BULGARIAN ACADEMY OF SCIENCES

CYBERNETICS AND INFORMATION TECHNOLOGIES • Volume **13**, No 2

Sofia • 2013

Print ISSN: 1311-9702; Online ISSN: 1314-4081
DOI: 10.2478/cait-2013-0015

Study on Deployment of Web Services for User Interaction in Multimedia Networks

Ivaylo Atanasov

Faculty of Telecommunications, Technical University of Sofia
Email: iia@tu-sofia.bg

Abstract: *The paper studies issues concerning the deployment of third party control on user interactions by Web Services. A mapping of Web Service interfaces onto control protocols in multimedia networks is described. Models of user interaction session are proposed. It is proved that both models expose equivalent behaviour. A use-case example is provided.*

Keywords: *Web Services, Internet Protocol Multimedia Subsystem, Interface to protocol mapping, Labelled Transition Systems, Bisimulation.*

1. Introduction

Opening the network allows third party applications to invoke communication capabilities, such as session management and access to information in public networks using application programming interfaces. The APIs provide an abstraction of underlying network technology and control protocols and enable easy development of innovative applications. Opening the networks provides a number of benefits for service developers, operators and end users. For network operators it is a way to increase the traffic in the network and therefore to increase the revenue. The service providers may develop applications using access to functions in the network without telecommunications knowledge. They may combine Information Technology (IT) applications with communications functions for end users who enjoy enhanced communication applications that are customized to their needs.

To achieve network programmability, Open Service Access (OSA) interfaces are defined. OSA APIs provide access to call and session control, messaging, user interaction control, charging, as well as access to information about user location and status. The abstraction provided by OSA APIs is relatively low which means

that the usage of OSA interfaces requires some telecommunications expertise by IT application developers. In order to make the process of application creation accessible for wider developer community and to shorten the way to the market, Parlay X APIs are defined. Parlay X is a set of Web Services that allow interfaces for programmability of communication resources [5, 10, 11]. The Parlay X interfaces are not as flexible as OSA interfaces but they are easier and simpler to use and provide high level of abstraction.

Deployment of open access to network capabilities requires implementation of a gateway which is a special type of application server [8]. The gateway exposes API towards third party applications and supports control protocols towards the network. It needs to translate the invocation of interface methods into control protocols messages and vice versa. In addition, it must maintain two state machines – one modelling the application view on the state of the network resources and another one modelling the control protocol view. Both state machines in the gateway must be synchronized to expose equivalent behaviour.

There exist two alternatives for deployment of Parlay X Web Services. One alternative exploits Parlay X gateway that interacts directly with a network node. The other alternative is to implement an OSA gateway as a mediator between Parlay X gateway and the network node. In the latter case, Parlay X APIs invocations are translated into OSA APIs ones which in turn are translated into control protocols messages.

In this paper, the focus is on open access to User Interaction (UI) control in all Internet protocol based multimedia networks. The Parlay X Audio Call is a web service that allows programmability of UI in the network [4]. The OSA UI APIs provide more flexible tool for controlling of UIs [2]. In IMS, session management including media control for user interaction rely on Session Initiation Protocol (SIP) signalling. The specification of OSA UI API does not define interface to SIP protocol mapping. Further, while the specification defines a model of UI as seen by application, SIP does not define any session management model. Some of the publications [16, 17] concerning OSA gateway implementation focus on aspects related to the application programming interfaces, while other authors [9, 13-15] discuss the evaluation of conformance of the basic session control mechanisms of an IMS out of the application context. This paper describes the mapping of Parlay X Audio Call interfaces onto OSA UI interfaces, and respectively onto SIP messages. A SIP model of user interaction in IMS is proposed. Both models representing the view of UI as seen by applications and by SIP are formally described and it is proved that both models behaviours are synchronized.

The paper is structured as follows. The next section describes functional architecture for deployment of Parlay X Web Services in multimedia networks. Section 3 presents in brief the Parlay X Third Party Call and Parlay X Audio Call Web Services. Section 4 provides a mapping of APIs for UI onto SIP messages. In Section 5, formal descriptions of UI session models as seen by third party application and by the network are provided and it is proved that both models expose equivalent behaviours. An example application that uses Parlay X APIs is presented in Section 6. Finally, the conclusion summarizes the paper contributions.

2. Deployment of parlay X web services in IMS

Internet protocol Multimedia Subsystem (IMS) is service control architecture intended to provide all types of multimedia services based on IP connectivity. In IMS control architecture, Application Servers run applications some of which may reside in a 3-rd party network. Fig. 1 shows one of the alternatives for deployment of Parlay X web services in IMS [6]. The interworking between 3-rd party control and network functions is provided by Parlay X gateway and OSA gateway. The Parlay X gateway translates the Parlay X APIs into OSA APIs. The OSA gateway communicates with Serving CSCF (S-CSCF) which is responsible for user registration and session management which rely on SIP signalling. Media Resource Function Controller (MRFC) and Media Resource Function Processor (MRFP) together provide mechanisms for media services such as conferencing, announcements to users or bearer transcoding in the IMS architecture. The MRFC handles SIP communication to and from the S-CSCF and controls the media resources of MRFP using H.248 protocol. The MRFP provides media resources requested and instructed by the MRFC. The Home Subscriber Server (HSS) is a database which stores the user profiles. The access to user data in the HSS is based on Diameter signalling.

It is also possible for network operator to connect the Parlay X gateway directly to the S-CSCF and HSS. In this case the Parlay X gateway needs to "talk" SIP and Diameter protocols.

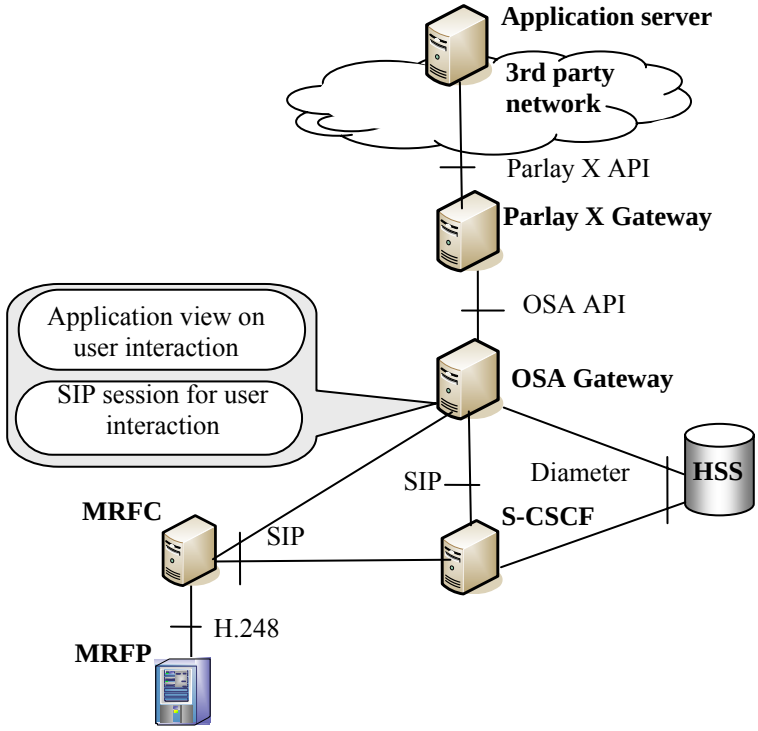

Fig. 1. Open access to media services in IMS

3. Parlay X interfaces for session management and media control

The Parlay X Third Party Call web service may be used to create and manage a call initiated by an application (third party call) [3]. The overall scope of this Web Service is to provide call handling functions to application developers to create a call in a simple way without detailed telecommunication knowledge. The underlying model of the service is based on the following entities:

- Call Session: A call (uniquely identified), to which participants can be added/removed.
- Call Participant: Each of the call parties (uniquely identified) involved in the call session.
- Media: the call can utilize multiple media types to support the participants' communication. In particular both audio and video streams are available, including the specific stream direction (i.e., incoming, outgoing, bidirectional).

An application setting up a call session must initially invoke the *makeCallSession*. The result of such invocation is the creation of a "context" that represents a call session with usually two participants, or at minimum one participant connected, a unique identifier is assigned to the just-created call session. Subsequently the application may wish to add, remove or transfer call participants. In order to do so, the operations *addCallParticipant*, *transferCallParticipant*, *deleteCallParticipant* can be used. Furthermore the call session or call participant status including the media details can be read. In order to do so the operations *getCallParticipantInformation*, and *getCallSessionInformation* can be used. It is also possible to retrieve the media details on its own using the *getMediaForParticipant* or *getMediaForCall* operations of the Audio Call web service. The application can also force the call session and all its participants to be terminated with the operation *endCallSession*.

The Parlay X Audio Call web service may be used for multimedia message delivery and the dynamic management of the media involved for the call participants. The interface is very simple, not requiring the developer to manage the creation of the call. There are several mechanisms which may be utilized for the message content:

- Text, to be rendered using a text-to-speech engine.
- Audio content (such as .wav content), to be rendered by an audio player.
- VoiceXML, to be rendered using a VoiceXML browser.
- Video, to provide video streaming to the user.
- Capture media input from the end user.

The service may provide one or more mechanisms, as determined by service policy. The service allows application control of the call participants' multimedia in a call:

- Allow multiple media types for each participant. In particular both audio and video, as well as chat and data.
- Add and delete media types.
- Control the specific media stream direction (i.e., incoming, outgoing, bidirectional) for each media type.

- Get the current media status of a single call participant or for all the call participants in a call.
- Control the media interactions for a call participant.

4. Interface to protocol mapping

In order to deploy Parlay X web services in the network, it is required to map interface operations onto OSA interface methods in case of OSA gateway usage or directly onto network protocols.

OSA APIs provide more flexibility in programming of call and session management and related user interactions. OSA provides two forms of call control interface. The Generic Call Control APIs support simple two-party voice calls. The three sets of interfaces are defined for control of multiparty, multimedia and conference calls. The Multiparty Call Control APIs provide methods required to set up, modify and clear multiparty calls [1]. The Multimedia Call Control APIs inherit all the methods of the multiparty interfaces and therefore form an enhanced interface that allows the control and manipulation of media within a multiparty call. The Conference Call Control APIs inherit the methods of the other two interfaces while adding specialized capabilities, for handling conference participants. Closely allied to call control APIs is the User Interaction APIs that allow the application to use IVR (Interactive Voice Response) capabilities.

In IMS, SIP is the control protocol for multimedia session establishment, modification and termination. Table 1 shows the suggested mapping of Third Party Call interfaces onto OSA Multiparty Cal, Control interfaces and respectively onto SIP messages.

Table 1 shows the suggested mapping of Audio Call interfaces. SIP signaling in the networks is concerned with playing announcement or media and reflects the SIP session management with MRFC. The announcement may be sent in the body of SIP INVITE request or SIP INFO request. If media services are used to prompt the user and collect information, then the user's answer is transmitted by the body of SIP response 200 OK of the related INVITE or with a dedicated INFO request and the related 200 OK response. SIP signaling in the network related to adding or removing media for call participant reflects the call handling.

5. Formal specification of user interaction models

The formal specification of finite state machines as Labeled Transition Systems allows proving the behavioral equivalence and hence the interworking of OSA User Interaction control and IMS media service control. This may be used for automatic generation of test cases during the OSA gateway verification.

5.1. Labeled Transition Systems and behavioral equivalence

To prove behavioural equivalence between state machines formally, the notion of *Labelled Transition Systems* is used [7].

Definition 1. A *Labelled Transition System* (LTS) is a quadruple (S, Act, $\rightarrow$, s_0), where S is a countable set of states, Act is a countable set of elementary actions, $\rightarrow \subseteq S \times \text{Act} \times S$ is a set of transitions, and $s_0 \in S$ is a set of initial states.

Table 1. Functional mapping of Third Party Call operations

Third Party Call operations	OSA Multi-Party Call Control methods	SIP messages
MakeCallSession	IpMultiPartyCallControlManager.createCall; IpMultiPartyCall.createAndRouteCallLegReq or IpMultiPartyCall.createCallLeg and IpCallLeg.routeReq	INVITE; 183; PRACK; 200[PRACK], UPDATE, 200[UPDATE], 180; 200[INVITE]; ACK
AddCallParticipant	IpMultiPartyCall.createAndRouteCallLegReq or IpMultiPartyCall.createCallLeg and IpCallLeg.routeReq	INVITE; 183; PRACK; 200[PRACK], UPDATE, 200[UPDATE], 180; 200[INVITE]; ACK
TransferCallParticipant	IpCallLeg.routeReq and IpCallLeg.release	INVITE; 200[INVITE]; BYE; 200[BYE]
GetCallParticipant Information	IpCallLeg.getInfoReq; IpCallLeg.getCall; IpCallLeg.getCurrentDestinationAddress; IpCallLeg.getProperties	INFO; 200[INFO]
GetCallSessionInformation	IpMultiPartyCall.getInfoReq; IpMultiPartyCall.getCallLegs	INFO; 200[INFO]
DeleteCallParticipant	IpCallLeg.release	BYE; 200[BYE]
EndCallSession	IpMultiPartyCall.release or IpCallLeg.release	BYE; 200[BYE]

Table 2. Functional mapping of Audio Call operations

Audio Call operations	OSA Multi-Party Call Control и User Interaction methods	SIP messages
PlayTextMessage, PlayAudioMessage, PlayVoiceXmlMessage, PlayVideoMessage	IpMultiPartyCallControlManager.createCall; IpMultiPartyCallControl.createAndRouteCallLeg Req or IpMultiPartyCall.createCallLeg, IpCallLeg.eventReportReq, IpCallleg.routeReq; IpUIManager.creteUICall; IpUICall.sendInfoReq	INVITE; 183; PRACK; 200[PRACK]; UPDATE; 200[UPDATE]; 180; 200[INVITE]; ACK
GetMessageStatus	IpAppUIManager.reportEventReq	INFO; 200[INFO]
EndMessage	IpUICall.abortActionReq	BYE; 200[BYE]
StartPlayAndRecord Interaction	IpMultiPartyCallControlManager.createCall; IpMultiPartyCallControl.createAndRouteCallLeg Req or IpMultiPartyCall.createCallLeg, IpCallLeg.eventReportReq, IpCallleg.routeReq; IpUIManager.creteUICall; IpUICall.sendInfoAndCollectReq	INVITE; 183; PRACK; 200[PRACK]; UPDATE; 200[UPDATE]; 180; 200[INVITE]; ACK; INFO; 200[INFO];
StopMediaInteraction	IpUICall.abortActionReq	BYE; 200[BYE]
AddMediaForParticipants	IpCallLeg.attachMediaReq	re-INVITE;200[INVITE];
DeleteMediaForParticipants	IpCallLeg.detachMediaReq	re-INVITE;200[INVITE];
GetMediaForParticipant	IpCallLeg.getInfoReq	INFO; 200[INFO]
GetMediaForCall	IpMultiPartyCall.getInfoReq	INFO; 200[INFO]

The following denotations are used:

– $s \xrightarrow{a} s'$ stands for the transition (s, a, s');

– $s \xrightarrow{a}$ means that $\exists\, s' : s \xrightarrow{a} s'$;

– $s \overset{\mu}{\Longrightarrow} s_n$, where $\mu = a_1, a_2, ..., a_n : \exists\, s_1, s_2, \ldots, s_n$, such that $s \xrightarrow{a_1} s_1 \ldots \xrightarrow{a_n} s_n$;

– $s \overset{\mu}{\Longrightarrow}$ means that $\exists\, s'$, such as $s \overset{\mu}{\Longrightarrow} s'$;

– $\overset{\hat{\mu}}{\Longrightarrow}$ means $\Rightarrow$ if $\mu \equiv \tau$ or $\overset{\mu}{\Longrightarrow}$ otherwise,

where τ is one or more internal (invisible) actions.

The concept of *bisimulation* [12] is used to prove that two LTS expose equivalent behaviour. The strong bisimulation possesses strong conditions for equivalence which are not always required. For example, there may be internal activities that are not observable. The strong bisimulation ignores the internal transitions.

Definition 2 [12]. Two labelled transition systems $T = (S, A, \rightarrow, s_0)$ and $T' = (S', A, \rightarrow', s_0')$ are *weekly bisimilar* if there is a binary relation $\mathrm{U} \subseteq S \times S'$, such that if $s_1 \mathrm{U}\, t_1 : s_1 \subseteq S$ and $t_1 \subseteq S'$ then $\forall a \in \mathrm{Act}$:

- $s_1 \overset{a}{\Longrightarrow} s_2$ implies $\exists\, t_2 : t_1 \overset{\hat{a}}{\Longrightarrow}' t_2$ and $s_2 \mathrm{U}\, t_2$;

- $t_1 \overset{a}{\Longrightarrow}' t_2$ implies $\exists\, s_2 : s_1 \overset{\hat{a}}{\Longrightarrow} s_2$ and $s_2 \mathrm{U}\, t_2$.

5.2. Formal description of OSA user interaction

The application view on UI object is defined in [2]. The behaviour of the UI object can be described by finite state machine. In Null state, the UI object does not exist. The UI object is created when the *create*UI() method is invoked or a network event is reported by *reportNotification*() method. In Active state, the UI object is available for requesting messages which have to be sent to the network. Both *sendInfoAndCollectReq*() and *sendInfoReq*() methods have a parameter indicating whether it is the final request and the UI object is to be released after the information has been presented to the user. In Active state, in case a fault is detected on the user interaction, an error is reported on all outstanding requests. A transition to Release Pending state is made when the application has indicated that after a certain message no further messages need to be sent to the end-user. There are, however, still a number of messages that are not yet completed. When the last message is sent or when the last user interaction has been obtained, the UI object is destroyed. In Finished state, the user interaction has ended. The application can only release the UI object. A simplified state transition diagram for UI object is shown in Fig. 2.

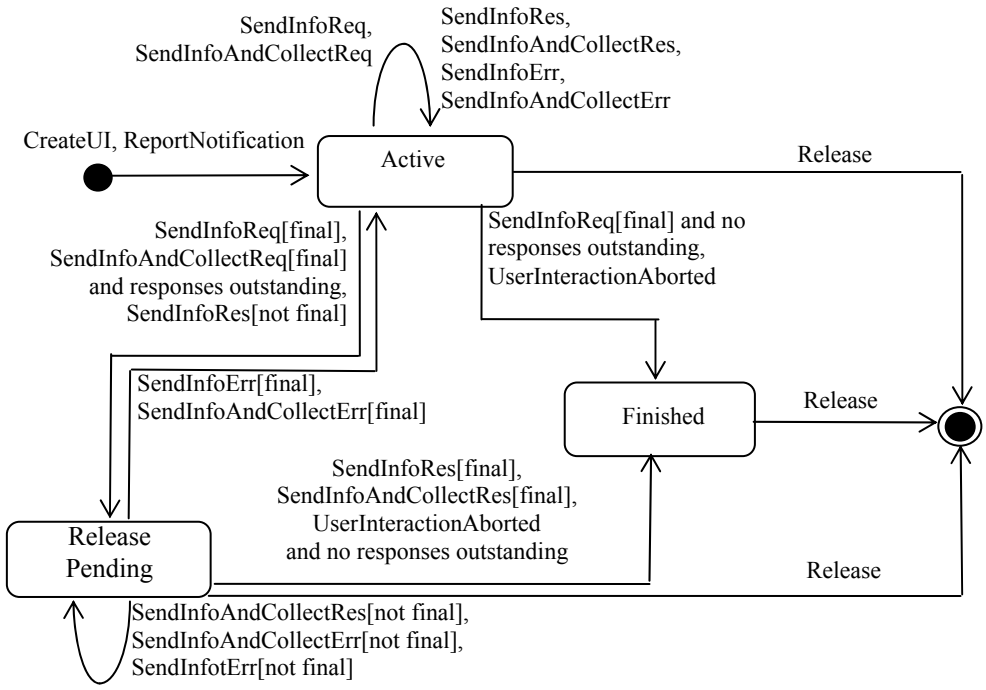

Fig. 2. OSA application view on the UI object

By $T_{\text{AppUI}} = (S_{\text{AppUI}}, \text{Act}_{\text{AppUI}}, \rightarrow_{\text{AppUI}}, s_0')$ we denote a LTS representing the OSA application view on UI object where:

- S_{AppUI} = { Null, Active, ReleasePending, Finished };
- $\text{Act}_{\text{AppUI}}$ = { createUI, reportNotification, sendInfoReq[final], sendInfoReq[not final], sendInfoAndCollectReq[final], sendInfoAndCollectReq[not final], sendInfoRes[final], sendInfoRes[not final], sendInfoErr[final], sendInfoErr[not final], sendInfoAndCollectRes[final], sendInfoAndCollectRes[not final], sendInfoAndCollectErr[final], sendInfoAndCollectErr[not final], userInteractionAborted, release };
- $\rightarrow_{\text{AppUI}}$ = { Null createUI Active,
  Null reportNotification Active,
  Active sendInfoReq[not final] Active,
  Active sendInfoRes[not final] Active,
  Active sendInfoAndCollectReq[not final] Active,
  Active sendInfoAndCollectRes[not final] Active,
  Active sendInfoErr[not final] Active,
  Active sendInfoAndCollectErr[not final] Active,
  Active release Null,
  Active sendInfoReq[final] ReleasePending,
  Active sendInfoRes[not final] ReleasePending ,
  ReleasePending sendInfoErr[final] Active,
  ReleasePending sendInfoErr[not final] ReleasePending,
  ReleasePending sendInfoRes[final] Finished,
  ReleasePending userInteractionAborted Finished,

ReleasePending release Null,
Finished release Null,
Active sendInfoAndCollectReq[final] ReleasePending,
ReleasePending sendInfoAndCollectErr[final] Active,
ReleasePending sendInfoAndCollectRes[not final] ReleasePending,
ReleasePending sendInfoAndCollectErr[not final] ReleasePending,
ReleasePending sendInfoAndCollectRes[final] Finished,
Active sendInfoReq[final] Finished,
Active userInteractionAborted Finished };

- $s_{0'}$ = { Null }.

5.3. Formal description of SIP Session with MRFC

We describe formally the SIP session with MRFC. By $T_{SIP} = (S_{SIP}, Act_{SIP}, \rightarrow_{SIP}, s_0)$ a LTS is denoted which represents a simplified SIP session state machine where

- S_{SIP} = { Idle, $Wait200_{INVITE}$, Established, $Wait200_{INFO}$, $Wait200_{BYE}$ };
- Act_{SIP} = { INVITE, 200_{INVITE}, INFO, 200_{INFO}, BYE, 200_{BYE}};
- $\rightarrow_{SIP}$ = { Idle INVITE $Wait200_{INVITE}$,
  $Wait200_{INVITE}$ 200_{INVITE} Established,
  Established INFO $Wait200_{INFO}$,
  $Wait200_{INFO}$ 200_{INFO} Established,
  Established BYE $Wait200_{BYE}$
  $Wait200_{BYE}$ 200_{BYE} Idle };
- s_0 = { Idle }.

5.4. User interaction models behavioral equivalence

To prove that both user interaction models in OSA and IMS running at the OSA gateway are synchronized, it must be proved that the state machine representing the OSA user interactions and the SIP state machine expose equivalent behaviour. The behavioural equivalence is proved using the concept of weak bisimilarity.

Proposition. The Labelled Transition Systems T_{AppUI} and T_{SIP} are weakly bisimilar.

Proof: To prove the bisimulation relation between two Labelled Transition Systems, it has to be proved that there is a bisimulation relation between their states. By U a relation is denoted between the states of T_{AppUI} and T_{SIP} where U = {(Null, Idle), (Active, Established)}. Table 3 presents the bisimulation relation between the states of T_{AppUI} and T_{SIP} which satisfies Definition 2. The mapping between the OSA User Interaction interface methods and SIP messages defined in Section 3 shows the action's similarity. Based on the bisimulation relation between the states of T_{AppUI} and T_{SIP} it can be stated that both systems expose equivalent behavior.

6. Use case of open access to user interaction

Let us consider an example application that sends greetings on occasion (e.g., on birthdays). The sequence diagram in Fig. 3 shows a "greeting message", in the form of an announcement, being delivered to a user as a result of a trigger from an

application. Typically, the application would be set to trigger at certain time, however, the application could also trigger on other types of events.

Table 3**.** Bisimulation relation between OSA user interaction and SIP media session

Transitions in T_{AppUI}	Transitions in T_{SIP}
Null createUI Active	Idle INVITE Wait200$_{INVITE}$, wait200$_{INVITE}$ 200$_{INVITE}$ Established
Null reportNotification Active	Idle INVITE Wait200$_{INVITE}$, Wait200$_{INVITE}$ 200$_{INVITE}$ Established
Active sendInfoReq[not final] Active	Established INFO Wait200$_{INFO}$, Wait200$_{INFO}$ 200$_{INFO}$ Established
Active sendInfoRes[not final] Active	Established INFO Wait200$_{INFO}$, Wait200$_{INFO}$ 200$_{INFO}$ Established
Active sendInfoAndCollectReq[not final] Active	Established INFO Wait200$_{INFO}$, Wait200$_{INFO}$ 200$_{INFO}$ Established
Active sendInfoAndCollectRes[not final] Active	Established INFO Wait200$_{INFO}$, Wait200$_{INFO}$ 200$_{INFO}$ Established
Active sendInfoErr[not final] Active	Established INFO Wait200$_{INFO}$, Wait200$_{INFO}$ 200$_{INFO}$ Established
Active sendInfoAndCollectErr[not final] Active	Established INFO Wait200$_{INFO}$, Wait200$_{INFO}$ 200$_{INFO}$ Established
Active release Null	Established BYE Wait200$_{BYE}$, Wait200$_{BYE}$ 200$_{BYE}$ Idle
Active sendInfoReq[final] ReleasePending, Active sendInfoRes[not final] ReleasePending, ReleasePending sendInfoErr[final] Active, ReleasePending sendInfoErr[not final] ReleasePending, ReleasePending sendInfoRes[final] Finished, ReleasePending userInteractionAborted Finished, ReleasePending release Null, Finished release Null	Established INFO Wait200$_{INFO}$, Wait200$_{INFO}$ 200$_{INFO}$ Established, Established INFO Wait200$_{INFO}$, Wait200$_{INFO}$ 200$_{INFO}$ Established, Established INFO Wait200$_{INFO}$, Wait200$_{INFO}$ 200$_{INFO}$ Established, Established INFO Wait200$_{INFO}$, Wait200$_{INFO}$ 200$_{INFO}$ Established, Established INFO Wait200$_{INFO}$, Wait200$_{INFO}$ 200$_{INFO}$ Established, Established INFO Wait200$_{INFO}$, Wait200$_{INFO}$ 200$_{INFO}$ Established, Established BYE Wait200$_{BYE}$, Wait200$_{BYE}$ 200$_{BYE}$ Idle, Established BYE Wait200$_{BYE}$, Wait200$_{BYE}$ 200$_{BYE}$ Idle
Active sendInfoAndCollectReq[final] ReleasePending, ReleasePending sendInfoAndCollectErr[final] Active, ReleasePending sendInfoAndCollectRes[not final] ReleasePending, ReleasePending sendInfoAndCollectErr[not final] ReleasePending, ReleasePending sendInfoAndCollectRes[final] Finished, ReleasePending userInteractionAborted Finished, ReleasePending release Null, Finished release Null	Established INFO Wait200$_{INFO}$, Wait200$_{INFO}$ 200$_{INFO}$ Established, Established INFO Wait200$_{INFO}$, Wait200$_{INFO}$ 200$_{INFO}$ Established, Established INFO Wait200$_{INFO}$, Wait200$_{INFO}$ 200$_{INFO}$ Established, Established INFO Wait200$_{INFO,}$ Wait200$_{INFO}$ 200$_{INFO}$ Established, Established INFO Wait200$_{INFO}$, Wait200$_{INFO}$ 200$_{INFO}$ Established, Established INFO Wait200$_{INFO}$, Wait200$_{INFO}$ 200$_{INFO}$ Established, Established BYE Wait200$_{BYE}$, Wait200$_{BYE}$ 200$_{BYE}$ Idle, Established BYE Wait200$_{BYE}$, Wait200$_{BYE}$ 200$_{BYE}$ Idle
Active sendInfoReq[final] Finished, Finished release Null,	Established INFO Wait200$_{INFO}$, Wait200$_{INFO}$ 200$_{INFO}$ Established, Established BYE Wait200$_{BYE}$, Wait200$_{BYE}$ 200$_{BYE}$ Idle
Active userInteractionAborted Finished, Finished release Null	Established INFO Wait200$_{INFO}$, Wait200$_{INFO}$ 200$_{INFO}$ Established, Established BYE Wait200$_{BYE}$, Wait200$_{BYE}$ 200$_{BYE}$ Idle

The application requests an audio message to be played to the user. The Parlay X gateway initiates a call to the user using the OSA Call Control interfaces. The call is established in the network and the Parlay X gateway is notified. Then using the OSA UI interfaces the Parlay X requests from OSA gateway to create a copy of UI object. A signaling connection to the MRFC is established in the network. The Parlay X gateway requests to play the audio message. The message is played and the result is reported to the Parlay X gateway and to the application. The application in turn requests the release of the call which is followed by resource release in the network.

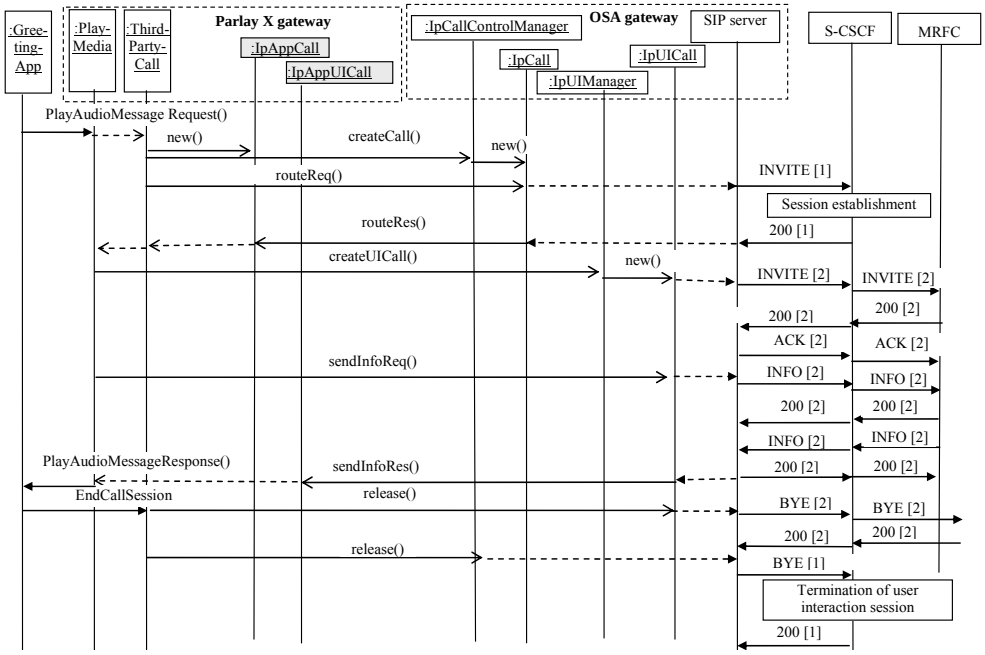

Fig. 3. Playing an audio message using Audio Call and Third Party Call interfaces

7. Conclusion

The usage of Web Services for session management makes the development process of application with enhanced communication functions easer and simpler. The high level of abstraction provided by Parlay X interfaces enables application creation by IT developers without comprehensive knowledge in telecommunications protocols.

Deployment of Parlay X web services requires by the network operator to implement a Parlay X gateway and OSA gateway which make the translation of interface methods into protocol messages and vice versa. The suggested mapping of Third Party Call and Audio Call operations SIP messages enables the usage of web services in all IP based multimedia networks.

The main issue in deployment of Web Services is related to implementation of OSA gateway. The OSA gateway needs to maintain two mutually synchronized sessions representing the application and network views. The suggested model of SIP user interaction session describes the network view. The formal description of both models representing the application and network views provides a method for formal verification of the functional behavior of OSA gateway. The method is useful in testing the conformance of implementation of OSA gateway with respect to the specification, in the context of reactive systems.

References

1. 3GPP TS 29.198-04-03, Open Service Access (OSA); Application Programming Interface (API). Part 4: Call Control Sub-Part 3: Multi-party Call Control Service Capability Feature; V**.** 9.0.0, 2009.
2. 3GPP TS 29.198-05, Open Service Access (OSA); Application Programming Interface (API). Part 5: User Interaction Service Capability Feature (SCF), V.9.0.0, 2009.
3. 3GPP TS 29.199-02, Open Service Access (OSA); Parlay X Web Services. Part 2: Third Party Call, V. 9.0.0, 2009.
4. 3GPP TS 29.199-11, Open Service Access (OSA); Parlay X Web Services. Part 11: Audio Call, V. 9.0.0, 2009.
5. Cabrera, L., F. Kurt, D. Box. An Introduction to the Web Services Architecture and Its Specifications, 2004. **http://msdn2.microsoft .com/en-us/library/ms996441.aspx**
6. Chen, R., E. Su, V. Shen, Y. Wang. Introduction to IP Multimedia Subsystem (IMS). Part 1: SOA Parlay X Web services. Computer and Information Science, 2006. **http://www.mendeley.com/research/introduction-ip-multimedia-subsystem-ims-part-1-soa-parlay-x-web-services/**
7. Chena, X., R. Nicola. Algebraic Characterizations of Trace and Decorated Trace Equivalences over Tree-Like Structures. – In: Theoretical Computer Science, 2001, 337-361.
8. Darvishan, A., H. Yeganeh, K. Bamasian, H. Ahmadian. OSA Parlay X Gateway Architecture for 3rd Party Operators Participation in Next Generation Networks. – In: Proc. of 12th International Conference on Advanced Communication Technology ICACT'2010, 2010, 75-80.
9. Islam, S., J.-C. Gregoire. Convergence of IMS and Web Services: A Review and a Novel Thin Client Based Architecture. – In: Proc. of 8th IEEE International Conference on Communication Networks and Services Research CNSR'10, 2010, 221-228.
10. Lofthouse, H., M. Yates, R. Strech. Parlay X Web Services. – BT Technology Journal, Vol. **22**, 2004, Issue 1, 81-86.
11. Moore, S., L. Liu. Web Services in Telecommunications. – IEEE Communications Magazine, Vol. **45**, 2007, Issue 7, 26-27.
12. Panangaden, P. Notes on Labelled Transition Systems and Bisimulation. 2004. **http://www.cs.mcgill.ca/~ prakash/Courses/comp330/Notes/lts09.pdf**
13. Tholo, S., H. Hanrahan. Expansion of Parlay X API for Enhanced Web Services. – In: Proc. of Southern Africa Telecommunication Networks and Applications Conference, SATNAC'2004, 2004, Western Cape, South Africa.
14. Vannucci, D., H. Hanrahan. OSA/Parlay-X Extended Call Control Telecom Web Services. – In: Proc. of Southern Africa Telecommunication Networks and Applications Conference, SATNAC'2007, 2007, Mauritius.
15. Yamato, Y., H. Ohnishi, H. Sunaga. Development of Service Control Server for Web-Telecom Coordination Service. – In: Proc. of IEEE International Conference on Web Services ICWS'08, 2008, 600-607.
16. Yang, J., H. Park. A Design of Open Service Access Gateway for Converged Web Service. – In: Proc. of 10th International Conference on Advanced Communication Technology, 2008, 1807-1810.
17. Yim, J., Y. Choi, B. Lee. Third Party Call Control in IMS using Parlay Web Service Gateway. – In: Proc. of 8th International Conference on Advanced Communication Technology ICACT'2006, 2006, 221-224.