Introducing Network-Awareness for Networked Multimedia and Multi-modal Applications

Miran Mosmondor[*], Ognjen Dobrijevic[+], Ivan Piskovic[+], Mirko Suznjevic[+], Maja Matijasevic[+], Sasa Desic[*]

[*]*Ericsson Nikola Tesla, Research and Development Center, Zagreb, Croatia*
{miran.mosmondor, sasa.desic}@ericsson.com

[+]*University of Zagreb, Faculty of Electrical Engineering and Computing, Zagreb, Croatia*
{ognjen.dobrijevic, ivan.piskovic, mirko.suznjevic, maja.matijasevic}@fer.hr

***Abstract* - Due to increased user/service requirements in terms of network quality of service (QoS) parameters, and heterogeneity of end-user access network options and terminal capabilities, introducing “network-awareness” into rich multimedia and multimodal networked applications could provide a critical advantage. An idea behind network-awareness is to let the applications indicate their requirements and to adapt to changing conditions in the network, as well as to let the network “know” of the applications’ resource demands. This approach is based on signaling, as a means to request special treatment for traffic in the network and to receive indications from the network of different conditions. Another important issue for the proposed solution is the simplicity of use. Providing developers with a reusable solution that, to much extent, removes the need for understanding a specific signaling protocol eases and quickens development of the network-aware applications. The project objective was to identify generic signaling functionality, and to create an application programming interface (API) which will enable application developers to create advanced multimodal networked services. The developed API was applied in a case study using a prototype application.**

***Index Terms* - Application Programming Interface, Dynamic service adaptation, End-to-end Quality of Service signaling, Multimedia and multimodal networked applications, IP Multimedia Subsystem**

I. Introduction

With ever more widespread multimedia and multimodal end-user equipment, ranging from devices specifically designed for a particular purpose to generic laptops and mobile phones, a wide range of new services may be envisioned to provide better quality of life, especially for the elderly and the disabled. Examples of such services include universally (“anywhere-anytime”) accessible and context-adaptive information services, medical monitoring and counseling services, and edutainment services based on collaborative virtual environments (CVE) [1]. A CVE may include various means of communication between its participants, including, but not limited to face and body gestures and behavior (performed via users’ representation in the virtual world, or, avatar), text chat, and, possibly, live voice communication. Further on, adaptation of the content presented to the user may be required in more than one way, taking into account the user's preferences, experience, and (dis)ability, as well as user's terminal capabilities/features, and network conditions. The interdependence of these requirements may be addressed through the "application aspect" and the "communication aspect" [12].

From the network point of view, such applications involving rich multimedia content and real-time interaction impose more strict requirements onto managing, delivering, and monitoring network performance. For example, a too long delay in service response or inability to adapt the content to terminal characteristics may render the service useless. In this work, we are particularly interested in services with multi-modal information being exchanged not only at the advanced human-computer interface, but also being transferred through the network. Such services need to go beyond the traditional approach (Fig. 1), where the quality guaranteed by the network is either predefined (e.g. voice quality in fixed telephony), or taken as random (e.g. delay in Web browsing), to a more “network-aware” approach. This means that a certain “control” component is needed at both the client and the service “ends”, which is capable of exchanging control information, or signaling, as illustrated in Fig. 2 below.

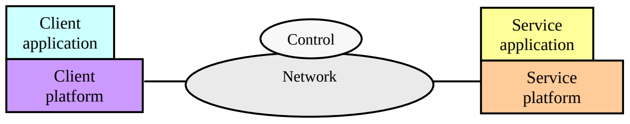


Figure 1. Traditional approach - network assumes predefined behavior

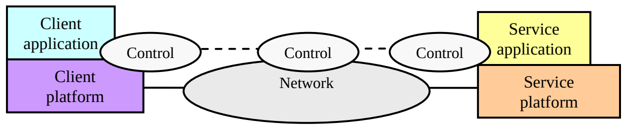


Figure 2. “Network-aware” approach - context and network adaptation

A. Dynamic service adaptation model

A dynamic service adaptation model (DSAM) has been proposed in our earlier work (more details in [10]), which takes into account the heterogeneity of access options and advanced multimedia services in next generation networks, and attempts to further describe and specify the (sets of) parameters referring to:

- end-user access network options and terminal capabilities (client platform)
- user preferences (client application, human-computer interface, personal preferences and/or (dis)ability)
- available resources and costs (network)
- service requirements (server application, server platform)

The proposed model focuses on the provisioning of end-to-end support for signaling QoS requirements at the session layer with the emphasis on virtual reality (VR) services. It includes the entire process of negotiation and renegotiation of QoS parameters, and service adaptation, from when an end-user accesses a VR service until (s)he terminates it. The model (shown in Fig. 3) is centered on a process in a client server architecture in which a client accesses an application server that hosts the service, and consists of a set of functionalities that are logically combined into three entities: *Client*, *Access and Control*, and *Application Server*.

After a user has initiated request for a specific VR service, the *Client* passes it to the *Access and Control* entity, specifying terminal capabilities and user preferences in a

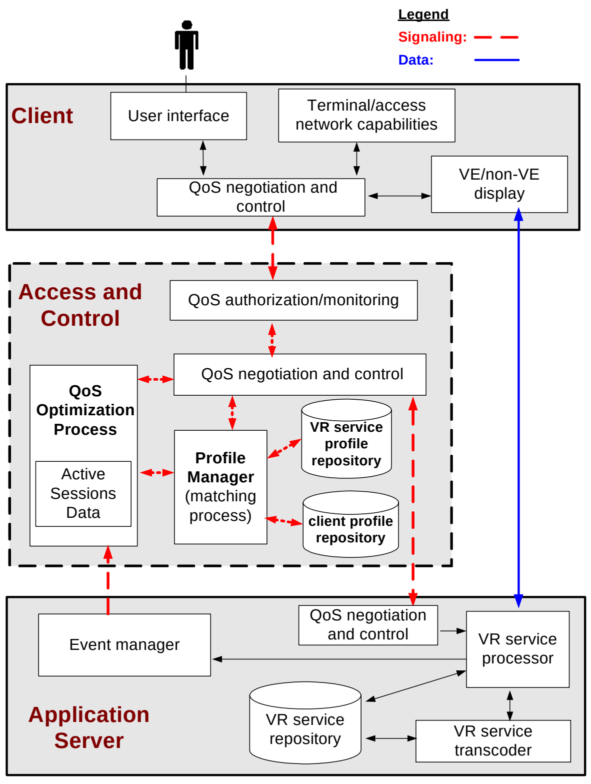

Legend
Signaling:
Data:
Client
User interface
Terminal/access network capabilities
QoS negotiation and control
VE/non-VE display
Access and Control
QoS authorization/monitoring
QoS negotiation and control
QoS Optimization Process
Active Sessions Data
Profile Manager (matching process)
VR service profile repository
client profile repository
Event manager
QoS negotiation and control
VR service processor
Application Server
VR service repository
VR service transcoder

Figure 3. Model for dynamic negotiation and adaptation of QoS

“client profile”. A client profile incorporates user preferences such as acceptable service format(s) and maximum download time, terminal hardware and software, and access network characteristics. The *Access and Control* entity represents a group of service control and management functionalities, and is responsible for identifying the client, authorizing requested network resources, and negotiation of QoS parameters for the service.

The *QoS negotiation and control* (QNC) receives the client request and invokes the *Profile Manager* (PM). The PM retrieves “service profiles” describing various service configurations for the requested service and matches parameters of the service profiles with constraints of the client profile in order to determine achievable service configurations. A service configuration is assumed achievable when: (1) a user’s terminal capabilities are able to support the requested service processing requirements; (2) the user’s access network is able to support the minimum network requirements for all required media elements; and (3) the user’s preferences in terms of desired media elements and acceptable download time can be met.

After the matching process, the PM extracts a set of potential session parameters (i.e. media formats and codec types) from service configurations that are feasible and forwards it to the QNC. The QNC sends offered session parameters to the *Client*, which in return indicates the subset of offered parameters it agrees to. Network entities authorize resources based on the agreed subset of parameters.

The returned parameter subset is sent back to the PM, which then orders the achievable service configurations according to quality based on user perceived quality. Quality of the achievable service configurations is influenced by user preferences (i.e. a user considers video to be of more importance than audio), and different configuration can be used if service degradation or upgrading is required. The service profile with the highest quality configuration is sent to the *QoS Optimization Process* (QOP).

The QOP determines the optimal service operating point and resource allocation taking into account constraints related to service requirements, terminal capabilities and user preferences, and network resource availability and cost. By the service operating point we assume the final configuration of the VR service (included media elements, associated media formats and codec types, etc.) that is to be delivered to the user.

After determination of the service operating point and resource allocation, the QOP sends the final service configuration profile to the QNC of the *Application Server*. The *Application Server* passes the final service profile to the *Client*. In addition, reservation of network resources is invoked.

The *Application Server* is responsible for retrieval and adaptation of hosted VR service based on the calculation carried out by the *Access and Control*. The QNC of the *Application Server* receives the final profile and sends it to the *VR service processor*. If necessary, service content is adapted, after which the *VR service processor* delivers it to the user. A generic sequence diagram of session establishment is shown in Fig. 4.

A user’s interest in particular virtual environment objects changes dynamically. Depending on provided interactions, a user may, for instance, choose to start video streaming. If an important change in user’s interest occurs, a need to determine the new service operating point and reallocate network resources may arise in order to meet new service requirements. A user interaction, or a change perceived by the service itself cause an event to be sent from the service to the *Event Manager* (EM). Using received events, the EM informs the QOP of new service conditions.

Negotiation/adaptation and optimization procedures are invoked throughout the service lifetime in response to significant network conditions and changes occurring in service requirements and constraints like network resource availability, network resource cost, and the client profile. Each of the parties involved - the client side, the server side, and the network - respond to dynamic changes in the system.

Three scenarios are specifically addressed:

- Changes in service requirements refer to addition or detraction of application components (for instance, starting or stopping video and audio streaming) which result in signaling, among rest, reservation or release of network resources.
- Changes in client profile refer to variations in any client profile parameter (user terminal hardware or software characteristics, access network characteristics, user preferences) and are simulated by sending new client profile versions from the client side.
- Changes in resource availability refer to variations of authorized network resources and result in signaling new conditions to the end-points.

B. *Dynamic service adaptation model implementation*

While the proposed model is independent of the particular network scenario, its applicability is of particular interest in the 3GPP’s (3rd Generation Partnership Project) IP Multimedia Subsystem (IMS) [7], a key path to providing the converged next generation network architecture. An implementation of the model was developed by mapping the DSAM model entities to different nodes of the IMS architecture (more details in [11]). For each of the conditions (session establishment, change in client profile, change in service requirements, and change in resource availability) covered by

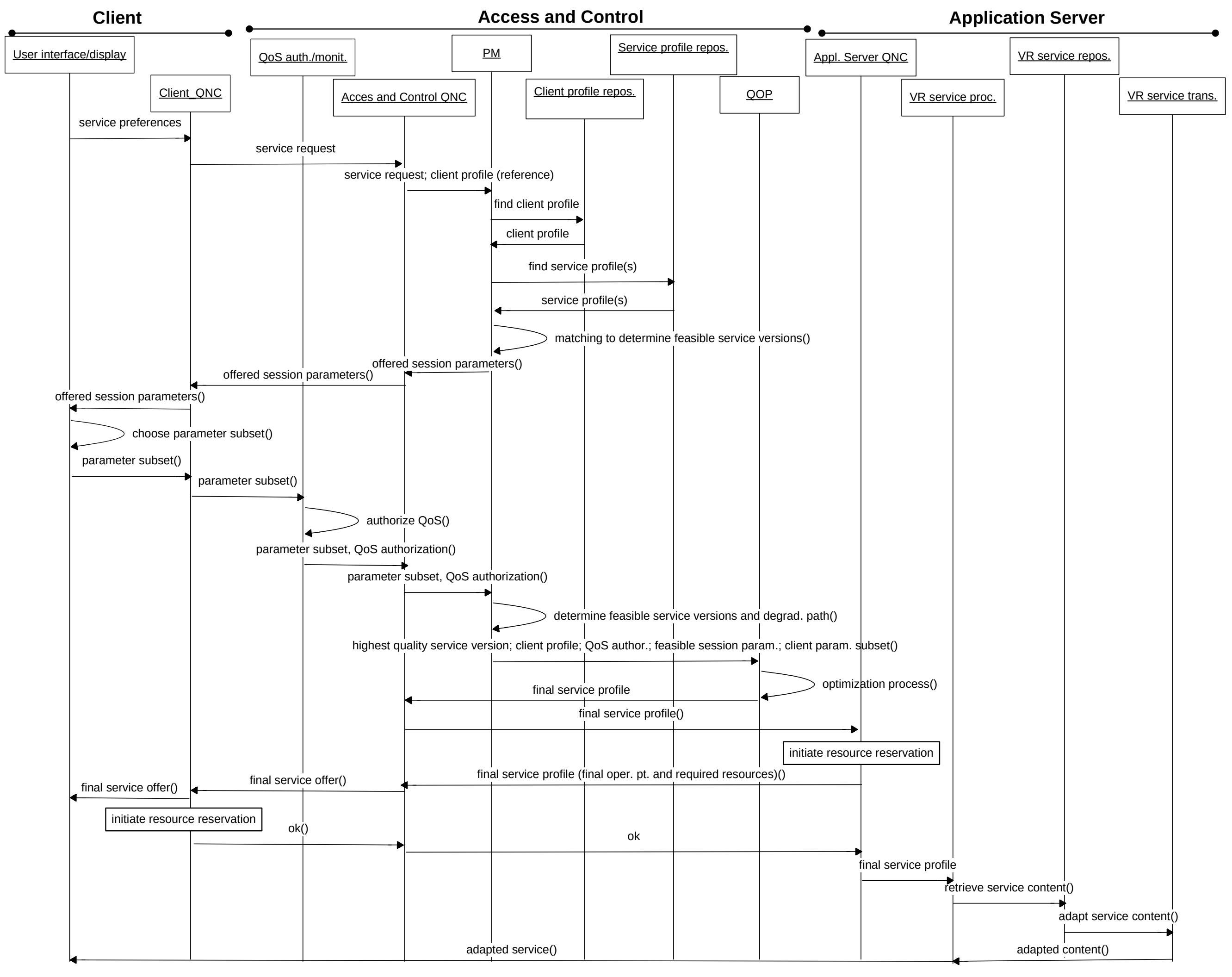

Client
Access and Control
Application Server
User interface/display
Client_QNC
QoS auth./monit.
Acces and Control QNC
PM
Client profile repos.
Service profile repos.
QOP
Appl. Server QNC
VR service proc.
VR service repos.
VR service trans.
service preferences
service request
service request; client profile (reference)
find client profile
client profile
find service profile(s)
service profile(s)
matching to determine feasible service versions()
offered session parameters()
offered session parameters()
offered session parameters()
choose parameter subset()
parameter subset()
parameter subset()
authorize QoS()
parameter subset, QoS authorization()
parameter subset, QoS authorization()
determine feasible service versions and degrad. path()
highest quality service version; client profile; QoS author.; feasible session param.; client param. subset()
optimization process()
final service profile
final service profile()
initiate resource reservation
final service profile (final oper. pt. and required resources)()
final service offer()
final service offer()
initiate resource reservation
ok()
ok
final service profile
retrieve service content()
adapt service content()
adapted service()
adapted content()

Figure 4. Generic sequence diagram for initial session establishment

the model a specific signaling scenario, that includes exchange of signaling messages between involved parties, has been defined according to the 3GPP specifications [4], [8], [9]. End-to-end signaling is preformed using widely adopted IETF’s Session Initiation Protocol (SIP) [2] that, in our case, is used to exchange XML-based client and service profiles. Implementation of this signaling functionality will be used as the basis for API development.

II. Dynamic Service Adaptation API

The goal of this project was to identify generic signaling functionality for application network-awareness and “fold” it into an API to be used by various multimedia and multimodal applications. The API, named Dynamic Service Adaptation (DSA) API, was designed with client/server architecture in mind meaning that one part of the API is to be used on the client side (DSA Client API related to client application with client platform) and the other part is to be used on the side of an application hosting the service(s) (DSA Server API related to server application with server platform), as shown in Fig. 5.

By using developed API, the application developers should be shielded from the signaling protocol specifics. The functionality of the API covers signaling service requirements (in our case service profile), client characteristics (in our case client profile) and final service configuration during session establishment (service invocation) and session update (service run-time phases) by exchanging messages, and capability of receiving notifications of various events that are related to changing conditions. Session update capability is initiated in response to changes occurring in service requirements, network resource availability and/or costs, and client capabilities - scenarios already referred to as change in service requirements, change in client profile, and change in resource availability. The effects of signaling may include network-aware service adaptation in response to varying conditions, as well as adequate network response to client and service requirements, with the overall goal of providing a better service to the user.

Fig. 6 portrays the building blocks of DSA API described hereafter.

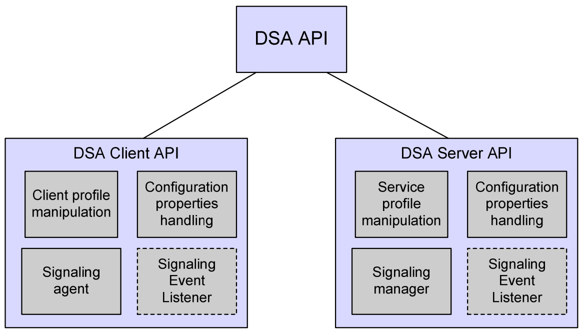

DSA API
DSA Client API
Client profile manipulation
Configuration properties handling
Signaling agent
Signaling Event Listener
DSA Server API
Service profile manipulation
Configuration properties handling
Signaling manager
Signaling Event Listener

Figure 6. DSA API architecture

1) *DSA Client API*

For the client part of DSA API several high-level functionalities were abstracted. Most importantly, the client part should handle all the signaling with involved parties in terms of exchanging signaling messages. This includes sending client profiles and session descriptions, as well as receiving notifications of events occurring in the network or at the server side. Furthermore, API implementation should ease client profile manipulation. With these requirements in mind, following modules were identified (Fig. 7):

- Signaling agent
- Signaling event listener
- Client profile handler

Signaling agent entity is the most important part of the Client API, responsible for handling all signaling messages. Several methods were identified as mandatory for this signaling capability. First one is *establishSession()* which initiates the signaling exchange with other network entities involved. As an input argument it should receive client profile description in a XML format that includes definition of client preferences and capabilities. Analogue to this method, the session can be terminated at any time by calling the *terminateSession()* method. The *changeInClientProfile()* method models a scenario when a change in client profile

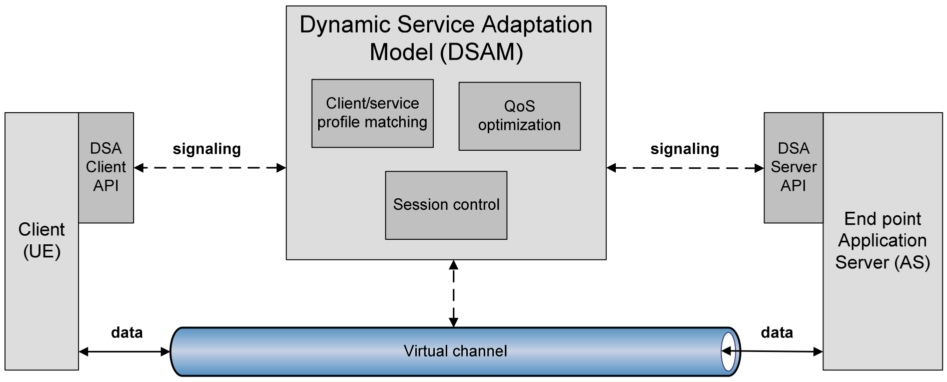


Figure 5. DSA API embedded in DSAM model architecture

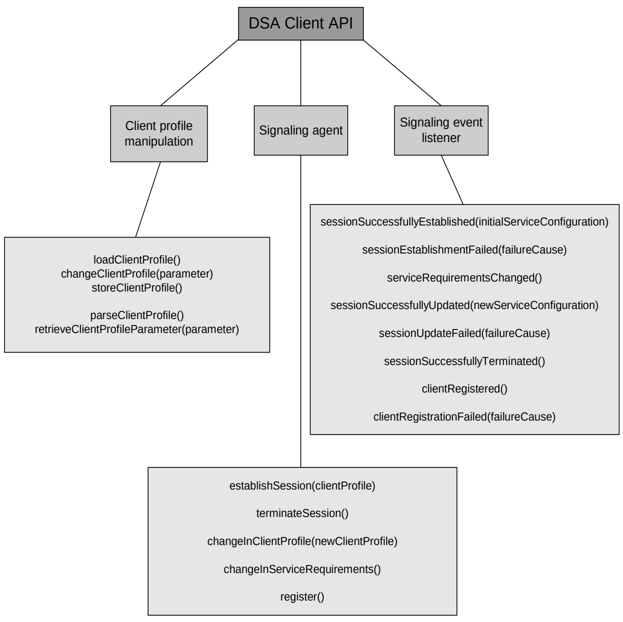

DSA Client API
Client profile manipulation
Signaling agent
Signaling event listener
loadClientProfile()
changeClientProfile(parameter)
storeClientProfile()
parseClientProfile()
retrieveClientProfileParameter(parameter)
sessionSuccessfullyEstablished(initialServiceConfiguration)
sessionEstablishmentFailed(failureCause)
serviceRequirementsChanged()
sessionSuccessfullyUpdated(newServiceConfiguration)
sessionUpdateFailed(failureCause)
sessionSuccessfullyTerminated()
clientRegistered()
clientRegistrationFailed(failureCause)
establishSession(clientProfile)
terminateSession()
changeInClientProfile(newClientProfile)
changeInServiceRequirements()
register()

Figure 7. DSA Client API specification

occurs, for instance due to change of access network or user preferences. Another method, the *changeInServiceRequirements()*, covers a scenario when the client side initiates change in service requirements related to signaling release of network resources reserved for (a) particular service component(s). If a user is required to register to use the network services, registration process is invoked by calling the *register()* method.

Signaling event listener entity is responsible for receiving events related to signaling progress. This includes basic notifications on state of session establishment (*session successfully established* and *session establishment failed*), session update (*session successfully updated* and *session update failed*), change in service requirements/signaling release of network resources (*service requirements changed*), and session termination (*session successfully terminated*), regardless of which entity initiated the signaling. Additionally, the client side can be notified of the registration process (*client successfully registered* and *client registration failed*).

Client profile manipulation module is responsible for the client profile creation and modification.

2) DSA Server API

DSA Server API was intended to provide applications with the means to specify service requirements and changes thereof in response to various user demands and network conditions. As specifying service requirements is done in terms of the service profile, this assumes the service parameters to be specified in a standard format after which they are embodied in the signaling messages and delivered to other entities.

Basic requirements of the DSA Server API included signaling capability, based on the proposed signaling functionality, and ability to receive and properly interpret indications from the network. Signaling capability refers to building blocks and methods that handle signaling specifics based on the proposed message flow diagrams. Indications from the network are based on the signaling progress in a particular scenario, and are meant to signal various network conditions of interest to application (changes in user preferences, capabilities of user terminals, access network conditions and network resource availability). Specification of the DSA Server API is shown in Fig. 8.

Signaling manager entity is directly associated with the signaling capability that is able to manage many clients (users). Its functionality takes care of processing and/or sending proper signaling message depending on developing network conditions or service requirements. Besides defining methods for starting and shutting down this entity, a method for handling changing service requirements in terms of signaling new service configurations has to be modeled. The latter only handles the case where application initiates signaling between involved parties, the rest is managed automatically.

Signaling event listener entity is, analogously to the client side, related to receiving events associated to signalization progress and, through it, to varying network conditions. *Session successfully established* and *session establishment failed* are to receive events specific to setting up a session between an application and a client. This process precedes initial service retrieval. *Session successfully terminated* manages events specific to session termination. *Session successfully updated* and *session update failed* are to handle events specific to session update. These events arise in response to changing network conditions and/or service requirements after session establishment. Any service component needs network resources to be reserved in order to be delivered to a user. As the signaling functionality assumes signaling network (transport) QoS requirements to underlying network entities in order to reserve necessary resources, *reserved network resources released* event has been introduced in order to indicate the release of those resources.

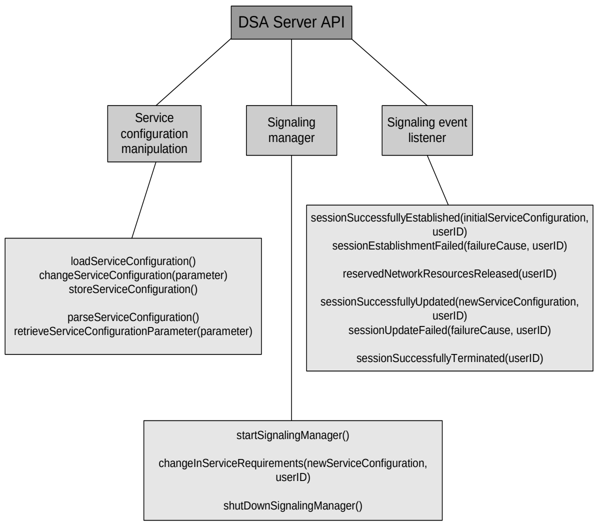

DSA Server API
Service configuration manipulation
Signaling manager
Signaling event listener
loadServiceConfiguration()
changeServiceConfiguration(parameter)
storeServiceConfiguration()
parseServiceConfiguration()
retrieveServiceConfigurationParameter(parameter)
sessionSuccessfullyEstablished(initialServiceConfiguration, userID)
sessionEstablishmentFailed(failureCause, userID)
reservedNetworkResourcesReleased(userID)
sessionSuccessfullyUpdated(newServiceConfiguration, userID)
sessionUpdateFailed(failureCause, userID)
sessionSuccessfullyTerminated(userID)
startSignalingManager()
changeInServiceRequirements(newServiceConfiguration, userID)
shutDownSignalingManager()

Figure 8. DSA Server API specification

Service configuration manipulation manages service configuration processing in terms of retrieving (*retrieve service configuration parameter*) or changing (*change service configuration*) a particular configuration parameter.

III. DSA API Implementation

DSA API Reference Implementation (RI) is based on the NIST-SIP API [3] and the 3GPP specifications [4], [8], [9] providing SIP signaling mechanisms and specifics.

A. DSA Client API RI

Reference implementation of DSA Client API relies on two *Java* packages:

- *hr.fer.tel.nims.dsa.client*, and
- *hr.fer.tel.nims.dsa.client.clientprofilehandler*.

First package (Fig. 9) contains *SignalingAgent* and *SignalingAgentException Java* classes, and the *SignalingEventListener Java* interface. *SignalingAgent* corresponds to the *Signaling agent* and is initialized with the reference to an implementation of the *SignalingEventListener Java* interface and a path to the configuration properties file. Methods implemented in the *SignalingAgent* are used for establishing, updating, and terminating the session, as described in the previous section. The *SignalingAgentException* was introduced in order to notify the client with a description of a problem related to the signaling.

The *SignalingEventListener Java* interface was modeled according to the *Signaling event listener* entity. An implementation of the *handleSessionEstablishedEvent()* method is notified of successful session establishment with a server hosting the service, and passed an initial service configuration for the session. An implementation of the *handleSessionEstablishmentFailedEvent()* method is notified of a failure during session establishment and passed a description of a failure cause. Event describing successful session termination is delivered by calling an implementation of the *handleSessionTerminatedEvent()* method. In response to changing network conditions, session has to be updated.

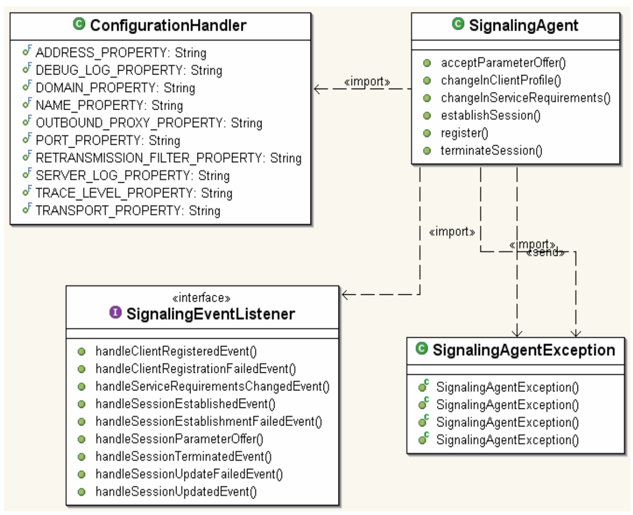

ConfigurationHandler
ADDRESS_PROPERTY: String
DEBUG_LOG_PROPERTY: String
DOMAIN_PROPERTY: String
NAME_PROPERTY: String
OUTBOUND_PROXY_PROPERTY: String
PORT_PROPERTY: String
RETRANSMISSION_FILTER_PROPERTY: String
SERVER_LOG_PROPERTY: String
TRACE_LEVEL_PROPERTY: String
TRANSPORT_PROPERTY: String
SignalingAgent
acceptParameterOffer()
changeInClientProfile()
changeInServiceRequirements()
establishSession()
register()
terminateSession()
«import»
«import»
«import»
«send»
«interface»
SignalingEventListener
handleClientRegisteredEvent()
handleClientRegistrationFailedEvent()
handleServiceRequirementsChangedEvent()
handleSessionEstablishedEvent()
handleSessionEstablishmentFailedEvent()
handleSessionParameterOffer()
handleSessionTerminatedEvent()
handleSessionUpdateFailedEvent()
handleSessionUpdatedEvent()
SignalingAgentException
SignalingAgentException()
SignalingAgentException()
SignalingAgentException()
SignalingAgentException()

Figure 9. Package *hr.fer.tel.nims.dsa.client**

The *handleSessionUpdatedEvent()* method notifies of successful update of the session by delivering new service configuration, while the *handleSessionUpdateFailedEvent()* method delivers a description of a failure cause along. The *handleServiceRequirementsChangedEvent()* method implementation is notified of network resource release for (a) particular service component(s). Methods related to registration process are implemented according to the behavior model described in the previous section.

Package *hr.fer.tel.nims.dsa.client.clientprofilehandler* contains functionalities for handling the client profile specifics. Main *Java* class for managing both client and service profiles is the *ProfileParser* which implements *ProfileInterface Java* interface. It comprises methods for managing profiles: *getParameter()*, *addParameters()*, *editParameters()*, and *getParameterValue()*. Parsing the profiles is done using the Simple API for XML (SAX) parser [6].

B. DSA Server API RI

Following API specification, reference implementation relies on three *Java* packages:

- *hr.fer.tel.nims.dsa.server*,
- *hr.fer.tel.nims.dsa.server.eventlistener*, and
- *hr.fer.tel.nims.dsa.profilemanipulation*.

First package contains *SignalingManager Java* class (Fig. 10) that corresponds to the *Signaling manager* entity. Its constructor is initialized with the configuration properties file and the reference to an implementation of the *SignalingEventListener Java* interface. The *startSignalingManager()* method starts, while the *shutDownSignalingManager()* method terminates the components of the manager. As explained previously, the *changeInServiceRequirements()* method initiates signaling new service requirements in terms of new service configurations.

The *hr.fer.tel.nims.dsa.server.eventlistener* defines *SignalingEventListener Java* interface (Fig. 10) that was modeled with the *Signaling event listener* entity. An implementation of the *handleSessionEstablishedEvent()* method is notified of successful session establishment with a particular client and passed an initial service configuration for the session. An implementation of the *handleSessionEstablishmentFailedEvent()* method is notified of a failure during session establishment with a particular client and passed a description of a failure cause. Event describing successful session termination with a particular client is delivered by calling an implementation of the *handleSessionTerminatedEvent()* method. In response to changing any network condition, session has to be updated. The *handleSessionUpdatedEvent()* method notifies of successful update of the session with a particular client by

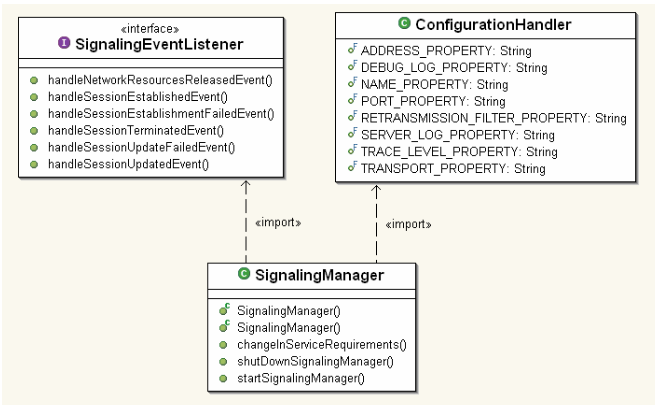

«interface»
SignalingEventListener
handleNetworkResourcesReleasedEvent()
handleSessionEstablishedEvent()
handleSessionEstablishmentFailedEvent()
handleSessionTerminatedEvent()
handleSessionUpdateFailedEvent()
handleSessionUpdatedEvent()
ConfigurationHandler
ADDRESS_PROPERTY: String
DEBUG_LOG_PROPERTY: String
NAME_PROPERTY: String
PORT_PROPERTY: String
RETRANSMISSION_FILTER_PROPERTY: String
SERVER_LOG_PROPERTY: String
TRACE_LEVEL_PROPERTY: String
TRANSPORT_PROPERTY: String
«import»
«import»
SignalingManager
SignalingManager()
SignalingManager()
changeInServiceRequirements()
shutDownSignalingManager()
startSignalingManager()

Figure 10. Package *hr.fer.tel.nims.dsa.server**

delivering new service configuration, while the *handleSessionUpdateFailedEvent()* method delivers a description of a failure cause along. An implementation of the *handleNetworkResourcesReleasedEvent()* method notifies of network resource release for a particular client. Each client must uniquely be identified with its IP address.

The *profilemanipulation* package (Fig. 10) consists of several *Java* classes that handle the service configuration format (service profile) explained in the first section. The *ProfileInterface Java* interface defines basic format/configuration manipulation methods, while the *ProfileParser Java* class implements methods for parsing it and modifying its values. The *XmlElement* Java class symbolizes a tag in the XML (Extensible Markup Language) structure with accompanying attributes.

IV. Case study

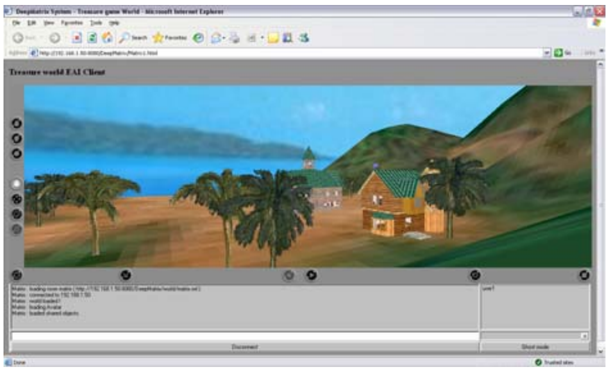
In order to show the applicability of the DSA API, a prototype Web-based application has been designed and developed. It hosts a 3D virtual world featuring a treasure hunt-like game and was extended with the signaling capability. Example client and service profiles that describe different user preferences, terminal capabilities, access network conditions, and service requirements were specified. Using the prototype application functionality of the API was tested in a laboratory testbed.

A. *The prototype application*

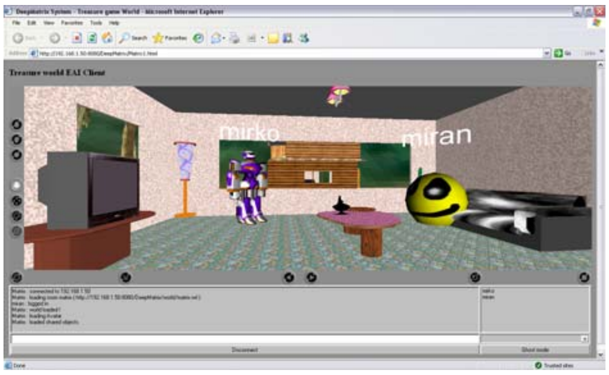
Our case study application, the *Inheritance Chase*, is a multiplayer game based on the client/server network architecture. The game scenario consists of a real-time adventure similar to a treasure hunt and is taking place in a 3D world developed using the *Virtual Reality Modeling Language* (VRML). Its plot is as follows. Players’ rich distant relative has deceased recently and left a vast inheritance. His last will is hidden somewhere in the virtual world and each player has to find it first in order to get the inheritance. To achieve that, they have to follow different audio and/or video clues.

The virtual world (Fig. 11) consists of two scenes: an island with two houses (Fig. 11a), which is a part of the world where most of the game takes place, and the scene containing a large chessboard (Fig. 11c), associated to one of the clues. After a player enters the game, the main scene is retrieved from the server side. Each player is represented with an avatar (virtual 3D character, Fig. 11b) which is visible to other players. As players explore the world, they come across the clues. Clues that lead players to finding the will were designed in different forms - some of them are streaming audio/video clips, others were implemented using special VRML elements bound to the scenes themselves. There are particular scene objects that are to be selected with the mouse in order to start “streaming” clues playing. All this service content (virtual 3D scenes, avatars, real-time streaming media, texture images) contribute to complexity of the system which, we believe, may serve as an example of an advanced multimedia and multimodal application, and its complex QoS requirements at the transport layer.

Implementation of the application hosting this service is divided into three parts. The first part refers to the SIP signaling functionality in the terms of specifying service requirements using service profiles. This logic was developed in a way to meet dynamic nature of the system and handle exchange of signaling messages as defined by dynamic negotiation and adaptation scenarios (chapter *1.A*). The second part is responsible for retrieving the 3D scenes, starting/stopping and displaying audio/video clips, synchronizing virtual world states among different players etc.

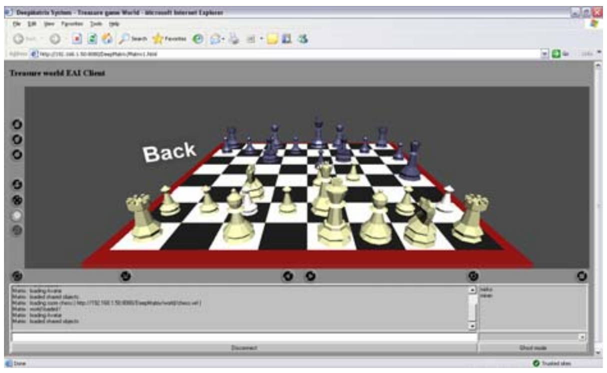


a. The island



b. Players’ avatars

Figure 11. The Inheritance Chase game



c. The chessboard

Figure 11. The Heritage Chase game

“Real-time media” hints are streamed and displayed using *Java Media Framework* (JMF) *API* based players. The third one is related to the service content and has been realized using an *Apache Tomcat* Web server. Multiplayer engine of the game is called DeepMatrix, and is written using *Java* programming language.

The *Inheritance Chase* game has been developed in three different service versions. Each of them regulates which service components, and in what form, are going to be delivered to a user, depending on the client side capabilities, network conditions, and service requirements. These three support:

(1) high quality audio and video streaming,

(2) low quality audio and video streaming, and

(3) low quality audio streaming only,

each with accompanying set of media codecs offered. Their configurations are stored as service profiles on server that hosts the service and organized according to predefined XML structure. For each service version this implementation provides only static transcoding, which means that service content has to be prepared in advance.

The client side is represented with several different client profiles based on various user terminal capabilities, access network characteristics, and user preferences. Client profile format is based on the SDPng [5].

B. *Laboratory setup and test scenarios*

As mentioned before, DSAM model was mapped to the 3GPP IMS architecture. This mapping was used as a reference for DSA API implementation. The API embedded in DSAM prototype implementation entities of a laboratory testbed is shown in Fig. 12. The *Session control element* is responsible for managing the signaling flows. It routes messages from the *Client* to the *End point Application Server* through the *QoS Matching and Optimization Node*. It is also responsible for registration and authentication procedures. The *QoS Matching and Optimization Node* is the central part of DSAM model, responsible for matching process and optimization calculations. The *Policy enforcement* element is used for reservation of the negotiated network resources. It also detects any change in network resource availability. The *Network control element* is used for forwarding signaling messages between the *Client* and the *Session control element*, and for passing negotiated resource reservation parameters to the *Policy enforcement* element. The *Virtual channel* emulates various network conditions.

The prototype application integrated with developed DSA API was tested with the following scenarios [10]:

- Session establishment,
- Change in service requirements,
- Change in client profile, and
- Change in network resource availability.

Session establishment is invoked by an end-user (a player). It includes procedure of registering a user-terminal to the network, negotiating initial service parameters (Fig. 13) in the terms of final service profile, and service retrieval (scene download, Fig. 14) in accordance with negotiated configuration. This scenario also comprises signaling

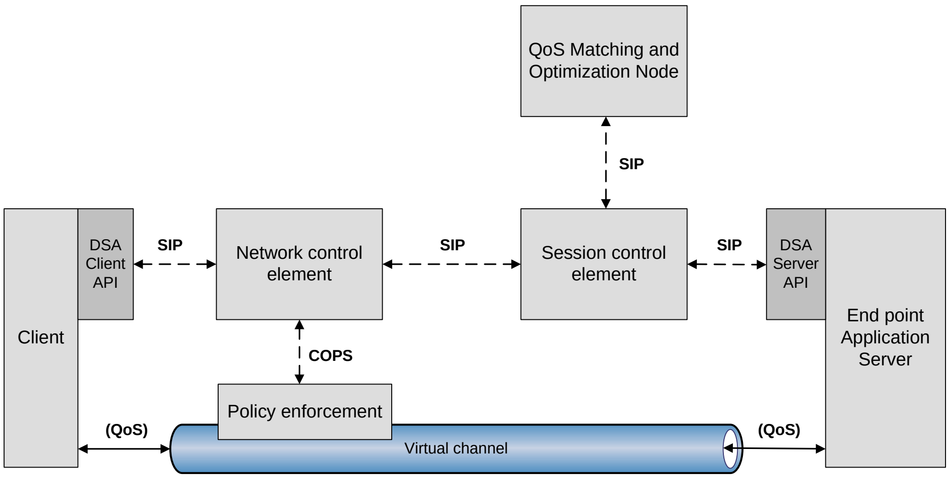
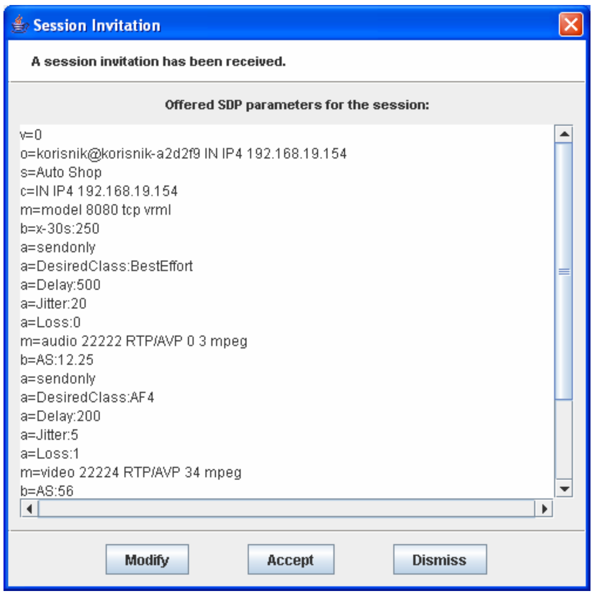


Figure 12. DSA API embedded in laboratory testbed implementation entities



A session invitation has been received.

Offered SDP parameters for the session:

```
v=0
o=korisnik@korisnik-a2d2f9 IN IP4 192.168.19.154
s=Auto Shop
c=IN IP4 192.168.19.154
m=model 8080 tcp vrml
b=x-30s:250
a=sendonly
a=DesiredClass:BestEffort
a=Delay:500
a=Jitter:20
a=Loss:0
m=audio 22222 RTP/AVP 0 3 mpeg
b=AS:12.25
a=sendonly
a=DesiredClass:AF4
a=Delay:200
a=Jitter:5
a=Loss:1
m=video 22224 RTP/AVP 34 mpeg
b=AS:56
```

Figure 13. Session parameters offered during initial session establishment

authorization, reservation, and release of network resources used for scene download, as needed.

Change in service requirements is caused by a user initiating an audio and/or video streaming. Signaling new service requirements is invoked by the server side, and new service configuration is negotiated based on information, carried in signaling messages, that are related to streams being requested. The *QoS Matching and Optimization Node* calculates optimal audio and video codec combination based on user preferences, user terminal constraints, network capabilities (bandwidth, delay, loss, etc.), resource cost, and service requirements. Prior to starting media streaming (Fig. 15), reservation of network resources is signalized. Through the *Policy enforcement* entity the *Network control element* reserves the calculated resources at the virtual channel.

The third scenario, change in client profile, is caused by an increase or a decrease in the user's access network bandwidth, which results in new negotiation and optimization process.

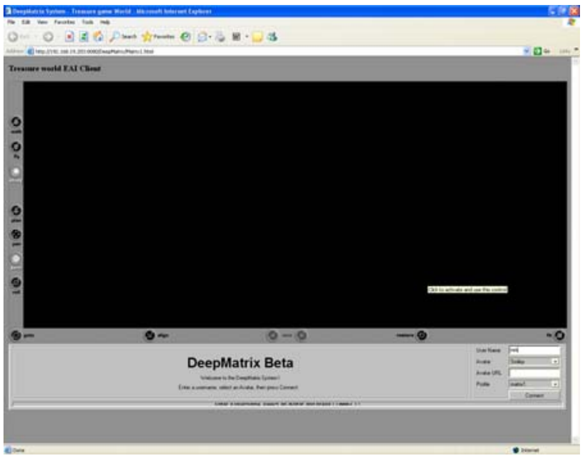


Figure 14. Service retrieval

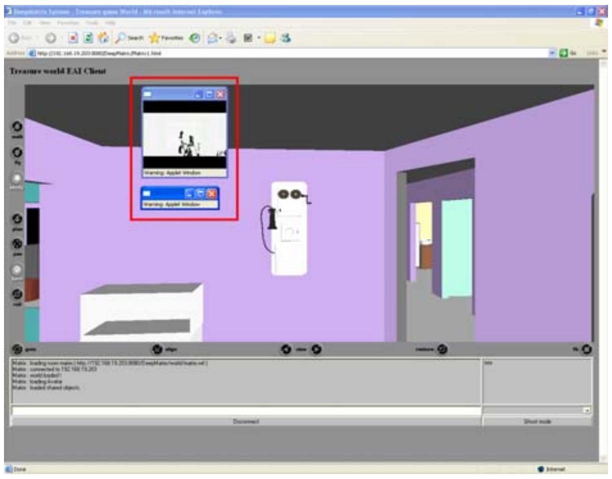


Figure 15. "Streaming" audio and video clues

This change is simulated by sending a new client profile configuration from the client side. If, for instance, media streaming is taking place at that instant, and if a variation of the bandwidth increase is significant, automatic change of the streaming quality/codecs (Fig. 16 and Fig. 17) will occur according to the new service configuration.

Change in network resource availability is detected by the *Network control element* (receives information from the *Virtual channel*). This again invokes negotiation and optimization process, which results in a new service configuration. Automatic changes of service parameters (i.e. audio codec due to a decrease of authorized network resources, Fig. 18 and Fig. 19) at the client and the server side occur in compliance with negotiated service profile.

During service run-time, changes in various parameters/constraints, related to the client side, service requirements, and the network resources can occur, and it was shown that each time an adapted version of all the service components will be delivered to a user.

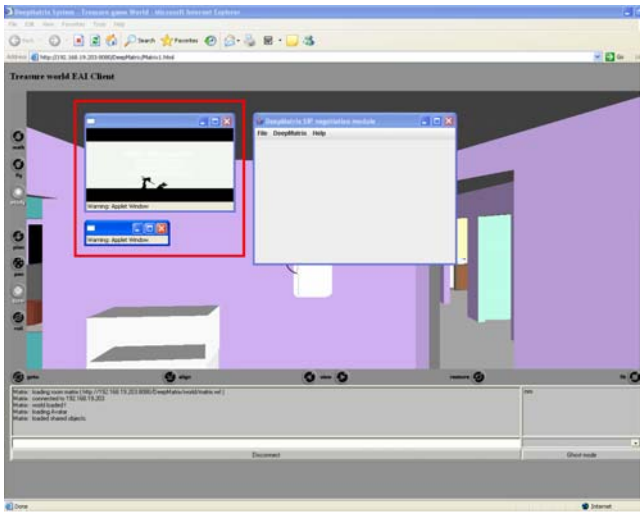
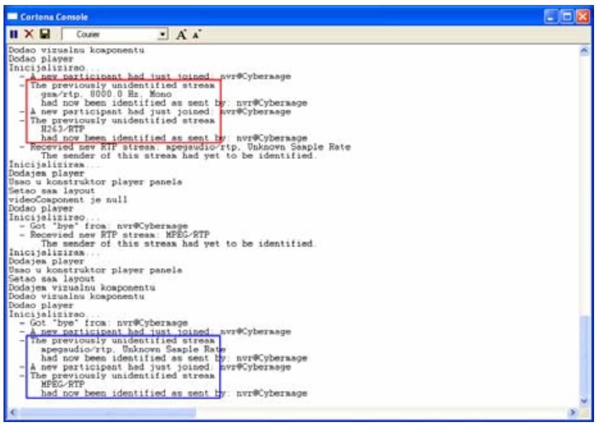


Figure 16. "Streaming" audio and video clues after codec change



```
Dodao vizualnu komponentu
Dodao player
Inicijalizirao...
  - A new participant had just joined: nvr@Cybermage
  - The previously unidentified stream
      gsm/rtp, 8000.0 Hz, Mono
      had now been identified as sent by: nvr@Cybermage
  - A new participant had just joined: nvr@Cybermage
  - The previously unidentified stream
      H263/RTP
      had now been identified as sent by: nvr@Cybermage
  - Recevied new RTP stream: mpegaudio/rtp, Unknown Sample Rate
      The sender of this stream had yet to be identified.
Inicijaliziram...
Dodajem player
Usao u konstruktor player panela
Setao sam layout
videoComponent je null
Dodao player
Inicijalizirao...
  - Got "bye" from: nvr@Cybermage
  - Recevied new RTP stream: MPEG/RTP
      The sender of this stream had yet to be identified.
Inicijaliziram...
Dodajem player
Usao u konstruktor player panela
Setao sam layout
Dodajem vizualnu komponentu
Dodao vizualnu komponentu
Dodao player
Inicijalizirao...
  - Got "bye" from: nvr@Cybermage
  - A new participant had just joined: nvr@Cybermage
  - The previously unidentified stream
      mpegaudio/rtp, Unknown Sample Rate
      had now been identified as sent by: nvr@Cybermage
  - A new participant had just joined: nvr@Cybermage
  - The previously unidentified stream
      MPEG/RTP
      had now been identified as sent by: nvr@Cybermage
```

Figure 17. Applied codec change after increase in network bandwidth

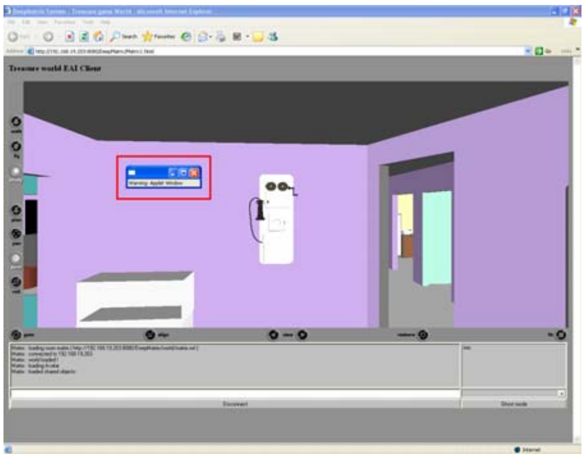
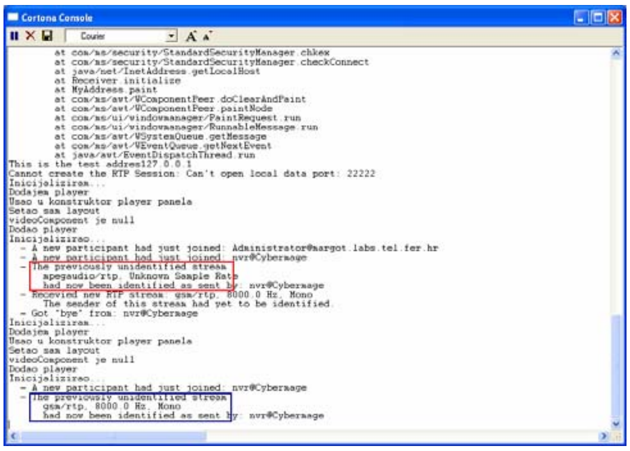


Figure 18. "Streaming" audio clue



```
        at com/ms/security/StandardSecurityManager.chkex
        at com/ms/security/StandardSecurityManager.checkConnect
        at java/net/InetAddress.getLocalHost
        at Receiver.initialize
        at MyAddress.paint
        at com/ms/awt/WComponentPeer.doClearAndPaint
        at com/ms/awt/WComponentPeer.paintNode
        at com/ms/ui/windowmanager/PaintRequest.run
        at com/ms/ui/windowmanager/RunnableMessage.run
        at com/ms/awt/WSystemQueue.getMessage
        at com/ms/awt/WEventQueue.getNextEvent
        at java/awt/EventDispatchThread.run
This is the test addres127.0.0.1
Cannot create the RTP Session: Can't open local data port: 22222
Inicijaliziram...
Dodajem player
Usao u konstruktor player panela
Setao sam layout
videoComponent je null
Dodao player
Inicijalizirao...
  - A new participant had just joined: Administrator@margot.labs.tel.fer.hr
  - A new participant had just joined: nvr@Cybermage
  - The previously unidentified stream
      mpegaudio/rtp, Unknown Sample Rate
      had now been identified as sent by: nvr@Cybermage
  - Recevied new RTP stream: gsm/rtp, 8000.0 Hz, Mono
      The sender of this stream had yet to be identified.
  - Got "bye" from: nvr@Cybermage
Inicijaliziram...
Dodajem player
Usao u konstruktor player panela
Setao sam layout
videoComponent je null
Dodao player
Inicijalizirao...
  - A new participant had just joined: nvr@Cybermage
  - The previously unidentified stream
      gsm/rtp, 8000.0 Hz, Mono
      had now been identified as sent by: nvr@Cybermage
```

Figure 19. Applied codec change after authorized resources decrease

V. Conclusion

The developed API offers various benefits to application developers. It may ease development of advanced multimodal and multimedia applications with network-aware adaptation and shorten the application development time. The proposed approach differs from current approaches, where applications either (1) do not use signaling at all (e.g. most Internet applications), or, (2) use a standard network and/or service specific signaling protocol (e.g. H.323, SIP) but have the signaling capability built into, and thus inseparable from, the client application or client platform. While the second approach enables the exchange of control information, it is practically impossible to reuse this functionality due to tight coupling with the application. Also, this approach assumes that the application developer knows the signaling protocol specifics very well, and is capable of building the signaling agent into each and every new application from scratch. Finally, once built into the application, signaling support can not be upgraded to, for instance, a more recent release of the signaling protocol without significant effort and rebuilding the whole application. The proposed approach solves these problems.

Acknowledgment

Ivan Piskovic was partially funded by SIMILAR, the European Network of Excellence on Multimodal Interfaces.

This work was supported by research project Networked Virtual Reality in IP Multimedia Subsystem (NIMS), conducted in the cooperation between the Faculty of Electrical Engineering and Computing, University of Zagreb, and the Ericsson Nikola Tesla company from Zagreb.

References

[1] S. Singhal and M. Zyda, *Networked Virtual Environments: Design and Implementation*, Addison-Wesley, 1999.
[2] J. Rosenberg et al., "SIP: Session Initiation Protocol", RFC 3261, June 2002.
[3] NIST-SIP [Online]. Available: http://snad.ncsl.nist.gov/proj/iptel/
[4] 3GPP TS 23.228: "IP Multimedia Subsystem (IMS); Stage 2", Release 7, June 2005.
[5] Kutscher, Ott, and Bormann, "Session Description and Capability Negotiation", draft-ietf-mmusic-sdpng-08.txt, TZI Universitaet Bremen, February 20, 2005.
[6] SAX [Online]. Available: http://www.saxproject.org/sax1-roadmap.html
[7] G. Camarillo and G.-M. Miguel-Angel, *The 3G IP Multimedia Subsystem (IMS): Merging the Internet and the Cellular Worlds*, John Wiley & Sons, 2004.
[8] 3GPP TS 23.218: "IP Multimedia (IM) session handling; IM call model; Stage 2", Release 7, June 2006.
[9] 3GPP TS 24.228: "IP multimedia call control protocol based on Session Initiation Protocol (SIP) and Session Description Protocol (SDP); Stage 3", Release 7, December 2005.
[10] L. Skorin-Kapov and M. Matijasevic, "Dynamic QoS Negotiation and Adaptation for Networked Virtual Reality Services" in *Proc. of the Sixth IEEE International Symposium on a World of Wireless and Mobile Multimedia*, Italy, 2005, pp. 344-351.
[11] L. Skorin-Kapov and M. Matijasevic, "End-to-end QoS Signaling for Future Multimedia Services in the NGN", Lecture Notes in Computer Science (0302-9743), St. Petersburg, 2006.
[12] M. Matijasevic, D. Gracanin, K. P. Valavanis, and I. Lovrek, "A framework for multi-user distributed virtual environments", *IEEE Transactions on Systems, Man and Cybernetics*, Part B: Cybernetics, 32(4):416–429, August 2002.

Miran Mosmondor received his Dipl. Ing. (2004) degree in electrical engineering from the University of Zagreb, Croatia. Since 2004 he has been employed as a research engineer in the Research and Development Center of the Ericsson Nikola Tesla company in Croatia, working in the area of networked virtual reality. Currently he is also working towards his Ph.D. degree at the Faculty of Electrical Engineering and Computing, University of Zagreb. As an undergraduate student with state scholarship he participated in the Summer Camp 2003: “Agent and Visualization Technologies” workshop, jointly organized by Ericsson Nikola Tesla in cooperation with the Faculty of Electrical Engineering and Computing, University of Zagreb. Later, he became one of the project leaders at Summer Camp 2005 "Exploring ICT Frontiers: Agents, IP Multimedia Subsystem, and Distributed Computing". He published several conference papers, some of which were awarded (IEEE 12th MELECON 2004; Best Student Paper Award). His main research interests are in the field of multimedia communications, virtual environments and mobile applications development.

Maja Matijasevic received her Dipl.-Ing. (1990), M. Sc. (1994), and Ph. D. (1998) degrees in Electrical Engineering from the University of Zagreb, Croatia, and the M. Sc. in Computer Engineering (1997) from the University of Louisiana at Lafayette, LA, USA. She is presently an Associate Professor in the Faculty of Electrical Engineering and Computing, University of Zagreb, Croatia. Her main research interests include networked virtual environments and advanced multimedia for next generation networks. She authored more than 40 publications, and served as a guest editor for several journal special issues. She has been involved in organization of several international conferences. Dr. Matijasevic is a senior member of IEEE, and member of ACM and Upsilon Pi Epsilon Honor Society in the Computing Sciences.

Sasa Desic received his Dipl. Ing., M.Sc. and Ph.D. degrees from the Faculty of Electrical Engineering and Computing, University of Zagreb, Croatia, in the years 1997, 1999 and 2004 respectively. From 1997 to 2000 he worked as a research assistant at Faculty of Electrical Engineering and Computing. He has been with Ericsson Nikola Tesla since 2000, currently employed as head of the Research Department. Previously he was the project leader of a joint research project conducted in cooperation with the Faculty of Electrical Engineering and Computing called “Remote operations management” investigating systems for software management on the remote locations. He actively participates in several research projects conducted in cooperation with the academia including project Location Based Services investigating the integration of different sources of location information. Since 2004, he is also an adjunct assistant professor at the University of Zagreb, Faculty of Electrical Engineering and Computing.

Ognjen Dobrijevic received his Dipl. Ing. degree in Electrical Engineering from the University of Zagreb in June 2004. In 2002 and 2003 he participated in the workshop, organized in the collaboration between the Research Department of the Ericsson Nikola Tesla company in Zagreb and the University of Zagreb, Faculty of Electrical Engineering and Computing, where he was engaged in projects Mobility in Advanced Network Architectures and Visualization Technologies. He has been a research assistant at the Faculty of Electrical Engineering and Computing in Zagreb since September 2004, where he is working towards his Ph.D. His main research interests include next generation networks, related QoS signaling issues, and adaptive multimedia applications. He is a member of the IEEE association.

Ivan Piskovic was born 1983 in Zagreb, Croatia. He finished high school in Zagreb in 2001 and is now an undergraduate student of the Faculty of Electrical Engineering and Computing, University of Zagreb. His interests include architecture of next generation networks and related multimedia session-control issues.

Mirko Suznjevic was born 1983 in Karlovac, Croatia. He finished high school in Glina in 2001 and is now an undergraduate student of the Faculty of Electrical Engineering and Computing, University of Zagreb. In 2005 he participated in Ericsson’s workshop, Summer Camp 2005 "Exploring ICT Frontiers: Agents, IP Multimedia Subsystem, and Distributed Computing", where he worked on development of a networked virtual environment application with multi-user support. His interests include networked virtual environments and network design.

APPENDIX: SOFTWARE DEMONSTRATION NOTES

In order to run demonstration example, installation and configuration of three software components has to be done. Instructions for running each of the provided components under Windows operating system will be described hereafter.

A. Instructions for installing the Inheritance Chase game and the End point Application Server

First step is to:

1) Copy the *matrix_server* folder (further on referred to as '+') to a machine that will run the game server application.

After copying the folder:

2) Create *C:\Documents and Settings\User\Desktop* destination folder, if one already does not exist, and copy *sdpcontent.txt* and *sdpParameters.txt* files from the *+\sdpmediadescription* folder to the destination folder.
3) Edit the *configuration.properties* file, in the *matrix_server* folder, to respond to IP address of the computer and wanted port number.

In order to run the game server, the computer furthermore needs to have the following software installed:

- Apache Tomcat Web server (version 5.5 preferred), and
- Java Media Framework (JMF) API (version 2.1.1e preferred). When the JMF is installed, the *jmf.jar* file from the *<JMF_folder>\lib* folder needs to be copied to the *+\lib* folder.

After installation of the Web server:

1) Modify the *gallery<1,2,3>.xml* files in the *+\serviceprofilerepository\gallery\profiles* folder so that IP address in the *url* parameters responds to IP address of the computer used as the game server.
2) Complete folder named *DeepMatrix* needs to be copied into the *<Tomcat_folder>\Apache Software Foundation\Tomcat 5.5\webapps\ROOT* folder (further on marked as '*').
3) Class file *EventManager* from the *+\bin\nvrcontentserver\vrserviceprocessor* folder needs to be copied into the **\WEB-INF\classes\nvrcontentserver\vrserviceprocessor* folder.
4) Class file *GalleryServlet* from the *+\bin\nvrcontentserver\vrserviceprocessor\...* *...galleryservice* folder needs to be copied into the **\WEB-INF\classes\nvrcontentserver\vrserviceprocessor\...* *...galleryservice* folder.
5) File *web.xml* in the *<Tomcat_folder>\Apache Software Foundation\Tomcat 5.5\conf* folder needs to be modified so that the comments "around" servlet *org.apache.catalina.servlets.InvokerServlet* and "around" servlet mapping for the invoker servlet (tag *servlet-name* equals to *invoker*) are deleted.
6) Files *Matrix.html* and *Matrix1.html* in the **\DeepMatrix* folder have to be modified so the IP addresses correspond to the machine that hosts the game server.
7) File *matrix.wrl* in the **\DeepMatrix\world* folder has to be modified in scripts *calling* and *calling2* so the IP address/computer name corresponds to the machine that hosts the game server. If an IP address is used, input format must be complied. Furthermore, modify IP address in the first two *url* parameters of the *Anchor* node in the *Object3* Transform node to match the computer being used as the game server.

B. Instructions for installing the QoS Matching and Optimization Node

For the purposes of matching client and service profiles, and optimizing final service configuration that is delivered to a user, the *QoS Matching and Optimization Node* (*QMON*) needs to be installed. First step is to:

1) Copy the *SIP-AS* folder (further on referred to as '#') to a machine that will run the component. It is recommended to use a different machine than one hosting the game server application, but it is not necessary.

After copying the folder:

2) Edit the *configuration.properties* file, in the *SIP-AS* folder, so the *javax.sip.IP_ADDRESS* and the *hr.fer.teletk.NETWORK_PRICE_QUOTATION_AD-DRESS* parameters respond to IP address of the computer, and *hr.fer.teletk.SIP_STACK_PORT* to a wanted port number. Furthermore, the *javax.sip.OUTBOUND_PROXY* parameter must respond to IP address and port number of the *Server*'s configuration.
3) Modify all *txt* files in the *#\SimulatedResponses* folder, so that the *ReceivingClientIP* tag responds to IP address of a computer running the client and IP address in the *url* parameter(s) matches IP address of the computer hosting the game server.
4) Add the absolute path of the *#\lib* folder to the *Path* system variable.

C. Instructions for installing the Client

First step is to:

1) Copy the *matrix_client* folder (further on referred to as '$') to a machine that will run the game client application. It is recommended to use a different machine than one hosting the *QMON*, but it is not necessary.

After copying the folder:

2) Edit the *configuration.properties* file, in the *matrix_client* folder, to respond to IP address of the

computer and wanted port number. Furthermore, the *NEXT_HOP* parameter must respond to IP address and the *hr.fer.teletk.SIP_STACK_PORT* port number of the *QMON*.

3) Modify the four files, in the *matrix_client* folder, that contain word *gallery* in their names so that IP address in the *Host* parameter corresponds to IP address of the computer hosting the game server.

In order to run the client application, the computer needs to have the following additional software installed:

- Microsoft Java Virtual Machine (MS JVM, version 5.00.3810 preferred),
- a Web browser (Internet Explorer preferred),
- VRML player/viewer for the particular Web browser, i.e. if Internet Explorer is used, Cortona VRML Client is preferred, and
- JMF API (version 2.1.1e preferred). When the JMF is installed, the *jmf.jar* file from the *<JMF_folder>\lib* folder needs to be copied to the *matrix_client\lib* folder.

When installing the JMF, mark all the checkboxes setup offers. In the *Tools->Internet Options…->Advanced* section of the Internet Explorer, usage of the compiler for Microsoft Virtual Machine has to be enabled.

D. Simple demonstration scenario

When software components have been properly configured, each of them can be started using accompanying batch file. Following simple demonstration scenario, a particular functionality of DSAM prototype implementation can be portrayed. Each client must uniquely be identified with its IP address. Before starting demonstration, it should be checked whether the Web server has been started.

Assumed demonstration scenario will reflect functionality of the software components related to first three scenarios of the case study - session establishment, change in service requirements, and change in client profile.

Session establishment is invoked by a user sending a particular client profile - in this case, for instance, the *Matrix audio and video LQ*. This particular client profile depicts conditions in UMTS access network with somewhat lower bandwidth and user's interest in both audio and video component of the service. Before sending the profile, the user has to specify IP address and port number of the game server. Next step in initial session establishment is offering session parameters to the user. For now, it is only allowed to accept offered parameters. After negotiating initial service parameters in the terms of final service profile, session establishment procedure successfully finishes and main 3D scene of the game is retrieved. It is displayed in the window of a Web browser when the user has logged in (use login *mm*).

Now change viewpoints to the scene until one containing a phone box is reached. Selecting it with the mouse starts "streaming" clues playing (adding audio/video streaming to the game refers to change in service requirements). Prior to streaming, signaling these new service requirements is invoked by the server and new service configuration is negotiated based on the information related to media streams being requested.

While the streaming takes place, the user could, for instance, send the *Matrix audio and video HQ* profile. This client profile depicts the same user's preferences (interest in both audio and video) but somewhat altered conditions in the access network related to a higher bandwidth. Sending new client profile results in new negotiation and optimization process. In this case, when the process finishes the streaming quality improves (audio and video codecs change, according to the new service configuration, which can be seen in the console of the VRML viewer).

Completion of the streaming also results in signaling changed service requirements, but this time associated to detraction of service components.