3GPP TSG-CN Meeting #24 **NP-040255**
02 – 04 June 2004, Seoul, KOREA

Source: **CN5 (OSA)**

Title: **3 Rel-4 CRs 29.198-04 OSA API Part 4: Call control (Correction of continueProcessing method for Generic Call Control Service)**

Agenda item: **7.10 (OSA Enhancements [OSA1])**

Document for: **APPROVAL**

Doc-1st-	Spec	CR	Rev	Phase	Subject	Cat	Version	Doc-2nd-	Workite
NP-040255	29.198-04	067	-	Rel-4	Correction of continueProcessing method for Generic Call Control Service (GCCS)	F	4.8.0	N5-040098	OSA1
NP-040255	29.198-04-2	012	-	Rel-5	Correction of continueProcessing method for Generic Call Control Service (GCCS)	A	5.6.0	N5-040099	OSA1
NP-040255	29.198-04-2	013	-	Rel-6	Correction of continueProcessing method for Generic Call Control Service (GCCS)	A	6.0.1	N5-040101	OSA1

joint-API-group (Parlay, ETSI Project OSA, 3GPP TSG_CN WG5) **N5-040099**
Meeting #26, Atlanta, GA, USA, 16-20 February 2004

CR-Form-v7

CHANGE REQUEST

⌘ **29.198-04-2** **CR** **012** ⌘ **rev** **-** ⌘ Current version: **5.6.0** ⌘

*For **HELP** on using this form, see bottom of this page or look at the pop-up text over the ⌘ symbols.*

Proposed change affects: UICC apps⌘ ☐ ME ☐ Radio Access Network ☐ Core Network **X**

Title:	⌘	Correction of continueProcessing method for Generic Call Control Service (GCCS)
Source:	⌘	CN5 NTT (Atsushi Iwasaki), Fujitsu (Yumi Suzuki), Incomit (Niklas Modin)
Work item code:	⌘	OSA1
Date:	⌘	20/02/2004
Category:	⌘	**A**
Release:	⌘	*REL-5*

Use one of the following categories:
F *(correction)*
A *(corresponds to a correction in an earlier release)*
B *(addition of feature),*
C *(functional modification of feature)*
D *(editorial modification)*
Detailed explanations of the above categories can be found in 3GPP TR 21.900.

Use one of the following releases:
2 (GSM Phase 2)
R96 (Release 1996)
R97 (Release 1997)
R98 (Release 1998)
R99 (Release 1999)
Rel-4 (Release 4)
Rel-5 (Release 5)
Rel-6 (Release 6)

Reason for change:	⌘	Currently it is not clear in the GCCS specification how the application resumes the call processing after receiving the notification or event of interrupt mode. In addition to that, there are some problems in the following cases:- - The application specifies the interrupt mode to the answer event of the routeReq() method to transfer the incoming call, and the applicatoin may just want to continue the call processing after some application's processes at the answer event without calling such as another routeReq() or deassignCall(). However the current specification does not allowed. - The enableCallNotification() can be set both P_EVENT_GCCS_ADDRESS_COLLECTED_EVENT and P_EVENT_GCCS_ADDRESS_ANALYSED_EVENT as intterupt mode. Even if the application request both events as intterupt mode and the gateway can detect both trigger, the application can only receive one or other of two events since the application have to call routeReq() method to continue the processing.
Summary of change:	⌘	To solve the above problem, we therefore propose to introduce continueProcessing() method to GCCS as well as MPCCS, and add some text to the Active State of State Transition Diagrams for IpCall for clarification of the way to resume the call processing from the interrupted status. We believe that there is no difference in the idea about interrupt mode between GCCS and MPCCS. In order to further clearify the usage of continueProcessing, methods that implicitly continues processing, i.e routeReq, releaseCall and deassignCall, should state this.
Consequences if not approved:	⌘	Can not support above cases.

Clauses affected:	⌘	6.3, New 6.3.9, 7.2, 7.2.4				
Other specs	⌘	**Y** X	**N**	Other core specifications	⌘	Rel-6 29.198-04-2

affected:		X	Test specifications
		X	O&M Specifications
Other comments:	⌘	Rel-5 Mirror CR of N5-040098. Rel-6 Mirror CR in N5-040101.	

6.3 Interface Class IpCall

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 in 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.

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.

This operation continues processing of the call implicitly.

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.

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.

This operation continues processing of the call implicitly.

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.

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

This operation continues processing of the call implicitly.

6.3.9 Method 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

7.2 State Transition Diagrams for IpCall

The state transition diagram shows the application view on the Call object for 3GPP.

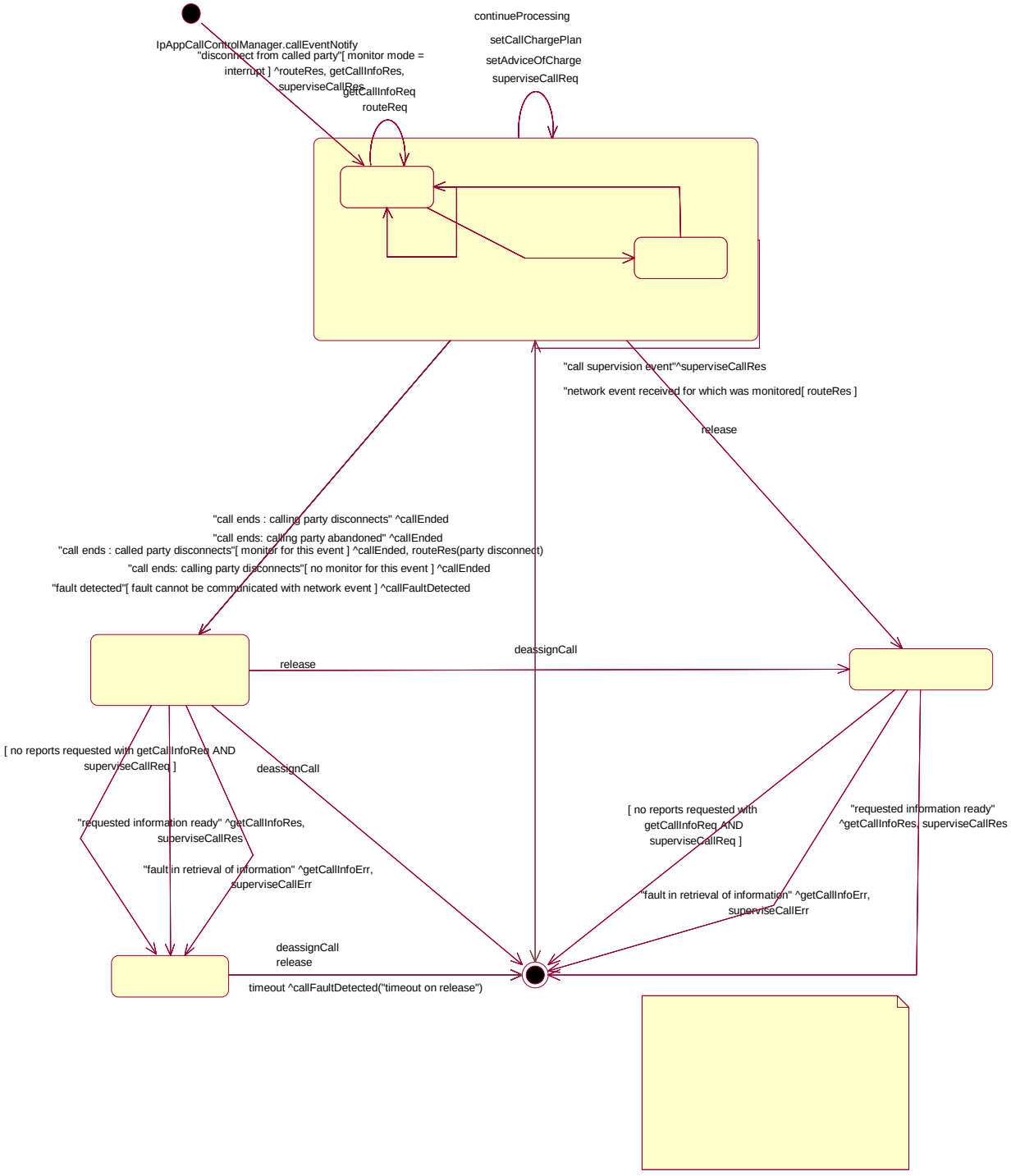


Figure : Application view on the IpCall object for 3GPP

7.2.1 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.

7.2.2 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.

7.2.3 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.

7.2.4 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(), ~~or~~ routeReq(), release() or deassignCall() method.

7.2.5 1 Party in Call State

When the Call is in this state a calling party is present. The application can now request that a connection to a called party be established by calling the method routeReq().

In this state the application can also 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 called party by means of routeReq().

When the calling party abandons the call before the application has invoked the routeReq() operation, the gateway informs the application by invoking callFaultDetected() and also the operation callEnded() will be invoked. 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().

When the called party answers the call, a transition will be made to the 2 Parties in Call state. In case the call can not be established because the application supplied an invalid address or the connection to the called party was unsuccessful while the application was monitoring for the latter in interrupt mode, the Call object will stay in this state

In this state user interaction is possible unless there is an outstanding routing request.

7.2.6 2 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.

CR-Form-v7

CHANGE REQUEST

⌘ **29.198-04-2** **CR** **013** ⌘ **rev** **-** ⌘ Current version: **6.0.1** ⌘

*For **HELP** on using this form, see bottom of this page or look at the pop-up text over the ⌘ symbols.*

Proposed change affects: UICC apps⌘ ☐ ME ☐ Radio Access Network ☐ Core Network **X**

Title:	⌘	Correction of continueProcessing method for Generic Call Control Service (GCCS)
Source:	⌘	CN5 NTT (Atsushi Iwasaki), Fujitsu (Yumi Suzuki), Incomit (Niklas Modin)
Work item code:	⌘	OSA1
Date:	⌘	20/02/2004
Category:	⌘	**A**
Release:	⌘	*REL-6*

Use one of the following categories:
F *(correction)*
A *(corresponds to a correction in an earlier release)*
B *(addition of feature),*
C *(functional modification of feature)*
D *(editorial modification)*
Detailed explanations of the above categories can be found in 3GPP TR 21.900.

Use one of the following releases:
2 (GSM Phase 2)
R96 (Release 1996)
R97 (Release 1997)
R98 (Release 1998)
R99 (Release 1999)
Rel-4 (Release 4)
Rel-5 (Release 5)
Rel-6 (Release 6)

Reason for change:	⌘	Currently it is not clear in the GCCS specification how the application resumes the call processing after receiving the notification or event of interrupt mode. In addition to that, there are some problems in the following cases:- - The application specifies the interrupt mode to the answer event of the routeReq() method to transfer the incoming call, and the applicatoin may just want to continue the call processing after some application's processes at the answer event without calling such as another routeReq() or deassignCall(). However the current specification does not allowed. - The enableCallNotification() can be set both P_EVENT_GCCS_ADDRESS_COLLECTED_EVENT and P_EVENT_GCCS_ADDRESS_ANALYSED_EVENT as intterupt mode. Even if the application request both events as intterupt mode and the gateway can detect both trigger, the application can only receive one or other of two events since the application have to call routeReq() method to continue the processing.
Summary of change:	⌘	To solve the above problem, we therefore propose to introduce continueProcessing() method to GCCS as well as MPCCS, and add some text to the Active State of State Transition Diagrams for IpCall for clarification of the way to resume the call processing from the interrupted status. We believe that there is no difference in the idea about interrupt mode between GCCS and MPCCS. In order to further clearify the usage of continueProcessing, methods that implicitly continues processing, i.e routeReq, releaseCall and deassignCall, should state this.
Consequences if not approved:	⌘	Can not support above cases.

Clauses affected:	⌘	6.3, New 6.3.9, 7.2, 7.2.4
Other specs	⌘	Y: ☐ N: **X** Other core specifications ⌘

affected:		Test specifications O&M Specifications
Other comments:	⌘	Rel-6 Mirror CR of N5-040098.

How to create CRs using this form:

6.3 Interface Class IpCall

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 in 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.

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.

This operation continues processing of the call implicitly.

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.

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.

This operation continues processing of the call implicitly.

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.

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

This operation continues processing of the call implicitly.

6.3.9 Method 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

7.2 State Transition Diagrams for IpCall

The state transition diagram shows the application view on the Call object for 3GPP.

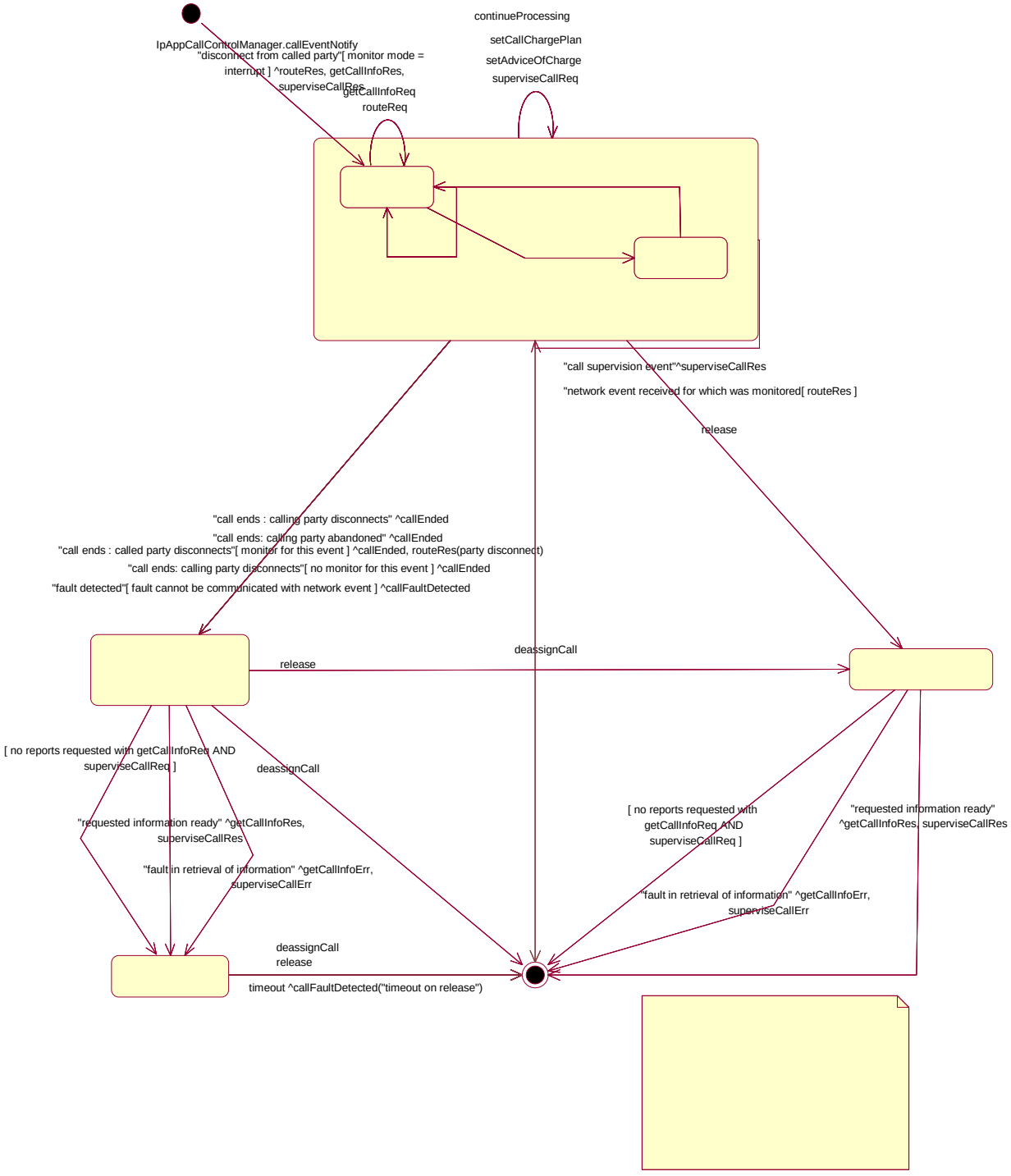


Figure : Application view on the IpCall object for 3GPP

7.2.1 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.

7.2.2 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.

7.2.3 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.

7.2.4 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(), ~~or~~ routeReq(), release() or deassignCall() method.

7.2.5 1 Party in Call State

When the Call is in this state a calling party is present. The application can now request that a connection to a called party be established by calling the method routeReq().

In this state the application can also 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 called party by means of routeReq().

When the calling party abandons the call before the application has invoked the routeReq() operation, the gateway informs the application by invoking callFaultDetected() and also the operation callEnded() will be invoked. 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().

When the called party answers the call, a transition will be made to the 2 Parties in Call state. In case the call can not be established because the application supplied an invalid address or the connection to the called party was unsuccessful while the application was monitoring for the latter in interrupt mode, the Call object will stay in this state

In this state user interaction is possible unless there is an outstanding routing request.

7.2.6 2 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.

joint-API-group (Parlay, ETSI Project OSA, 3GPP TSG_CN WG5) **N5-040098**
Meeting #26, Atlanta, GA, USA, 16-20 February 2004

CR-Form-v7

CHANGE REQUEST

⌘ **29.198-04** **CR** **067** ⌘ **rev** **-** ⌘ Current version: **4.8.0** ⌘

*For **HELP** on using this form, see bottom of this page or look at the pop-up text over the ⌘ symbols.*

Proposed change affects: UICC apps⌘ ☐ ME ☐ Radio Access Network ☐ Core Network **X**

Title:	⌘	Correction of continueProcessing method for Generic Call Control Service (GCCS)
Source:	⌘	CN5 NTT (Atsushi Iwasaki), Fujitsu (Yumi Suzuki), Incomit (Niklas Modin)
Work item code:	⌘	OSA1
Date:	⌘	20/02/2004
Category:	⌘	**F**
Release:	⌘	*REL-4*

Use one of the following categories:
F *(correction)*
A *(corresponds to a correction in an earlier release)*
B *(addition of feature),*
C *(functional modification of feature)*
D *(editorial modification)*
Detailed explanations of the above categories can be found in 3GPP TR 21.900.

Use one of the following releases:
2 (GSM Phase 2)
R96 (Release 1996)
R97 (Release 1997)
R98 (Release 1998)
R99 (Release 1999)
Rel-4 (Release 4)
Rel-5 (Release 5)
Rel-6 (Release 6)

Reason for change:	⌘	Currently it is not clear in the GCCS specification how the application resumes the call processing after receiving the notification or event of interrupt mode. In addition to that, there are some problems in the following cases:- - The application specifies the interrupt mode to the answer event of the routeReq() method to transfer the incoming call, and the applicatoin may just want to continue the call processing after some application's processes at the answer event without calling such as another routeReq() or deassignCall(). However the current specification does not allowed. - The enableCallNotification() can be set both P_EVENT_GCCS_ADDRESS_COLLECTED_EVENT and P_EVENT_GCCS_ADDRESS_ANALYSED_EVENT as intterupt mode. Even if the application request both events as intterupt mode and the gateway can detect both trigger, the application can only receive one or other of two events since the application have to call routeReq() method to continue the processing.
Summary of change:	⌘	To solve the above problem, we therefore propose to introduce continueProcessing() method to GCCS as well as MPCCS, and add some text to the Active State of State Transition Diagrams for IpCall for clarification of the way to resume the call processing from the interrupted status. We believe that there is no difference in the idea about interrupt mode between GCCS and MPCCS. In order to further clearify the usage of continueProcessing, methods that implicitly continues processing, i.e routeReq, releaseCall and deassignCall, should state this.
Consequences if not approved:	⌘	Can not support above cases.

Clauses affected:	⌘	4.1.1, 4.2.2.4				
Other specs	⌘	**Y** ☒	**N** ☐	Other core specifications	⌘	Rel-5/6 29.198-04-2

affected:		Test specifications O&M Specifications
Other comments:	⌘	Rel-5/6 Mirror CRs 29.198-04-2 in N5-040099/101.

4.1.1 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.

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 in 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.

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.

This operation continues processing of the call implicitly.

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.

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.

This operation continues processing of the call implicitly.

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.

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

This operation continues processing of the call implicitly.

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

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.

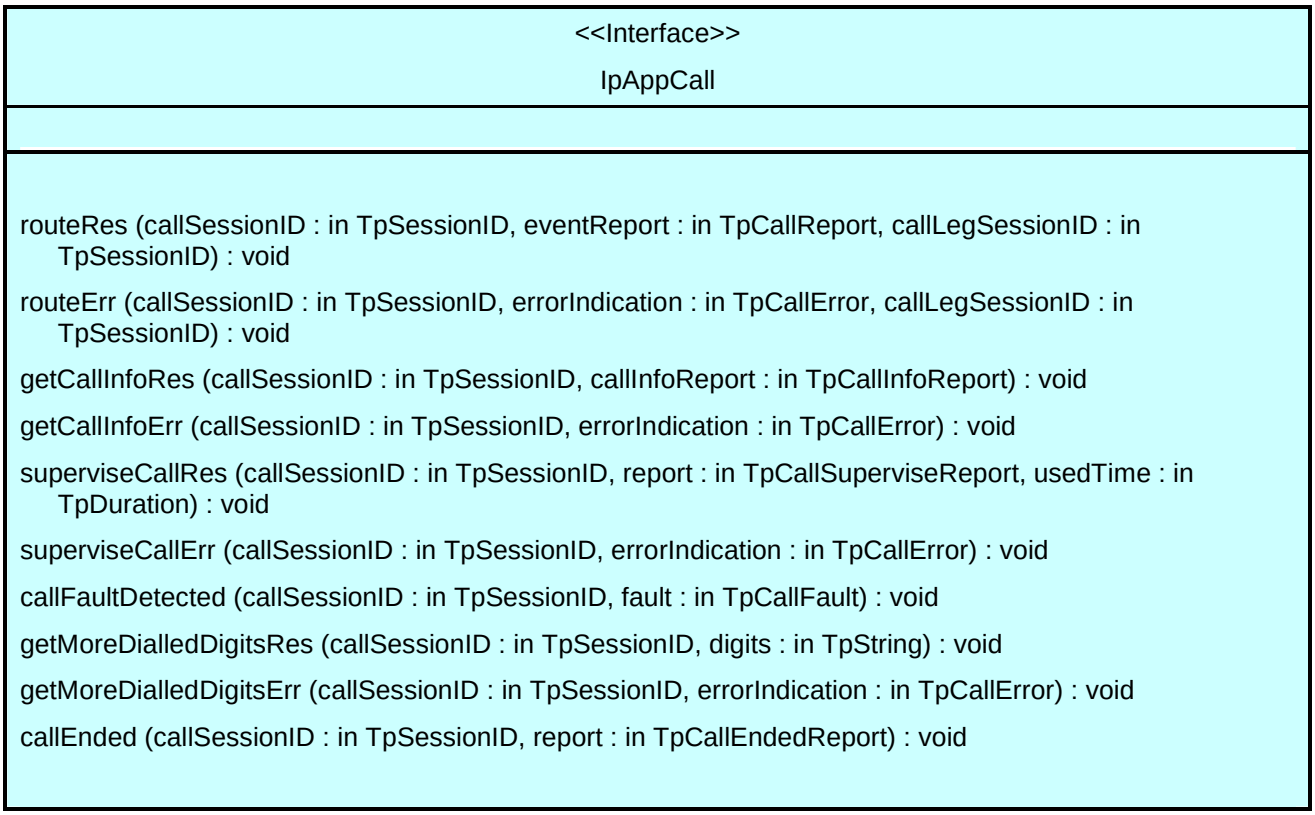
Raises

TpCommonExceptions, P_INVALID_SESSION_ID, P_INVALID_NETWORK_STATE

4.1.2 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.



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.

4.2 Generic Call Control Service State Transition Diagrams

4.2.1 State Transition Diagrams for IpCallControlManager

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

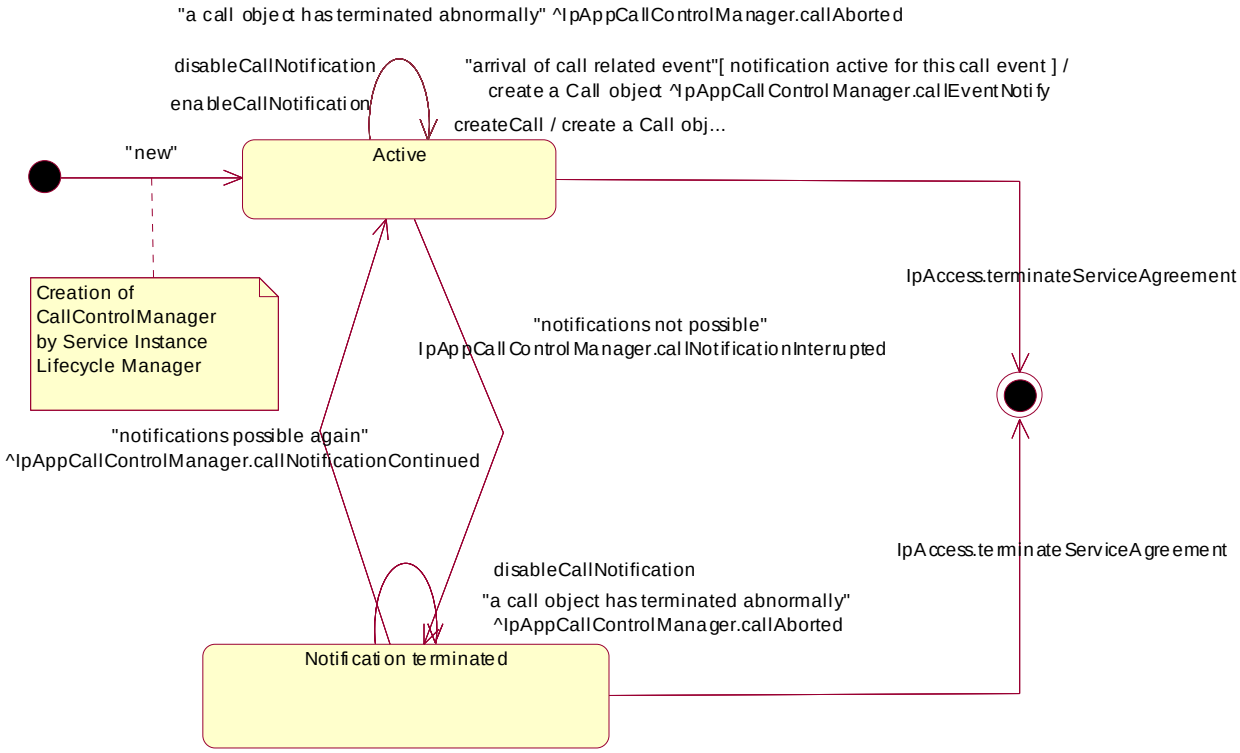


Figure : Application view on the Call Control Manager

4.2.1.1 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().

4.2.1.2 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.

4.2.2 State Transition Diagrams for IpCall

The state transition diagram shows the application view on the Call object for 3GPP.

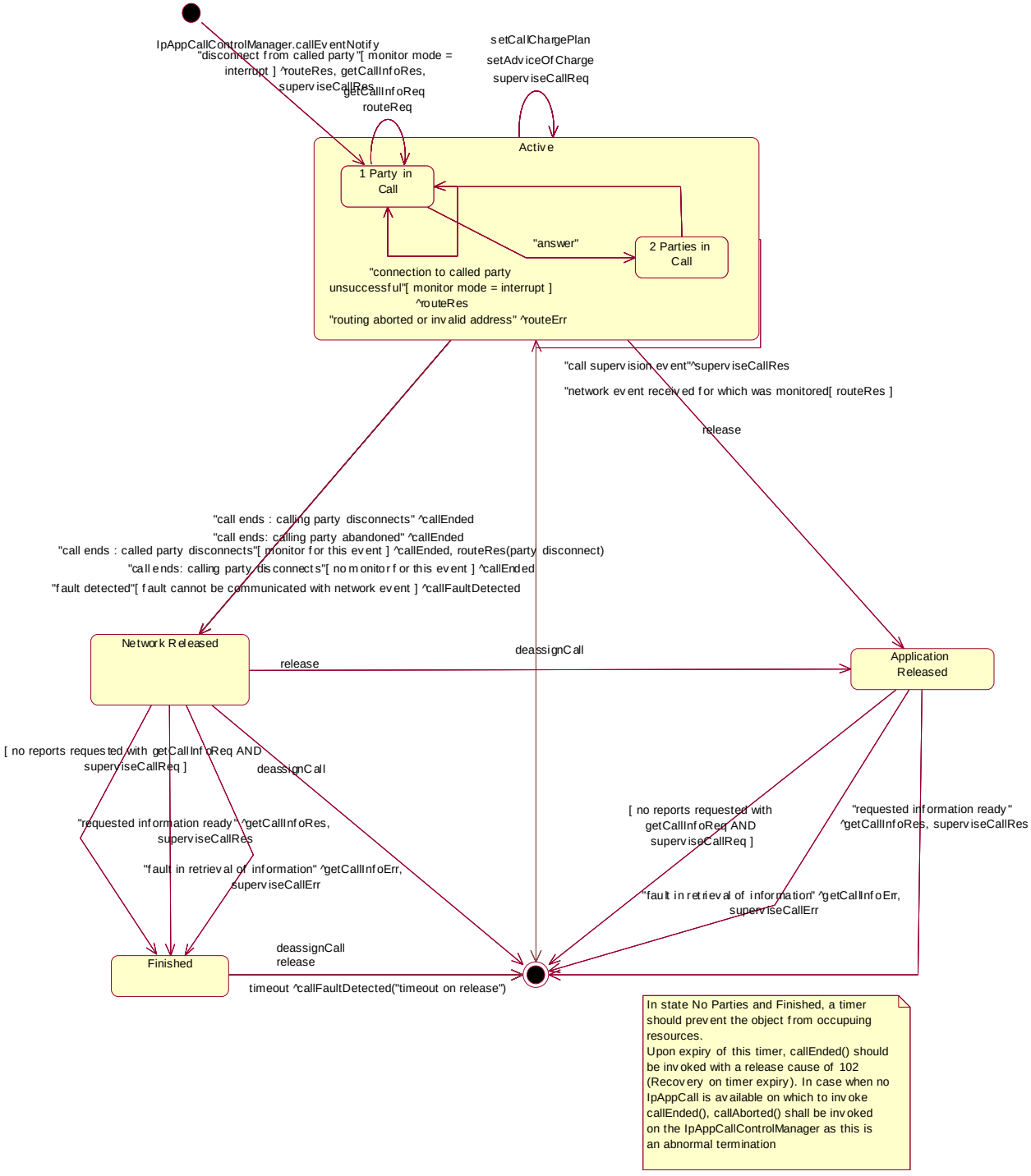


Figure : Application view on the IpCall object for 3GPP

4.2.2.1 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.

4.2.2.2 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.

4.2.2.3 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.

4.2.2.4 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(), ~~or~~ routeReq(), release() or deassignCall() method.

4.2.2.5 1 Party in Call State

When the Call is in this state a calling party is present. The application can now request that a connection to a called party be established by calling the method routeReq().

In this state the application can also 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 called party by means of routeReq().

When the calling party abandons the call before the application has invoked the routeReq() operation, the gateway informs the application by invoking callFaultDetected() and also the operation callEnded() will be invoked. 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().

When the called party answers the call, a transition will be made to the 2 Parties in Call state. In case the call can not be established because the application supplied an invalid address or the connection to the called party was unsuccessful while the application was monitoring for the latter in interrupt mode, the Call object will stay in this state

In this state user interaction is possible unless there is an outstanding routing request.

4.2.2.6 2 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.

Annex B (informative): Change history

Change history							
Date	**TSG #**	**TSG Doc.**	**CR**	**Rev**	**Subject/Comment**	**Old**	**New**
Mar 2001	CN_11	NP-010134	047	-	CR 29.198: for moving TS 29.198 from R99 to Rel 4 (N5-010158)	3.2.0	1.0.0
June 2001	CN_12	NP-010327	--	--	Approved at TSG CN#12 and placed under Change Control	2.0.0	4.0.0
Sep 2001	CN_13	NP-010467	001	--	Changing references to JAIN	4.0.0	4.1.0
Sep 2001	CN_13	NP-010467	002	--	Correction of text descriptions for methods enableCallNotification and createNotification	4.0.0	4.1.0
Sep 2001	CN_13	NP-010467	003	--	Specify the behaviour when a call leg times out	4.0.0	4.1.0
Sep 2001	CN_13	NP-010467	004	--	Removal of Faulty state in MPCCS Call State Transition Diagram and method callFaultDetected in MPCCS in OSA R4	4.0.0	4.1.0
Sep 2001	CN_13	NP-010467	005	--	Missing TpCallAppInfoSet description in OSA R4	4.0.0	4.1.0
Sep 2001	CN_13	NP-010467	006	--	Redirecting a call leg vs. creating a call leg clarification in OSA R4	4.0.0	4.1.0
Sep 2001	CN_13	NP-010467	007	--	Introduction of MPCC Originating and Terminating Call Leg STDs for IpCallLeg	4.0.0	4.1.0
Sep 2001	CN_13	NP-010467	008	--	Corrections to SetChargePlan() Addition of PartyToCharge parmeter	4.0.0	4.1.0
Sep 2001	CN_13	NP-010467	009	--	Corrections to SetChargePlan()	4.0.0	4.1.0
Sep 2001	CN_13	NP-010467	010	--	Remove distinction between final- and intermediate-report	4.0.0	4.1.0
Sep 2001	CN_13	NP-010467	011	--	Inclusion of TpMediaType	4.0.0	4.1.0
Sep 2001	CN_13	NP-010467	012	--	Corrections to GCC STD	4.0.0	4.1.0
Sep 2001	CN_13	NP-010467	013	--	Introduction of sequence diagrams for MPCC services	4.0.0	4.1.0
Sep 2001	CN_13	NP-010467	014	--	The use of the REDIRECT event needs to be illustrated	4.0.0	4.1.0
Sep 2001	CN_13	NP-010467	015	--	Corrections to SetCallChargePlan()	4.0.0	4.1.0
Sep 2001	CN_13	NP-010467	016	--	Add one additional error indication	4.0.0	4.1.0
Sep 2001	CN_13	NP-010467	017	--	Corrections to Call Control – GCCS Exception handling	4.0.0	4.1.0
Sep 2001	CN_13	NP-010467	018	--	Corrections to Call Control – Errors in Exceptions	4.0.0	4.1.0
Dec 2001	CN_14	NP-010597	019	--	Replace Out Parameters with Return Types	4.1.0	4.2.0
Dec 2001	CN_14	NP-010597	020	--	Removal of time based charging property	4.1.0	4.2.0
Dec 2001	CN_14	NP-010597	021	--	Make attachMedia() and detachMedia() asynchronous	4.1.0	4.2.0
Dec 2001	CN_14	NP-010597	022	--	Correction of treatment datatype in superviseReq on call leg	4.1.0	4.2.0
Dec 2001	CN_14	NP-010597	023	--	Corrections to Call Control Data Types	4.1.0	4.2.0
Dec 2001	CN_14	NP-010597	024	--	Correction to Call Control (CC)	4.1.0	4.2.0
Dec 2001	CN_14	NP-010597	025	--	Amend the Generic Call Control introductory part	4.1.0	4.2.0
Dec 2001	CN_14	NP-010597	026	--	Correction in TpCallEventType	4.1.0	4.2.0
Dec 2001	CN_14	NP-010597	027	--	Addition of missing description of RouteErr()	4.1.0	4.2.0
Dec 2001	CN_14	NP-010597	028	--	Misleading description of createAndRouteCallLegErr()	4.1.0	4.2.0
Dec 2001	CN_14	NP-010597	029	--	Correction to values of TpCallNotificationType, TpCallLoadControlMechanismType	4.1.0	4.2.0
Dec 2001	CN_14	NP-010695	030	--	Correction of method getLastRedirectionAddress	4.1.0	4.2.0
Mar 2002	CN_15	NP-020106	031	--	Add P_INVALID_INTERFACE_TYPE exception to IpService.setCallback() and IpService.setCallbackWithSessionID()	4.2.0	4.3.0
Mar 2002	CN_15	NP-020106	032	--	Correction of Event Subscription/Notification Data Type	4.2.0	4.3.0
Mar 2002	CN_15	NP-020106	033	--	Correction of parameter name in IpCallLeg.routeReq() and in IpCallLeg.setAdviceOfCharge()	4.2.0	4.3.0
Mar 2002	CN_15	NP-020106	034	--	Clarification of ambiguous Event handling rules	4.2.0	4.3.0
Jun 2002	CN_16	NP-020180	035	--	Correction to TpCallChargePlan	4.3.0	4.4.0
Jun 2002	CN_16	NP-020180	036	--	Correction to CAMEL Service Property values	4.3.0	4.4.0
Sep 2002	CN_17	NP-020424	057	--	Correction on use of NULL in Call Control API	4.4.0	4.5.0
Mar 2003	CN_19	NP-030020	058	--	Correction of status of methods to interfaces in clause 6.3	4.5.0	4.6.0
Mar 2003	CN_19	NP-030020	059	--	Correction to TpReleaseCauseSet in Multi Party Call Control	4.5.0	4.6.0
Mar 2003	CN_19	NP-030020	060	--	Correction to Sequence Diagrams to remove incorrect Framework references	4.5.0	4.6.0
Mar 2003	CN_19	NP-030020	061	--	Correction to User Interaction Prepaid Sequence Diagrams	4.5.0	4.6.0
Mar 2003	CN_19	NP-030020	062	--	Correction to remove unused TpCallChargeOrder	4.5.0	4.6.0
Mar 2003	CN_19	NP-030020	063	--	Correction to TpCallEventCriteriaResult in Generic Call Control	4.5.0	4.6.0
Mar 2003	CN_19	NP-030020	064	--	Correction of status of methods to interfaces in clause 7.3	4.5.0	4.6.0
Jun 2003	CN_20	NP-030238	065	--	Correction of the description for callEventNotify & reportNotification	4.6.0	4.7.0
Dec 2003	CN_22	NP-030544	066	--	Correction of description in superviseRes and superviseCallRes	4.7.0	4.8.0