ICT-287760

Vconect

Video Communication for Networked Communities

Specific targeted research project

ICT – Networked Media

D2.3 Open Reference Architecture and Deployment - Interim Specification

Due date of deliverable: 30.06.2013

Actual submission date: 05.08.2013

Start date of project: 1 December 2011 Duration: 36 months

Lead contractor for this deliverable: Fraunhofer IIS

Version 0.6 of 2 August 2013

Confidentiality status: Public

Abstract

This technical deliverable includes an interim specification of the Vconect platform for 3rd party developers. The platform is described from the perspective of a black box: the functions and features which are currently available are clearly described and the Application Programming Interface (API) is specified and illustrated with code examples. It is also described how client- and server components are linked together and communicate with the Vconect platform. This deliverable targets in particular third parties interested in using or exploring the capabilities of the Vconect platform, such as service providers of a social network. After reading this document, a web developer should be able to assess the value and effort of integrating video conferencing into his existing web application when using the Vconect platform API.

Target audience

This document is public. It is intended for the technical community interested in video communication for networked communities. More specifically, it is targeted to 3rd party developers, who are interested in using the Vconect technology in the context of social networks.

Impressum

Full project title: Video Communication for Networked Communities

Title of the workpackage: WP2 Architecture and Platform Integration

Document title: D2.3 Open Reference Architecture and Deployment - Interim Specification

Editors: Nikolaus Färber, Fraunhofer IIS; Dennis Dams, Alcatel-Lucent Bell Labs

Workpackage Leader: Vilmos Zsombori, Goldsmiths, University of London

Project Co-ordinator: Peter Stollenmayer, Eurescom

Technical Project Leader: Marian Ursu, Goldsmiths

This project is co-funded by the European Union through the ICT programme under FP7.

Executive Summary

The Vconect vision is the adoption of high-quality videoconferencing as a medium for mass communication within communities.

To achieve this vision, Vconect is building a video communication platform which models and supports the complex communication topologies that characterise conversations between group members. The platform will take intelligent decisions to mediate the communication at the level of audio-visual choices, screen layout and network capabilities. This reasoning process, termed *Orchestration*, is the key research contribution of Vconect.

One important application is the integration of video conferencing into social networking services, for which an Application Programming Interface (API) is provided. After reading this document, a web developer should be able to access the value and effort of integrating video conferencing into his existing web application when using the Vconect platform API. The following table list all available API methods, which are explained briefly below.

Server-API (REST/JSON, base-URL http://vc-server.eu/ is hypothetical)	
`http://vc-server.eu/createSession`	create a session, get new sessionID
`http://vc-server.eu/sessionInfo`	get sessionIDs and userIDs of current sessions
`http://vc-server.eu/deleteSession`	delete a session, given sessionID
Client-API (JavaScript, include vConect.js in head of HTML doc)	
`vConect(divID,sessionID,userID)`	init VC Client plugin and embed in
`vConect.startClient()`	start VC Client plugin, join session
`vConect.setViewCleanCut()`	set view-mode to Clean-Cut
`vConect.setViewTiles()`	set view-mode to Tiles
`vConect.setViewStandard()`	set view-mode to Standard
`vConect.setMicrophoneOff()`	mute own mic, others cannot hear you
`vConect.setMicrophoneOn()`	un-mute mic, you can be heard again
`vConect.stopClient()`	stop VC Client, leave session

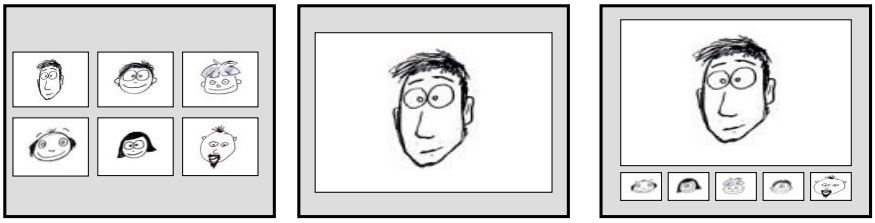
There are two platforms that have to be integrated with each other, namely the Social Network (SN) platform and the Vconect (VC) platform. Each of those is having server components and client components, such that four basic components are involved (SN-Client, SN-Server, VC-Client, VC-Server).

On the server side, the VC-Server offers a REST/JSON API to the SN-Server through which it can create and delete video conferencing sessions. When creating a session through the *createSession* request, the VC platform allocates all required resources in the back end and returns a *sessionID*, which is comparable to the number/PIN of a conventional phone conference. It is in the responsibility of the SN platform to distribute the *sessionID* to all clients who want to join this session.

On the client side, the starting point is the SN-Client, i.e. a web application running on a browser, which is typically implemented in HTML/CSS/JavaScript and communicates to the SN-Server via HTML/AJAX. Video conferencing is integrated into this web page by creating a element with an arbitrary *divID* and size. The VC-Client is implemented as a browser plugin, which is embedded into the by calling the JavaScript function *vConect(divID,sessionID, userID)*. The *userID* is a unique identifier for the user in the social network and is provided by the SN platform. Joining a video conference is then triggered through a call to *vConect.startClient()*. All API methods for the Client-API are defined in a JavaScript library Vconect.js which has to be included in the <head> of the HTML page. If the *createSession*, *vConect()* and *vConect.startClient()* are successful, then this is all that the web developer needs to do. I.e. at this point of time there should be live audio and video flowing between the participants who can immediately start their conversation.

The corresponding methods to leave a session and delete a session are *vConect.stopClient()* on the Client-API and *deleteSession* on the Server-API. Besides those and basic microphone control for muting (*vConect.setMicrophoneOff()*, *vConect.setMicrophoneOn()*), the only remaining API call to consider is the setting of view-modes.

A *view-mode* is a high level layout style, which describes the basic video composition and orchestration approach. As illustrated in the figure below, there are three view-modes which can be selected by the web developer and/or user. In view-mode *Tile*, all participants are displayed simultaneously in tiles of equal size. There is no orchestration active as all participants are always visible. In view-mode *Clean-Cut* only a single participant is shown on the screen at any point in time and his video is scaled as big as the given allows. The orchestration makes sure that the person who is currently talking is shown. The view-mode *Standard*, is a compromise between the above two options and very similar to Google+ (Google Plus) Hangouts. It shows the person who is currently talking in a larger video while still displaying the other participants in small videos ("thumb nails"). The transition during turn taking is animated. The corresponding methods are listed in the table above.



Tile Clean-Cut Standard

The described API is an interim specification and subject to change. However, the high level API as described above is expected to be relatively stable and mature. A lot of effort will be needed to make the underlying technology work robustly in difficult network conditions. But this development is happening "under the hood" and will not affect the API. Some extensions of the API are nevertheless planned and include *Additional view-modes*, *Sub-Grouping and Private Conversations*, *Security and Authentication*, *Access to User Profiles*, *Multiple Domains*, and *Text and Graphic Overlays*. It will be a challenging task of the Vconect management to determine the right priorities and set the roadmap for the second half of the project until end of 2014.

Perhaps the most significant extension of the 3rd party API would be to open up further components of the Vconect architecture. At this point in time, however, the definition of component boundaries and their APIs is still work in progress and too immature to be described with value to a 3rd party developer. Therefore we focus on the high-level API for now, which has reached a good level of maturity and stability. Later documents may also describe the component APIs and make the architecture thus more open.

List of Authors

Fraunhofer IIS
Niko Färber
Yaroslav Kryvyi

Goldsmiths, University of London
Vilmos Zsombori

SAPO, Portugal Telecom Group
Pedro Torres
Jorge Braz
João Abreu

Table of contents

1 Introduction

The Vconect vision is the adoption of high-quality videoconferencing as a medium for mass communication within communities.

To achieve this vision, Vconect is building a video communication platform which models and supports the complex communication topologies that characterise conversations between group members. The platform will take intelligent decisions to mediate the communication at the level of audio-visual choices, screen layout and network capabilities. One important application is the integration of video conferencing into social networking services.

Since the start of the Vconect project there has been a remarkable increase in interest in this topic, most prominently visible through Google+ Hangouts and WebRTC. Though it is difficult for a European research project to compete on all grounds with such global industry efforts, there are many white spaces where value can be added. WebRTC, for example, is not supporting group communication very well while Hangouts cannot be integrated in other social networks easily. Therefore, Vconect tries to provide an open web interface similar to WebRTC while supporting video conferencing as efficiently as Hangouts. Furthermore, Vconect performs fundamental research on how to improve the user experience for dynamic communication structures and how to make more efficient use of network resources. Though those basic research questions are investigated on the Vconect platform they are applicable to other video communication platforms as well.

1.1 Objectives of the Deliverable

This deliverable publishes an interim API specification of the Vconect platform for 3rd party developers. The platform is described from the perspective of a black box: the functions and features which are currently available are described and the Application Programming Interface (API) is specified and illustrated with code examples. It is also described how client- and server components are linked together and communicate with the Vconect platform. This deliverable targets in particular third parties interested in using or exploring the capabilities of the Vconect platform, such as service providers of a social network. After reading this document, a web developer should be able to assess the value and effort of integrating video conferencing into his existing web application when using the Vconect platform API.

It should be noted that the Vconect platform is designed as an open architecture. This means that individual components within the platform are accessible and can be exchanged if desired. This in turn requires us to also describing the APIs of each component in detail. At this point in time, however, the definition of component boundaries and their APIs is still work in progress and too immature to be described with value to a 3rd party developer. Therefore we focus on the top-level API in this document, which has reached a good level of matureness and stability. Later documents may also describe the component APIs and make the architecture truly open.

1.2 Structure of the Deliverable

The main content of this deliverable is included in Section 3, in which we describe the actual API and provide code examples to illustrate its usage. Before that, however, we first describe the Vconect components and architecture in Section 2 as background information. This section provides a look inside the black bock and what it does “under the hood”. Web developers who are just interested in what they can do with the API can jump to Section 3. Finally, Section 4 describes future extensions that are currently on our roadmap.

2 Vconect Components and Architecture

This section provides an overview of the Vconect components and architecture. Most of the information contained in this section is not essential for a 3rd party developer who sees the Vconect platform as a black-box and is only interested in what it can do rather than how it does it. However, it is always helpful to understand the underlying functionality in order to better understand (and appreciate) the external API. Hence, we provide some background information in the following. Web developers who are eager to get their hands on the API may jump to Section 3.

The Vconect platform follows a client-server architecture which is illustrated in Fig. 1. The client components are visible on the left hand side, and server components on the right. The functionality is grouped into four layers, working up from a basic stream level (content layer) to higher-level control logic (reasoning layer). The social network is shown as a grey box to the right.

Vconect envisages two different client configurations: a light-weight *Home-Client* for use by individuals, and a multi-camera *Studio-Client* which can be used by groups of performers on stage or in a theatre. Each client configuration shares the same server-side components. In this document we only address the Home Client, and therefore refer to it simply as the Vconect Client.

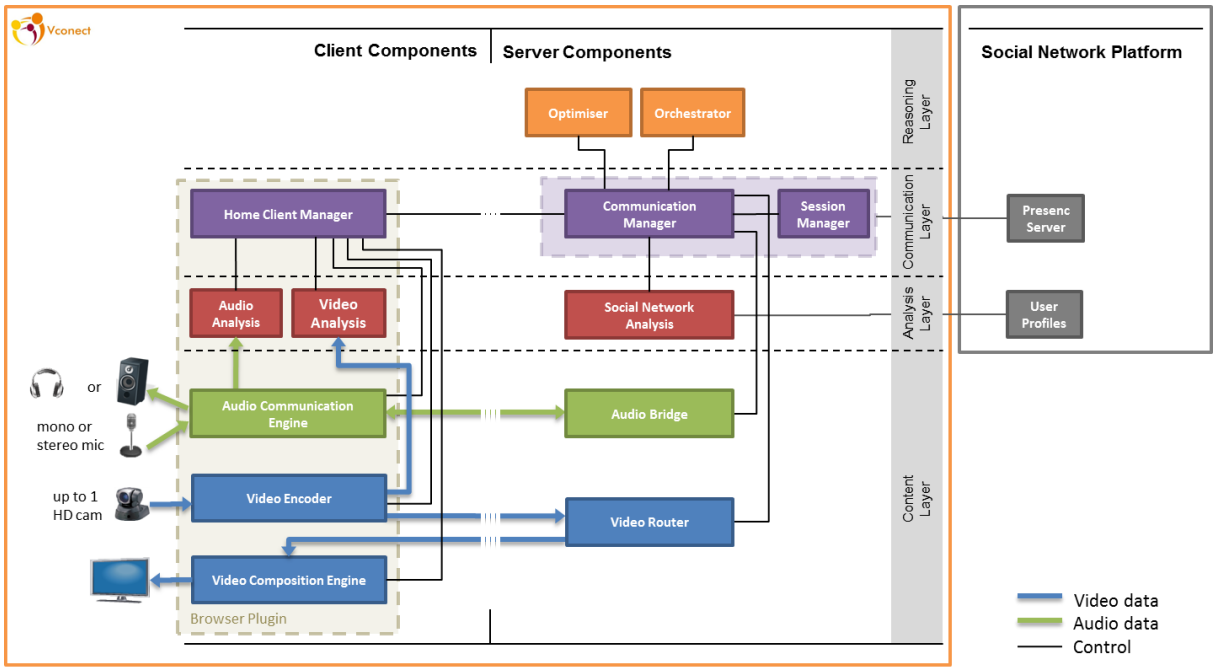


Figure 1: Overview of Vconect Architecture

2.1 Client Components

At the Content Layer, the client side of the platform includes components for capturing, encoding and transmitting video streams from a high definition web cam and audio streams from a microphone. These components are largely equivalent to those found in conventional video conferencing systems. Encoding is based on H.264 video and AAC-ELD [6] audio and IP encapsulation is based on RTP. Video is encoded in multiple resolutions and bit rates, such that adaptation to the network and client resources becomes possible. The Audio Communication Engine (ACE) is a VoIP engine handling AAC-ELD coding, IP streaming, and echo control in a single module [7]. This layer also incorporates

the Video Composition Engine (VCE) which receives, decodes and composes multiple video streams for presentation to the user.

Moving up to the Analysis Layer, the Vconect client includes components for the automated analysis of the captured audio and video streams, from which cues can be generated as an input to the Orchestrator. As a result, the Vconect platform can e.g. detect which participant is currently talking and show his video enlarged.

A central Client Manager component is provided to coordinate the distribution of messages to all the client components, and provide necessary coordination and control functions.

All of the above components and functionality is integrated into a browser plugin, such that integration into a social network becomes easily possible.

2.2 Server Components

At the Content Layer, two main components provide scalable transmission of audio-visual streams: the Multipoint Control Unit for the ACE (ACE-MCU) and the Video Router. The ACE-MCU is an audio bridge, which decodes and mixes multiple audio streams and distributes them to all clients in a Vconect session after re-encoding. The Video Router is an efficient packet switch and replicator which connects multiple source video streams to multiple client targets. It also measures the quality of the network through RTCP and can be cascaded to form an overlay network. Note that the Video Router does not perform any H.264 decoding or re-encoding and has therefore very low complexity. A Media Server can also be connected to the Video Router in order to inject streams of additional pre-recorded video, or to record and play back any captured stream.

At the Analysis layer, the profiles from a social network can be accessed and analysed in the Social Network Analysis. Specific knowledge about users' interests, social relationships, and communication characteristics are extracted in order to support better decisions about the representation of group communication. For example, the Orchestration can make sure that two users who are good friends are displayed in a preferred way.

The Communication Layer includes the Communication Manager and the Session Manager. Together, these provide the hub of the communication framework in Vconect, enabling messages to be transmitted between components, and enabling users to find each other and join a Vconect session. While each videoconference session has its own instance of a Communication Manager, the Session Manager is the central entry point for starting new sessions.

2.3 Orchestration

Finally, the Reasoning Layer contains the Optimiser and the Orchestrator. Though they are "normal" server components, they are treated in this special subsection because they are unique to the Vconect platform and relate to the central research question in the Vconect project. They work together to interpret cues from the lower-level components and generate instructions which control how audiovisual streams are presented at each Vconect client in order to make optimal use of the network at lowest cost, while maximising the Quality of Experience for users. In other words, they represent the brain of the system. The term Orchestration is used to describe a functionality that is also referred to as automatic floor control. For example, assuming a conference with more than 10 participants, the Orchestration decides who is shown and at what size. Participants who are only listening may not be shown at all, while the dominant talker is shown in a larger video. This is something already known from conventional video conferencing systems and Hangouts, but we believe that there is still a lot to learn and improve in this area.

After this short overview of the Vconect components and architecture we now close the black-box and focus on what can be done with it.

3 Application Programming Interface

This section describes the Application Programming Interface (API) from the perspective of a web developer who wants to integrate group video communication into his web site. Without loss of generality, we assume that this web site implements a Social Network (SN) with the required server infrastructure. The goal is to start a videoconference in a similar way as creating an image using the HTML <img> tag. But instead of seeing a JPG image on the web site, the user will see a live videoconference and be able to participate in the conversation. The API is based on JavaScript and REST/JSON and described in detail using code examples.

3.1 Overview

Before the relevant interfaces are covered in detail, it is important to get an understanding of the overall architecture and message flow between the components. This also serves the purpose of defining the terms that will be used in the following.

As illustrated in Fig. 2, there are four basic components, which work together when establishing a video conference in a social network.

1. **SN Client**
   The Social Network Client is what the user sees and experiences when being on the social network. It is the web application running on the browser of the user and is typically implemented using HTML/CSS/JS. We assume that there is an existing web application that shall be extended with videoconferencing.
2. **SN Server**
   The Social Network Server comprises all server components of the social network. This is where e.g. all user data is stored (pictures, messages, profiles) and the HTML content is served from. While the user is logged on the social network, the SN Client is talking to the SN Server through the SN-API, e.g. using HTTP and AJAX. This interface is out of scope for the Vconect platform but needs to provide some basic functionality, such as exchanging user identities.
3. **VC Client**
   The Vconect Client is responsible for the actual video conference and is therefore transmitting and receiving live audio and video streams. It can be controlled from the SN Client with JavaScript through the Client API (VC-API). Multiple videos from multiple participants are composed and rendered into a rectangular area ( ) of the SN Client. The VC Client is implemented as a browser plugin based on the Firebreath framework [3].
4. **VC Server**
   The Vconect Server comprises all server components of the Vconect platform, including video routers, audio bridges, and session management etc. All communication between the Vconect Client and Server is aggregated in the VC-API. This includes all media streams and control messages. In addition, the VC Server and SN Server also talk directly to each other, e.g. to create new sessions. This is done via the Server API (S-API), which is implemented in REST/JSON.

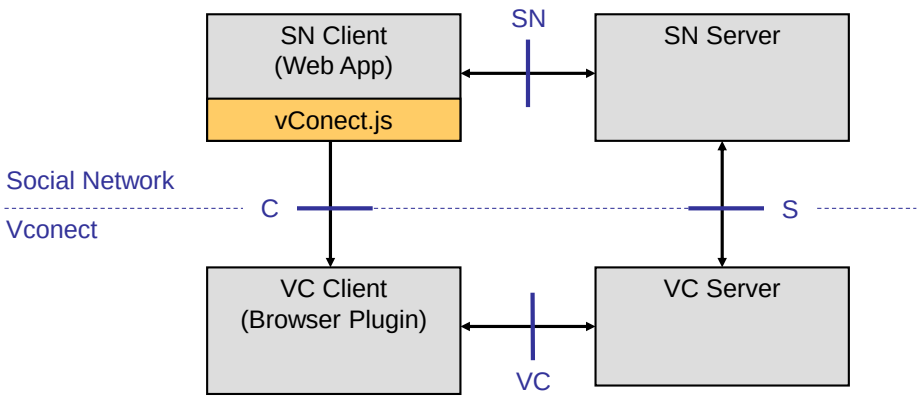


Figure 2: Components and Interfaces between the Social Network (SN) and the Vconect platform (VC).

In order to integrate Vconect into the social network, there are two interfaces that have to be considered, i.e. the Client API and Server API. The social network platform will have to use both in concert to establish a videoconference.

1. **Client API (C)**
   The Client API (C-API) is a JavaScript interface that controls the browser plugin, which is implemented using the Firebreath framework [3]. The SN Client loads the JavaScript library vConect.js from the Vconect Server which provides a convenient API to the web developer. It can be seen as glue code or wrapper code hiding some of the complexity in embedding the plugin and communicating with it.
2. **Server API (S)**
   The two server systems communicate directly to each other through the Server API (S-API). For example, the SN Server will ask the VC Server to create a new session, i.e. a new video conference. This interface is using REST/JSON as a communication protocol. Currently it is focused on session creation/deletion and can provide information on running sessions and their participants.

The detailed message flow of the VC-API and SN-API is out of scope for this document. On the one hand, the implementation of the SN-API is up to the social network platform and should not be of concern to the Vconect platform. We simply assume that the relevant information can be exchanged, e.g. using AJAX. On the other hand, the details of the VC-API are hidden from the web developer. He does not have to care how media is routed and which control messages are transmitted in the VC back end.

Table 1 lists all methods on the Client-API and Server-API with a short description and a reference to the section in which it is described in more detail. In the following sections we will go through the individual steps that are needed to start and control a videoconference. Those are discussed roughly in chronological order.

Table 1: Overview of API

API Method	Description	Section
Server-API (REST/JSON, base-URL http://vc-server.eu/ is hypothetical)		
`http://vc-server.eu/createSession`	create a session, get new sessionID	3.2
`http://vc-server.eu/sessionInfo`	get sessionIDs and userIDs of current sessions	3.6
`http://vc-server.eu/deleteSession`	delete a session, given sessionID	3.7
Client-API (JavaScript, include vConect.js in head of HTML doc)		
`vConect(divID,sessionID,userID)`	init VC Client plugin and embed in	3.3
`vConect.startClient()`	start VC Client plugin, join session	3.3
`vConect.setViewCleanCut()`	set view-mode to Clean-Cut	3.4
`vConect.setViewTiles()`	set view-mode to Tiles	3.4
`vConect.setViewStandard()`	set view-mode to Standard	3.4
`vConect.setMicrophoneOff()`	mute own mic, others cannot hear you	3.5
`vConect.setMicrophoneOn()`	un-mute mic, you can be heard again	3.5
`vConect.stopClient()`	stop VC Client, leave session	3.6

3.2 Creating a Session

The first thing that needs to be done before a video conference can be started in the browser is to create a session via the S-API. In the following we use the term *session* to refer to a video conference in Vconect, i.e. the time period during which audio and video is flowing between clients. Note that there is also a session running in parallel on the social network, which typically lasts longer.

There are several actions that the VC Platform has to take before a session can be started. It must launch several server components, such as the audio bridge(s) and video router(s). In addition, each session has its dedicated session manager which controls all components for one session. All these steps are triggered with a REST/JSON request to the VC Server on the S-API. In the following we assume that the VC Server is reachable at http://vc-server.eu, in which case the request looks like:

```
http://vc-server.eu/createSession
```

There are two possible HTTP responses to this request. In case of success, the VC Server responds with a unique sessionID, such as:

```
{
    "message" : "createSession",
    "sessionId" : "a431de84014aceb1a4d7aeccb11",
    "status" : "successful"
};
```

This sessionID is required by all VC Clients who want to join this session (see Section 3.3). It is comparable to the phone number and PIN of a conventional conference bridge, which the participants need to know to dial in. It is in the responsibility of the SN platform to distribute the sessionID to all clients and maybe associate other metadata with it (e.g. who created the session or what the topic of the session is about). Such metadata is out of scope for the Vconect platform.

If the VC Server is not able to create the session, e.g. because of resource unavailability, it then replies with a failure status, e.g.

```
{
    "message" : "createSession",
    "error" : "out of resources",
    "status" : "failed"
};
```

Note that the following rules are consistent for all REST/JSON responses: First, the name of the request is returned in the "message" field. Second, the "status" field contains either "successful" or "failed". In case of a failed request, the "error" field includes a human-readable message.

In the following we assume that the session with ID `a431de84014aceb1a4d7aeccb11` is created successfully.

If the SN Server has its own internal mechanism of generating unique sessionIDs, that is easily distributed to SN Clients (maybe because this functionality is already implemented within the SN infrastructure), these sessionIDs can be used for creating Vconect communication sessions, passing the sessionID to the createSession call as a GET parameter:

```
http://vc-server.eu/createSession?sessionId=SN-specific-unique-sessionID
```

The VC Server double-checks whether there is already a session created with the required sessionID, according to which the subsequent response may be one of the following two (similar to above):

```
{
    "message" : "createSession",
    "sessionId" : "SN-specific-unique-sessionID",
    "status" : "successful"
};
```

in case of successful session creation, or

```
{
    "message" : "createSession",
    "error" : "Session ID [SN-specific-unique-sessionID] is already in use.",
    "status" : "failed"
};
```

in case of failure.

3.3 Joining a Session

In order to start a videoconference from a web page, the vConect.js JavaScript library first needs to be included in the `<head>` tag of the HTML document:

```
<script src="http://vc-server.eu/vConect.js"></script>
```

In addition, the area in which the video will be displayed must be created in the `<body>` of the HTML document using a `<div>` tag:

```
<div id="videoWindow" width=600px height=400px>
```

The ID is needed in the next step to reference the for embedding the plugin. In the following we use “videoWindow” but the web developer is free to select a name. The width and height of the video area can be set as desired and changed dynamically later.

Next, the browser plugin needs to be embedded in the video window and initialized, which is accomplished through a call to the vConect.js libraray:

```
<script>
  // unique sessionID obtained from VC Server
  // (typ. forwarded via SN Server to SN Client)
  var sessionID = "a431de84014aceb1a4d7aeccb11";
  // unique userID obtained from SN Server
  var userID = "alan";
  vConect("videoWindow", sessionID, userID);
</script>
```

Note that the first parameter to vConect() is the ID of the video window and the second parameter is the sessionID as obtained from the REST/JSON call, see Section 3.2. The userID is a unique identifier for the user in the social network. It is in the responsibility of the SN platform to maintain and obtain those. The IDs will typically be hash strings similar to the sessionID, but for simplicity we use `alan` in this example.

As a final step, the plugin needs to be started, which is accomplished by:

```
<script>
  vConect.startClient();
</script>
```

This function is also defined in the vConect.js library. At this point in time, audio and video should be available and the user can join the conversation.

The call to vConect.startClient() will fail if there is no session with the specified sessionID or if the maximum number of clients per session is reached. Currently, up to six participants are supported, which, however, is only a preliminary limitation. If a user is joining a session while he is still in another session, he will automatically be removed from the old session and added to the new session.

Note that the Vconect platform does not manage the process of joining a session, i.e. who is allowed to join a certain session and how this process is handled from a user interface aspect. The design of this process and the UI are up to the SN platform. Only after this process is completed and the SN platform has decided to add a user to a session, the vConect.startClient() method shall be called.

3.4 Controlling the Screen Layout

In order to control the design and look and feel of the web site, it is important to understand how the videoconference is displayed on the screen. In this section we therefore describe how the videos from the remote participants are rendered to the browser window of the SN Client.

As a general rule, all output from the VC Client is constrained to the , i.e. the plugin will not render any graphic or text elements outside the . In addition, the is showing only the live videos. Any UI elements, such as buttons or text have to be added by the SN Client outside the . This is illustrated in Fig. 3 using the social network SAPO Campus from Portugal Telecom Group as an example [4]. The area of the is marked in red, while the UI elements are placed below.

As can be seen from Fig. 3, there are multiple videos from the remote participants included in the area of the . However, the layout and composition of those is in the responsibility of the Vconect platform and the web developer does not have to worry about it. This also includes the orchestration,

i.e. the decision to enlarge the video of a particular participant when he is talking (very similar to Google+ Hangouts). The paradigm followed here by Vconect is to provide the best user experience but release the developer from the burden to implement the details. In other words, the web developer only provides a rectangular area while the Vconect platform will fill it in an optimal way.

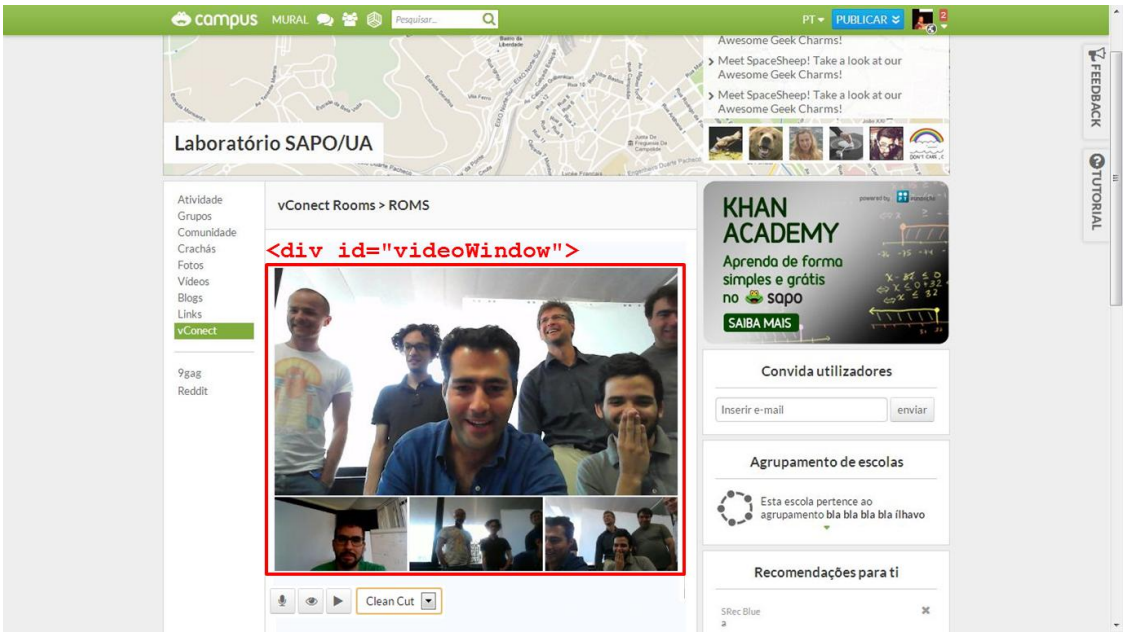


Figure 3: Screen layout of social network client with the including the Vconect client marked in red. Screen shot from social network of SAPO Campus used as example.

View-modes

The VC Client supports three *view-modes* that can be selected by the web developer. A view-mode is a high level layout style, which describes the basic video composition and orchestration approach. The details are again hidden from the web developer for his convenience, though later releases may also include a low-level API with fine grained control over the size, position, and stacking order of the videos. The three view-modes are illustrated in Fig. 4 and described briefly in the following.

1. **Tile**
   In view-mode Tile, all participants are displayed simultaneously in tiles of equal size. There is no orchestration active;all participants are always shown at the same size.

2. **Clean-Cut**
   In view-mode Clean-Cut only a single participant is shown on the screen at any point in time and his video is scaled as big as the given area allows. The orchestration makes sure that the person who is currently talking is shown big.

3. **Standard**
   The view-mode Standard, is a compromise between the above two options. It shows the person who is currently talking in a larger video while still displaying the other participants in small videos (“thumb nails”). The transition during turn taking is animated. This view-mode is very similar to Google+ Hangouts.

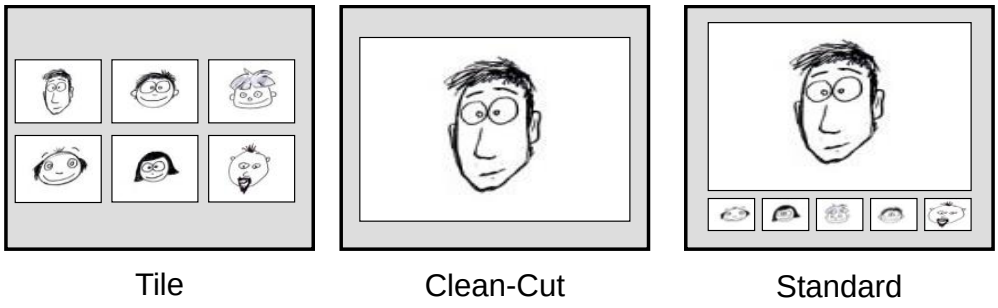


Figure 4: Illustration of view-modes for 6 participants

The view-modes support different communication needs in different ways and are to some extent a personal preference. The Tile view-mode, for example, allows always seeing all participants and therefore provides a good perception of who is in the group. However, as the group size grows, the individual videos become small and the facial expressions may not be visible very well. This is where the Clean-Cut view-mode has it strength, i.e. it allows seeing every detail of the talking person. The Standard view-mode is a good compromise between the two communication needs and therefore intended as the standard or default view-mode.

The view-modes can be changed dynamically during the session. It is up to the web developer to either make the choice available to the user through a UI or decide on the view-mode on behalf of the user, e.g. because the developer believes that a given application is best supported by e.g. using tiles. The following JavaScript code illustrates how to set the view-mode.

```
<script>
  vConect.setViewCleanCut(); // set view-mode to clean-cut
  vConect.setViewTiles();    // set view-mode to tiles
  vConect.setViewStandard(); // set view-mode to standard
</script>
```

Scaling and Resizing

The size of the including the Vconect plugin can be changed dynamically during the session and the plugin will be scaled accordingly. It may take 1-2 seconds until the plugin detects the new size of the enclosing and therefore the scaling is not completely smooth. Hence, it is not recommended to use resizing as an animation effect or interactively but rather for changing the size in bigger steps and more permanently, e.g. to change from/to full-screen (in the sense that the Vconect plugin covers a major part of the browser window).

Because the composition of the videos from individual participants is controlled by the VC Client, the resulting layout may not fit exactly into the aspect ratio of the in which it is embedded. In this case, the layout is scaled to be as big as possible given the constraints from the and placed in the centre. The remaining area is filled with a background colour as illustrated in Fig. 5.

At this point it is worth noting that all videos are captured and displayed in an aspect ratio of 3:2 (width:height). Hence, if a single video is displayed in Clean-Cut mode and shall fill the completely, the should also have an aspect ratio of 3:2. In the view-modes Tile and Standard, the layout gets more complicated, but in all cases the aspect ratio of the participant videos is 3:2. The video is captured from the web cam in a native resolution of 1080:720p. Hence, if sufficient bit rate is available the VC Client provides HD quality. However, it will reduce the video bit rate and video resolution if the system is lacking channel or CPU resources.

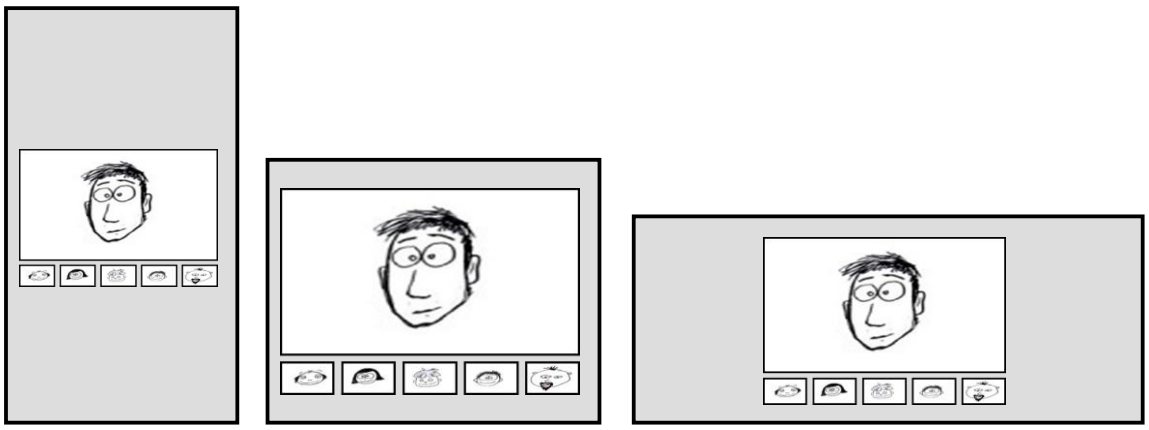


Figure 5: Placement of composed videos into embedded for different aspect ratios

3.5 Controlling Audio

Compared to video, there is less to explain with respect to audio. Audio is transmitted in Full-HD Voice quality [5] based on the AAC Enhanced Low Delay codec (AAC-ELD, [6]). This codec supports the full audio bandwidth of 20 kHz and therefore provides CD-like quality while also supporting stereo. Within a session, all participants can hear each other on equal terms, i.e. there is currently no audio orchestration supported.

The only control option for audio is to mute the own microphone, e.g. when there is too much noise in the background or when the participant needs to quickly talk to someone in his room. Muting can be controlled on the Client API using

```
<script>
  vConect.setMicrophoneOff(); // mute own mic
  vConect.setMicrophoneOn();  // unmute own mic
</script>
```

Note that this method can also be used by the SN platform without direct interaction from the user. For example, one participant may have the role of a session administrator or host and is able control the microphone of all participants if they misbehave. Such logic, however, is again in the responsibility of the SN platform.

Future releases will provide more control on audio. For example, this will allow controlling the stereo image of the audio (who is heard left/right) or intermediate private conversations between two participants in the session. Such audio orchestration is for further study and currently not exposed on the API.

3.6 Getting Information about a Session

The VC Server keeps track of all running sessions and knows which participants are in each session. This information is typically needed to provide a *session directory* to the user. Note that the SN platform can theoretically keep track of this state information on its own, as it does initiate all events that create sessions and join users. However, to avoid a potential mismatch, this information can be obtained from the VC Server on the S-API. To get a list of all on going sessions, a request to

```
http://vc-server.eu/sessionInfo
```

will return a JSON encoded response, such as

```
{
    "message" : "sessionInfo",
    "sessions" : [
       "a431de84014aceb1a4d7aeccb11",
       "a443a5234e9a893f0cd3979e2be",
       "49549644cfdb2f41a6c79e0308c"
    ],
    a431de84014aceb1a4d7aeccb11: { users: ["alice","bob"] },
    a443a5234e9a893f0cd3979e2be: { users: ["chris","david","emily"] },
    49549644cfdb2f41a6c79e0308c: { users: [ ] },
    "status" : "successful"
};
```

which is listing three sessions and for each one, the users which have joined (one session is empty). As noted earlier, the userIDs are typically more cryptic and do not show the user name as clear text (though they could, as long as they are unique in the system). However, the SN platform can associate user names and other profile information to the userID independently from the Vconect platform.

If the SN Server is only interested in a particular session, it can add the sessionID as a GET parameter to the request, e.g.

```
http://vc-server.eu/sessionInfo?sessionId=a443a5234e9a893f0cd3979e2be
```

In which case it will get a list of users as in the following example response:

```
{
    "message" : "sessionInfo",
    "users" : ["chris","david","emily"],
    "status" : "successful"
};
```

If the sessionID does not exist the request will fail, resulting in:

```
{
    "message" : "sessionInfo",
    "erros" : "session does not exist",
    "status" : "failed"
};
```

Using the above requests, the SN Server can get an overview of the current sessions and status of users. It can then forward this information to the SN Client, which can display it to the user.

3.7 Leaving and Deleting a Session

Though creating and joining a session is typically more exciting, it is also necessary to leave a session and delete the complete session. Otherwise, the resources of the VC platform would never be released and eventually block any new session.

On the one hand, the SN Client can stop the plugin, e.g. when a user decides to leave the current discussion or when the conference is over. This can be done via the C-API using another method of the vConect.js library:

```
<script>
  vConect.stopClient();
</script>
```

If there are no more clients in a session then it can be deleted completely from the VC Server. This is done on the S-API using the following request:

```
http://vc-server.eu/deleteSession?sessionId=a443a5234e9a893f0cd3979e2be
```

which will release all resources that were required for the session with the given ID. More specifically, this will close down the audio bridge(s), the video router(s) and the internal Vconect components responsible for automatic editing (the Orchestrator) and network optimisation.

The response can be either:

```
{
    "message" : "deleteSession",
    "status" : "successful"
};
```

if the session has successfully been deleted, or:

```
{
    "message" : "deleteSession",
    "error" : "Session [s0] doesn't exist, therefore it can't be deleted.",
    "status" : "failed"
};
```

It is important to note that the deletion of sessions is enforced by the VC Server even when there are participants still in the session. It is therefore in the responsibility of the SN platform to assure that certain precautions are made before this call is requested. For example, it can assure by extra logic that a session can only be deleted if it is empty, or it can only be deleted by the user who initiated the session. However, such checking of preconditions and logic is up the SN platform.

Because the Vconect platform cannot rely on clients to properly close a session, it will look for clients that are inactive and remove those automatically from a session after a timeout. For example, if the browser is closed or the OS crashes, then the Vconect platform will not receive proper stopClient() calls. Nevertheless, it has to release the resources after a while. The same is true for sessions that do not have any participants for a longer period of time. How this “garbage collection” is implemented in detail is out of scope for the SN platform and does not affect the API. However, because the resources are released more quickly and more controlled when this is done explicitly through stopClient() and deleteSession, it is in the interest of the SN platform to do so.

4 Future Extensions

As the name of this document implies, the presented API only describes an interim status. More methods for 3rd party developers will be added until the end of the project and hopefully also beyond the life time of the project. Despite being an intermediate snapshot, it is worth noting that the API is complete and tested. We therefore expect that it is fairly stable and can be extended in a backwards compatible way. Nevertheless there are several extensions on our roadmap, which will be described in this section.

Perhaps the most significant extension of the 3rd party API would be to open up further components of the Vconect architecture. The current API as exposed on the Vconect front-end (C-API, S-API) is actually only a very small part of the overall protocols defined and developed in the Vconect project. Currently all those other APIs are hidden from the web-developer in the Vconect back-end. In order to work towards an open architecture, more of the internal components must be made accessible. For example, if the API towards the Orchestrator would be open, then a SN platform could replace it and use its own implementation of the Orchestrator. Basically, the Orchestrator becomes a plugin to the Vconect platform which can be controlled by the service provider. The same is true for e.g. the Video Router, Optimizer, and also the VClient. However, because those internal APIs are currently still under development and the module boundaries are not yet stable we decided not to cover those in this interim specification. The final specification may do so and discuss APIs to individual Vconect components.

As a final remark before discussing some future extensions, we note that it is well possible that there will indeed be very little change to the described API. This is because it is a high-level API which is hiding a lot of internal complexity. A lot of effort will be needed to make the current functionality work robustly over the Internet. This includes NAT-traversal, bandwidth estimation, adaptive streaming, network monitoring, routing between several Video Routers building an overlay network, failover, or garbage collection. Though the API is tested for a best case network scenario, there is much work needed to make that functionality available robustly and for a heterogeneous client- and network environment. Depending on resources, those may have priority over other “nice to have” extensions.

Though there are many extensions in the back-end which need to be worked on, there are much fewer changes that affect the 3rd party API for web developers. Those include:

1. **Additional view-modes**
   It is likely that through discussion with 3rd party developers we will see the need for additional view-modes and a more flexible control of the layout. This may include a fine grain control of individual videos with respect to their position, size, and stacking order.

2. **Sub-Grouping and Private Conversations**
   In a conventional group conversation with more than 6 participants it often happens that sub-groups emerge or two people have an intermediate private conversation. During those parallel tracks the other people are still audible and one can react to events outside the own conversation, e.g. when the others are laughing very hard. The API may support this by e.g. selecting one of the participants for a private conversation.

3. **Security and Authentication**
   The current API is completely open, which means that everybody can e.g. delete a session. The only thing he needs is the base-URL. This is of course not suitable for a commercial deployment or trial and needs consideration.

4. **Access to User Profiles**
   This extension is related to the Server API and will allow the VC Server to obtain profile information for a given user of the social network. This information is exploited by the

Orchestrator to improve the video communication experience. For example, if the Orchestrator knows that one participant is a teacher, while the others are students, then it can modify the view-mode to fit the needs of special roles. For example, the teacher is always shown big, while the students are shown small.

5. **Multiple Domains**
   Currently all social networks who want to use the Vconect platform will access it through a central entry point, which is a message broker (using activeMQ). Currently, the URL of this broker is hard coded in the VC Client. In future it should be a parameter that can be set during initialization (similar as a VoIP client registers to a SIP server).

6. **Text and Graphic Overlays**
   The element is currently limited to displaying live videos. There are several use cases in which a basic text or graphic overlay would make sense. For example, it would be nice to overlay the name of a user onto his video or draw a border around the video to highlight the participant when speaking. Another use of such functionality would be to display a profile picture in case that the connection is failing or the participant has joined in audio-only mode.

Obviously, this list can be extended and is only limited by the imagination of the web developer and enthusiasm of Vconect engineers. It will be a challenging task of the Vconect management to determine the right priorities and set the roadmap for the second half of the project until end of 2014.

References

[1] http://www.eurescom.de/

[2] Home page of Apache activeMQ project
http://activemq.apache.org/

[3] Home page of the FireBreath project
http://www.firebreath.org

[4] Home page of SAPO Campus
http://campus.sapo.pt/

[5] Home page of Full-HD Voice project
http://www.full-hd-voice.com/

[6] Information on AAC-ELD
http://www.iis.fraunhofer.de/en/bf/amm/produkte/audiocodec/audiocodecs/aaceld.html

[7] White paper on the Audio Communication Engine (ACE), available at
http://www.iis.fraunhofer.de/en/bf/amm/download/produktbr.html

[8] Home page of the Vconect project
http://www.vconect-project.eu/