Model of OSA / Parlay Gateway For Call Control

Hristo E. Froloshki, Evelina N. Pencheva[1]

Abstract – **Present paper focuses on Parlay/OSA call control functionality. Generic Call Control methods and parameters used in communication with applications were thoroughly analyzed with primary research goal set on definition of OSA compliant generic call agent. Some considerations and guidelines concerning the specialization of the model in supporting different networks are presented as well.**

Keywords – **OSA interfaces, Call Agent, Call Control**

I. Introduction

Parlay/OSA (Open Service Access) architecture is telecommunication industry's response to the challenge of offering flexible and attractive services to customers. It joins IT and telecom efforts in defining a comprehensive set of Application Programming Interfaces (APIs), with final goal set to bring wide developer community in the area of service creation. The approach hides operator network's complexity through strictly defined APIs integrated with well known development environments or coming as software development kits (SDKs) using popular programming language like JAVA. APIs are used by developers to access objects abstracting network resources – these objects are usually run on service platforms [1], directly connected to particular network(s). Although intended for UMTS networks, the principles of Parlay/OSA are applicable in the next generation network domain as well (Figure 1). Some very attractive network capabilities become available (for applications) through Parlay/OSA interfaces: location, mobility, call control, etc. OSA enhances the traditional Intelligent Network (IN) approach of defining building blocks [2] by offering developers objects abstracting network capabilities, enabling them to define the next generation of services for both UMTS and fixed networks. Call control capabilities are split in three Service Capability Features (SCFs): Generic Call Control (GCC), Multiparty Call Control (MPCC), and Multimedia Call Control (MMCC).

Present paper focuses on objects defined in GCC, with the aim to model their behavior in the context of a working Parlay/OSA gateway. OSA specifications [3]-[5] define call control through interfaces of objects, implementing the particular functionality. Implementation of OSA gateway functionality requires generic call agent model, capable of communicating both with application and underlying network.

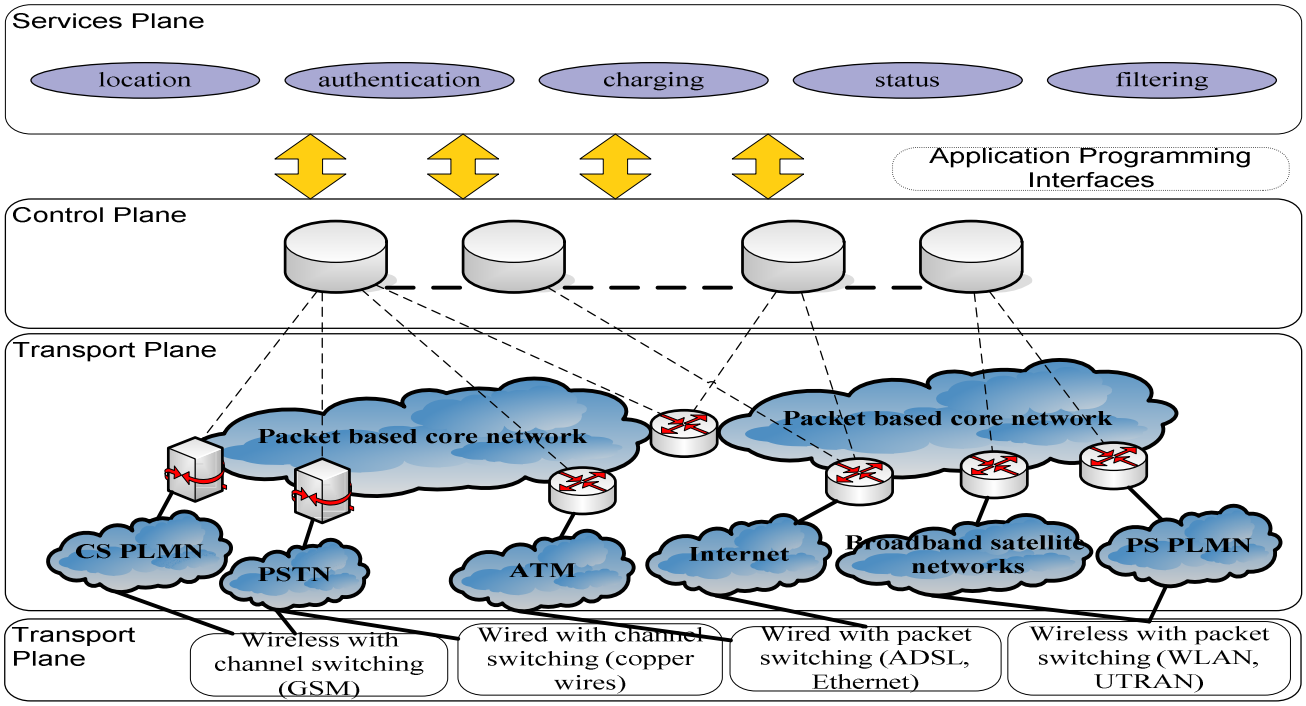

Figure 1. Next Generation Network

[1] [1]Hristo Froloshki and Evelina Pencheva are with Faculty of Communications, Technical University of Sofia, 8, "Kliment Ohridsky" blvd, Sofia 1000, Bulgaria E-mail:, hef@tu-sofia.bg, enp@tu-sofia.bg

The proposed generic object model is suitable for adaptation to different transport technologies, through the use of generalization and specialization approaches. The paper presents object-oriented call agent model, enabled for application interaction. Possible specializations aiming at compatibility with diverse underlying networks are considered.

II. OSA CALL CONTROL INTERFCES

Generic Call Control SCF provides definition of objects, needed both by entities abstracting underlying network (SCS) and applications [6]. The object responsible for handling application notification is IpCallControlManager. It has the ability to set/remove load control on particular address range but its primary task is the creation of IpCall objects. IpCall is the actual interface, allowing an application to control a call in the underlying network (route/release call, gather charging information, etc.).

Each application should pass successfully authentication and service selection steps, and then the framework instructs service lifecycle manager to create an instance of requested service manager (IpCallControlManager in our case). IpCallControlManager in turn creates IpCall to provide the application with control over a call that matched certain criteria. Most objects in Parlay/OSA utilize asynchronous methods to transfer notifications. An essential step in each object's creation is the setting of a reference to its peer object on application part (i.e. the callback interface).

Functionality described so far is relevant for the service layer of Parlay/OSA architecture. However, a functional call agent model should be able to translate the methods invoked on call abstracting objects into protocol (ISUP, INAP, SIP, MAP, etc.) messages, understandable for the nodes, residing on the resource level.

III. GENERIC CALL AGENT MODEL

IpCallControlManager is the primary interface providing management functions to the generic call control service. It is implemented by Generic Call Control SCF and it must support at least the methods createCall(), enableCallNotification(), disableCallNotification(). An example flow of invoked methods during its operation is presented on Figure 2. Service Instance Lifecycle Manager creates an instance of IpCallControlManager, which enters the "Active" state, where it expects requests from application logic. An application has two options to declare its interest in events associated with a particular call: register its callback interface via "enableCallNotification()" method or explicitly setting the address of the callback interface on IpCallControlManager. This callback interface will be used to deliver event or state information (party busy, answer, no answer, etc.) to application logic. It is possible for an application to invoke the method "enableCallNotificaiton" several times with different value for the callback interface – each time setting a new callback interface. If the first one fails the next (in order of creation) will be used to deliver event notifications.

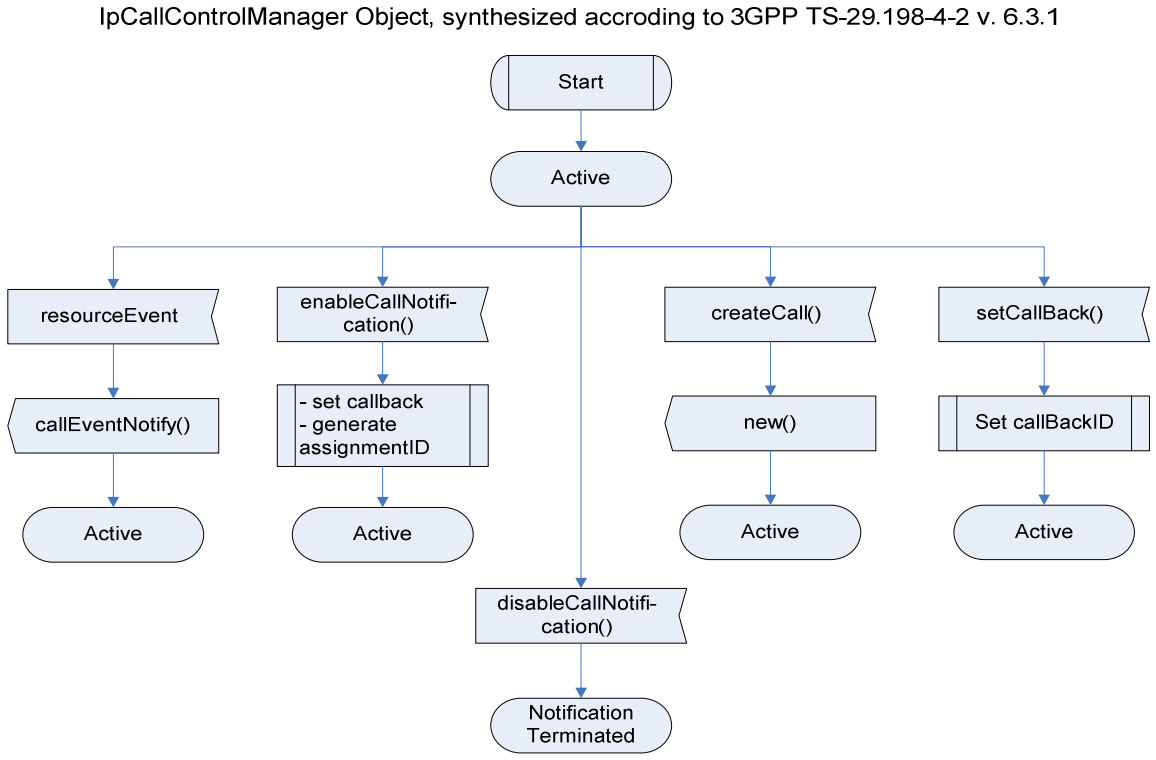

Figure 2. Model of IpCallControlManager

Each "enableCallNotificaiton" request sets an "assignmentID" – to identify the particular event(s) of interest. Having set the callback interface, IpCallControlManager is able to create a call object to represent an actual call taking place in the underlying network. Each IpCall object is assigned unique sessionID value and IpAppCallRef – address of a callback interface to the application. IpCallControlManager then returns to its "Active" state.

In case IpCallControlManager receives "resource Event", for a call monitored by a certain application, it invokes the method "callEventNotify" on the address pointing the callback reference for IpAppCallControlManager. Exchanged information consists mainly of reference to the object, representing the call and description of the event that occurred in the underlying network.

Application which is no longer interested in certain event invokes "DisableCallNotification()" method with parameter "assignmentID", in order to terminate monitoring of event in question. IpCallControlManager responds with a transition from "Active" into "NotificaitonTerminated" state.

Minimum requirements for IpCall interface include implementations of the following methods: routeReq(), release() and deassignCall(). IpCall is created by IpCallControlManager, acting on behalf of an application. In its "Active" state IpCall is able to receive some requests (superviseCallReq() and setAdviceOfCharge()), which although executed do not lead to change in state. SuperviseCallReq() method gives an application the opportunity to set a predefined time interval and supervise the call. Important parameters are callSessionID, time (duration) and treatment – defining how the underlying network should process the particular call after timer expiration. SetAdviceOfCharge() method sends charging information to terminals capable of interpreting it. Important parameters for this method are: callSessionID and aOCInfo.

One of the methods causing state transition is "release()" – if IpCall is in "Active" state, an application may invoke the method and send the object in "Application released" state (Figure 3.). If IpCall is subscribed for information regarding the call (previously invoked getCallInfoReq or superviseCallReq) it needs to wait and forward it to IpAppCall object. In case there is no information to for collection the IpCall object is purged.

In another case the underlying network may trigger an event ("Event From Network"), indicating that a call is terminated by one of the calling parties. This makes IpCall invoke "callEnded()" method on IpAppCall to inform the application. If application is subscribed for additional call info, IpCall enters "Network Released State", and waits for the reports - when they arrive next state is "Finished" and "release()" or "deassignCall()" are legitimate methods for invocation – both lead to the destruction of IpCall and related objects for the particular call. The difference between these two methods is in what happens to the actual call – "deassignCall()" is used when application is no longer interested in controlling it, and frees resources at the OSA/Parlay gateway (call remains in network). Release() causes both call and controlling objects to be released by network and service lifecycle manager respectively. State transition caused by "release()" is possible between "Network released" and "Application released" (not shown on the figure, due to space limitations) – this may happen when call was released in network, but controlling application waits for call-related information to be sent (e.g. for charging purposes). This model reflects the application (service) view of call control functionality accessible through APIs. The main purpose of the model is to hide network protocol complexity from applications.

In order to reflect the specificity of underlying network the model has to be redefined as a specialization of generic functionality. Some guidelines for considering network specificity are given in the next section.

IV. GUIDELINES FOR REDEFINITION OF CALL AGENT BEHAVIOR

The idea of a generic call agent model presented encompasses common properties of call control. In terms of SDL this model has to be defined as virtual type that can be redefined in subtypes. The subtypes defined represent the specific network functionality. For example, the part of the model considering call routing is specific for circuit-switched and packet-switched networks. In the model call routing is presented by time delay. Actually the routeReq() transition is virtual type transition that may be redefined and the routing process has to be presented as a procedure. In case of circuit-switched network the procedure includes exchange of Initial Address message (IAM), Address Complete Message (ACM) and Answer Message (ANM) between SCS implementing ISUP and a switch [7]. The sending of routeRes() is triggered by translation of ANM message (indicating that a conversation between parties has begun). Application is informed by the call agent (through routeRes() method) about completing of routing and current call status.

V. CONCLUSION

We present a generic model of call control agent that can be implemented in NGN. The synthesized model is based on OSA APIs and reflects the call control functionality by the application side. The model describes the call agent behavior by means of methods provided by Call Control interface. The model presents common functionality and hides the protocol complexity of the underlying network. Some guidelines are given to show how the model can be adapted to reflect the network specificity. The full specification of the model including its specializations can be used in implementing OSA gateway.

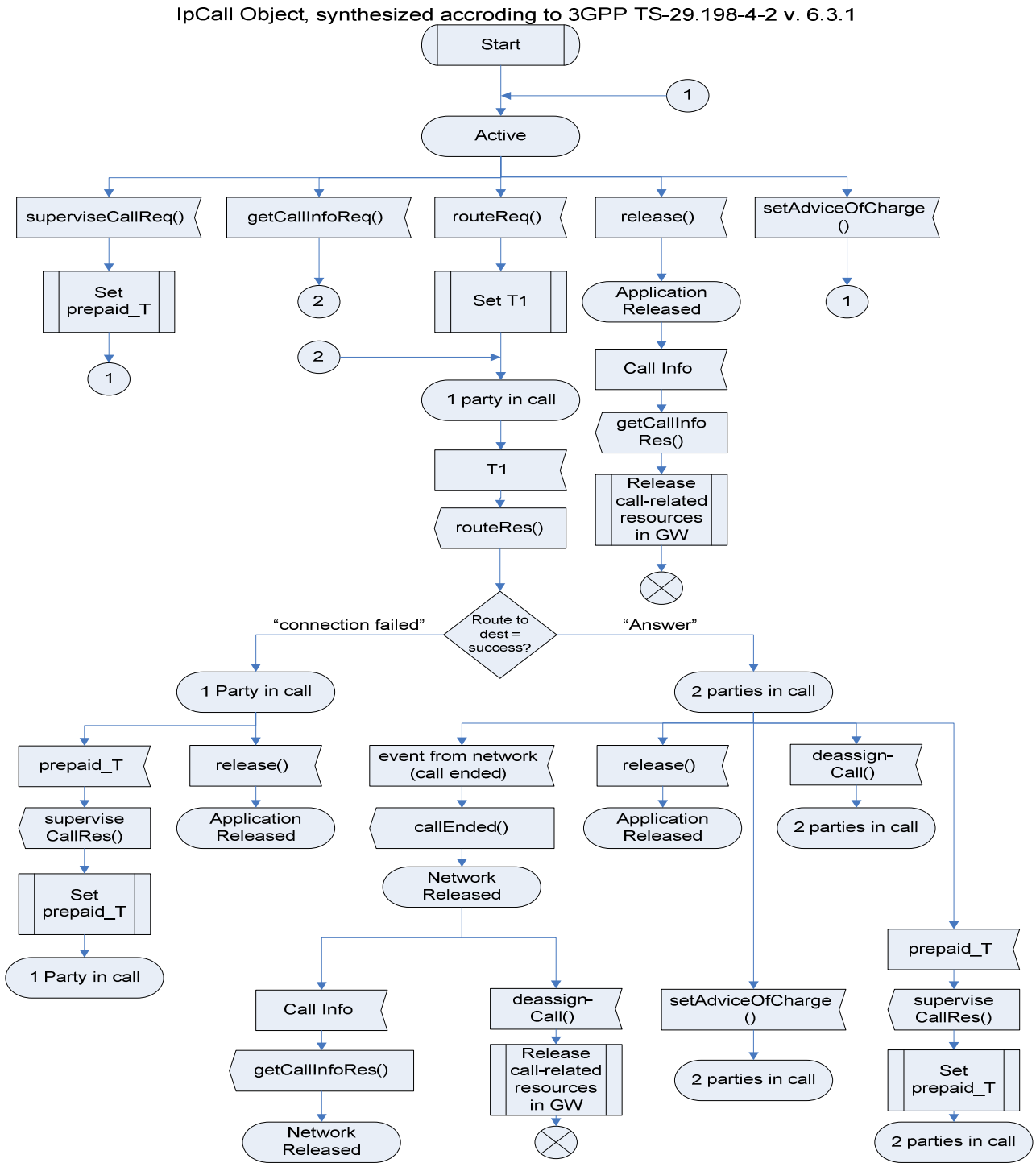

Figure 3. Model of IpCall

REFERENCES

[1] E. S. Chaniotakis, A. E. Papadakis, "Parlay and Mobile Agents in a Homogenized Service Provision", *2nd European Conference on Universal Multi-Service Networks*, Conference Proceedings, pp 150-154, Colmar, France, 2002

[2] ITU-T Rec. Q.1224, Distributed functional plane for intelligent network Capability Set 2

[3] 3GPP TS 29.198-4-2 Open Service Access, Application Programming Interface, Generic Call Control, v6.4.1

[4] 3GPP TS 29.198-4-3 Open Service Access, Application Programming Interface, MultiParty Call Control, v6.4.1

[5] 3GPP TS 29.198-4-4 Open Service Access, Application Programming Interface, MultiMedia Call Control, v6.4.1

[6] J. Zuidweg, *Next Generation Intelligent Networks*, Artech House Inc., 2002

[7] http://www.pt.com/tutorials/ss7/isup.html