

Universal multimedia framework for online videoconferencing

Radical Chat

Jaromir Sivic

Bachelor's Thesis
November 2009

Degree Programme in Information Technology
School of Technology



JYVÄSKYLÄN AMMATTIKORKEAKOULU
JAMK UNIVERSITY OF APPLIED SCIENCES



Author(s) SIVIC, Jaromir	Type of publication Bachelor's Thesis	Date 7.12.2009
	Pages	Language English
	Confidential () Until	Permission for web publication GRANTED
Title UNIVERSAL MULTIMEDIA FRAMEWORK FOR ONLINE VIDEOCONFERENCING		
Degree Programme Information Technology		
Tutor(s) MANNINEN, Pasi		
Assigned by LÄHDE, Ville - Radical Pictures oy		
Abstract <p>This thesis describes the implementation of Universal multimedia framework for online videoconferencing.</p> <p>The final result should present the first real suitable solution for managing and mastering live audio-video communication over RTMP protocol (used by Adobe Flash) from such programming languages as PHP or ASP.NET (C#). Multimedia videoconferencing framework is designed and implemented by universal way to enable almost any platform and programming language to use it. Officially supported technologies are PHP and ASP.NET. Integration and cooperation of all components together in this highly heterogeneous environment are supported by Web Services technology, SOAP protocol and Real Time Messaging Protocol.</p> <p>This solution allows (PHP,ASP.NET) developers to create live Flash multimedia applications without any knowledge about Flash, ActionScript 3.0, RTMP protocol or media server API written in Java. It extremely speeds up the process of development and deployment of final applications with an increased emphasis on performance of the system. Two main programming languages (Actionscript 3, Java) are used for framework implementation and another two (PHP and ASP.Net – C#) for demonstration, how to write applications in this universal multimedia videoconferencing framework. The client or user of this framework may be anyone who has an internet-capable device with installed Adobe Flash Player 10 compatible plug-in.</p>		
Keywords Adobe Flash Player, ActionScript 3, Java, Red5, Wowza media server, RTMP, PHP, Flash Chat.		
Miscellaneous		

CONTENTS

1. INTRODUCTION	6
1.1 Thesis background.....	6
1.2 Motivation.....	6
1.3 Radical Pictures OY.....	7
1.4 Structure of the thesis.....	7
2 SELECTION OF TOOLS AND TECHNOLOGIES.....	8
2.1 Who are the users of the framework?	8
2.2 Selection of the proper technology.....	9
2.2.1 Technologies overview	9
2.2.2 Java platform.....	9
2.2.3 .NET platform & Silverlight	10
2.2.4 Adobe Flash platform	10
2.2.5 Final decision.....	11
2.2.6 Wowza media server Pro.....	11
3 SERVER SIDE DESIGN AND IMPLEMENTATION.....	12
3.1 Three-tier architecture.....	12
3.2 System Logic.....	13
3.3 List of events.....	16
3.3.1 Global events.....	16
3.3.2 Application events.....	17
3.3.3 Logging in/out events.....	17
3.3.4 AV stream handling events.....	18
3.3.5 Client related events	19
3.3.6 Chat room related events.....	20
3.3.7 Universal events	20
3.4 Parallel, asynchronous, event-driven programming.....	21
3.5 The problems of parallel data processing.	22
4 CLIENT SIDE DESIGN AND IMPLEMENTATION.....	26
4.1 Radical Flash Chat.....	26
4.2 XML configuration file	26
4.3 Static skin.....	26
4.4 RPSF positioning system.....	29
4.5 Strict rules for RPSF positioning system.....	31
4.6 RPSF implementation.	32

4.7	SkinObject class extensions.....	32
4.7.1	Universal components.....	32
4.7.2	BasicPanel.....	33
4.7.3	BasicButton.....	35
4.7.4	HtmlTextArea	38
4.7.5	Specialized components.....	39
4.7.6	ListOfUsers.....	40
4.7.7	ChatText.....	40
4.7.8	ChatInput.....	42
4.8	Dynamic skinning.....	43
4.8.1	Introduction to dynamic skinning.....	43
4.8.2	Implementation of dynamic skinning.....	44
4.8.3	Three levels of synchronization.....	45
4.8.4	Threats and limits of dynamic skinning.....	46
5	CONCLUSION	47
5.1	Results	47
5.2	Personal experience	47
5.3	Further improvements	48
6	REFERENCES	49

FIGURES

FIGURE 1.	Technologies.....	8
FIGURE 2.	Radical Chat architecture	12
FIGURE 3.	Hierarchy	14
FIGURE 4.	Radical Chat deployment.....	15
FIGURE 5.	Radical Chat basic interactions.....	16
FIGURE 6.	onCheckClients	19
FIGURE 7.	Sequence of events	20
FIGURE 8.	Parallel data processing problems	24
FIGURE 9.	Layers.....	27
FIGURE 10.	RPSF positioning system.....	30
FIGURE 11.	RPSF dependency graph	31
FIGURE 12.	Universal components.....	33
FIGURE 13.	NineGrid.....	34
FIGURE 14.	Specialized components	39

Terminology

Multimedia framework

A multimedia framework (MMF) is a software framework that handles media on a computer and through a network. A good multimedia framework offers an intuitive API and a modular architecture to easily add and support a lot of codecs, container formats and transmission protocols. It is meant to be used by applications such as media players and audio or video editors, but can also be used to build Videoconferencing applications, media converters and other multimedia tools. MMF is available for different operating systems.

(http://en.wikipedia.org/wiki/Multimedia_framework, November 2009)

Rich Internet Applications

(RIAs) are web applications that have most of the characteristics of desktop applications, typically delivered by way of standards based web browser plug-ins or independently via sandboxes or virtual machines. Examples of RIA frameworks include Curl, GWT, **Adobe Flash/Adobe Flex/AIR**, Java/JavaFX, Mozilla's XUL and Microsoft Silverlight. The term was introduced in March 2002 by vendors like Macromedia who were addressing limitations at the time in the "richness of the application interfaces, media and content, and the overall sophistication of the solutions" by introducing proprietary extensions." Key characteristics: Accessibility, Advanced communications, Complexity, Consistency, Installation and Maintenance, Security, Performance, Richness.

(http://en.wikipedia.org/wiki/Rich_Internet_application, September 2009)

Adobe Flash

Adobe Flash (formerly Macromedia Flash) is a multimedia platform originally acquired by Macromedia and currently developed and distributed by Adobe Systems. Since its introduction in 1996, Flash has become a popular method for adding animation and interactivity to web pages. Flash is commonly used to create animation, advertisements, and various web page Flash components, to integrate video into web pages, and more recently, to develop rich Internet applications. Flash can manipulate vector and raster graphics, and supports bidirectional streaming of audio and video. It contains a scripting language called ActionScript. Several software products, systems, and devices are able to create or display Flash content, including Adobe Flash Player, which is available free for most common web browsers, some mobile phones and for other electronic devices (Flash Lite). (http://en.wikipedia.org/wiki/Adobe_Flash, September 2009)

Real Time Messaging Protocol (RTMP)

Real Time Messaging Protocol (RTMP) is a proprietary protocol developed by Adobe Systems for streaming audio, video and data over the Internet, between a Flash player and a server.

The RTMP protocol has three variations:

- The "plain" protocol which works on top of TCP and uses port number 1935.
- RTMPT which is encapsulated within HTTP requests to traverse firewalls.
- RTMPS which works just like RTMPT, but over a secure HTTPS connection.

While the primary motivation for RTMP was a persistent protocol for Flash, it is also used in some other applications, such as the Adobe LiveCycle Data Services.

On Monday, June 15, 2009 Adobe released the RTMP Specification, which could be seen on <http://www.adobe.com/devnet/rtmp/>.

(http://en.wikipedia.org/wiki/Real_Time_Messaging_Protocol, November 2009)

ActionScript

ActionScript is a scripting language based on ECMAScript. ActionScript is used primarily for the development of websites and software using the Adobe Flash Player platform (in the form of SWF files embedded into Web pages). Originally developed by Macromedia, the language is now owned by Adobe (which acquired Macromedia in 2005). ActionScript was initially designed for controlling simple 2D vector animations made in Adobe Flash (formerly Macromedia Flash). Later versions added functionality allowing for the creation of Web-based games and rich Internet applications with streaming media (such as video and audio).

Flash Player 9 added **ActionScript 3.0** with the advent of a new virtual machine, called **AVM2 (ActionScript Virtual Machine 2)**, which coexists with the previous AVM1 needed to support legacy content. Performance increases were a major objective for this release of the player including a new just in time compiler. Support for binary sockets, E4X XML parsing, full-screen mode and Regular Expressions were added. This is the first release of the player to be titled **Adobe Flash Player**.

Flash Player 10 (initially called Astro): Added basic 3D manipulation, such as rotating on the X, Y, and Z axis, and a 3D drawing API. Ability to create custom filters using Adobe Pixel Bender. Several visual processing tasks are now offloaded to the **GPU** which gives a noticeable decrease to rendering time for each frame, resulting in higher frame rates, especially with H.264 video. There is a new sound API which allows for custom creation of audio in flash, something that has never been possible before. Furthermore, Flash Player 10 supports Peer to Peer (P2P) communication with Real Time Media Flow Protocol (RTMFP).

(<http://en.wikipedia.org/wiki/ActionScript>, November 2009)

1. INTRODUCTION

1.1 Thesis background

Nowadays, the Internet and mobile networks play a key role in connecting people and transferring information between them. Although it appears that electronic mail (email), text chats and social networks dominate the modern world, there are certain reasons why do not to use these technologies in the commercial sphere. Emails are still unwisely transmitted using unencrypted connections and electronic communication (protocol SMTP) and they can be tapped on the fly. The same thing applies to mobile calls. Social networks (e.g. Facebook, Twitter, ...) or various chat systems (e.g. ICQ, Skype, ...) are designed, owned and administered by foreign corporations. It is almost impossible to fully adapt such systems to match all company needs. The only other alternative is to develop a completely new communication tool, aimed at concrete company demands. In praxis that is mostly done by a simple PHP or ASP web forum or online chat, usually secured by https transfer protocol.

Audiovisual web conferences are not very common. The main reason why it is so is the lack of experienced developers in the area of online live broadcasted audio and (live) video. The Second very important reason is that until now there no specialized multimedia framework has been created to simplify developers' work.

1.2 Motivation

The main motivation is to create the first universal online multimedia videoconferencing framework ever (MMVCF). This bachelor's thesis describes a unique solution, which simplifies and speeds up the development process of online audiovisual multimedia applications. It deals with a concrete implementation of a complex MMVCF that enables developers to create applications like an online text chats, audio-video conference systems, secured tele-bridges, etc. much more easily

The expected result of this Bachelor's thesis project will be more or less experimental, but highly innovative.

1.3 Radical Pictures OY

Company Radical Pictures OY was founded in year 2004 and its residence is situated in Tampere, Finland. The company's main businesses are multimedia and software development.

The project Radical Chat - multimedia videoconferencing framework was at first developed as an open source multimedia framework for regional television Polar (Czech Republic) by Jaromir Sivic. Lately in year 2009 it was improved and extended to match all needs of the company Radical Pictures OY. Some selected parts of the source code can be shown to the readers of this thesis; however, not the whole solution.

Radical Chat project was deployed on many servers owned and managed by Radical Pictures OY. Radical Pictures OY uses Linux OS on all servers – however, Radical Chat could be installed also on Windows or Mac OS operating systems.

1.4 Structure of the thesis

The following chapter (chapter two) determines a target group of developers (customers - users of the framework), their needs and problems to focus on. Subsequently the proper technologies and tools are selected - of course the most suitable one with the best future perspectives. After that, in chapter three and four, the design, implementation and some problems related to implementation of the project are described in detail. Each theoretical paragraph (from the part Design and implementation) is followed by an example from the praxis. Finally, in the last chapter Conclusion – there is a summary of the whole project.

2 SELECTION OF TOOLS AND TECHNOLOGIES

2.1 Who are the users of the framework?

To enable a massive expansion of the framework, developers' demands i.e. their wishes need to be listen to and the focus should be on the comfort of end users who will use the final applications. The target groups of customers are the web pages - online web services developers and developers of an information systems. The most popular and most common programming languages in this area are PHP, ASP/ASP.NET and Java (See Figure 1).

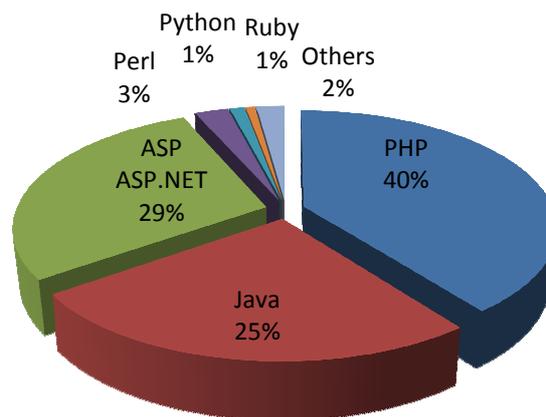


FIGURE 1. Technologies

When talking about the final products - applications written in the MMVCF (multimedia videoconferencing framework), almost all commonly used operating systems are supported such as Windows (XP, 2003, Vista, Windows7), Mac OS 10.X, Linux (kernel 2.4 or above), Unix, BSD compatible as well as PDA and smart phones light weight operating systems - iPhone OS, Windows mobile, Symbian.

2.2 Selection of the proper technology

Following paragraphs shortly describe differences between technologies which could have been used for MMVCF development.

2.2.1 Technologies overview

The most important question, which must be answered before analysis and implementation of the project are started is: what is the most suitable technology(platform) for the multimedia videoconferencing framework? There are three real alternatives.

1. Java platform and Java Applets.
2. Everything could be established on Microsoft .NET platform (actual version of the .NET framework 3.5) and Windows Presentation Foundation technology for rich internet applications called Silverlight in cooperation with Windows Media Server.
3. Great alternative for Microsoft Silverlight & WMS could be an Adobe Flash & Adobe Flash Media Interactive Server.

The advantages and disadvantages of each of these technologies are discussed, before making a final decision.

2.2.2 Java platform

Java applet + JSP server-side application would certainly be a very interesting choice. Java is now supported on all operating systems and almost in all web browsers. Some problems could be found in the speed and form of data processing. Java applets are not primarily designed to work with cameras and live audio-video streams. They are not implicitly accelerated by GPU (graphical processing unit). The transport layer and live data transfer could possibly be a big problem - recorded live data needs to be buffered and encoded in packages in performer's computer memory, then transported to the server via such protocols as FTP or HTTP, then downloaded into the client-viewer's computer memory, decoded and played as quickly as possible. It is very hard to make it without any proprietary real-time data transfer protocol. Signal delay (latency) would be relatively long, as well as high traffic bandwidth.

2.2.3 .NET platform & Silverlight

A much better choice would be if the project was written in .NET framework (programming language C#) for Silverlight. In cooperation with Windows Media Server. Plug-in Silverlight supports the latest video codec WMV3. Silverlight is primarily designed for creating online entertainment and multimedia applications in the Internet environment. It can be accelerated by hardware, especially by GPU. On the server side Microsoft Windows Media Server could be used. WMS is a stable, and easily configurable software product. The main disadvantage is that Silverlight does not support capturing web cameras or microphones. That can be done only by some third party applications. Since Silverlight does not work on all operating systems and in all web browsers and compared to Adobe Flash Player is not so widespread, it could not be recommended to use this technology for implementation of the multimedia videoconferencing framework yet.

2.2.4 Adobe Flash platform

The last option would be to create MMVCF in Adobe Flash and use the Flash Media Interactive Server (FMIS) on the server side. Adobe Flash Player is installed on 99% of internet-enabled computers all over the world (http://www.adobe.com/products/player_census/flashplayer/, November 2009) and is supported on all commonly used operating systems, including smart OS of phones. This plug-in is primarily designed for creating online multimedia applications; it also enables capturing web cameras and microphones and drawing is accelerated by GPU. In this case, server part of the MMVCF has to be implemented in Flash Media Interactive Server, and its API somehow exposed or connected to PHP, ASP.NET or Java. So far, this solution seems like the best one, but there are certain reasons, why such multimedia videoconferencing framework has not been created yet, and why nobody ever tried to develop it.

The most likely major problem is the high price of a FMIS server (US\$ 4500, November 2009) as well as the lack of skilled developers, who know and fully understand FMIS API. Up to year 2008 Flash supported only a relatively slow and old scripting language called Action Script 2. Fortunately in years 2008-2009 AVM2 (Actionscript Virtual Machine 2 - just in time compiler) and the programming language Action Script 3 (with excellent performance and quick data processing)

were added into the Flash Player. Also on 15th of June 2009 Adobe published the first official documentation of RTMP protocol, which is used for communication between Adobe Flash Player and FMIS. After that, a lot of third party stream servers were created even under GNU-GPL licenses.

2.2.5 Final decision

The differences between technologies mentioned above are obvious, but still it is very hard to make a final decision. If it is taken into account that Flash technology (Flash Player) is the world's most pervasive software platform and during last two years was many times upgraded, performance improved, API extended and if some kind of much cheaper stream server like Wowza or Red5 will be used instead of overpriced FMIS, then Flash platform is a clear choice. Therefore, the final decision is relatively interesting and unexpected: the multimedia videoconferencing framework will be implemented in Action Script 3 (Adobe Flash platform) in cooperation with Wowza media server (Java platform) and lately Red5 media server (also Java platform).

2.2.6 Wowza media server Pro

Wowza Media Server Pro is a high-performance, extensible and a fully interactive Flash media server for live and on-demand streaming, chat, recording and much more. Wowza Pro lets developers take Flash streaming to new heights with its full set of features and exclusive capabilities like Wowza's H.264/HE-AAC live streaming with non-Flash RTSP/RTP and MPEG-TS encoders. Wowza media server Pro comparison to Adobe Flash Interactive Server could be found on web page <http://www.wowzamedia.com/comparison.html>.

3 SERVER SIDE DESIGN AND IMPLEMENTATION

This chapter is divided into two sections. The first section presents the idea of Universal Multimedia Videoconferencing Framework in general as well as a basic overview of the designed API. The second section, starting with paragraph 3.4, deals with the solution of selected (most interesting) parts of this MMVCF.

3.1 Three-tier architecture

The entire universal multimedia videoconferencing framework (codename Radical Chat) is divided into three parts (See Figure 2).

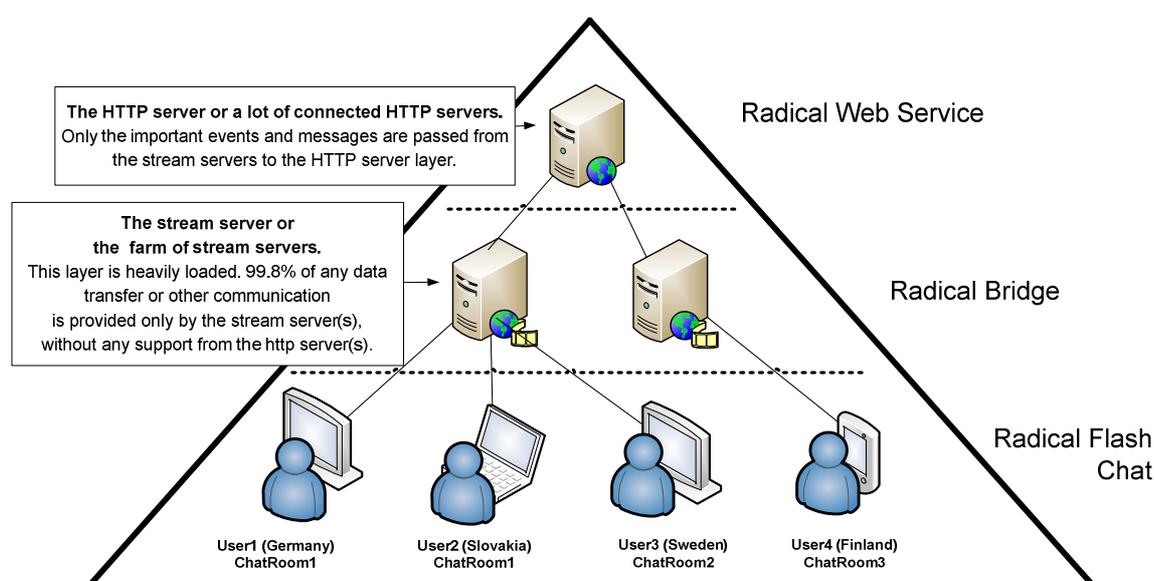


FIGURE 2. Radical Chat architecture

The first (bottom) level contains the client part of the system (codename Radical Flash Chat) implemented in Action Script 3 for Adobe Flash Player 10. The behavior of the Radical Flash Chat is controlled by a lightweight application written in Java, API for Wowza Media Server & RED5 (codename Radical Bridge). Its main task is to filter out or respond for less important client requests and to forward the really important requests to the PHP, or ASP.NET web service on the top of the system

hierarchy. Of course, when started, Radical Bridge is running only in system memory as quickly as possible. It is not connected to any database or a file stored on a hard drive (except for logging, which could be turned off). The most important and unique part of whole framework is the top level of the system, where the chat rules are implemented. This highest level is a very complex web service with an exactly defined structure described by WSDL document. It may seem obvious to use web services technology for communication with an external part of the system implemented by other developers. In this case, customers (developers) do not implement “web service consumer”, but “web service server”. That allows the Radical Chat to be used by almost any programming language/platform. On the other hand some clients (developers) may be annoyed by such policy where the customers must spend a weeks, or months by implementing the correct, precise web service server (provider) before they can use the purchased product. That is the reason, why Radical Chat is delivered together with pre-implemented, tested Radical Web Service for PHP and ASP.NET (C#) programming languages. Radical Web service is an abstract web service with an attached suitable API, which allows the developers to create their own videoconferencing systems or chats in a few hours or days without any further knowledge about Flash, Action Script 3, RTMP protocol or stream server API.

3.2 System Logic

All online videoconferences or chats have some common elementary rules and structures. Radical Chat is not an exception. The system recognizes two basic types of objects - a chat room and a client. Each client, before he/she enters the system receives a globally unique identifier – clientUID. This identification string is assigned via FlashVars variable. Each Adobe Flash Player with Radical Flash Chat application in the web browser must be recognized by exactly one unique clientUID. Each client is associated with just one room specified by globally unique identifier roomUID. That does not mean that a client from one chat room cannot contact any other clients from another chat room of the system. The entire structure looks like a simple tree diagram. (See Figure 3).

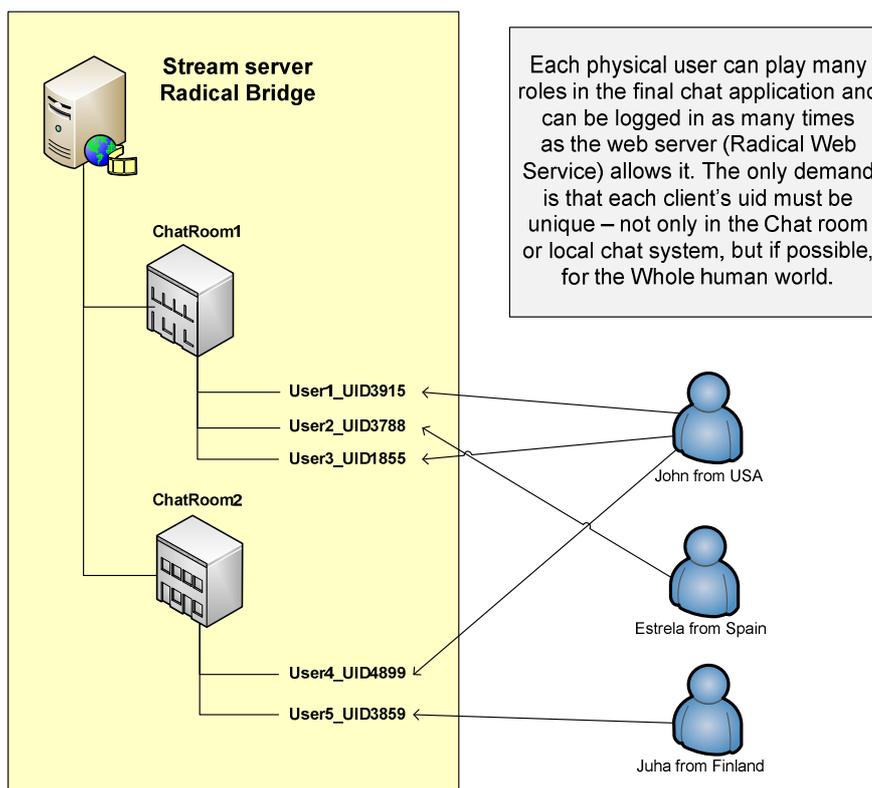


FIGURE 3. Hierarchy

When the SWF application is loaded into the Flash Player, Radical Flash Chat tries to download configuration XML file including design specification (Skin), description of the user interface behavior and some other eventualities like stream server IP address, port, Radical Bridge app. name etc. For heavily loaded systems, the entire RADICAL BRIDGE stream server farm can be used. The same applies to Radical Web Service on the HTTP server. After all skin images and sounds are loaded, Radical Flash Chat tries to establish connection to the Radical Bridge and log client into the chat room with specified roomUID.

If the client requests an access to the chat room, which does not exist in the memory yet, then the chat room is automatically created. Lately, when the last client leaves the chat room - the system will recognize it and immediately remove the room and other dependent resources from the memory. It should be remembered that the whole Radical Bridge runs only in RAM memory of the stream server computer. It is true, that stream server remembers the status of all clients and chat rooms and automatically creates (updates) all necessary data structures, but when the last client leaves the room - dependent allocated memory is released.

When the next client enters the same chat room (even if the interval between the last client disconnection and the new client connection is extremely small - a few milliseconds), then the whole new chat room is created and eventually all properties must be set again. It is very important to have enough available RAM on the stream server computer as well as sufficient network bandwidth capabilities. (See Figure 4).

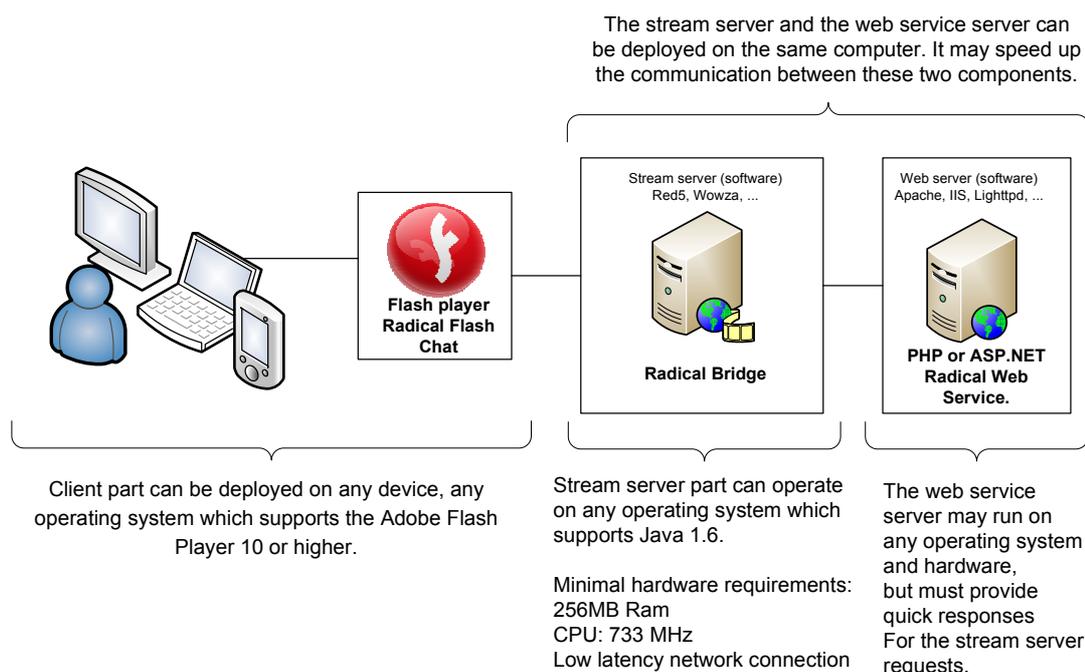


FIGURE 4. Radical Chat deployment

Although Radical Bridge handles 99% of data communication and 95% of client's requests, the most important core functions of each videoconferencing chat (such as logging in/out, configuration of privileges for playback or video publishing, ...) and another additional rules (like kicking out, private chat invitation, ...) are mostly wanted to be implemented by final developers (users of the MMVCF). However, which way is the most appropriate for this task? How should this part of the framework be created to enable developers to interact with the system from different programming languages? The answer is: by web services technology. In this case, the customer's part is a web service server (provider) instead of a web service consumer as obvious. (See Figure 5).

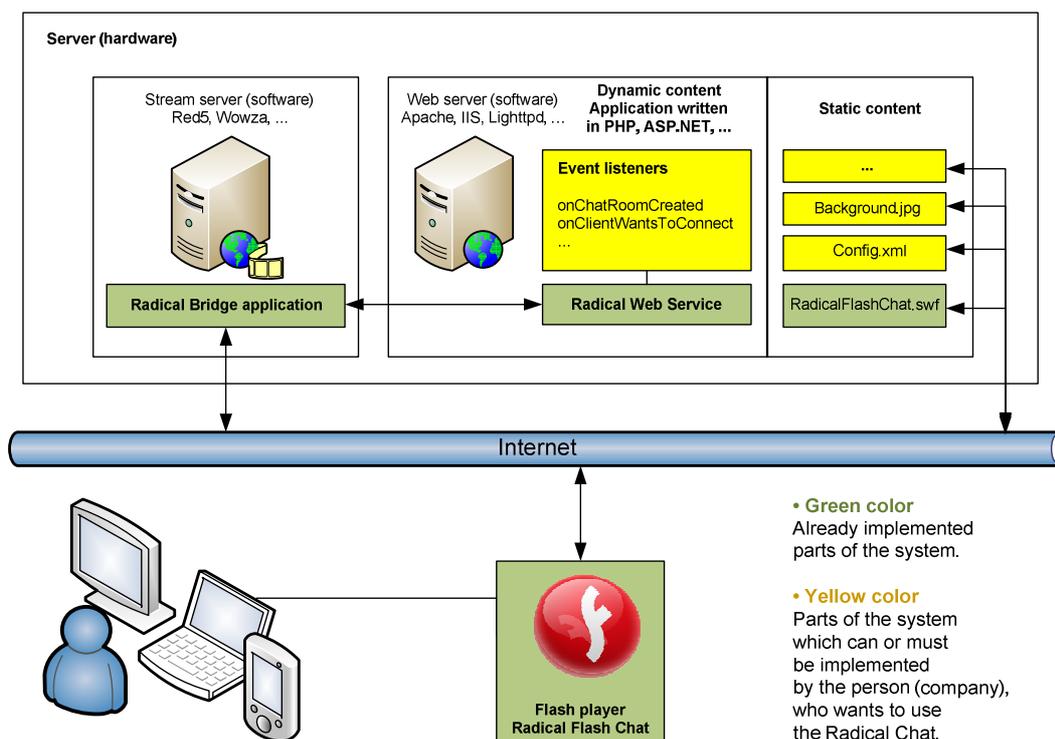


FIGURE 5. Radical Chat basic interactions

Because such a strategy would never have any chance for success on real market, Radical Chat in enterprise edition contains “Radical Web Service” for PHP and ASP.NET (C#). “Radical Web service” is an abstract web service with attached suitable API. Any time when pre-implemented Radical Web Service is called by Radical Bridge - exactly one of following 18 events (functions in service.php) is triggered.

3.3 List of events

3.3.1 Global events

Global events are dispatched directly by the stream server, when it is starting or stopping.

- **GetAPICode** event is triggered at the moment when the stream server (Wowza or Red5) is started. This function returns a string - version of the API interface implemented by the PHP (ASP.NET) Radical Web Service.

- **GetImplementedFunctions** is called after GetAPICode and returns a list of event names separated by comma that have been implemented by the developer (user of MMVCF). This function is triggered only once per Radical Bridge Application life cycle, just before onApplicationStarted. It is recommended to not mention any functions (events), which body have not been implemented - PHP Radical Web Service recognizes implemented, non-implemented events and assembles the list automatically. In other programming languages, it must be done manually.

3.3.2 Application events

The application events are dispatched by Radical Bridge application after its loaded to the system memory and started.

- **OnApplicationStarted** event is triggered when Radical Bridge application is loaded into the stream server memory.
- **OnApplicationStopped** event is triggered a few moments before Radical Bridge application is unloaded from the stream server memory. Typically it is just before the stream server is properly terminated, or also in case that there are no connected clients for a long time.

WARNING: onApplicationStopped is not triggered if the stream server crashes down.

3.3.3 Logging in/out events

Logging in/out events are very important for successful chat management. They are dispatched before a client enters the system or after he/she leaves.

- **OnClientWantsToConnect** event is triggered when a client attempts to connect to the Radical Bridge (before he/she is assigned into the chat room). Developers should always implement this function. "AcceptConnection" or "RejectConnection" method must be called somewhere in the body of this function.

- **OnClientConnectionWasAccepted** event comes after `onClientWantsToConnect`, but only if a client was accepted by “AcceptConnection” method. Usually it is not necessary to implement the body of this function.
- **OnClientConnectionWasRejected** event comes after `onClientWantsToConnect` if the client was fired (kicked out) by “RejectConnection” method. Usually it is not necessary to implement the body of this function.
- **OnClientDisconnected** event occurs when the client is disconnected by “RejectConnection” or if a connection is somehow terminated, for example because of network malfunction or because the client turned off the computer or closed the web browser.

3.3.4 AV stream handling events

AV stream handling events are dispatched by the Radical Bridge when client attempts to play or publish audio-video stream.

- **OnClientWantsToStartStreaming** event is triggered every time when any client wants to publish live audio or video stream. Mostly it is a live video from a web camera.
- **OnClientStoppedStreaming** is a notification event that indicates that the client stopped streaming.
- **OnClientWantsToPlayStream** event is triggered when some client attempts to play live audio or video.
- **OnClientStoppedPlayingStream** is a notification event triggered when some client stopped playing live audio or video.

3.3.5 Client related events

Other important client related events.

- **OnClientSendMessage** function is called every time when a client sends a text message, but only in case that the client's flag `dispatchEventIfClientSendTextMessage` is set to true.
- **OnCheckClients** - This event is executed by Radical Bridge at a periodic time interval, which depends on the property `NextCheck` of each client in the system. Every one second Radical Bridge decrements `NextCheck` value of each client in all stream server chat rooms by one. When `NextCheck` value is equal to 0, client's manipulator (handler) is pushed to the associative array and passed to the `onCheckClients` event (See Figure 6) - there the property `NextCheck` must be reset, otherwise the concrete client will never more call `onCheckClients` again. This function is very useful for such systems, where on the beginning each user has an account with some amount of money or credits and later, if he/she runs out of credit, then his/her connection to the chat is terminated.

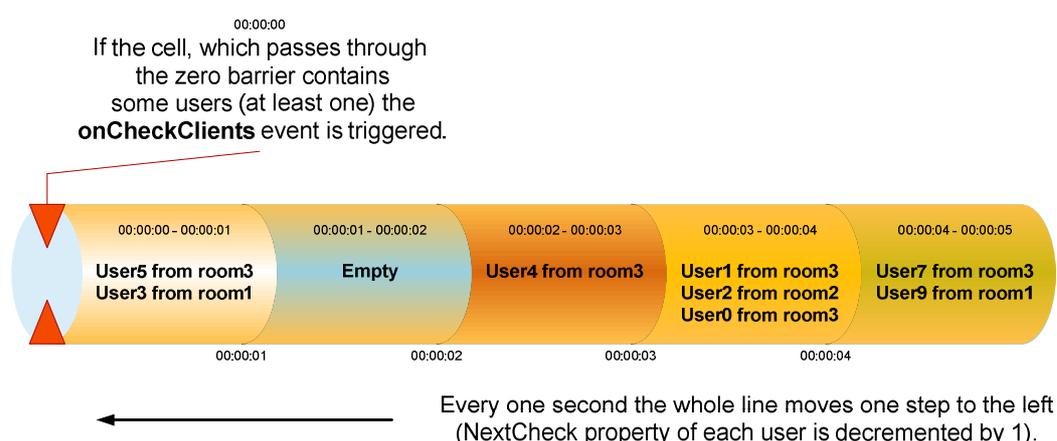


FIGURE 6. `onCheckClients`

3.3.6 Chat room related events

Events created for manipulation and management of a chat rooms.

- **OnChatRoomCreated** event occurs when new chat room is created just before first client attempts to enter it.
- **OnChatRoomClosed** event is invoked when a room is about to close, just after the last client left the room.
- **OnCheckChatRooms** - very similar to onCheckClients, but associative array contains chat room handlers instead of client handlers. This function is very useful for online SMS chats.

3.3.7 Universal events

Universal events are events designed for dynamic interactions with client's graphic user interface.

- **OnUniversalCall** event is triggered when custom BasicButton in "Radical Flash Chat" application (application in client's web browser) is pressed.

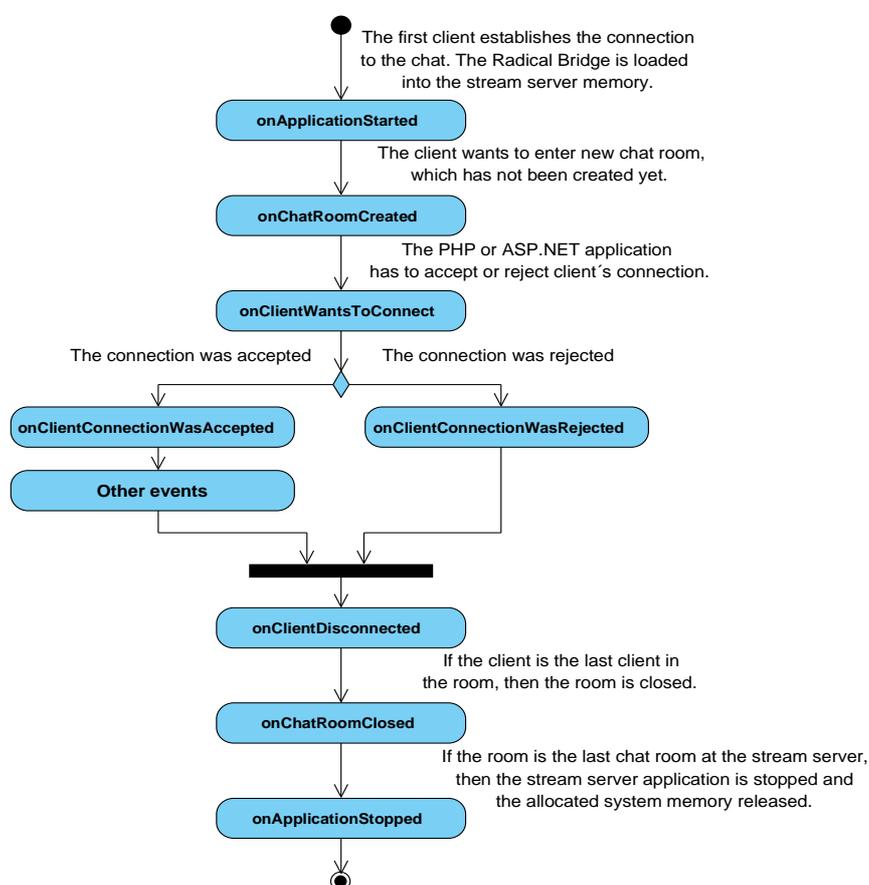


FIGURE 7. Sequence of events

The events (functions) and associated API above forms the highest level of the MMVCF Radical Chat. As was mentioned before, Radical Web Service is not an obvious web service provider, but uses the asynchronous event driven programming technology. It is possible that ordinary web developers (PHP developers, or ASP.NET developers), which have never got in touch with Actionscript 3 or Win32 API programming are not familiar with this technology. Therefore, it is described in the following paragraphs.

Programming languages and techniques for web pages programming like AJAX are established on a very simple “request – response” mechanism. For any request - one instant response (which somehow changes user's web browser content) is sent. In the last years, MVC model (which separates web application appearance, data and code behind) has become popular. Also AJAX technology is getting famous and is used almost everywhere, if it is possible. However, in both cases the used technology is the same: (one) request - (one) response. This is about to change with the revolutionary conception of the Radical Chat.

3.4 Parallel, asynchronous, event-driven programming

The reader is encourage to a futuristic system, where the user can click on any button on the web page, and according to that action the content of a web browser of another user(visitor) is instantly changed. Another more unbelievable scenario - the user sends SMS message from the cell phone. Message arrives through the mobile operator's gate and executes the PHP or C# script causing an immediate change of the application skin and behavior for all logged users. Such a thing cannot be realized by the old request - response mechanism, but can be easily implemented using an asynchronous event driven programming in Radical Chat. Flash technology and Adobe Flash Player (since version 9.x and above) support programming language ActionScript 3, where any event (action/reaction) is basically asynchronous. When a connection between Adobe Flash Player and media server is established, Adobe Flash Player communication port remains opened until explicitly closed. That means the Flash Player may receive new data and commands from the stream server without any previous request (this method is called response without request). Moreover, each Adobe Flash Player request to the stream server is

basically sent by method request without response - when the request is sent, the response can arrive after a few milliseconds, seconds or never. Until this very moment the asynchronous, event driven programming was encapsulated only in the Adobe Flash technology (Adobe Flash Player and stream server). Multimedia videoconferencing framework Radical Chat brings it to the ASP.NET and PHP environment.

The communication between Radical Bridge and Radical Web Service is secured by the manipulators or handlers (client handlers, chat room handlers). Once Radical Bridge receives an important request from Radical Flash Chat, which must be processed by PHP or ASP.NET script, the Radical Bridge relays the request to the Radical Web Service with attached client handler, chat room handler or stream server handler. Handler virtually represents client, chat room or stream server structure. Subsequently Radical Web Service executes `service.php` or `service.cs` and triggers exactly one event (function) in the script file (see List of events 3.3).

Handlers and enclosed set of methods are passed to the function(s) as parameter(s). If any handler's method or operation is called inside event function body, then nothing special happens immediately. All performed operations are stored in the command queue, which is processed later after the PHP or ASP.NET script ends. Each request is processed in a separate thread to enhance Radical Bridge and Radical Web Service performance. It may happen that multiple events are executed in parallel at once. This fact may cause many unpredictable collisions. In case that data are stored in a database (like MySQL, MSSQL,...), it is highly recommended to use transaction lock for any operation. In other cases – semaphores alias mutexes should do the trick. The Mutex class is included in the kernel of "Radical Web Service API" for PHP.

3.5 The problems of parallel data processing.

Although Radical Bridge runs only in the memory, it must solve many problems related to parallel data processing. Most of these problems are solved only by partial synchronization, as for example text chatting or logging in/out events. Occasionally some specific problems like dynamic skinning are solved by the pseudo complete synchronization.

Partial synchronization is very fast and does not consume a lot of RAM or CPU time, but in rare cases may leads to very strange results. See the following implementation of onClientWantsToConnect event in PHP.

```
function _onClientWantsToConnect(&$client)
{
    Logger::func("onClientWantsToConnect");
    //accept client's connection
    $client->AcceptConnection();
    $client->SendPrivateMessage("Admin", $client->getUID(),
        "000000", "Welcome in the room.");
    $client->SendRoomMessage("Admin", "000000",
        "User \"\" . $client->getNickName() .
        "\"\" has entered the room.");
    $client->SendPrivateMessage("Admin", $client->getUID(), "000000",
        "Please do not use unpolite words in this chat.");
    $client->SendRoomMessage("Admin", "000000",
        "Remember this room will be closed at 9pm.");
}
```

What happens if two users enter the room exactly at the same time? Since the text chat is synchronized only partially, there are mixed public and private messages in the function above. The result in User1 and User2 web browser should look like:

User1:

```
Admin: User "User2" has entered the room.
Admin: Remember this room will be closed at 9pm.
Admin: User "User1" has entered the room.
Admin: Remember this room will be closed at 9pm.
Admin>> User1: Welcome in the room.
Admin>> User1: Please do not use unpolite words in this chat.
```

User2:

```
Admin: User "User2" has entered the room.
Admin: Remember this room will be closed at 9pm.
Admin>> User2: Welcome in the room.
Admin>> User2: Please do not use unpolite words in this chat.
Admin: User "User1" has entered the room.
Admin: Remember this room will be closed at 9pm.
```

How is it possible? Each Flash Player after its connected to the Radical Bridge attaches two Shared Objects (SO). A shared object is a data object dynamically shared between the stream server and Flash Player. The first one is a public – room SO and is shared for the whole chat room and all logged users, the second one is private - designed only for one client.

Once the Radical Bridge starts to process commands from Radical Web Service, and recognizes the private or public message signature, then that message is automatically inserted into the appropriate private or public shared object. Radical Bridge works in many parallel threads, so the requests and incoming commands could be processed at once in parallel (See Figure 8).

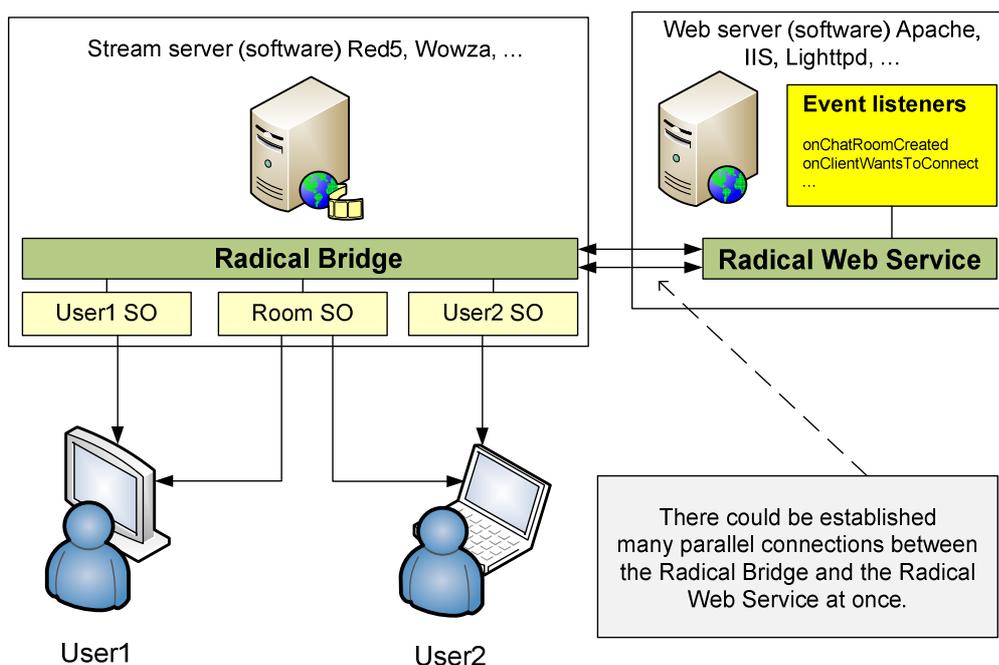


FIGURE 8. Parallel data processing problems

After that, the stream server immediately synchronizes the changed shared objects with Adobe Flash Players. Unfortunately, synchronization needs some time and usually public room SO is synchronized before private user SO. Since in example above private message is sent as the first, but room shared object with a public data and messages is synchronized with the Flash Player (Radical Flash Chat) before

private messages - the correct messages order is lost. That may leads to an unexpected strange results. Because this problem appears only in case, that a lot of users from one chat room send a lot of public and private messages in a very short time period, and because usually the order does not matter – this problem is solved only by a partial synchronization. The same way is used for publishing live audio and video – in case that two users turn on their web cameras at precisely the same moment - it does not matter if user2 is announced to the audience as the first online performer or as the second one. However, there exists one problem, which cannot be solved by a simple partial synchronization. The problem, where the order of commands and sequence of changes matters a lot is called “dynamic skinning”. Its main idea is uncovered at the end of chapter 4.

4 CLIENT SIDE DESIGN AND IMPLEMENTATION

This chapter presents the client's part of Radical Chat multimedia videoconferencing framework – "Radical Flash Chat". It describes techniques, how to create an application user interface and its behavior.

4.1 Radical Flash Chat

Radical Flash Chat is the only part of Radical Chat system exposed to the final users and loaded directly into their web browsers. It is very stable, efficient and could be deployed on almost any device or computer, which is Adobe Flash Player 10 compatible, yet still, allows the developers to manage issues and control the chat without any further knowledge about Flash Technology or RTMP protocol.

4.2 XML configuration file

Radical Flash Chat (radicalflashchat.swf 74KB) does not include any pre-implemented skin, text labels or set of concrete behavior. Everything must be designed by final developers or graphic designers. Fortunately, Radical Flash Chat provides an incredibly simple way to easily make application skin and basic functionality. Everything is base on XML configuration file inspired by modern UI (user interface) techniques as Windows Presentation Foundation or Flex. The complete description of all possible options of the configuration file is on www.cze.cz.

4.3 Static skin

The user interface, based on (standalone) XML configuration file, is called static skin. Its construction is somewhat similar to the construction of an HTML document. Static skin is loaded into the web browser memory before Flash Player establishes connection to the stream server. Static skin is independent on Radical Bridge (see figure 5). Radical Chat preserves MVC design patter - application user interface is strictly separated from the code behind. Therefore, graphic-designers can work separately of PHP and ASP.NET developers.

The whole application skin (element `<Skin>` in `config.xml`) is divided into three layers BG (background layer), ML (main layer, or middle layer) and FG (foreground layer).

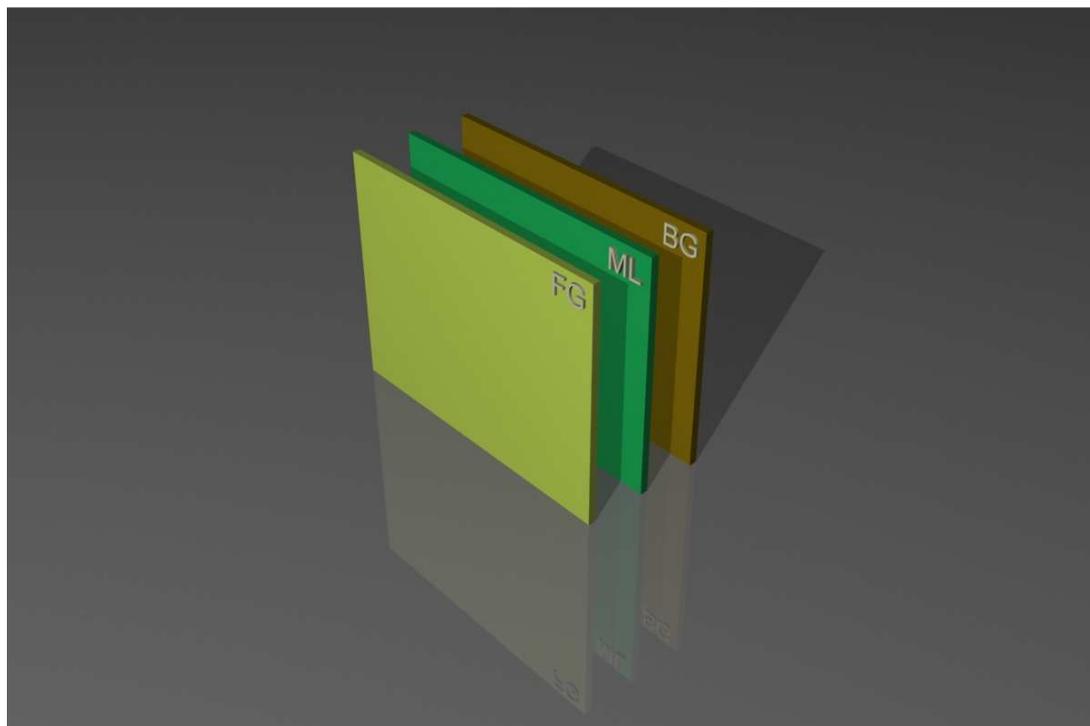


FIGURE 9. Layers

Layers BG and FG support universal set of components (classes) derived from `SkinObject`. Practically layers BG and FG are subjects to the same rules and features. All elements there (images, buttons, ...), can be subsequently modified or re-designed by “dynamic skinning”. On the other hand the main layer ML that lies in front of the layer BG and behind the layer FG, contains support for only a few specialized classes (`TextChat`, `ListOfUsers`, `ChatInput`), which cannot be instantiated more than once at a time.

All visible components (alias types or classes in this terminology) in all layers are derived from one basic abstract parent called `SkinObject`. `SkinObject` contains a set of essential properties common for all derived subclasses. Some of them are obligatory and a value must be explicitly assigned to them (in `config.xml`), the others are optional.

Following code demonstrates, how many properties SkinObject has:

```

<RadicalChat>
...
  <Skin>
    ...
    <BG>
      <SkinObject>
        <!--
          Type specify which derived subclass of SkinObject
          will be used at this place. Type is obligatory parameter.
        -->
        <Type></Type>
        <!-- Name of the object - obligatory parameter. -->
        <Name>Panell</Name>
        <!--Visibility (optional, default value true). -->
        <Visible>true</Visible>
        <!-- Relative position - obligatory section. -->
        <Position>
          <Left>0%+10+{AnotherObject.Right}</Left>
          <Top>0%+10</Top>
          <Right>100%-10, min=400</Right>
          <Bottom>100%-10, min=300, max=900</Bottom>
        </Position>
        <!-- Basic effects - optional section. -->
        <LookAndFeel>
          <!-- Alpha value - real value between 0 and 1 -->
          <AlphaBlend>1</AlphaBlend>
          <!--
            Blend mode, like in Adobe Photoshop.
            Allowed values: normal, multiply, etc...
          -->
          <BlendMode>normal</BlendMode>
          <!-- Color transformation -->
          <ColorTransform>
            <Color redMultiplier="0.1" greenMultiplier="0.4"
              blueMultiplier="1" alphaMultiplier="1"
              redOffset="128" greenOffset="128"
              blueOffset="128" alphaOffset="0" />
          </ColorTransform>
        </LookAndFeel>
      </SkinObject>
    ...
  </BG>
  ...
</Skin>
</RadicalChat>

```

Basic obligatory parameter `<Type>` determines which class will be instantiated in place of the `<SkinObject>` element. Type may be a `BasicPanel`, `BasicButton`, `HtmlTextArea`, ... All these classes are described in the text below. If `<Type>` is specified, `SkinObject` usually gains a lot of new parameters and properties. Some of

them might be obligatory. For example: if type is basic panel

<Type>BasicPanel</Type>, then "Pattern" or "NineGrid" element must be set in section <LookAndFeel> - so LookAndFeel XML element is no longer optional for all components derived from BasicPanel.

Example:

```
<LookAndFeel>
  <NineGrid>
    <SliceImgSequence>
      ./skin/panels/rsqshadow1_0{1-9}.png
    </SliceImgSequence>
  </NineGrid>
</LookAndFeel>
```

Another obligatory parameter <Name> contains the local object identifier (identifier unique for entire <Skin> element). The name is extremely important for “dynamic skinning” and also for innovative “RPSF positioning system” built in Radical Flash Chat core.

4.4 RPSF positioning system

RPSF is an acronym for Relative Position Specified by Formula. RPSF technology was invented and described in year 2008 by the author of this thesis. RPSF gets rid of “align” parameter from HTML language and simplifies the graphical design process. RPSF was for the first time implemented in TV framework for online broadcasting (See web portals <http://tv.cd.cz> or <http://www.tvportaly.cz>) and is also used in Radical Chat. Although anybody may think that positioning of rectangular objects is a simple, many times implemented software gadget - the truth is that RPSF comes up with a completely different idea than the one used in HTML or in Win32 API. This solution is partially compatible with CSS, but enables developers to do much more.

The basic idea is that any rectangular object can be placed into the document by a set of six values (left, top, width, height, right, bottom), where only four of them must be specified. The remaining two are dependent and may be calculated later by the formulas below, please see Figure 10.

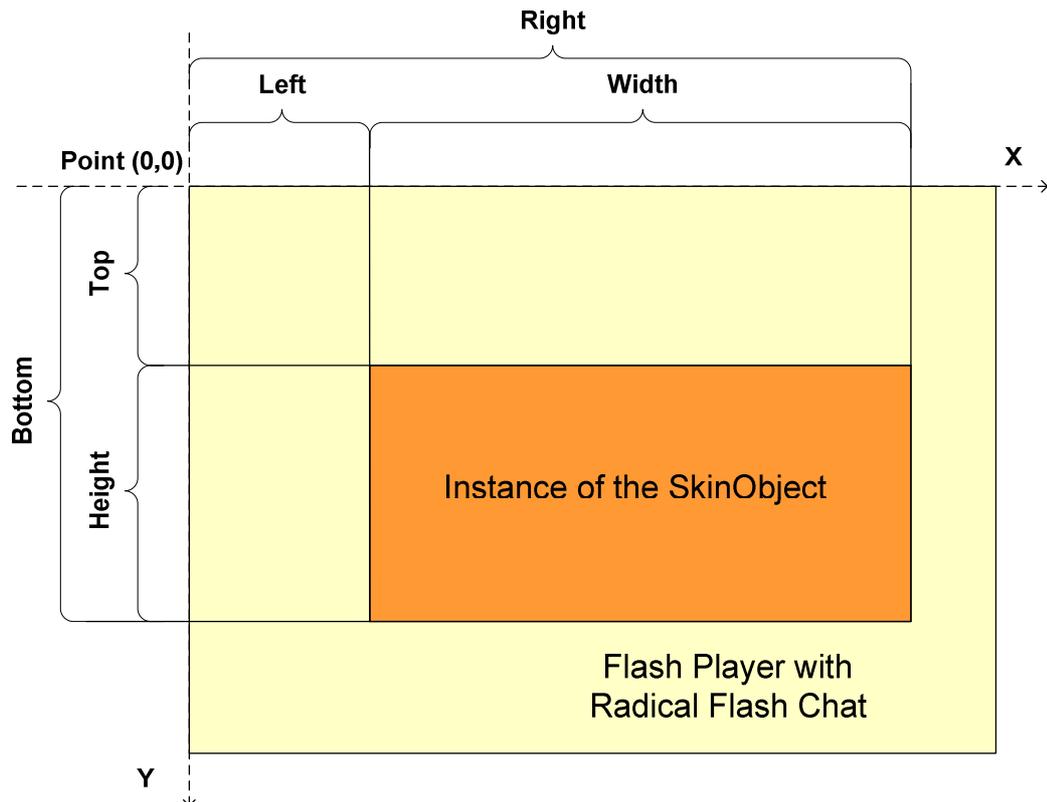


FIGURE 10. RPSF positioning system

$$\text{Width} = \text{Right} - \text{Left}$$

$$\text{Height} = \text{Bottom} - \text{top}$$

What is very important in RPSF system is the fact that variables are not only a simple numbers but complex expressions with a constraints like min or max.

For example:

```
<left> 80% - 50 + {Panel1.Left}, min = 10% + 70, max = 500 </ Left>
```

The calculation process of the expression above, in case that the width of the Flash Player (Web Browser) is 800 pixels and Panel1.Left is 100 pixels is following:

- 1 Compute expression base:

$$\text{Left}_{\text{Base}} = 80\% - 50 + \{\text{Panel1.Left}\} = 80 * \frac{800}{100} - 50 + 100 = 690$$

- 2 If specified, compute minimum, otherwise min is set to $-(2 * 10^9)$.

$$\text{Left}_{\text{min}} = 10\% + 70 = 10 * \frac{800}{100} + 70 = 150$$

3 If specified, compute maxime, otherwise max is set to $(2 * 10^9)$.

$$\text{Left}_{\text{max}} = 500$$

4 Compute the result.

$$\begin{aligned} \text{Left} &= \text{Min}(\text{Max}(\text{Left}_{\text{Base}}, \text{Left}_{\text{min}}), \text{Left}_{\text{max}}) = \text{Min}(\text{Max}(690, 150), 500) = \\ &= \text{Min}(690, 500) = 500 \end{aligned}$$

4.5 Strict rules for RPSF positioning system.

RPSF system is very powerful, flexible and easy to learn. On the other hand, it does not tolerate any mistakes. Components, placed into the document using RPSF positioning system must follow following rules:

- i. It is allowed to combine percentages, pixels and inches, but only using operations addition or subtraction (sign plus or minus). If the target device is a screen, then the result value will always be in pixels.
- ii. It is possible to use min or max constraints. If it is so, then the result value will always be from interval $\langle \text{min}; \text{max} \rangle$.
- iii. It is allowed to refer to properties (Left, Top, Width, Height, Right, Bottom) of other objects, but the final dependency graph must always be an oriented graph without any loop (See Figure 11).

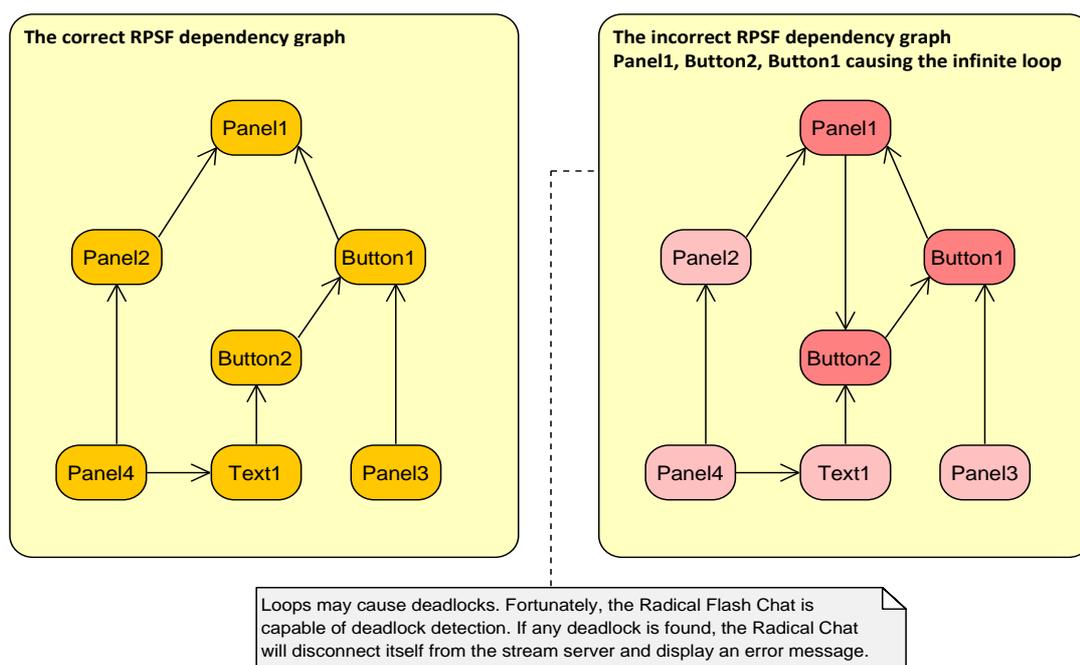
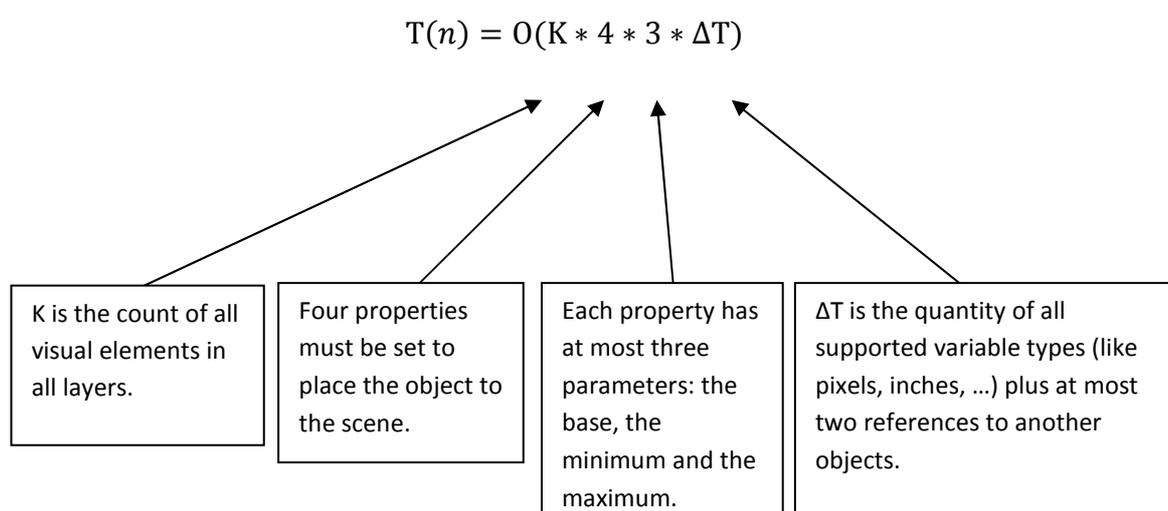


FIGURE 11. RPSF dependency graph

4.6 RPSF implementation.

RPSF positioning system time complexity is linear. It works with the same speed as the outdated HTML positioning. If the position of any object is dynamically changed or the web browser resized, then positions of all dependent components are recomputed.



For more information see RelativePosition.as source file.

4.7 SkinObject class extensions.

SkinObject is a general abstract class inherited and implemented by many other subclasses. Those subclasses are divided into two main groups - universal components and specialized components.

4.7.1 Universal components

Universal components (BasicPanel, BasicButton, HtmlTextArea, LiveStream, MyCamera) - can be used in layers <BG> and <FG>. There is no default limit for them, so any class (component) can be instantiated as many times as graphic-designer consider it necessary. Default behavior and appearance are strictly determined only by the graphic-designer. New objects can be dynamically added, updated or destroyed on the fly by “dynamic skinning”. The appearance and behavior may be interactively modified. Some components have specialized “Look And Feel” section with some new properties (See Figure 12).

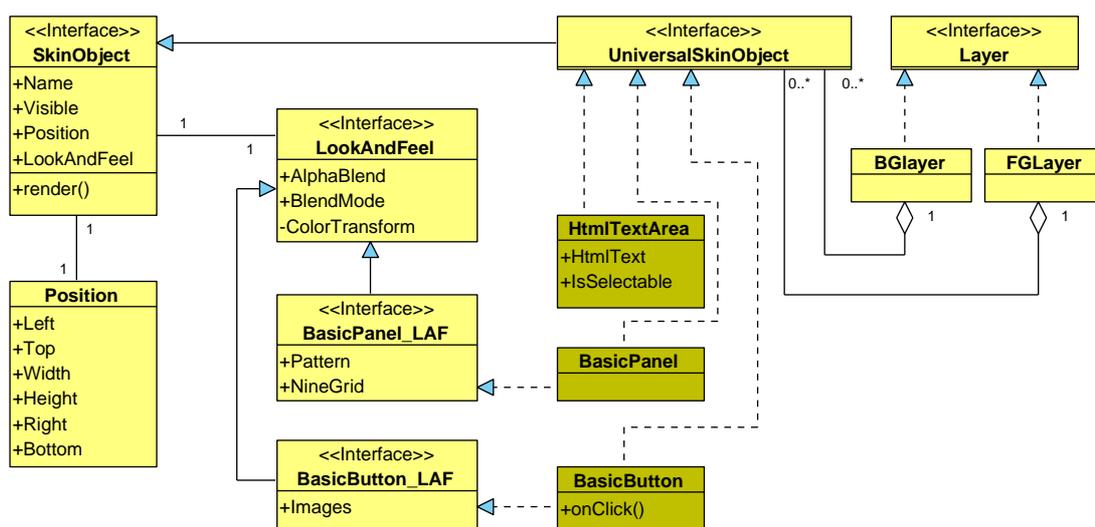


FIGURE 12. Universal components

4.7.2 BasicPanel

Specialized type(class) BasicPanel is derived from the SkinObject and extended by two new features: <Pattern> and <NineGrid>. Only one of them must be used, when an object is instantiated. Basically, BasicPanel is a very sophisticated image.

Mode <Pattern>

If mode is <Pattern>, then the area specified by <Position> is filled by texture <LookAndFeel> <Pattern> <Image>. In the example above, the picture is walltexture.jpg. Mode pattern is usually used for backgrounds and watermarks. To preserve a nice look and feel it is recommended to use seamless images. Please see the source code below.

```

<SkinObject>
  <Type>BasicPanel</Type>
  <Name>Panel</Name>
  <Position>
    <Left>0</Left>
    <Top>0</Top>
    <Right>100%</Right>
    <Bottom>100%</Bottom>
  </Position>
  <LookAndFeel>
    <Pattern>
      <Image>walltexture.jpg</Image>
    </Pattern>
  </LookAndFeel>
</SkinObject>
  
```

Mode <NineGrid>

If mode is <NineGrid>, then area specified by <Position> is filled by seamless graphic shape composed of 9 parts (slices - see Figure 13 below).

```
<SkinObject>
  <Type>BasicPanel</Type>
  <Name>Panel</Name>
  <Position>
    <Left>0</Left>
    <Top>0</Top>
    <right>100%</Right>
    <Bottom>100%</Bottom>
  </Position>
  <LookAndFeel>
    <NineGrid>
      <slice1>slice01.png</Slice1>
      <slice2>slice02.png</Slice2>
      <slice3>slice03.png</Slice3>
      <slice4>slice04.png</Slice4>
      <slice5>slice05.png</Slice5>
      <slice6>slice06.png</Slice6>
      <slice7>slice07.png</Slice7>
      <slice8>slice08.png</Slice8>
      <slice9>slice09.png</Slice9>
    </NineGrid>
  </LookAndFeel>
</SkinObject>
```

It is not necessary to state all nine slices, but at least one slice must be set. Also, It is allowed to use short description:

```
<NineGrid>
  <SliceImgSequence>slice0{1-9}.png</SliceImgSequence>
</NineGrid>
```

The expression {1-9} or {0-8} is replaced by a sequence of consecutive numbers.

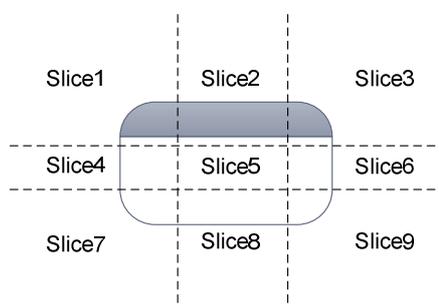


FIGURE 13. NineGrid

4.7.3 BasicButton

BasicButton is a specialized class derived from SkinObject. BasicButton is a multifunction button. Its behavior is set in `<OnClick>` element of XML configuration file. Any click on BasicButton is considered as a “**time protected action**”, which means that no other button can be pressed during one second time interval after the previous button click - it is a safety regulation to protect heavily loaded systems. Of course, the button’s look and feel may be adjusted by the graphic-designer using properties `<NormalImage>`, `<OverImage>`, `<DownImage>`. See an example of BasicButton definition below:

```
<SkinObject>
  <Type>BasicButton</Type>
  <Name>Button1</Name>
  <OnClick>
    <Action>onUniversalCall_WithSelectedClient</Action>
  </OnClick>
  <Position>
    <Left>10</Left>
    <Top>100%-40</Top>
    <Width>80</Width>
    <Height>30</Height>
  </Position>
  <LookAndFeel>
    <Images>
      <NormalImage>./skin/images/private_btn_01.png</NormalImage>
      <OverImage>./skin/images/private_btn_02.png</OverImage>
      <DownImage>./skin/images/private_btn_03.png</DownImage>
    </Images>
  </LookAndFeel>
</SkinObject>
```

The action is specified in element `<OnClick>` `<Action>` and determines the behavior after the user presses the button. Radical Flash Chat currently supports the following set of actions:

onUniversalCall bubbles through the system and triggers onUniversalCall (Radical Web Service) event, if implemented. The size of an array `$clients` is always one and the first item contains the manipulator of the client who pressed the button. See the following “code behind” for Button1 (used language PHP):

```
function _onUniversalCall(&$clients, $senderName, $eventName, $value)
{
  if ($eventName==ChatEvents::$EVENT_BASICBUTTONCLICKED)
  {
    //if Button1 was pressed
    if ($senderName=="Button1")
```

```

    {
        //retrieve manipulator of the client who pressed the button
        $client = $clients[0];
        //send a message to the client's chatbox
        $client->SendPrivateMessage("Admin", $client->getUID(),
                                   "000000", "Button1 was pressed.");
    }
}
}

```

onUniversalCall_WithSelectedClients is very similar to onUniversalCall action above, but in this case the parameter `$clients` contains (from index 1 to index `n`) handlers of all clients selected in the component "ListOfUsers". Besides `$clients[0]` is still the manipulator of the client, who performed the button click. This action is very useful for operations such as "kicking out" or "private chat invitation".

```

function _onUniversalCall(&$clients, $senderName, $eventName, $value)
{
    if($eventName ==
        ChatEvents::$EVENT_BASICBUTTONCLICKED_WITHSELECTEDCLIENTS)
    {
        //if Button1 was pressed
        if($senderName=="Button1")
        {
            //get manipulator of client who performed this action
            $clientWhoIsKickingOut = $clients[0];
            $clientWhoIsKickingOut->SendPrivateMessage(
                "Admin",
                $clientWhoIsKickingOut->getUID(),
                "000000",
                "You are attempting to KickOut some clients."
            );
            //kickout all clients selected in ListOfUsers
            for($a = 1; $a<sizeof($clients); $a++)
            {
                $clients[$a]->SendPrivateMessage(
                    "Admin",
                    $clients[$a]->getUID(),
                    "000000",
                    "You were kicked out by " .
                    $clientWhoIsKickingOut->getNickName()
                );
                //disconnect the user
                $clients[$a]->RejectConnection();
            }
        }
    }
}

```

If action is **ShowDialog_SelectEmoticon**, then after the button is pressed a dialog with a set of smiles or emoticons is shown.

ShowDialog_SelectColor forces the Radical Flash Chat to display a dialog, where the user can choose a text color - used only for the chat message text. The color palette must be explicitly defined by the graphic-designer. The minimum number of colors is 1, the maximum number is 20.

```
<OnClick>
  <Action>ShowDialog_SelectColor</Action>
  <!-- Palette can contain at most 20 colors -->
  <Palette>
    <Color>990000</Color>
    <Color>FF0000</Color>
    <Color>CC9933</Color>
    <Color>FA9100</Color>
    <Color>000000</Color>
    <Color>006600</Color>
    <Color>00CC00</Color>
    <Color>99CC00</Color>
    <Color>000066</Color>
    <Color>0000CC</Color>
    <Color>0099FF</Color>
    <Color>9933FF</Color>
  </Palette>
</OnClick>
```

ShowDialog_SelectCamera - display a dialog, where the user can select the camera, microphone and configuration of the multimedia stream. The designer-graphic must specify all properties of all available stream configurations (element `<StreamingConfiguration><AVOption>` - at least one AVOption must be defined). If the user decides to enable web camera capturing by selecting one of the streaming configurations and clicking on apply button, then event `onClientStartStreaming` (Radical Web Service, See paragraph 3.3.4) is triggered. Subsequently the live audio video stream is published.

```
<OnClick>
  <Action>ShowDialog_SelectCamera</Action>
  <!--
    Dialog could be shown immediatelly after
    XML configuration is loaded into the Radical Flash Chat
  -->
  <AutoShowDialog>>false</AutoShowDialog>
  <!-- Configuration for the streaming -->
  <StreamingConfiguration>
    <!-- Name of the stream -->
    <StreamName>Stream1</StreamName>
    <!-- Is user able to stop streaming? -->
    <AVOption_NoCamera>>false</AVOption_NoCamera>
    <!--
      User could select one of the following AVOptions
      in modal dialog. There must be at least one AVOption.
    -->
```

```

<AVOption>
  <!-- Short text representation of this option in dialog -->
  <Title>Standart configuration</Title>
  <!-- Longer description of this option -->
  <Description>
    Standart configuration 320x240 pixels, 10 FPS, high quality.
  </Description>
  <!-- only one AVOption could be default -->
  <IsDefault>>true</IsDefault>
  <!-- audio setup -->
  <Audio>
    <!-- Audio kilobit rate -->
    <Kbitrate>32</Kbitrate>
    <!-- Silence level, int number in interval from 0 to 100 -->
    <SilenceLevel>5</SilenceLevel>
  </Audio>
  <!-- video setup -->
  <Video>
    <!-- Resolution of the video broadcasted to the room -->
    <Resolution>
      <Width>320</Width>
      <Height>240</Height>
    </Resolution>
    <!-- Video kilo bit rate -->
    <Kbitrate>320</Kbitrate>
    <!-- Video frames per second -->
    <FPS>10</FPS>

    <!-- Quality of the video. Min value 10, max value 100. -->
    <Quality>80</Quality>
    <!-- Interval of the video key frame -->
    <KeyFrameInterval>30</KeyFrameInterval>
  </Video>
</AVOption>

<AVOption>
  ...
</AVOption>

<AVOption>
  ...
</AVOption>
</StreamingConfiguration>
</OnClick>

```

Action **SendMessage** performs “send chat message” operation, which transfers the text from `<ChatInput>` component to the chat room `<ChatText>` component.

4.7.4 HtmlTextArea

HtmlTextArea is a sophisticated text label. The text inside may be formatted by HTML tags and some other rules typical of hypertext documents. Since it is strictly

denied to use characters as "<" lower, ">" greater in XML element body - the graphics-designers must use substitution "{{" lower, "}}" greater instead of "<", ">".
 HtmlTextArea example is below:

```
<SkinObject>
  <Type>HtmlTextArea</Type>
  <Name>HtmlText1</Name>
  <Position>
    <Left>15</Left>
    <Top>100</Top>
    <Width>400</Width>
    <Height>100</Height>
  </Position>
  <HtmlText>
    {{FONT SIZE="12" FACE="_sans" COLOR="#FFAA00" LETTERSPACING="0"}}
    SOME TEXT
    {{/FONT}}
  </HtmlText>
  <IsTextSelectable>>false</IsTextSelectable>
</SkinObject>
```

4.7.5 Specialized components

Specialized components (ListOfUsers, ChatText, ChatInput) may be used only in layer <ML> and instantiated only once at most. Obviously specialized objects cannot be dynamically created or deleted and the possibility of their interactive changes by dynamic skinning is very limited.

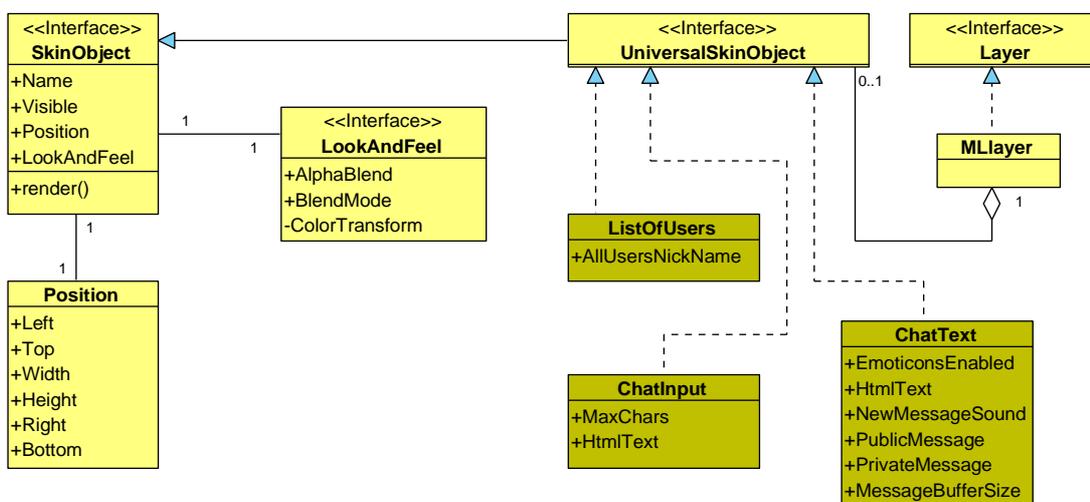


FIGURE 14. Specialized components

4.7.6 ListOfUsers

ListOfUsers is a rectangular widget with an optional scrollbar. It contains a list of all users in the current chat room.

```
<SkinObject>
  <Type>ListOfUsers</Type>
  <Name>ListOfUsers</Name>
  <Visible>true</Visible>
  <AllUsers>
    <NickName>All users</NickName>
  </AllUsers>
  <Position>
    <Left>15</Left>
    <Top>410</Top>
    <Right>125</Right>
    <Bottom>100%-15</Bottom>
  </Position>
</SkinObject>
```

4.7.7 ChatText

The most important specialized component in the main layer is ChatText. It is a dynamic read only text field designed for displaying public or private text messages. ChatText contains included HTML text shader technology. The text shader enables the graphic-designer to determine how the text messages will look like, when they appear in the ChatText. A reminder here is: chat text messages can originate only at Radical Flash Chat or at Radical Web Service, but through Radical Bridge they pass in a plain message format. After that, they are relayed to the client(s), transformed from plain format into HTML format and finally, displayed in the web browser. The plain message format is always encoded in UTF-8 character set. The message packet has enclosed information about client, who wrote the text, addressee and message significant color. This information can be used by text shader to shift message from plain format into the HTML formatted text. Everything is controlled by <PublicMessage><Template> and <PrivateMessage><Template> elements. See following example:

```
<SkinObject>
  <Type>ChatText</Type>
  <Name>ChatText</Name>
  <Visible>true</Visible>
```

```

<Position>
  <Left>430</Left>
  <Top>45</Top>
  <Right>100%-10</Right>
  <Bottom>100%-15</Bottom>
</Position>
<EmoticonsEnabled>>false</EmoticonsEnabled>
<!-- Default invitation text -->
<HtmlText>
  {{FONT SIZE="12" COLOR="#FFAA00"}}Welcome in the room.{{/FONT}}
</HtmlText>
<!-- Sound which is played, when a new message arrive. -->
<NewMessageSound>./skin/sounds/newmessage.mp3</NewMessageSound>
<!--
Public message HTML text shader.
You can use:
{message.getColor()} - return color of the message,
{message.sender.getNickName()} - return nick name of the sender,
{message.sender.getUID()} - return uid of the sender,
{message.getText()} - return text of the message
{datetime.format("string")} - format actual time
  %a - lowercase am or pm,           %A - uppercase AM or PM
  %d - day of month 01-31 (leading 0), %D - day of month 1-31
  %g - 12-hour 00-11 (leading 0),     %G - 12-hour 0-11
  %h - 24-hour 00-23 (leading 0),    %H - 24-hour 0-23
  %i - minutes 00-59 (leading 0),    %I - minutes 0-59
  %m - numeric month 01-12 (leading 0), %M - numeric month 1-12
  %N - month (January),              %n - month (Jan)
  %s - seconds 00-59 (leading 0),    %S - seconds 0-59
  %y - 2-digit year,                 %Y - 4-digit year
-->
<PublicMessage>
  <Template>
    {{P ALIGN="LEFT"}}
    {{FONT SIZE="12" COLOR="#{message.getColor()}" LETTERSPACING="1" }}
    {{B}}{message.sender.getNickName()}{{/B}}
    {{/FONT}}
    {{/P}}

    {{P ALIGN="LEFT"}}
    {{FONT FACE="_sans" SIZE="12" COLOR="#{message.getColor()}"}}
    {message.getText()}
    {{/FONT}}
    {{/P}}

    {{P ALIGN="RIGHT"}}
    {{FONT FACE="_sans" SIZE="8" COLOR="#000000"}}
    {datetime.format("%Y.%m.%d %h:%i:%s")}
    {{/FONT}}
    {{/P}}
  </Template>
</PublicMessage>

<!--
Same options as for public message element
with following new options
{message.receiver.getNickName()}
{message.receiver.getUID()}
-->
<PrivateMessage>
  <Template>
    {{P ALIGN="LEFT"}}
    {{FONT SIZE="12" COLOR="#{message.getColor()}" LETTERSPACING="1" }}

```

```

{{B}}{{I}}
{message.sender.getNickName()} &lt;&gt;&lt;&gt;
{message.receiver.getNickName()}
{{/I}}{{/B}}
{{/FONT}}
{{/P}}

{{P ALIGN="LEFT"}}
{{FONT FACE="_sans" SIZE="12" COLOR="#{message.getColor()}"}}
{message.getText()}
{{/FONT}}
{{/P}}

{{P ALIGN="RIGHT"}}
{{FONT FACE="_sans" SIZE="8" COLOR="#000000" }}
{datetime.format("%Y.%m.%d %h:%i:%s")}
{{/FONT}}
{{/P}}
</Template>
</PrivateMessage>
<MessagesBufferSize>15</MessagesBufferSize>
</SkinObject>

```

4.7.8 ChatInput

ChatInput component is a simple text input field, where the user can type a message and later send it. This input field can be styled by HTML tags. There is a `<MaxChars>` property to limit the maximum count of letters the user can type. The example below illustrates this feature:

```

<SkinObject>
  <Type>ChatInput</Type>
  <Name>ChatInput</Name>
  <Visible>true</Visible>
  <MaxChars>512</MaxChars>
  <SendMessageWhenEnterKeyPressed>
    true
  </SendMessageWhenEnterKeyPressed>
  <HtmlText>
    {{FONT SIZE="14" FACE="Courier" COLOR="#FFAA00"
      LETTERSPACING="0" KERNING="0"}}
  </HtmlText>
  <MessageSentSound>./skin/sounds/sending.mp3</MessageSentSound>
  <Position>
    <Left>145</Left>
    <Top>410</Top>
    <Right>405</Right>
    <Bottom>100%-125</Bottom>
  </Position>
</SkinObject>

```

4.8 Dynamic skinning

4.8.1 Introduction to dynamic skinning

Dynamic skinning is the last revolutionary innovation in the current version of Radical Chat. Dynamic skinning enables developers to change the application design and its behavior on the fly. Moreover, the data flow is designed "upside down" - changes can originate only on the server (Radical Web Service) and are passed down to the client'(s) web browser(s) without any previous interaction or request from the user(s). That means: application skin and logic could be changed any time and for many clients at once (e.g. as a response to mobile call, mobile SMS, or as a scheduled task). For convenience of users, the dynamic skinning technique was encapsulated only into two functions.

To alter the skin of any Radical Flash Chat application the developer needs a kernel class Skin (included in servicehandler.php), which contains two important methods: updateSkinObject and deleteAllSkinObjectUpdates. See the following demonstration (used programming language - PHP):

```
function _onClientWantsToStartStreaming(&$client, $stream)
{
    Logger::func("onClientWantsToStartStreaming");
    //client can publish the stream
    $client->setStreamingAccepted();

    //get StreamServerCommand class for communication
    //with Radical Bridge
    $ssc = $client->getStreamServerCommands();
    //try to retrieve Skin manipulator for $client
    $skin = $ssc->client_getSkin($client->getUID());
    //update or create SkinObject Panel5 in client's web browser
    $skin->updateSkinObject("
        <Skin>
        <BG>
        <SkinObject>
        <Type>BasicPanel</Type>
        <Name>Panel5</Name>
        <Visible>true</Visible>
        <Position>
        <Left>50</Left>
        <Top>-40</Top>
        <Right>410</Right>
        <Bottom>360</Bottom>
        </Position>
        <LookAndFeel>
        <NineGrid>
        <SliceImgSequence>
        ./skin/panels/rsp1_0{1-9}.png
        </SliceImgSequence>
```

```

        </NineGrid>
        </LookAndFeel>
        </SkinObject>
        </BG>
    </Skin>
    " );

    //get room skin manipulator
    $room_skin = $ssc->room_getSkin($client->getRoomUID());
    //destroy dynamically created object ButtonStartStreaming
    $room_skin->deleteAllSkinObjectUpdates("ButtonStartStreaming");
}

```

These two functions are absolutely sufficient for any skin and behavioral dynamic changes, but there are some rules which must be followed:

- I. updateSkinObject method can update only one SkinObject at most. If the developer wants to update more SkinObjects, then updateSkinObject must be executed several times.
- II. The given XML path must be the full path to the SkinObject including XML elements <Skin> and layer(<BG><ML><FG>). It is possible to change all properties of SkinObject at once. If SkinObject with a given <Name> does not exist in the Radical Flash Player, then it is dynamically created, otherwise the existing object is updated. <Type> of the old and new SkinObject must be the same.
- III. deleteAllSkinObjectUpdates function returns SkinObject back to the stage before Radical Flash Chat established connection with stream server. That means - if an object was created dynamically, then it is deleted, otherwise the object is restored to the state specified in XML configuration file (see paragraph 4.3 Static skin).

4.8.2 Implementation of dynamic skinning

From PHP or ASP.NET developer point of view it may seem that dynamic skinning is a trivial thing. However, its implementation in Radical Bridge and Radical Flash Chat is extremely complicated. Radical Bridge runs in many parallel threads, processing many requests at once, therefore it is very hard to preserve the correct order of skin changes. Moreover, each client has different CPU speed and different connection bandwidth to the Internet – so, the final synchronization of room skin changes is very tricky. Fortunately, if the rules settled above (See paragraph 4.8.1) are

complied, then and only then it is possible to reconstruct the correct sequence of dynamic skin changes, even if a room and a client skin changes are mixed together.

4.8.3 Three levels of synchronization

The first (highest) level: before the first attempt to change dynamic skin is made, Radical Web Service and Radical Bridge together negotiate so-called SCUIDs (sequence of command unique identifiers). SCUIDs are booked in the protected synchronized section unaffected by any parallel problems, so the sequence of command identifiers blocked for the triggered event function is unique within whole Radical Chat system. SCUID number constantly increases by 1, which is the basic element that preserves the correct sequence of changes.

The second synchronization moment is built in the Radical Bridge core. When chat room skin updates arrive from the Radical Web Service, they are pushed into RSCS (room skin changes stack), but only in case that their SCUIDs are greater than the SCUIDs of the commands on the top of the stack. RCSC is very helpful while a new client enters the chat room - all changes of all SkinObjects from the top of the stack are used for automatic construction of a complementary XML configuration file and immediately sent to the newly connected client to update (synchronize) his/her application skin and behavior with other users in the room.

The last synchronization moment is built in the kernel of Radical Flash Chat. Every time, when a new dynamic skin command arrives and `<LookAndFeel>` section of the object is changed, then Radical Flash Chat cannot know, if the command is actual, or if it was replaced by a new one, which is right on the way. So it starts constructing a new invisible SkinObject according to the last known `updateSkinObject` order, and when the construction is completed Radical Flash Chat waits another 250 milliseconds before the change is published. During the loading period Radical Flash Chat freezes the screen (hourglasses are displayed). Only in case that `<LookAndFeel>` section of the updated object is not modified, then the screen is not frozen and changes are done immediately.

4.8.4 Threats and limits of dynamic skinning

The maximum number of all dynamic skin updates, which can be done in one PHP or ASP.NET triggered event is 32 (by default). All other updates are ignored.

It is not recommended to use relative links to dynamically created objects (in RPSF positioning system), because RPSF system has higher priority and is used before skin updates are put into the correct order – that may occasionally cause unexpected errors and illogical infinite loops in dependency graph (See figure 11).

It is not recommended to use dynamic skinning for one client, or whole chat room in intervals lower than three seconds.

5 CONCLUSION

5.1 Results

The Radical Chat multimedia videoconferencing framework – version 1.0.0 rc2 (revision 1347) functions very well within the specified parameters. The most important features supported by the recent version of the framework are RPSF positioning system and dynamic skinning. Each part of the Radical Chat was implemented as well as was possible with an increased emphasis on the performance of the system. Right now Radical Chat is used by three enterprise solutions.

5.2 Personal experience

This was for first time, when I worked as a contractor for a Finnish company. From the beginning, our work was well advised. We used RUP (Rational Unified Process) and UML modeling language to state all demands on the system. SVN (Subversion technology) was used as a source code synchronization tool. Our cooperation went very well and the framework and dependent applications were done in four months and two weeks. I hope I have met all company requirements and needs.

During my work I have learned many new practical things about Adobe Flash technology including stream server API and RTMP protocol. I have extended stream server (written in java) and redeveloped the way for communication with PHP and ASP.NET in a highly heterogeneous environment. The framework and applications written in it are now stable.

This project was the most challenging one in my life, but the work did not stop. I am still adding new functions, revising source code and extending the framework in all dimensions.

5.3 Further improvements

There are a few new demands on the system functionality. The praxis proved that some improvements and upgrades are necessary. I am about to create a sound engine to enable developers manage the sound effects and music – usage will be very similar to “dynamic skinning”. I am glad to announce that the new universal component “trackbar” will be included in the next version of the Radical Chat. Finally, a future version of Radical Chat (May or June 2010) might be published under the “free for noncommercial use” license.

6 REFERENCES

Adobe Action Message Format 0, November 2009,

http://download.macromedia.com/pub/labs/amf/amf0_spec_121207.pdf

Adobe Action Message Format 3, November 2009,

http://download.macromedia.com/pub/labs/amf/amf3_spec_121207.pdf

Adobe Actionscript 3 reference, October 2009,

<http://livedocs.adobe.com/flash/9.0/ActionScriptLangRefV3/>

Adobe Actionscript 3, November 2009,

<http://en.wikipedia.org/wiki/ActionScript>

Adobe Flash, September 2009,

http://en.wikipedia.org/wiki/Adobe_Flash

Adobe Real Time Messaging Protocol, November 2009,

<http://www.adobe.com/devnet/rtmp/>

Adobe Real Time Messaging Protocol RTMP, November 2009,

http://en.wikipedia.org/wiki/Real_Time_Messaging_Protocol

Multimedia framework, November 2009,

http://en.wikipedia.org/wiki/Multimedia_framework

Rich internet applications, September 2009,

http://en.wikipedia.org/wiki/Rich_Internet_application

Wowza Media Server Pro, November 2009,

<http://www.wowzamedia.com/>