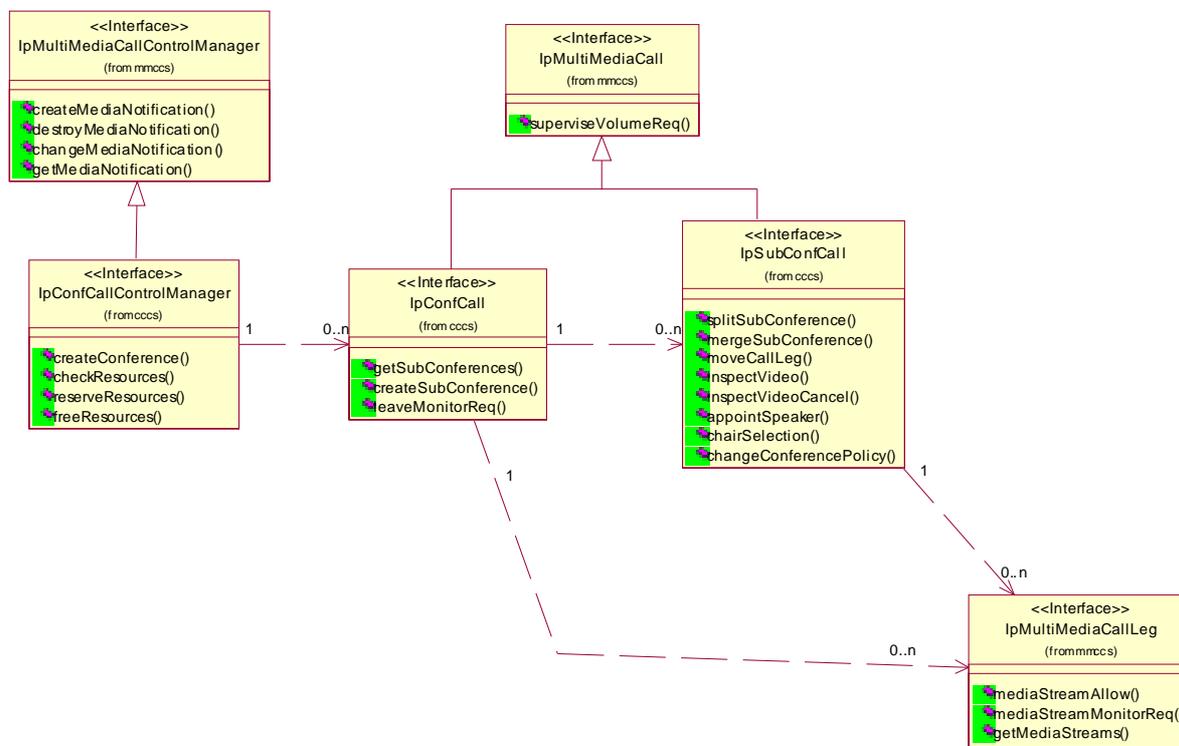


Figure 18: Service Interfaces

## 9.3 Conference Call Control Service Interface Classes

The Conference Call Control Service enhances the multi-media call control service. The conference call control service gives the application the ability to manipulate subconferences within a conference. A subconference defines the grouping of legs within the overall conference call. Only parties in the same subconference have a bearer connection (or media channel connection) to each other (e.g. can speak to each other). The application can:

- create new subconferences within the conference, either as an empty subconference or by splitting an existing subconference in two subconferences;
- move legs between subconferences;
- merge subconferences;
- get a list of all subconferences in the call.

The generic conference also gives the possibility to manipulate typical multi-media conference details, such as:

- interworking with network signalled conference protocols (e.g. H.323);
- manipulation of the media in the MCU, e.g. broadcasting of video;
- handling of multi-media conference policies, e.g. how video should be handled, voice controlled switched or chair controlled.

Furthermore the conference call control service adds support for the reservation of resources needed for conferencing. The application can:

- reserve resources for a predefined time period;
- free reserved resources;
- search for the availability of conference resources based on a number of criteria.

There are two ways to initiate a conference:

- the conferences can be started on the pre-arranged time by the service, at the start time indicated in the reservation. The application is notified about this. The application can then add parties to the conference and/or parties can dial-in to the conference using the address provided during reservation;
- the conference can be created directly on request of the application using the createConference method in the IpConfCallControlManager interface.

Each Conference Call Control interface inherits from a Multi Media Call Control interface, which in turn inherits from Multi Party Call Control. It is possible to implement conference call control without any multi-media features, using only those inherited methods which come from Multi Party Call Control, in addition to the Conference Call Control methods. The minimum required method set for each Conference Call Control interface reflects this possibility, by not requiring the Multi Media Call Control methods.

### 9.3.1 Interface Class IpConfCallControlManager

Inherits from: IpMultiMediaCallControlManager.

The conference Call Control Manager is the factory interface for creating conferences. Additionally it takes care of resource management.

This interface shall be implemented by a Conference Call Control SCF. As a minimum requirement, either the createConference() method shall be implemented, or the reserveResources() and freeResources() methods shall be implemented. The minimum required methods from IpMultiPartyCallControlManager are also required.

<<Interface>> IpConfCallControlManager
<pre> createConference (appConferenceCall : in IpAppConfCallRef, numberOfSubConferences : in TpInt32,   conferencePolicy : in TpConfPolicy, numberOfParticipants : in TpInt32, duration : in TpDuration) :   TpConfCallIdentifier checkResources (searchCriteria : in TpConfSearchCriteria) : TpConfSearchResult reserveResources (appInterface : in IpAppConfCallControlManagerRef, startTime : in TpDateAndTime,   numberOfParticipants : in TpInt32, duration : in TpDuration, conferencePolicy : in TpConfPolicy) :   TpResourceReservation freeResources (resourceReservation : in TpResourceReservation) : void </pre>

#### *Method*

#### **createConference ( )**

This method is used to create a new conference. If the specified resources are not available for the indicated duration the creation is rejected with P\_RESOURCES\_UNAVAILABLE.

Returns conference : Specifies the interface reference and sessionID of the created conference.

### *Parameters*

**appConferenceCall : in IpAppConfCallRef**

Specifies the callback interface for the conference created.

**numberOfSubConferences : in TpInt32**

Specifies the number of subconferences that the user wants to create automatically. The references to the interfaces of the subconferences can later be requested with `getSubConferences`.

The number of subconferences should be at least 1.

**conferencePolicy : in TpConfPolicy**

Specifies the policy to be applied for the conference, e.g. are parties allowed to join (call into) the conference?

Note that if parties are allowed to join the conference, the application can expect `partyJoined()` messages on the `IpAppConfCall` interface.

**numberOfParticipants : in TpInt32**

Specifies the number of participants in the conference. The actual number of participants may exceed this, but these resources are not guaranteed, i.e. anything exceeding this will be best effort only and the conference service may drop or reject participants in order to fulfil other committed resource requests. By specifying 0, the application can request a best effort conference.

**duration : in TpDuration**

Specifies the duration for which the conference resources are reserved. The duration of the conference may exceed this, but after the duration, the resources are no longer guaranteed, i.e. parties may be dropped or rejected by the service in order to satisfy other committed resource requests. When the conference is released before the allocated duration, the reserved resources are released and can be used to satisfy other resource requests. By specifying 0, the application requests a best effort conference.

### *Returns*

**TpConfCallIdentifier**

### *Raises*

**TpCommonExceptions**

### *Method*

**checkResources ( )**

This method is used to check for the availability of conference resources.

The input is the search period (start and stop time and date) - mandatory.

Furthermore, a conference duration and number of participants can be specified - optional.

The search algorithm will search the specified period for availability of conference resources and tries to find an optimal solution.

When a match is found the actual number of available resources, the actual start and the actual duration for which these are available is returned. These values can exceed the requested values.

When no match is found a best effort is returned, still the actual start time, duration, number of resources are returned, but these values now indicate the best that the conference bridge can offer, e.g. one or more of these values will not reach the requested values.

Returns result : Specifies the result of the search. It indicates if a match was found. If no exact match was found the best attempt is returned.

*Parameters***searchCriteria : in TpConfSearchCriteria**

Specifies the boundary conditions of the search. E.g. the time period that should be searched, the number of participants.

*Returns***TpConfSearchResult***Raises***TpCommonExceptions***Method***reserveResources ( )**

This method is used to reserve conference resources for a given time period. Conferences can be created without first reserving resources, but in that case no guarantees can be made.

Returns resourceReservation : Specifies a structured data type which contains two fields:

ResourceID: The address with which the conference can be addressed, both in the methods of the interface and in the network, i.e. if joinAllowed is TRUE, parties can use this address to join the conference.

If no match is found the ResourceID contains an empty address.

ReservationID: Specifies the reservation made. It should be unique in a particular resource.

*Parameters***appInterface : in IpAppConfCallControlManagerRef**

Specifies the callback interface to be used when the conference is created in the network. The application will receive the conferenceCreated message when a conference is created in the network.

**startTime : in TpDateAndTime**

Specifies the time at which the conference resources should be reserved, i.e. the start time of the conference.

**numberOfParticipants : in TpInt32**

Specifies the number of participants in the conference. The actual number of participants may exceed this, but these resources are not guaranteed, i.e. anything exceeding this will be best effort only and the conference service may drop or reject participants in order to fulfil other committed resource requests.

**duration : in TpDuration**

Specifies the duration for which the conference resources are reserved. The duration of the conference may exceed this, but after the duration, the resources are no longer guaranteed, i.e. parties may be dropped or rejected by the service in order to satisfy other committed resource requests. When the conference is released before the allocated duration, the reserved resources are released and can be used to satisfy other resource requests.

**conferencePolicy : in TpConfPolicy**

The policy to be applied for the conference, e.g. are parties allowed to join (call into) the conference? Note that if parties are allowed to join the conference, the application can expect partyJoined() messages on the appConfCall.

*Returns*

**TpResourceReservation**

*Raises*

**TpCommonExceptions**

*Method*

**freeResources ( )**

This method can be used to cancel an earlier made reservation of conference resources.

This also means that no ConferenceCreated events will be received for this conference.

*Parameters*

**resourceReservation : in TpResourceReservation**

Specifies the ResourceID and the ReservationID that were received during the reservation.

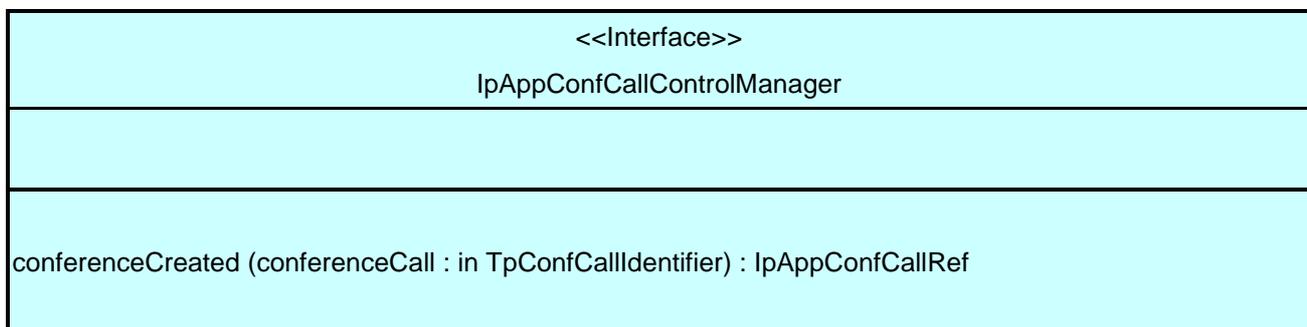
*Raises*

**TpCommonExceptions**

### 9.3.2 Interface Class IpAppConfCallControlManager

Inherits from: IpAppMultiMediaCallControlManager.

The conference call control manager application interface provides the application with additional callbacks when a conference is created by the network (based on an earlier reservation).



*Method*

**conferenceCreated ( )**

This method is called when a conference is created from an earlier resource reservation.

Returns appInterface : Specifies a reference to the application interface which implements the callback interface for the new conference. If the application has previously explicitly passed a reference to the callback using a setCallbackWithSessionID() invocation, this parameter may be null, or if supplied must be the same as that provided during the setCallbackWithSessionID().

*Parameters*

**conferenceCall : in TpConfCallIdentifier**

Specifies the reference to the conference call interface to which the notification relates and the associated sessionID.

*Returns*

**IpAppConfCallRef**

### 9.3.3 Interface Class IpConfCall

Inherits from: IpMultiMediaCall

The conference call manages the subconferences. It also provides some convenience methods to hide the fact of multiple subconferences from the applications that do not need it. Note that the conference call always contains one subconference. The following inherited call methods apply to the conference as a whole, with the specified semantics:

- setCallback; changes the callback interface reference.
- release; releases the entire conference, including all the subconferences and detached legs.
- deassignCall; de-assigns the complete conference. No callbacks will be received by the application, either on the conference, or on any of the contained subconferences or call legs.
  - getInfoReq; request information over the complete conference. The conference duration is defined as the time when the first party joined the conference until when the last party leaves the conference or the conference is released.
- setChargePlan; set the chargeplan for the conference. This chargeplan will apply to all the subconferences, unless another chargeplan is explicitly overridden on the subconference.
  - superviseReq; supervise the duration of the complete conference.
  - getCallLegs; return all the call legs used within the conference.
  - superviseVolumeReq; supervises and sets a granted data volume for the conference.

Other methods apply to the default subconference. When using multiple subconferences, it is recommended that the application calls these methods directly on the subconference since this makes it more explicit what the effect of the method is:

- createAndRouteCallLegReq
- createCallLeg

This interface shall be implemented by a Conference Call Control SCF. As a minimum requirement, the getSubConferences() and createSubConference() methods shall be implemented. The minimum required methods from IpMultiPartyCall are also required.

<<Interface>> IpConfCall
<pre> getSubConferences (conferenceSessionID : in TpSessionID) : TpSubConfCallIdentifierSet createSubConference (conferenceSessionID : in TpSessionID, appSubConference : in   IpAppSubConfCallRef, conferencePolicy : in TpConfPolicy) : TpSubConfCallIdentifier leaveMonitorReq (conferenceSessionID : in TpSessionID) : void </pre>

#### Method

#### **getSubConferences ( )**

This method returns all the subconferences of the conference.

Returns subConferenceList : Specifies the list of all the subconferences of the conference.

#### Parameters

**conferenceSessionID : in TpSessionID**

Specifies the sessionID of the conference.

*Returns***TpSubConfCallIdentifierSet***Raises***TpCommonExceptions, P\_INVALID\_SESSION\_ID***Method***createSubConference ( )**

This method is used to create a new subconference. Note that one subconference is already created together with the conference.

Returns subConference : Specifies the created subconference (interface and sessionID).

*Parameters***conferenceSessionID : in TpSessionID**

Specifies the sessionID of the conference.

**appSubConference : in IpAppSubConfCallRef**

Specifies the call back interface for the created subconference.

**conferencePolicy : in TpConfPolicy**

Conference Policy to be used in the subconference. Optional; if undefined, the policy of the conference is used. Note that not all policy elements have to be applicable for subconferences.

*Returns***TpSubConfCallIdentifier***Raises***TpCommonExceptions, P\_INVALID\_SESSION\_ID***Method***leaveMonitorReq ( )**

This method is used to request a notification when a party leaves the conference.

*Parameters***conferenceSessionID : in TpSessionID**

Specifies the session ID of the conference.

*Raises***TpCommonExceptions, P\_INVALID\_SESSION\_ID**

### 9.3.4 Interface Class IpAppConfCall

Inherits from: IpAppMultiMediaCall.

The Conference Call application interface allows the application to handle call responses and state reports. Additionally it allows the application to handle parties entering and leaving the conference.

<<Interface>> IpAppConfCall
partyJoined (conferenceSessionID : in TpSessionID, callLeg : in mpccs::TpCallLegIdentifier, eventInfo : in TpJoinEventInfo) : mpccs::IpAppCallLegRef leaveMonitorRes (conferenceSessionID : in TpSessionID, callLeg : in TpSessionID) : void

*Method***partyJoined( )**

This asynchronous method indicates that a new party (leg) has joined the conference. This can be used in, e.g. a meetme conference where the participants dial in to the conference using the address returned during reservation of the conference.

The Leg will be assigned to the default subconference object and will be in a detached state. The application may move the call Leg to a different subconference before attaching the media.

The method will only be called when joinAllowed is indicated in the conference policy.

Returns appCallLeg : Specifies the call back interface that should be used for callbacks from the new call Leg.

*Parameters*

**conferenceSessionID : in TpSessionID**

Specifies the session ID of the conference that the party wants to join.

**callLeg : in mpccs::TpCallLegIdentifier**

Specifies the interface and sessionID of the call leg that joined the conference.

**eventInfo : in TpJoinEventInfo**

Specifies the address information of the party that wants to join the conference.

*Returns*

**mpccs::IpAppCallLegRef**

*Method***leaveMonitorRes( )**

This asynchronous method indicates that a party (leg) has left the conference.

*Parameters*

**conferenceSessionID : in TpSessionID**

Specifies the session ID of the conference that the party wants to leaves.

**callLeg : in TpSessionID**

Specifies the sessionID of the call leg that left the conference.

### 9.3.5 Interface Class IpSubConfCall

Inherits from: IpMultiMediaCall.

The subconference is an additional grouping mechanism within a conference. Parties (legs) that are in the same subconference have a speech connection with each other. The following inherited call methods apply to the subconference as a whole, with the specified semantics:

- setCallback; changes the callback interface reference.
- release; releases the subconference, including all currently attached legs. When the last subconference in the conference is released, the conference is implicitly released as well.
- deassignCall; de-assigns the subconference. No callbacks will be received by the application on this subconference, nor will the gateway accept any methods on this subconference or accept any methods using the subconference as a parameter (e.g. merge). When the subconference is the last subconference in the conference, the conference is deassigned as well. In general it is recommended to only use deassignCall for the complete conference.
- getInfoReq; request information over the subconference. The subconference duration is defined as the time when the first party joined the subconference until when the last party leaves the subconference or the subconference is released.
- setChargePlan; set the charge plan for the subconference.
- superviseReq; supervise the duration of the subconference. It is recommended that this method is only used on the complete conference.
- superviseVolumeReq; supervises and sets a granted data volume for the subconference.
- getCallLegs; return all the call legs in the subconference.
- createCallLeg; create a call leg.
- createAndRouteCallLegReq; implicitly create a leg and route the leg to the specified destination.

This interface shall be implemented by a Conference Call Control SCF. As a minimum requirement, either the moveCallLeg() method shall be implemented, or the splitSubConference() and mergeSubConference() methods shall be implemented. The minimum required methods from IpMultiPartyCall are also required.

<<Interface>> IpSubConfCall
<pre> splitSubConference (subConferenceSessionID : in TpSessionID, callLegList : in TpSessionIDSet,   appSubConferenceCall : in IpAppSubConfCallRef) : TpSubConfCallIdentifier mergeSubConference (subConferenceCallSessionID : in TpSessionID, targetSubConferenceCall : in   TpSessionID) : void moveCallLeg (subConferenceCallSessionID : in TpSessionID, targetSubConferenceCall : in TpSessionID,   callLeg : in TpSessionID) : void inspectVideo (subConferenceSessionID : in TpSessionID, inspectedCallLeg : in TpSessionID) : void inspectVideoCancel (subConferenceSessionID : in TpSessionID) : void appointSpeaker (subConferenceSessionID : in TpSessionID, speakerCallLeg : in TpSessionID) : void chairSelection (subConferenceSessionID : in TpSessionID, chairCallLeg : in TpSessionID) : void changeConferencePolicy (subConferenceSessionID : in TpSessionID, conferencePolicy : in TpConfPolicy) :   void           </pre>

#### *Method*

#### **splitSubConference()**

This method is used to create a new subconference and move some of the legs to it.

Returns newSubConferenceCall : Specifies the new subconference that is implicitly created as a result of the method.

*Parameters***subConferenceSessionID : in TpSessionID**

Specifies the session ID of the subconference.

**callLegList : in TpSessionIDSet**

Specifies the sessionIDs of the legs that will be moved to the new subconference.

**appSubConferenceCall : in IpAppSubConfCallRef**

Specifies the application call back interface for the new subconference.

*Returns***TpSubConfCallIdentifier***Raises***TpCommonExceptions, P\_INVALID\_SESSION\_ID***Method***mergeSubConference()**

This method is used to merge two subconferences, i.e. move all our legs from this subconference to the other subconference followed by a release of this subconference.

*Parameters***subConferenceCallSessionID : in TpSessionID**

Specifies the session ID of the subconference.

**targetSubConferenceCall : in TpSessionID**

The session ID of target subconference with which the current subconference will be merged.

*Raises***TpCommonExceptions, P\_INVALID\_SESSION\_ID***Method***moveCallLeg()**

This method moves one leg from this subconference to another subconference.

*Parameters***subConferenceCallSessionID : in TpSessionID**

Specifies the session ID of the source subconference.

**targetSubConferenceCall : in TpSessionID**

Specifies the sessionID of the target subconference.

**callLeg : in TpSessionID**

Specifies the sessionID of the call leg to be moved.

*Raises***TpCommonExceptions, P\_INVALID\_SESSION\_ID**

*Method***inspectVideo()**

This method can be used by the application to select which video should be sent to the party that is currently selected as the chair.

Whether this method can be used depends on the selected conference policy.

*Parameters*

**subConferenceSessionID : in TpSessionID**

Specifies the session ID of the multi media subconference.

**inspectedCallLeg : in TpSessionID**

Specifies the sessionID of call leg of the party whose video stream should be sent to the chair.

*Raises*

**TpCommonExceptions, P\_INVALID\_SESSION\_ID**

*Method***inspectVideoCancel()**

This method cancels a previous inspectVideo. The chair will receive the broadcasted video.

Whether this method can be used depends on the selected conference policy.

*Parameters*

**subConferenceSessionID : in TpSessionID**

Specifies the session ID of the multi media subconference.

*Raises*

**TpCommonExceptions, P\_INVALID\_SESSION\_ID**

*Method***appointSpeaker()**

This method indicates which of the participants in the conference has the floor. The video of the speaker will be broadcast to the other parties.

Whether this method can be used depends on the selected conference policy.

*Parameters*

**subConferenceSessionID : in TpSessionID**

Specifies the session ID of the multi media subconference.

**speakerCallLeg : in TpSessionID**

Specifies the sessionID of the call leg of the party whose video stream should be broadcast.

*Raises*

**TpCommonExceptions, P\_INVALID\_SESSION\_ID**

*Method***chairSelection()**

This method is used to indicate which participant in the conference is the chair. E.g. the terminal of this participant will be the destination of the video of the inspectVideo method.

Whether this method can be used depends on the selected conference policy.

*Parameters*

**subConferenceSessionID : in TpSessionID**

Specifies the session ID of the multi media subconference.

**chairCallLeg : in TpSessionID**

Specifies the sessionID of the call leg of the party that will become the chair.

*Raises*

**TpCommonExceptions, P\_INVALID\_SESSION\_ID**

*Method***changeConferencePolicy()**

This method can be used to change the conference policy in an ongoing conference.

- Multi media conference policy options available. E.g.;
- chair controlled video/voice switched video;
- closed conference/open conference;
- Composite video (different types)/only speaker.

*Parameters*

**subConferenceSessionID : in TpSessionID**

Specifies the session ID of the multi media subconference.

**conferencePolicy : in TpConfPolicy**

New Conference Policy to be used in the subconference.

*Raises*

**TpCommonExceptions, P\_INVALID\_SESSION\_ID**

### 9.3.6 Interface Class IpAppSubConfCall

Inherits from: IpAppMultiMediaCall.

The Sub Conference Call application interface allows the application to handle call responses and state reports from a sub conference.

<<Interface>> IpAppSubConfCall
chairSelection (subConferenceSessionID : in TpSessionID, callLegSessionID : in TpSessionID) : void floorRequest (subConferenceSessionID : in TpSessionID, callLegSessionID : in TpSessionID) : void

*Method***chairSelection( )**

This method is used to inform the application about the chair selection requests from the network. The application can grant the request by calling the chairSelection method on the subconference.

*Parameters*

**subConferenceSessionID : in TpSessionID**

Specifies the session ID of the subconference where the chair request originates.

**callLegSessionID : in TpSessionID**

Specifies the session ID of the call leg making the chair request.

*Method***floorRequest( )**

This method is used to inform the application about the floor requests from the network. The application can grant the request by calling the appointSpeaker method.

*Parameters*

**subConferenceSessionID : in TpSessionID**

Specifies the session ID of the subconference where the floor request originates.

**callLegSessionID : in TpSessionID**

Specifies the session ID of the call leg making the floor request.

## 9.4 Conference Call Control Service State Transition Diagrams

There are no State Transition Diagrams for the Conference Call Control Service package.

## 9.5 Conference Call Control Data Definitions

This clause provides the Conference call control data definitions necessary to support the API specification.

The general format of a data definition specification is described below:

- Data Type

This shows the name of the data type.

- Description

This describes the data type.

- Tabular Specification

This specifies the data types and values of the data type.

- Example

If relevant, an example is shown to illustrate the data type.

All data types referenced but not defined in this clause are either in the common call control data definitions clause of the present document (clause 10) or in the common data definitions which may be found in ES 201 915-2.

## 9.5.1 Event Notification Data Definitions

No specific event notification data.

## 9.5.2 Conference Call Control Data Definitions

### 9.5.2.1 IpConfCall

Defines the address of an IpConfCall Interface.

### 9.5.2.2 IpConfCallRef

Defines a Reference to type IpConfCall.

### 9.5.2.3 IpAppConfCall

Defines the address of an IpAppConfCall Interface.

### 9.5.2.4 IpAppConfCallRef

Defines a Reference to type IpAppConfCall.

### 9.5.2.5 IpSubConfCall

Defines the address of an IpSubConfCall Interface.

### 9.5.2.6 IpSubConfCallRef

Defines a Reference to type IpSubConfCall.

### 9.5.2.7 IpAppSubConfCall

Defines the address of an IpAppSubConfCall Interface.

### 9.5.2.8 IpAppSubConfCallRef

Defines a Reference to type IpAppSubConfCall.

### 9.5.2.9 TpSubConfCallIdentifierSet

Defines a Numbered Set of Data Elements of IpSubConfCallIdentifier.

### 9.5.2.10 TpConfCallIdentifier

Defines the Sequence of Data Elements that unambiguously specify the Conference Call object.

Sequence Element Name	Sequence Element Type	Sequence Element Description
ConfCallReference	IpConfCallRef	This element specifies the interface reference for the conference call object.
ConfCallSessionID	TpSessionID	This element specifies the session ID of the conference call.

### 9.5.2.11 TpSubConfCallIdentifier

Defines the Sequence of Data Elements that unambiguously specify the SubConference Call object.

Sequence Element Name	Sequence Element Type	Sequence Element Description
SubConfCallReference	IpSubConfCallRef	This element specifies the interface reference for the subconference call object.
SubConfCallSessionID	TpSessionID	This element specifies the session ID of the subconference call.

### 9.5.2.12 IpAppConfCallControlManager

Defines the address of an IpAppConfCallControlManager Interface.

### 9.5.2.13 IpAppConfCallControlManagerRef

Defines a Reference to type IpAppConfCallControlManager.

### 9.5.2.14 TpConfPolicyType

Defines policy type for the conference.

If undefined the gateway will select an appropriate default.

If a mono media conference policy is specified for a multi-media conference, the gateway will select appropriate defaults for the multi-media policy items.

If a multi-media policy is selected for a mono-media (voice-only) conference, the multi-media conference items will be ignored.

Name	Value	Description
P_CONFERENCE_POLICY_UNDEFINED	0	Undefined
P_CONFERENCE_POLICY_MONOMEDIA	1	CCCS - monomedia conference policy
P_CONFERENCE_POLICY_MULTIMEDIA	2	MMCCS - multimedia conference policy

### 9.5.2.15 TpConfPolicy

Defines the Tagged Choice of Data Elements that specify the policy that needs adhered to by the conference.

Tag Element Type
TpConfPolicyType

Tag Element Value	Choice Element Type	Choice Element Name
P_CONFERENCE_POLICY_MONOMEDIA	TpMonoMediaConfPolicy	MonoMedia
P_CONFERENCE_POLICY_MULTIMEDIA	TpMultiMediaConfPolicy	MultiMedia

### 9.5.2.16 TpMonoMediaConfPolicy

Defines the type of conference policy as a sequence of Policy Items and their values.

For mono media there are only two types of conference policies; specified, i.e. the application provides the policy, or undefined, i.e. the GW may choose a default conference policy.

Sequence Element Name	Sequence Element Type	Description
JoinAllowed	TpBoolean	Specifies if dial-in to the conference is allowed. Parties can dial-in to the conference using the address returned during reservation. If this is specified the application will receive partyJoined for each participant dialling into the conference.

### 9.5.2.17 TpJoinEventInfo

Defines the Sequence of Data Elements that specify the information returned to the application in a Join event notification.

Sequence Element Name	Sequence Element Type
DestinationAddress	TpAddress
OriginatingAddress	TpAddress
OriginalDestinationAddress	TpAddress
RedirectingAddress	TpAddress
CallAppInfo	TpCallAppInfoSet

### 9.5.2.18 TpConfSearchCriteria

Defines the Sequence of Data Elements that specify the criteria for doing a search for available conference resources.

Sequence Element Name	Sequence Element Type
StartSearch	TpDateAndTime
StopSearch	TpDateAndTime
RequestedResources	TpInt32
RequestedDuration	TpDuration

### 9.5.2.19 TpConfSearchResult

Defines the Sequence of Data Elements that specifies the result of a search for available conference resources.

Sequence Element Name	Sequence Element Type
MatchFound	TpBoolean
ActualStartTime	TpDateAndTime
ActualResources	TpInt32
ActualDuration	TpDuration

### 9.5.2.20 TpMultiMediaConfPolicy

Sequence of items for multi-media conferences.

Sequence Element Name	Sequence Element Type	Description
JoinAllowed	TpBoolean	Specifies if dial-in to the conference is allowed. Parties can dial-in to the conference using the address returned during reservation. If this is specified the application will receive partyJoined for each participant dialling into the conference.
MediaAllowed	TpMediaType	Specifies the media that are allowed to be used by the participants. E.g. this can be used to limit the conference to audio only, even when all participants support video.
Chaired	TpBoolean	Specifies whether the conference is chaired or free. In a chaired conference the application or one of the participants acting as chair has special privileges; e.g. can control the video distribution.
VideoHandling	TpVideoHandlingType	Specifies how the video should be handled.

### 9.5.2.21 TpResourceReservation

Defines the Sequence of Data Elements that specifies the result of a search for available conference resources.

Sequence Element Name	Sequence Element Type	Sequence Element Description
ResourceID	TpAddress	The address with which the conference can be addressed
ReservationID	TpInt32	Specifies the reservation made. It should be unique in a particular resource

### 9.5.2.22 TpVideoHandlingType

Defines how video should be handled in the conference.

Name	Value	Description
P_MIXED_VIDEO	0	Video is mixed, no special treatment of speaker
P_SWITCHED_VIDEO_CHAIR_CONTROLLED	1	Video is switched, chair determines the speaker
P_SWITCHED_VIDEO_VOICE_CONTROLLED	2	Video is switched automatically based on audio output of the speaker

---

## 10 Common Call Control Data Types

The following data types referenced in this clause are defined in ES 201 915-5:

TpUIInfo

All other data types referenced but not defined in this clause are common data definitions which may be found in ES 201 915-2.

### 10.1 TpCallAlertingMechanism

This data type is identical to a TpInt32, and defines the mechanism that will be used to alert a call party. The values of this data type are operator specific.

## 10.2 TpCallBearerService

This data type defines the type of call application-related specific information (Q.931: Information Transfer Capability, and TS 122 002).

Name	Value	Description
P_CALL_BEARER_SERVICE_UNKNOWN	0	Bearer capability information unknown at this time
P_CALL_BEARER_SERVICE_SPEECH	1	Speech
P_CALL_BEARER_SERVICE_DIGITALUNRESTRICTED	2	Unrestricted digital information
P_CALL_BEARER_SERVICE_DIGITALRESTRICTED	3	Restricted digital information
P_CALL_BEARER_SERVICE_AUDIO	4	3,1 kHz audio
P_CALL_BEARER_SERVICE_DIGITALUNRESTRICTED TONES	5	Unrestricted digital information with tones/announcements
P_CALL_BEARER_SERVICE_VIDEO	6	Video

## 10.3 TpCallChargePlan

Defines the Sequence of Data Elements that specify the charge plan for the call.

Sequence Element Name	Sequence Element Type	Description
ChargeOrderType	TpCallChargeOrderCategory	Charge order
TransparentCharge	TpOctetSet	Operator specific charge plan specification, e.g. charging table name/charging table entry. The associated charge plan data will be sent transparently to the charging records. Only applicable when transparent charging is selected.
ChargePlan	TpInt32	Pre-defined charge plan. Example of the charge plan set from which the application can choose could be : (0 = normal user, 1 = silver card user, 2 = gold card user). Only applicable when predefined charge plan is selected.
AdditionalInfo	TpOctetSet	Descriptive string which is sent to the billing system without prior evaluation. Could be included in the ticket.
PartyToCharge	TpCallPartyToChargeType	Identifies the entity or party to be charged for the call or call leg.
PartyToChargeAdditionalInfo	TpCallPartyToChargeAdditionalInfo	Contains additional information regarding the charged party.

## 10.4 TpCallPartyToChargeAdditionalInfo

Defines the Tagged Choice of Data Elements that identifies the entity or party to be charged.

Tag Element Type
TpCallPartyToChargeType

Tag Element Value	Choice Element Type	Choice Element Name
P_CALL_PARTY_ORIGINATING	NULL	Undefined
P_CALL_PARTY_DESTINATION	NULL	Undefined
P_CALL_PARTY_SPECIAL	TpAddress	CallPartySpecial

## 10.5 TpCallPartyToChargeType

Defines the type of call party to charge.

Name	Value	Description
P_CALL_PARTY_ORIGINATING	0	Calling party, i.e. party that initiated the call. For application initiated calls this indicates the first party of the call
P_CALL_PARTY_DESTINATION	1	Called party
P_CALL_PARTY_SPECIAL	2	An address identifying e.g. a third party, a service provider

## 10.6 TpCallChargeOrderCategory

Defines the type of charging to be applied.

Name	Value	Description
P_CALL_CHARGE_TRANSPARENT	0	Operator specific charge plan specification, e.g. charging table name/charging table entry. The associated charge plan data will be send transparently to the charging records
P_CALL_CHARGE_PREDEFINED_SET	1	Pre-defined charge plan. Example of the charge plan set from which the application can choose could be : (0 = normal user, 1 = silver card user, 2 = gold card user).

## 10.7 TpCallEndedReport

Defines the Sequence of Data Elements that specify the reason for the call ending.

Sequence Element Name	Sequence Element Type	Description
CallLegSessionID	TpSessionID	The leg that initiated the release of the call. If the call release was not initiated by the leg, then this value is set to -1.
Cause	TpReleaseCause	The cause of the call ending.

## 10.8 TpCallError

Defines the Sequence of Data Elements that specify the additional information relating to a call error.

Sequence Element Name	Sequence Element Type
ErrorTime	TpDateAndTime
ErrorType	TpCallErrorType
AdditionalErrorInfo	TpCallAdditionalErrorInfo

## 10.9 TpCallAdditionalErrorInfo

Defines the Tagged Choice of Data Elements that specify additional call error and call error specific information. This is also used to specify call leg errors and information errors.

	Tag Element Type	
	TpCallErrorType	

Tag Element Value	Choice Element Type	Choice Element Name
P_CALL_ERROR_UNDEFINED	NULL	Undefined
P_CALL_ERROR_INVALID_ADDRESS	TpAddressError	CallErrorInvalidAddress
P_CALL_ERROR_INVALID_STATE	NULL	Undefined
P_CALL_ERROR_RESOURCE_UNAVAILABLE	NULL	Undefined

## 10.10 TpCallErrorType

Defines a specific call error.

Name	Value	Description
P_CALL_ERROR_UNDEFINED	0	Undefined; the method failed or was refused, but no specific reason can be given.
P_CALL_ERROR_INVALID_ADDRESS	1	The operation failed because an invalid address was given
P_CALL_ERROR_INVALID_STATE	2	The call was not in a valid state for the requested operation
P_CALL_ERROR_RESOURCE_UNAVAILABLE	3	There are not enough resources to complete the request successfully

## 10.11 TpCallInfoReport

Defines the Sequence of Data Elements that specify the call information requested. Information that was not requested is invalid.

Sequence Element Name	Sequence Element Type	Description
CallInfoType	TpCallInfoType	The type of call report.
CallInitiationStartTime	TpDateAndTime	The time and date when the call, or follow-on call, was started.
CallConnectedToResourceTime	TpDateAndTime	The date and time when the call was connected to the resource. This data element is only valid when information on user interaction is reported.
CallConnectedToDestinationTime	TpDateAndTime	The date and time when the call was connected to the destination (i.e. when the destination answered the call). If the destination did not answer, the time is set to an empty string. This data element is invalid when information on user interaction is reported with an intermediate report.
CallEndTime	TpDateAndTime	The date and time when the call or follow-on call or user interaction was terminated.
Cause	TpReleaseCause	The cause of the termination.

A callInfoReport will be generated at the end of user interaction and at the end of the connection with the associated address. This means that either the destination related information is present or the resource related information, but not both.

## 10.12 TpCallInfoType

Defines the type of call information requested and reported. The values may be combined by a logical "OR" function.

Name	Value	Description
P_CALL_INFO_UNDEFINED	00h	Undefined
P_CALL_INFO_TIMES	01h	Relevant call times
P_CALL_INFO_RELEASE_CAUSE	02h	Call release cause

## 10.13 TpCallLoadControlMechanism

Defines the Tagged Choice of Data Elements that specify the applied mechanism and associated parameters.

Tag Element Type
TpCallLoadControlMechanismType

Tag Element Value	Choice Element Type	Choice Element Name
P_CALL_LOAD_CONTROL_PER_INTERVAL	TpCallLoadControlIntervalRate	CallLoadControlPerInterval

## 10.14 TpCallLoadControlIntervalRate

Defines the call admission rate of the call load control mechanism used. This data type indicates the interval (in milliseconds) between calls that are admitted.

Name	Value	Description
P_CALL_LOAD_CONTROL_ADMIT_NO_CALLS	0	Infinite interval (do not admit any calls)
	1 - 60000	Duration in milliseconds

## 10.15 TpCallLoadControlMechanismType

Defines the type of call load control mechanism to use.

Name	Value	Description
P_CALL_LOAD_CONTROL_PER_INTERVAL	0	admit one call per interval

## 10.16 TpCallMonitorMode

Defines the mode that the call will monitor for events, or the mode that the call is in following a detected event.

Name	Value	Description
P_CALL_MONITOR_MODE_INTERRUPT	0	The call event is intercepted by the call control service and call processing is interrupted. The application is notified of the event and call processing resumes following an appropriate API call or network event (such as a call release)
P_CALL_MONITOR_MODE_NOTIFY	1	The call event is detected by the call control service but not intercepted. The application is notified of the event and call processing continues
P_CALL_MONITOR_MODE_DO_NOT_MONITOR	2	Do not monitor for the event

## 10.17 TpCallNetworkAccessType

This data defines the bearer capabilities associated with the call. (TS 124 002) This information is network operator specific and may not always be available because there is no standard protocol to retrieve the information.

Name	Value	Description
P_CALL_NETWORK_ACCESS_TYPE_UNKNOWN	0	Network type information unknown at this time
P_CALL_NETWORK_ACCESS_TYPE_POT	1	POTS
P_CALL_NETWORK_ACCESS_TYPE_ISDN	2	ISDN
P_CALL_NETWORK_ACCESS_TYPE_DIALUPINTERNET	3	Dial-up Internet
P_CALL_NETWORK_ACCESS_TYPE_XDSL	4	xDSL
P_CALL_NETWORK_ACCESS_TYPE_WIRELESS	5	Wireless

## 10.18 TpCallPartyCategory

This data type defines the category of a calling party. (Q.763: Calling Party Category/Called Party Category).

Name	Value	Description
P_CALL_PARTY_CATEGORY_UNKNOWN	0	calling party's category unknown at this time
P_CALL_PARTY_CATEGORY_OPERATOR_F	1	operator, language French
P_CALL_PARTY_CATEGORY_OPERATOR_E	2	operator, language English
P_CALL_PARTY_CATEGORY_OPERATOR_G	3	operator, language German
P_CALL_PARTY_CATEGORY_OPERATOR_R	4	operator, language Russian
P_CALL_PARTY_CATEGORY_OPERATOR_S	5	operator, language Spanish
P_CALL_PARTY_CATEGORY_ORDINARY_SUB	6	ordinary calling subscriber
P_CALL_PARTY_CATEGORY_PRIORITY_SUB	7	calling subscriber with priority
P_CALL_PARTY_CATEGORY_DATA_CALL	8	data call (voice band data)
P_CALL_PARTY_CATEGORY_TEST_CALL	9	test call
P_CALL_PARTY_CATEGORY_PAYPHONE	10	payphone

## 10.19 TpCallServiceCode

Defines the Sequence of Data Elements that specify the service code and type of service code received during a call. The service code type defines how the value string should be interpreted.

Sequence Element Name	Sequence Element Type
CallServiceCodeType	TpCallServiceCodeType
ServiceCodeValue	TpString

## 10.20 TpCallServiceCodeSet

Defines a Numbered Set of Data Elements of TpCallServiceCode.

## 10.21 TpCallServiceCodeType

Defines the different types of service codes that can be received during the call.

Name	Value	Description
P_CALL_SERVICE_CODE_UNDEFINED	0	The type of service code is unknown. The corresponding string is operator specific.
P_CALL_SERVICE_CODE_DIGITS	1	The user entered a digit sequence during the call. The corresponding string is an ASCII representation of the received digits.
P_CALL_SERVICE_CODE_FACILITY	2	A facility information element is received. The corresponding string contains the facility information element as defined in ITU Q.932
P_CALL_SERVICE_CODE_U2U	3	A user-to-user message was received. The associated string contains the content of the user-to-user information element.
P_CALL_SERVICE_CODE_HOOKFLASH	4	The user performed a hookflash, optionally followed by some digits. The corresponding string is an ASCII representation of the entered digits.
P_CALL_SERVICE_CODE_RECALL	5	The user pressed the register recall button, optionally followed by some digits. The corresponding string is an ASCII representation of the entered digits.

## 10.22 TpCallSuperviseReport

Defines the responses from the call control service for calls that are supervised. The values may be combined by a logical "OR" function.

Name	Value	Description
P_CALL_SUPERVISE_TIMEOUT	01h	The call supervision timer has expired
P_CALL_SUPERVISE_CALL_ENDED	02h	The call has ended, either due to timer expiry or call party release. In case the called party disconnects but a follow-on call can still be made also this indication is used.
P_CALL_SUPERVISE_TONE_APPLIED	04h	A warning tone has been applied. This is only sent in combination with P_CALL_SUPERVISE_TIMEOUT
P_CALL_SUPERVISE_UI_FINISHED	08h	The user interaction has finished.

## 10.23 TpCallSuperviseTreatment

Defines the treatment of the call by the call control service when the call supervision timer expires. The values may be combined by a logical "OR" function.

Name	Value	Description
P_CALL_SUPERVISE_RELEASE	01h	Release the call when the call supervision timer expires
P_CALL_SUPERVISE_RESPOND	02h	Notify the application when the call supervision timer expires
P_CALL_SUPERVISE_APPLY_TONE	04h	Send a warning tone to the originating party when the call supervision timer expires. If call release is requested, then the call will be released following the tone after an administered time period

## 10.24 TpCallTeleService

This data type defines the tele-service associated with the call. (Q.763: User Teleservice Information, Q.931: High Layer Compatibility Information, and TS 122 003).

Name	Value	Description
P_CALL_TELE_SERVICE_UNKNOWN	0	Teleservice information unknown at this time
P_CALL_TELE_SERVICE_TELEPHONY	1	Telephony
P_CALL_TELE_SERVICE_FAX_2_3	2	Facsimile Group 2/3
P_CALL_TELE_SERVICE_FAX_4_I	3	Facsimile Group 4, Class I
P_CALL_TELE_SERVICE_FAX_4_II_III	4	Facsimile Group 4, Classes II and III
P_CALL_TELE_SERVICE_VIDEOTELEX_SYN	5	Syntax based Videotex
P_CALL_TELE_SERVICE_VIDEOTELEX_INT	6	International Videotex interworking via gateways or interworking units
P_CALL_TELE_SERVICE_TELEX	7	Telex service
P_CALL_TELE_SERVICE_MHS	8	Message Handling Systems
P_CALL_TELE_SERVICE_OSI	9	OSI application
P_CALL_TELE_SERVICE_FTAM	10	FTAM application
P_CALL_TELE_SERVICE_VIDEO	11	Videotelephony
P_CALL_TELE_SERVICE_VIDEO_CONF	12	Videoconferencing
P_CALL_TELE_SERVICE_AUDIOGRAPH_CONF	13	Audiographic conferencing
P_CALL_TELE_SERVICE_MULTIMEDIA	14	Multimedia services
P_CALL_TELE_SERVICE_CS_INI_H221	15	Capability set of initial channel of H.221
P_CALL_TELE_SERVICE_CS_SUB_H221	16	Capability set of subsequent channel of H.221
P_CALL_TELE_SERVICE_CS_INI_CALL	17	Capability set of initial channel associated with an active 3,1 kHz audio or speech call.
P_CALL_TELE_SERVICE_DATATRAFFIC	18	Data traffic.
P_CALL_TELE_SERVICE_EMERGENCY_CALLS	19	Emergency Calls
P_CALL_TELE_SERVICE_SMS_MT_PP	20	Short message MT/PP
P_CALL_TELE_SERVICE_SMS_MO_PP	21	Short message MO/PP
P_CALL_TELE_SERVICE_CELL_BROADCAST	22	Cell Broadcast Service
P_CALL_TELE_SERVICE_ALT_SPEECH_FAX_3	23	Alternate speech and facsimile group 3
P_CALL_TELE_SERVICE_AUTOMATIC_FAX_3	24	Automatic Facsimile group 3
P_CALL_TELE_SERVICE_VOICE_GROUP_CALL	25	Voice Group Call Service
P_CALL_TELE_SERVICE_VOICE_BROADCAST	26	Voice Broadcast Service

## 10.25 TpCallTreatment

Defines the Sequence of Data Elements that specify the treatment for calls that will be handled only by the network (for example, call which are not admitted by the call load control mechanism).

Sequence Element Name	Sequence Element Type
CallTreatmentType	TpCallTreatmentType
ReleaseCause	TpReleaseCause
AdditionalTreatmentInfo	TpCallAdditionalTreatmentInfo

## 10.26 TpCallTreatmentType

Defines the treatment for calls that will be handled only by the network.

Name	Value	Description
P_CALL_TREATMENT_DEFAULT	0	Default treatment
P_CALL_TREATMENT_RELEASE	1	Release the call
P_CALL_TREATMENT_SIAR	2	Send information to the user, and release the call (Send Info & Release)

## 10.27 TpCallAdditionalTreatmentInfo

Defines the Tagged Choice of Data Elements that specify the information to be sent to a call party.

Tag Element Type
TpCallTreatmentType

Tag Element Value	Choice Element Type	Choice Element Name
P_CALL_TREATMENT_DEFAULT	NULL	Undefined
P_CALL_TREATMENT_RELEASE	NULL	Undefined
P_CALL_TREATMENT_SIAR	TpUIInfo	InformationToSend

## 10.28 TpMediaType

Defines the media type of a media stream. The values may be combined by a logical "OR" function.

Name	Value	Description
P_AUDIO	1	Audio stream
P_VIDEO	2	Video stream
P_DATA	4	Data stream (e.g. T.120)

---

## Annex A (normative): OMG IDL Description of Call Control SCF

The OMG IDL representation of this interface specification is contained in text files (`common_cc_data.idl`, `gcc_data.idl`, `gcc_interfaces.idl`, `mpcc_data.idl`, `mpcc_interfaces.idl`, `mmccs.idl`, `ccs.idl` contained in archive `es_20191504v010501m0.zip`) which accompany the present document.

---

## Annex B (informative): Contents of 3GPP OSA R4 Call Control

All items in Generic Call Control, clause 6 and all items in MultiParty Call Control are relevant for TS 129 198-4 V4 (Release 4).

## Annex C (informative): Record of changes

The following is a list of the changes made to this specification for each release. The list contains the names of all changed, deprecated, added or removed items in the specifications and not the actual changes. Any type of change information that is important to the reader is put in the *Others* part of this annex.

Changes are specified as changes to the prior major release, but every minor release will have its own part of the table allowing the reader to know when the actual change was made.

### C.1 Interfaces

#### C.1.1 New

Identifier	Comments
	Interfaces added in ES 201 915-4 version 1.3.1 (Parlay 3.2)
	Interfaces added in ES 201 915-4 version 1.4.1 (Parlay 3.3)
	Interfaces added in ES 201 915-4 version 1.5.1 (Parlay 3.4)

#### C.1.2 Deprecated

Identifier	Comments
	Interfaces deprecated in ES 201 915-4 version 1.3.1 (Parlay 3.2)
	Interfaces deprecated in ES 201 915-4 version 1.4.1 (Parlay 3.3)
	Interfaces deprecated in ES 201 915-4 version 1.5.1 (Parlay 3.4)

#### C.1.3 Removed

Identifier	Comments
	Interfaces removed in ES 201 915-4 version 1.3.1 (Parlay 3.2)
	Interfaces removed in ES 201 915-4 version 1.4.1 (Parlay 3.3)
	Interfaces removed in ES 201 915-4 version 1.5.1 (Parlay 3.4)

## C.2 Methods

### C.2.1 New

Identifier	Comments
<b>Methods added in ES 201 915-4 version 1.3.1 (Parlay 3.2)</b>	
<b>Methods added in ES 201 915-4 version 1.4.1 (Parlay 3.3)</b>	
<b>Methods added in ES 201 915-4 version 1.5.1 (Parlay 3.4)</b>	
IpCall.continueProcessing	

### C.2.2 Deprecated

Identifier	Comments
<b>Methods deprecated in ES 201 915-4 version 1.3.1 (Parlay 3.2)</b>	
<b>Methods deprecated in ES 201 915-4 version 1.4.1 (Parlay 3.3)</b>	
<b>Methods deprecated in ES 201 915-4 version 1.5.1 (Parlay 3.4)</b>	

### C.2.3 Modified

Identifier	Comments
<b>Methods modified in ES 201 915-4 version 1.3.1 (Parlay 3.2)</b>	
<b>Methods modified in ES 201 915-4 version 1.4.1 (Parlay 3.3)</b>	
<b>Methods modified in ES 201 915-4 version 1.5.1 (Parlay 3.4)</b>	

### C.2.4 Removed

Identifier	Comments
<b>Methods removed in ES 201 915-4 version 1.3.1 (Parlay 3.2)</b>	
<b>Methods removed in ES 201 915-4 version 1.4.1 (Parlay 3.3)</b>	
<b>Methods removed in ES 201 915-4 version 1.5.1 (Parlay 3.4)</b>	

## C.3 Data Definitions

### C.3.1 New

Identifier	Comments
<b>Data Definitions added in ES 201 915-4 version 1.3.1 (Parlay 3.2)</b>	
TpCallPartyToChargeAdditionalInfo	TpCallPartyToCharge renamed TpCallPartyToChargeAdditionalInfo
<b>Data Definitions added in ES 201 915-4 version 1.4.1 (Parlay 3.3)</b>	
TpMultiMediaCallLegIdentifierSet	Missing from document, although used - already present in IDL
<b>Data Definitions added in ES 201 915-4 version 1.5.1 (Parlay 3.4)</b>	

### C.3.2 Modified

Identifier	Comments
<b>Data Definitions modified in ES 201 915-4 version 1.3.1 (Parlay 3.2)</b>	
TpCallChargePlan	
<b>Data Definitions modified in ES 201 915-4 version 1.4.1 (Parlay 3.3)</b>	
TpReleaseCauseSet	Corrected in documentation to be a set of TpReleaseCause
TpCallEventCriteriaResult	Corrected in documentation to match IDL
<b>Data Definitions modified in ES 201 915-4 version 1.5.1 (Parlay 3.4)</b>	

### C.3.3 Removed

Identifier	Comments
<b>Data Definitions removed in ES 201 915-4 version 1.3.1 (Parlay 3.2)</b>	
TpCallPartyToCharge	TpCallPartyToCharge renamed TpCallPartyToChargeAdditionalInfo
<b>Data Definitions removed in ES 201 915-4 version 1.4.1 (Parlay 3.3)</b>	
TpCallChargeOrder	No longer used.
<b>Data Definitions removed in ES 201 915-4 version 1.5.1 (Parlay 3.4)</b>	

## C.4 Service Properties

### C.4.1 New

Identifier	Comments
<b>Service Properties added in ES 201 915-4 version 1.3.1 (Parlay 3.2)</b>	
<b>Service Properties added in ES 201 915-4 version 1.4.1 (Parlay 3.3)</b>	
<b>Service Properties added in ES 201 915-4 version 1.5.1 (Parlay 3.4)</b>	
P_NOTIFICATION_ADDRESS_RANGES	Replaces P_TRIGGERING_ADDRESSES

## C.4.2 Deprecated

Identifier	Comments
Service Properties deprecated in ES 201 915-4 version 1.3.1 (Parlay 3.2)	
Service Properties deprecated in ES 201 915-4 version 1.4.1 (Parlay 3.3)	
Service Properties deprecated in ES 201 915-4 version 1.5.1 (Parlay 3.4)	
P_TRIGGERING_ADDRESSES	Replaced by P_NOTIFICATION_ADDRESS_RANGES

## C.4.3 Modified

Identifier	Comments
Service Properties modified in ES 201 915-4 version 1.3.1 (Parlay 3.2)	
P_TRIGGERING_EVENT_TYPES	Multi Party Call Control
P_DYNAMIC_EVENT_TYPES	Multi Party Call Control
Service Properties modified in ES 201 915-4 version 1.4.1 (Parlay 3.3)	
Service Properties modified in ES 201 915-4 version 1.5.1 (Parlay 3.4)	

## C.4.4 Removed

Identifier	Comments
Service Properties removed in ES 201 915-4 version 1.3.1 (Parlay 3.2)	
Service Properties removed in ES 201 915-4 version 1.4.1 (Parlay 3.3)	
Service Properties removed in ES 201 915-4 version 1.5.1 (Parlay 3.4)	

---

## C.5 Exceptions

### C.5.1 New

Identifier	Comments
Exceptions added in ES 201 915-4 version 1.3.1 (Parlay 3.2)	
Exceptions added in ES 201 915-4 version 1.4.1 (Parlay 3.3)	
Exceptions added in ES 201 915-4 version 1.5.1 (Parlay 3.4)	

## C.5.2 Modified

Identifier	Comments
	Exceptions modified in ES 201 915-4 version 1.3.1 (Parlay 3.2)
	Exceptions modified in ES 201 915-4 version 1.4.1 (Parlay 3.3)
	Exceptions modified in ES 201 915-4 version 1.5.1 (Parlay 3.4)

## C.5.3 Removed

Identifier	Comments
	Exceptions removed in ES 201 915-4 version 1.3.1 (Parlay 3.2)
	Exceptions removed in ES 201 915-4 version 1.4.1 (Parlay 3.3)
	Exceptions removed in ES 201 915-4 version 1.5.1 (Parlay 3.4)

---

## C.6 Others

None.

---

## History

<b>Document history</b>		
V1.1.1	February 2002	Publication
V1.2.2	July 2002	Publication
V1.3.1	October 2002	Publication
V1.4.1	July 2003	Publication
V1.5.1	November 2004	Membership Approval Procedure    MV 20050128: 2004-11-30 to 2005-01-28