BULGARIAN ACADEMY OF SCIENCES

CYBERNETICS AND INFORMATION TECHNOLOGIES • Volume **7**, No 1

Sofia • 2007

Enabling Architecture for Third Party Applications in Intelligent Network

Hristo Froloshki, Evelina Pencheva

Department of Telecommunications, Technical University – Sofia
e-mail: hef@tu-sofia.bg enp@tu-sofia.bg

Abstract: *Intelligent network (IN) is well known architecture for delivering value-added services. In conventional telephony networks with IN functionality no other party than network operator or the vendor of the IN platform can deploy new services. The Parlay/OSA (Open service architecture) concept opens network interfaces for 3 rd parties so that others than network operator can create and deploy services. The research concerns applicability of Parlay/OSA generic call control interface, toward fixed circuit-switched IN network. Mapping between Parlay/OSA and IN call models is performed with the purpose of achieving common view of a call. Two scenarios are proposed for realization of the application initiated call functionality. The scenarios use signaling supported by the IN.*

Keywords: *Intelligent network, Open service architecture, Parlay/OSA applications.*

1. Introduction

Intelligent network (IN) is well known and mature software architecture capable of delivering value-added services in circuit-switched telephone networks [1]. The cornerstone of this concept is decoupling of call control from service control. This means that a computer program can interfere and control the behavior of the network. Such capability provides the network operator an opportunity to offer content rich services in addition to traditional ones.

During the 90ies most operators invested heavily and upgraded their circuit-switched networks to intelligent ones. However, the growth of both IT industry and the Internet led to new business requirements and deregulation of the telecom sector. Unfortunately, the IN concept is restricted to circuit switched networks and is not applicable in packet environment. The centralized approach of service control assumes that the network is governed by a single operator and does not allow 3rd party control. It became clear that new evolution path should emerge.

One of the initiatives that pave the way for evolution of the IN is Parlay/OSA (Open Service Access). Parlay/OSA is service architecture defined for 3rd generation mobile networks [2] but it is also applicable to next generation networks also. The idea is to provide services in a unique way independent from underlying networks. This is achieved by open interfaces that hide network specifics and protocol complexity from service developers. 3rd party service developers can create new attractive application having access to network function through application programming interfaces.

As a new service concept, Parlay/OSA should provide interworking capabilities with the existing architectures. IN services are related to call events. It is really important to map an abstract Parlay/OSA call model onto the IN call model in order to allow Parlay/OSA applications control network resources. There exist some mappings between OSA/Parlay multi-party call control and call models defined in Java call control [3] and Session Initiation Protocol (SIP) [4]. Further in [5] a possible interaction between IN enabled resources and Session Initiation Protocol (SIP) entities is considered. Still the proved IN basic call state model is not integrated in the vision of Parlay/OSA abstract model.

The present research concentrates on capabilities of Parlay/OSA applications to control resources in IN structured networks. Here we investigate the applicability of Parlay/OSA generic call control interface for calls occurring in IN enabled circuit switched network. Scenarios for application initiated IN calls are considered.

First in the paper we explain in a simplistic way the IN and Parlay/OSA service control and suggest possible architecture for interworking between IN and Parlay/OSA. Then a mapping between call models of IN and OSA is established. Different call flows for accessing Parlay/OSA applications from IN networks are considered.

2. Interworking architecture for IN and Parlay/OSA

Fig. 1 shows a simplified IN architecture, where the telephone switches are called Service Switching Points (SSP) and there are also dedicated nodes for service control called Service Control Points (SCPs). SSPs and SCPs are connected via signaling system No 7 network and the dialog between them is based on Intelligent Network Application Protocol (INAP). The call control between switching nodes is based on ISDN user part (ISUP) protocol.

At some points the call can be interrupted and the SSP transfers control to SCP where the service resides. The SCP controls service execution sending instructions to SSP how to proceed with the call. The service control is separated from call control but it is still in the network operator domain.

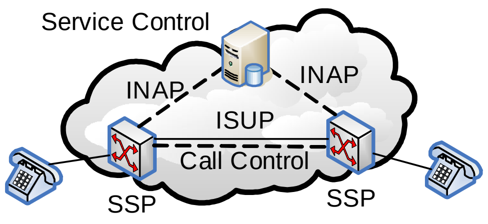


Fig. 1. Operator-centric decoupling of service control in Intelligent Network

Fig. 2 shows the new vision where the service control is exported outside the network boundaries. Service logic resides on so called application servers. External applications access network resources by application programming interfaces (API). The APIs provide programmability of telecommunication resources by defining these resources in terms of objects and methods, data types and parameters that operate on those objects. Using APIs methods, the application server receives event notifications for the calls it is interested in, and sends instructions for particular procedures (dial a number, release call, activate user interaction, etc.). The access to network resources is through a gateway. In terms of Parlay/OSA this gateway is regarded as service capability server (SCS) providing functions for call control.

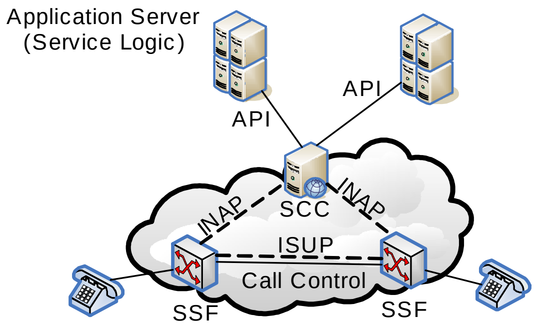


Fig. 2. Exporting service control outside the operator network in Parlay/OSA

To access Parlay/OSA applications from IN structured network the SCS has to integrate functions of SCP and call control API. On the side of the network the SCS has to "talk" INAP, and on the side of application the SCS has to support methods of call control interface. Further, the SCS has to provide both IN and Parlay/OSA call models. Such OSA-enabled SCP, operating in synchrony with the events occurring at IN level and interacting with Parlay/OSA applications, is depicted in Fig. 3.

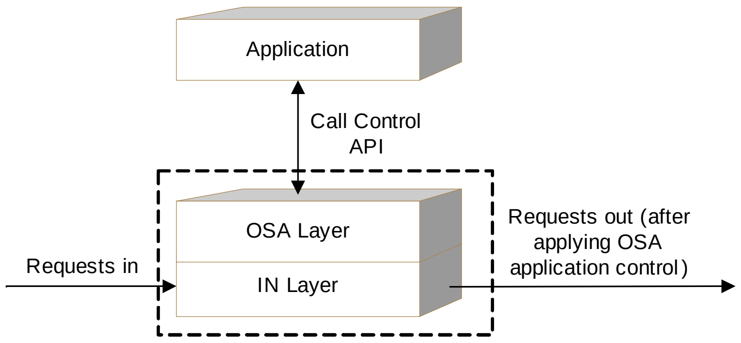


Fig. 3. IN enabled call control SCS

The Parlay/OSA application registers its interest for call events occurring in the IN layer. Event notifications are passed to OSA layer for further application-based processing. The OSA layer proceeds through its states for call object and sends notifications to the application waiting for further instructions for call treatment.

Once the call treatment has been determined, the IN layer is informed and processes the transaction accordingly.

To allow this kind of control, translation between call control APIs methods and INAP procedures is needed. Not all interface methods map directly to INAP procedures. Some methods require functional mapping since the SCS is required to track states and provide polymorphic behavior for some interface methods.

3. Parlay/OSA generic call model against IN basic call state model

In order to maintain active control association between the application server and IN network, the SCS must keep the application view of call states for calls of interest.

Parlay/OSA defines rather high level model [6], representing application's view of the call. This model does not consider the network's view of the call, since the Parlay/OSA interfaces by design are open and network-agnostic. Call control API methods need to be translated into protocol messages in underlying network, with specific view of the call.

Parlay/OSA generic call control model consists of 4 states:

- *Network released* – in this state the call has been ended and the SCS collects the possible call information
- *Finished* – in this state the call has ended and no call related information is to be send to the application
- *Application released* – in this state the application has requested to release the call object and the SCS collects the possible call information
- *Active* – in this state a call between two parties is being setup or present. This state is divided in two substates. The call is in *1 party in call* state when the calling party is present. The call is in *2 parties in call* when a connection between two parties has been established.

To access Parlay/OSA application the SCS must provide both call control manager and call object classes. The call control manager is responsible for starting and managing of an instance of call object. The call control manager is used to request call-related event notifications like address information is analyzed or called party is busy. The call object offers methods to control resources in the network. The call object is a simple call control interface, which allows the setup only of traditional two-party telephone calls.

When the Parlay/OSA application lifecycle manager creates a new call control manager it enters active state. In this state a relation between application and the call control service has been established. As depicted in Figure 4, the application uses *enableCallNotifications()* method to subscribe for events of interest such as arrival of call related events. In case such event occurs, the call control manager creates a call object and informs the application by invoking *callEventNotify()* method on callback interface. The application can also indicate it is no longer interested in certain call related events by invoking *disableCallNotification()* method. In this case the call control manager moves to notification terminated state where event will not be forwarded to the application.

On the side of IN, the SCS must be capable to react to events occurring in the telephony network. To allow control on originating and terminating calls, the SCS has to talk with SSPs at both ends of the network (Fig. 5). To control originating calls an originating basic call state model (O_BCSM) is started. A terminating basic call state model (T_BCSM) is to provide terminating side services.

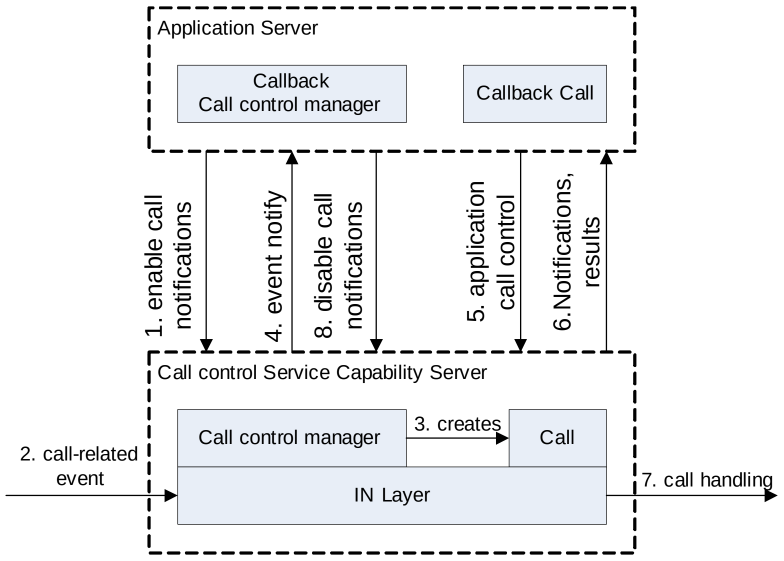


Fig. 4. Interaction between call related objects and their callbacks objects

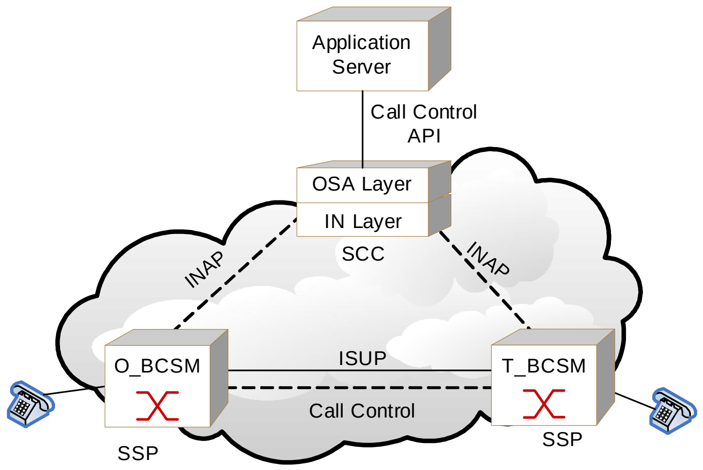


Fig. 5. Application control of IN originating and terminating calls

O_BCSM and T_BCSM models implement the idea of half-call, associated with each party in a two-party conversation [7]. Each of the models represents finite state machine which includes break points that allow the call routing to be paused for originating calls and terminating calls correspondingly. Each one of these triggers is separately settable either statically or dynamically by service logic, to allow call process to be interrupted as various stages in call processing (called points in call (PICs)).

When an originating call attempt is detected in SSP, an O_BCSM is started.

O_BSCM involves the following 11 PICs:

• O_Null: initial state; no call exists, supervision is being provided.
• Auth_Orig_Attempt: initiation of desire to place outgoing call is detected.
• Collect_Info: information is being collected from originating party.

• Analyze_Info: information is analyzed and routing address and call type are determined.

• Select_Route: routing address and call type are being interpreted and the route is selected

• Auth_Call_Setup: authority of originating party to place this particular call is being verified.

• Send_Call: call is being processed by the terminating half BCSM.

• O_Alerting: the terminating party is being alerted

• O_Active: connection is established between originating and terminating parties; call supervision is being provided.

• O_Disconnect: connection is torn down.

• O_Exception: an exceptional condition is detected.

Special treatment might be applied for terminating call also. For terminating calls a T_BCSM is started.

T_BSCM involves the following 9 PICs:

• T_Null: initial state; no call exists, supervision is being provided.

• T_Auth_Term_Attempt: given an indication of an incoming call received from O_BCSM, authority to route this call to the terminating party is being verified.

• Select_facility: a particular available resource is been selected.

• Present_call: terminating resource is informed of incoming call.

• Select_Route: routing address and call type are being interpreted and the

• T_Alerting: the terminating party is being alerted

• T_Active: connection is established between originating and terminating parties; call supervision is being provided.

• T_Disconnect: connection is torn down.

• T_Exception: an exceptional condition is detected.

The next section provides mapping between Parlay/OSA generic call state machine and O_BCSM and T_BCSM of IN.

4. Application control on IN originating calls

By making initial subscription for call notifications, the application can set detection points in O_BCSM. In case of originating call, the application is notified and takes control over the call in order to fulfill application logic. The interworking architecture between IN and OSA for originating calls is shown in Fig. 6.

The call control manager at OSA layer creates a new call object and notifies its callback call control manager at application side about the originating call. The application creates the corresponding callback call object. Control shuttles between O_BSCM and OSA generic call state machine while the call is being serviced.

When a call request arrives, call object is being initialized in *1 Party in call* state. The *1 Party in call* state accommodates the following O_BCSM PICs:

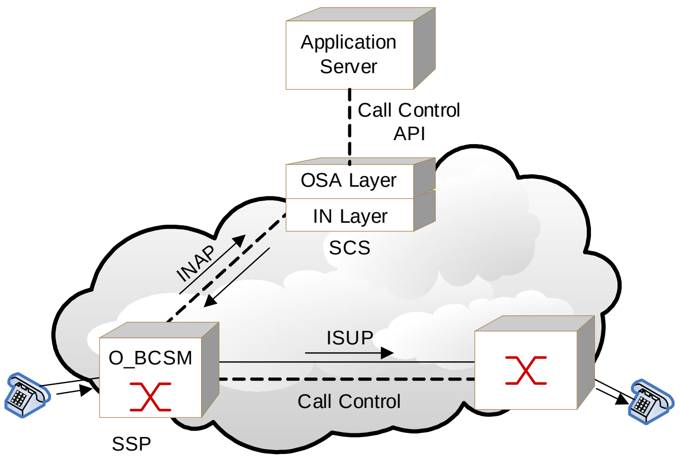


Fig. 6. OSA application control of originating calls

Auth_Orig_Attempt, Collect_Info, Analyse_Info, Select_Route, Auth_Call_Setup, Send_Call and *O_Alerting* as illustrated in figure 7.

Interruption of the call in *Auth_Orig_Att* PIC allows an application to verify the right of the user to access network resources. This step could include a query to external database. A possible reason to deny access could be a report indicating unpaid bill or administratively forbidden subscriber.

If the authentication is positive, the *Collect_Info* PIC is entered. The waiting time for this point in ISDN networks is zero (the dialed destination address is encoded in the *Setup* message). Still the point may be used in a "Hotline" service, where a user picks up the hook and waits for a few seconds until a timer set by the application expires and *Setup* message with predefined (by the user) destination address is sent.

In the *Analyse_Info* PIC the application can affect routing address for example to determine a new destination.

The *Select_Route* PIC provides the application with the ability to set alternative route for the call in case the one chosen by the telephone switch fails.

The application can verify the authority of the subscriber to place the particular call in the *Auth_Call_Setup* PIC. This check is performed to ensure that the subscriber is allowed to dial international numbers, premium services, etc. Typically it also includes database lookup.

The *Send_Call* PIC is used to send the call to the terminating party. At this PIC a message might be received indicating that the called party is busy, which opens the possibility for implementation of the service like "call completion on busy" or "call forwarding on busy".

In the *O_Alerting* state the calling party is notified with audio signal that the called party's equipment rings. Potential services making use are "call completion on no answer" and "call forwarding on no reply".

When the called party answers the *O_Active* PIC is entered and the application is notified. The application is notified by *routeRes()* method and the call object moves to *2 parties in call* state. In this state the application can monitor the call.

Exceptions in the IN layer leading to the *O_Exception* PIC are reported to the application by invoking its *CallFaultDetected()* method.

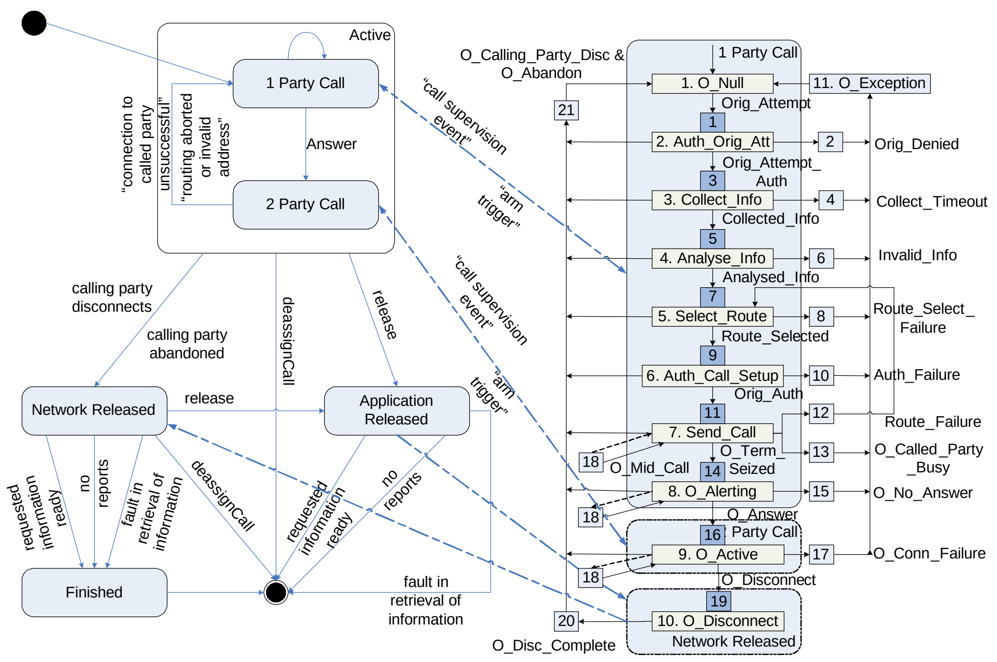


Fig. 7. Mapping between OSA generic call control model and O_BCSM model

In case the application has explicitly requested routing toward certain destination address and that routing failed, the application is reported by *routeErr()* method.

In case the call ends, the application is notified and enters the *Network Released* state. In this state the application has the ability to collect call information for purposes of charging, logging etc. using *getCallInfoReq()* and/or *superviseCallReq* methods.

The *Application Released* state is entered when certain criteria for the call are met (e.g. not enough debit in a pre-paid card) and application logic issues a *release()* method to release the call. In this state the application also can request possible call information.

5. Application control on IN terminating calls

In case of terminating call, the application control is being associated with the terminating party. The application has to register its interest for specific events related with the terminating party in advance. Control shuttles between T_BSCM and OSA generic call state machine while the call is being serviced. The interworking architecture between IN and OSA for terminating calls is shown in Fig. 8.

When an IN terminating call arrives in the SSP, the T_BCSM is started. The call control manager creates a new call object and initiates it in *1 party in call* state.

The *1 party in call* state in the OSA layer includes functionality for *Auth_Term_Att, Select_Facility, Present_Call,* and *T_Alerting* PICs in the IN layer (Fig. 9).

Potentially *Auth_Term_Att* PIC can be used by applications to check whether the address of the caller is not on a black list, defined by the terminating party, or by served network operator.

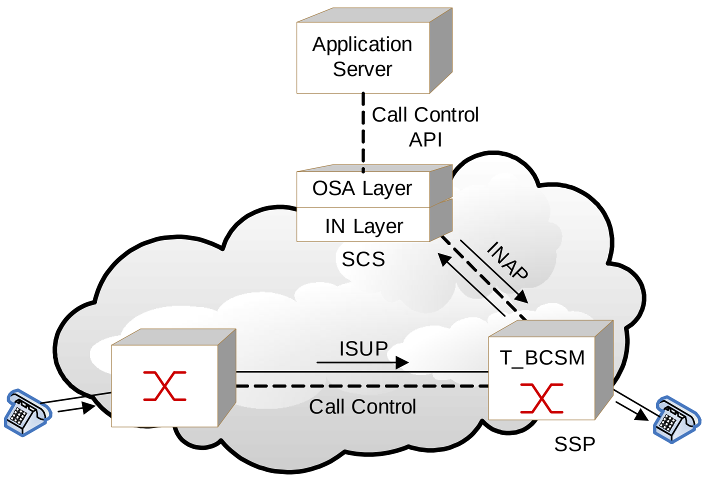


Fig. 8. OSA application control of terminating calls

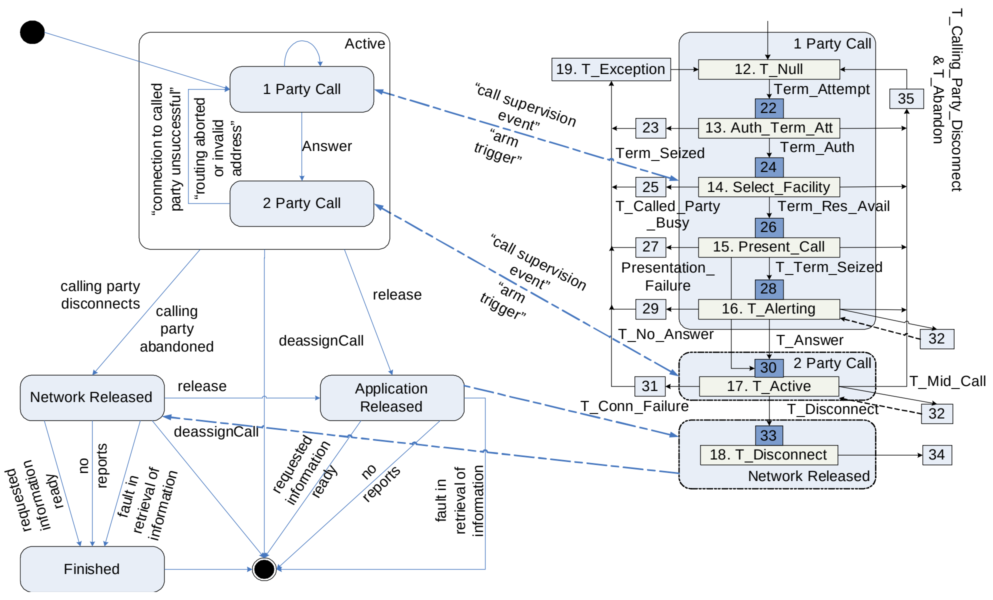


Fig. 9. Mapping between OSA generic call control model and T_BCSM model

After positive authentication T_BCSM enters the *Select_Facility* PIC, where status of the terminating party is examined. This is particularly useful for services such as "call waiting".

Present_Call PIC is entered when the called party is not busy and there are resources to terminate the call. Terminating resources are informed of incoming call.

T_Alerting PIC is the point where indication is sent to O_BCSM, that the terminating side is being notified for the call. Services may use a timer in this PIC and on its expiration to forward the call to a voice mailbox or a mobile number.

The T_BCSM moves to *T_Active* PIC when the terminating party answers. The application is notified and the call object moves to active *2 Parties in call* state.

Transition to *T_Disconnect* PIC is induced when disconnect event in the network occurs. The call object in OSA layer enters the *Network Released* state. The application can collect the possible call information.

6. Application Initiated Call

In Parlay/OSA, 3 rd party call is possible, for example alarm call.

In case of application initiated call, the application requests from call control manager to create a new call object. Using *routeReq()* method the application requests the call to be routed to each of the parties in call. Both parties in call are treated as terminating sides. INAP capable network elements, on the other hand use the *InitiateCall* procedure, issued by the SCP toward SSP in the IN [8]. However the IN assigns O-BCSM and T-BCSM, even though the call is established by the network.

The SSP creates an instance of O_BCSM and suspends its processing, until an explicit *Continue* procedure is called to actually route the call between originating and terminating parties.

Here we consider two approaches. The first one assumes that in addition to INAP, the SCS "talks" ISUP protocol as well. This approach is suitable for IN networks where just simple call control is possible and call party handling is not allowed. The other approach relies only on INAP enabled SCS but assumes more complex call handling including manipulation of individual parties in call.

6.1. Simple call handling with INAP and ISUP signaling

An example dialogue between SCS and SSP is given in figure 10. It represents simplified INAP message exchange needed for implementing the first approach. The application invokes *routeReq()* method on the SCS which starts *InitiateCall* procedure for the first party. Since both parties are terminating sides of the call, in this case we replace the traditional naming convention A and B with B1 and B2.

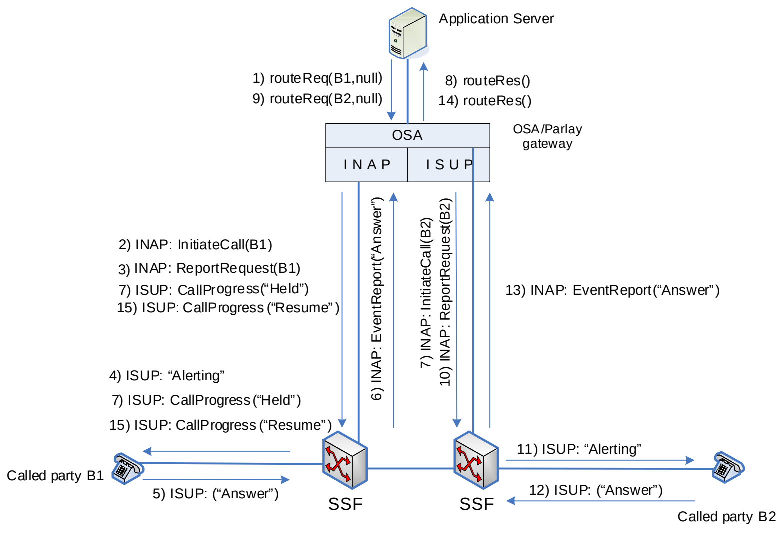
At least one event detection point, needed for further execution of the service (*T_Answer*) is also set. An event is generated when B1 answers – it is reported to the SCS so it must determine if B2 has answered as well. If B1 is the first to answer the call from the network the SCS puts the call on hold with the ISUP *CallProgress(held)* message [9].

When the SCS detects a second event report indicating that B2 has picked up the phone it sends an ISUP *CallProgress(resume)* message is sent, indicating the establishment of connection between B1 and B2.

In this scenario it is essential to notice the necessity of direct interaction between the SCS and the SSP, which bypasses the functionality provided for service logic, provided by INAP.

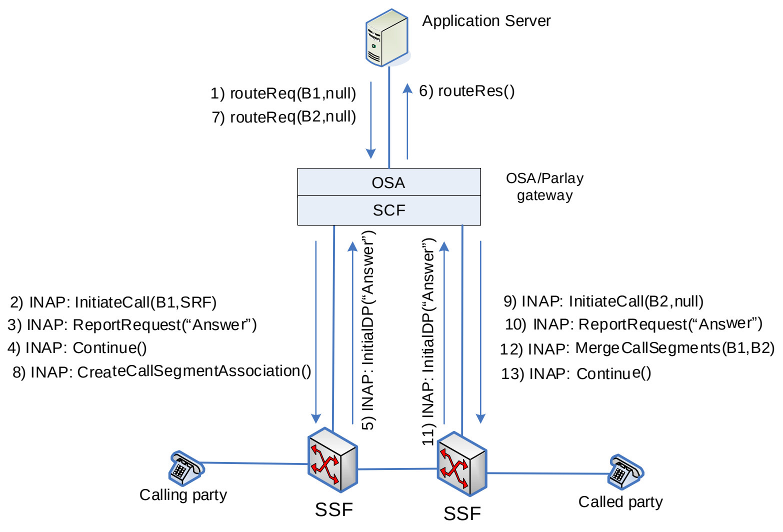
6. 2. Complex call handling with INAP signaling

Fig. 11 represents the dialogue based solely on INAP. The application invokes *routeReq()* method to establish connection with B1. The SCS issues an *InitiateCall* request toward the SSP with address of B1. The SSP only creates an instance of BCSM for B1 and suspends call processing until explicit *Continue* message is sent by the SCS.



Application Server
1) routeReq(B1,null)
9) routeReq(B2,null)
8) routeRes()
14) routeRes()
OSA
INAP
ISUP
OSA/Parlay gateway
2) INAP: InitiateCall(B1)
3) INAP: ReportRequest(B1)
7) ISUP: CallProgress("Held")
15) ISUP: CallProgress ("Resume")
6) INAP: EventReport("Answer")
7) INAP: InitiateCall(B2)
10) INAP: ReportRequest(B2)
13) INAP: EventReport("Answer")
4) ISUP: "Alerting"
7) ISUP: CallProgress ("Held")
15) ISUP: CallProgress ("Resume")
Called party B1
5) ISUP: ("Answer")
SSF
SSF
11) ISUP: "Alerting"
12) ISUP: ("Answer")
Called party B2

Fig. 10. Application initiated call using simple call control



Application Server
1) routeReq(B1,null)
7) routeReq(B2,null)
6) routeRes()
OSA
SCF
OSA/Parlay gateway
2) INAP: InitiateCall(B1,SRF)
3) INAP: ReportRequest("Answer")
4) INAP: Continue()
8) INAP: CreateCallSegmentAssociation()
5) INAP: InitialDP("Answer")
11) INAP: InitialDP("Answer")
9) INAP: InitiateCall(B2,null)
10) INAP: ReportRequest("Answer")
12) INAP: MergeCallSegments(B1,B2)
13) INAP: Continue()
Calling party
SSF
SSF
Called party

Fig. 11. Application initiated call using simple call party handling

Then control is transfered to the SSP. When B1 answers, the application is notified by *routeRes()* method.

The application requests establishing connection toward B2. The SCS receives creates a call segment association and sends *InitiateCall* request with the address of B2. When B2 answers, the SCS merges the two call segments sending *MergeCallSegments()* and invokes a *Continue* procedure for B2. Both parties are now in active session.

This approach is more elegant than the first one, since it does not introduce additional requirements on the protocols, supported by the call control function of the SCS. The problem is with the more complex call control not supported by traditional intelligent networks [10].

7. Conclusions

Research conducted for this article shows that mapping can be done between OSA/ Parlay generic call control and basic call state machines of IN. This mapping is needed in order to allow 3rd party control for calls in networks structured as intelligent ones. The IN concept provides a way of decoupling call control from service control but it is operator centric. With Parlay/OSA network interfaces are open for external service providers.

The call control Parlay/OSA interface is quite suitable for services, requiring event reports and minimal routing intervention on behalf of the application server. In addition to control of originating and terminating IN calls, Parlay/OSA allows application initiated calls. Two scenarios are proposed with the aim of illustrating guidelines on solutions for different call handling function supported by IN nodes.

R e f e r e n c e s

1. ITU-T Rec. Q.1218. Interface Recommendation for intelligent network Capability Set 1.
2. 3GPP TS 29.198-1. Open Service Access, Application Programming Interface. Overview, V. 6.3.2.
3. T a n a k a, S., H.S h i n a, T. Y a m a d a, S. S h i r a i s h i. High Performance Platform for multiple OpenAPIs. – In: 10th International Conference on Telecommunications, 23 February-March 2003. ICT 2003, Vol. **2**, 1259-1263.
4. RFC3261 SIP: Session Initiation Protocol. J. Rosenberg, H. Schulzrinne et al., June 2002.
5. RFC3976 Interworking SIP and Intelligent Network (IN) Applications. January, 2005.
6. 3GPP TS 29.198-4-2 Open Service Access, Application Programming Interface, Generic Call Control, V. 6.4.1.
7. ITU-T Rec. Q.1224, Distributed functional plane for intelligent network Capability Set 2.
8. ITU-T Rec. Q.1228, Interface Recommendation for intelligent network Capability Set 2.
9. ITU-T Rec. Q.733.2 Description For Call Completion Supplementary Services Using SS No 7.
10. Z u i d w e g, J. Next Generation Intelligent Networks, Artech House Inc., 2002.