ulm university universität ulm

Technical Report VS-R08-2010

Aspectix Research Team | Institute of Distributed Systems | University of Ulm, Germany

RAPIX
A plug-in based RIA for multimedia communication

Jan-Patrick Elsholz · Eduard Seibel · Franz J. Hauck

2010

ART
Aspectix Research Team

RAPIX: A plug-in based RIA for Multimedia Communication

Jan-Patrick Elsholz, Eduard Seibel, Franz J. Hauck
Institute of Distributed Systems
Ulm University, Germany
Email: {jan-patrick.elsholz,eduard.seibel,franz.hauck}@uni-ulm.de

***Abstract*—Multimedia communication like voice and video calls over IP or Video on Demand grows in popularity today [1], [2]. However, the development of such applications is a complex and error-prone process, as only basic framework support is available. In this paper we introduce RAPIX, a plug-in based Rich Internet Application (RIA) that eases this development. Furthermore, our web-based approach supports a wide range of devices enabling even mobile phones for multimedia communication. We prove our concept with a prototype implementation based on the Rich AJAX Platform [3] enriched with Adobe Flash to support multimedia playback and access audio and video capture from the web browser. This enables RAPIX to cope with multiple, hot pluggable applications and takes the complexity of the web browser integration off the application programmer. Common functionality like window management, a notification system, and a generic resource list are integrated as well.**

I. Introduction

Multimedia communication becomes more and more widespread in the mobile world; consider for example the availability of Skype for mobile phones. However, such applications for the mobile market are mostly targeted at specific hardware platforms like the iPhone. Several approaches have been made to form a common base on all devices, e. g. Android. Alternatively, a web browser is already available on most devices today. Thus, RAP for Instant-X (RAPIX) is a plug-in based Rich Internet Application (RIA), capable of multimedia playback and access to client devices like webcam and microphone. It offers a Java-only Application Programming Interface (API) to hide the complexity of web browser integration. All necessary visual components are transparently created and automatically displayed. The application developer writes exclusively Java code to deploy applications for RAPIX. Furthermore, pluggable applications can be added to and removed from RAPIX even at runtime. To enable this hot pluggable integration, we use OSGi [4] in our prototype implementation. RAPIX is based on the Rich AJAX Platform (RAP) [3] enriched with Adobe Flash to enable multimedia playback and device access. Furthermore, common functions of applications for multimedia communication like a generic resource list and a notification system are integrated into the API of RAPIX. We advocate, that there are several advantages of a RIA for multimedia communication over a native client. First, the high availability of web browsers on a wide range of platforms supports the platform independency and thus highly eases application development. Second, applications can be used worldwide from any web browser without installation and maintenance issues. And third, the RIA has a consistent look-and-feel across all devices and platforms, improving user acceptance and orientation in the User Interface (UI).

The paper is structured as follows. First, we show related work and categorise it related to placement of application logic and access to system resources. Then, basic technology is introduced. Section IV presents our prototype implementation with device access, multimedia playback, a notification system and a generic resource list. Finally, we conclude in Section V.

II. Related Work

Many frameworks and different technologies support the development of RIAs for multimedia communication. In Figure 1 we place those according to their access to system resources and the residence of application logic. We distinguish the latter in arrange from thin client to fat client. On a thin client the application runs inside a web browser, whereas on a fat client the application might run without any connection to the Internet or even outside a web browser. The system resources ascend from simple I/O to graphics processing unit (GPU), each time adding the specific resource access to the others.

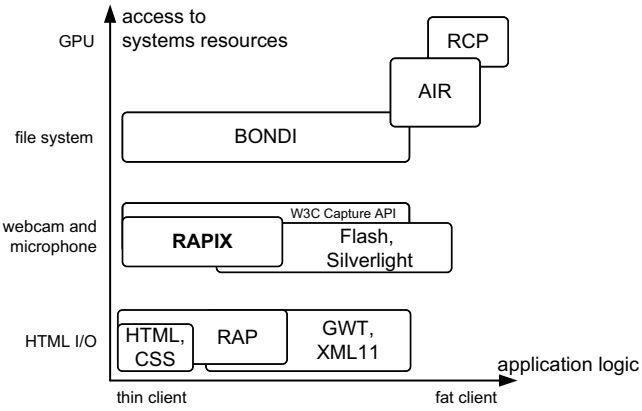

Fig. 1. RAPIX in comparison to related technologies.

In the lower left corner HyperText Markup Language (HTML) [5] and Cascading Styling Sheets (CSS) [6] represent the standard web client with access to screen and input values like forms. Opposite in the upper right corner Rich Client

Platform (RCP) [7] represents applications running outside of a web browser with full access to any hardware resource and high interactivity with the user. The Java frameworks RAP [3], XML11 [8] and Google Web Toolkit (GWT) [9] offer tools to create web applications but no hardware access. The web browser plug-ins Adobe Flash [10] and Microsoft Silverlight [11] offer access to webcam and microphone using proprietary protocols and programming languages. The capture API [12] of the World Wide Web Consortium (W3C) combined with HTML 5 [13], BONDI [14] and PhoneGap [15] are JavaScript based APIs to access hardware on client devices like mobile phones. The technologies Adobe Integrated Runtime (AIR) [16], JavaFX [17] and Google Gears [18] make web applications available offline. Therefore, most of the application logic resides on client side. Due to computational limitations on mobile devices we focus on thin clients still capable of accessing webcam and microphone. Thus, we merged RAP and Adobe Flash to RAPIX.

III. Basic Technologies

In the following section we describe basic technologies and concepts our prototype is based on.

A. MVC

Our implementation follows the Model-View-Controller (MVC) pattern [19]. A strict separation of presentation, data access and business logic allows decoupling all three of them. In our case, the view is based on RAP and implemented partly in JavaScript, Adobe Flash, HTML and CSS, whereas model and controller are exclusively written in Java. Thus, we gain a loose coupling of components and a transparent usage of different programming languages. This highly eases development of pure Java applications and allows adaptation of the View according to device capabilities, e. g. Tabbed Document Interface (TDI) instead of the desktop metaphor for mobile devices with small screens.

B. OSGi

OSGi [4] is a specification of a lean, Java-based component framework defined by the OSGi Alliance. It offers life-cycle management and automatic dependency resolution of components at runtime. Components are standard Java archives containing a manifest file for metadata. It is used to share common functionality and libraries between components in terms of imported and exported Java packages. Optionally, components export and import *services*. They implement a regular Java interface and are registered with optional metadata at a *service registry*. However, the resolution of service dependencies is left to the component developer. Therefore, OSGi provides a *service tracker* for monitoring the availability of service components. The OSGi implementation Equinox introduced the extension point (EP) mechanism [20] as a declarative plug-in concept for the Rich Client Platform Eclipse [21]. Similar to OSGi services, EPs support dynamic extension of applications at runtime [22]. In contrast, EPs use XML declarations to import and export the extension. Thus, an *extension registry* and an *extension tracker* are provided by Equinox. The OSGi specification later introduced a similar concept named *declarative services* [23]. All of these mechanisms support the development of highly dynamic applications. We focus on EPs due to the usage in RAP.

C. Instant-X

Our idea is based on the multimedia middleware Instant-X [24], [25]. It offers a service-oriented architecture (SOA) based approach for the development of applications for multimedia communication. Common functionality like data transmission and signalling is encapsulated into services. This eases the development of such applications. Furthermore, applications become independent of specific protocols by using the offered APIs exclusively. Thus, applications, API and protocol implementations are encapsulated into OSGi components.

D. RCP

The Rich Client Platform [7] is a framework for the development of rich client applications. It derived from the Eclipse project [21] and is based on OSGi. Amongst others, it offers an abstract programming model for a graphical user interface (GUI) based on the Standard Widget Toolkit (SWT) [26] following the MVC pattern. This takes the cross-platform burden off the application developer. The EP mechanism is used to cope with the dynamic integration of applications at runtime. However, web clients are not supported.

E. RAP

The Rich AJAX Platform [3] brings RCP [7] to the web browser [27]. It is based on the open source Asynchrounous JavaScript and XML (AJAX) [28] implementation Qooxdoo [29] and enables the development of AJAX-enabled RIAs like Gmail [30]. Similar to RCP, it offers an abstract Java-only API [31] for the development of GUIs. Thus, it hides the complex web browser integration from the application developer [8]. No knowledge of HTML, CSS or JavaScript is needed. Following the MVC pattern, multiple views are allowed to access a single model. Therefore, it is possible to use a single model for multiple users of a single application as well. Also, RAP offers no support for multimedia communication.

F. RTMP

The Real Time Messaging Protocol (RTMP) [32] by Adobe is a multimedia streaming protocol. It uses in-band control messages for the playback in the Adobe Flash player. We use the open source implementation Flazr [33] as server side software to access the Adobe Flash based device API (Section IV-A)

IV. Implementation

In this section we introduce the prototype implementation of RAPIX. It is based on RAP and extended for multimedia communication. In our prototype we use a desktop metaphor with a taskbar (Figure 2, no. 1) and freely resizable windows (Figure 2, no. 2) for each application. Any other window

Fig. 2. Screenshot of RAPIX.

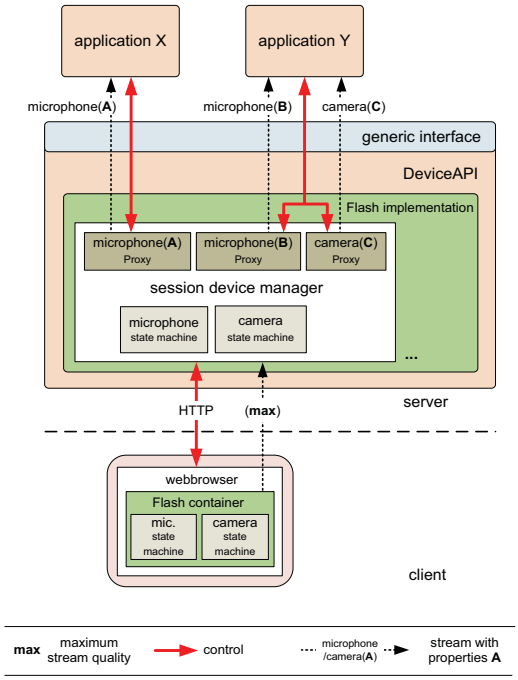

Fig. 3. Architecture of the device access

management is possible and dynamically adjustable, e. g. one tab per application as commonly seen on mobile devices. Leaving details of the window management aside, we focus on device access, multimedia playback (Figure 2, no. 3), our notification system (Figure 2, no. 4) and a generic resource list (Figure 2, no. 5).

A. Device access of webcam and microphone

The device access is based on Adobe Flash. We developed a custom widget and integrated Adobe Flash into RAP. This is done using the `ExternalInterface` of ActionScript 3.0 [34] as a bridge between JavaScript and Adobe Flash. Thus, it is possible to access microphone and webcam on any device capable of running a web browser and Adobe Flash.

1) Architecture for device access: Figure 3 shows the architecture of the device access. The main idea is to use proxies for all physical devices allowing simultaneous access from multiple applications.

Application X wants to access the client's microphone. Thus, a proxy (A) is created. The hot plugged-in application Y wants to access the same microphone and the client's camera. Another proxy (B) for the microphone and a proxy (C) representing the camera are created. Proxies are created by the `session device manager`, which is responsible for all devices in a single session. Each session represents one web browser displaying RAPIX as view of the MVC model. A duplicated state machine on client and server side is used to synchronise the device and its proxies. The server side proxy is accessible through a Java-only API, hiding the concrete implementation. The multimedia stream from the client device to the `session device manager` is of maximum quality, enabling each proxy to scale to the appropriate formats acquired by the application. In our prototype we utilise Adobe Flash to implement this device access. Other implementations like Microsoft Silverlight are possible as well.

2) Duplicated state machine: In our architecture, the client device and its proxies may be on different machines. Thus, duplicated state machines synchronise the state machine of the proxy with the current state of the client device. Additionally, we implemented a kind of reference counting, as releasing a proxy must not release the device. Figure 4 shows the states of our duplicated state machine.

We identified six states for accessing client devices. First, all available devices are discovered and synchronised with the server. For each device its state machine is in `INIT` state. Calling `start` acquires the appropriate device to be used within RAPIX. This is represented by the `OCCUPYING DEVICE` state. On success the state changes to `REQUESTING ACCESS`. This is necessary due to security and privacy reasons. The user of the client device has to permit its access explicitly. If either of the latter state changes fails, we fall back to the `INIT` state. Next, we try to establish the RTMP connection between server and client in the `CONNECTING` state. On success, we pass on to `CONNECTED`. Calling `start` and `stop` in this state initiates and stops a maximum quality stream represented by the `SENDING STREAM` state. For any error or calling `release` transfers the state machine to the `INIT` state. All related applications are informed and the device is released.

3) Device API: Figure 5 shows how to access the microphone from an application using RAPIX.

In line 1 we obtain the appropriate `session device manager` for the client. Line 2 creates a proxy for the microphone. In line 3 we get the multimedia stream of the

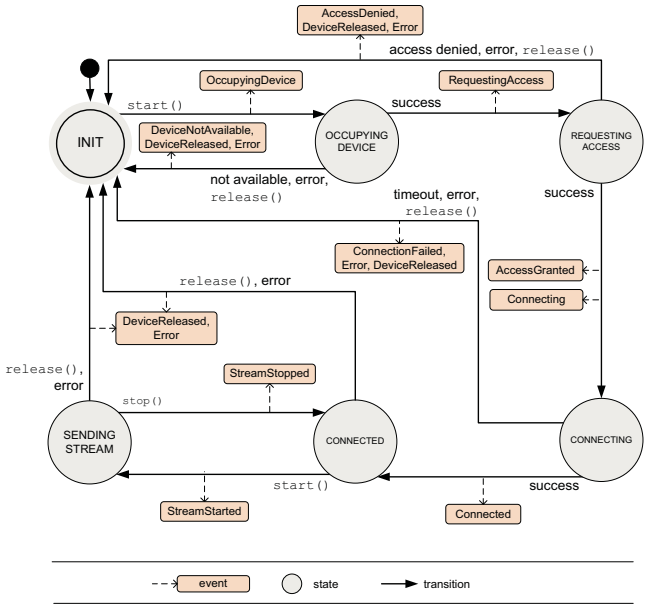

Fig. 4. Duplicated state machine of the device API.

```
1  DeviceManager dm = DeviceManagerFactory.INSTANCE;
2  Device microphone = dm.getDefaultMicrophone();
3  Stream micStream = microphone.getStream();
4  micStream.addStreamListener( new StreamListener() ){
5      void mediaPacketReceived( StreamEvent event ){
6          // event.packet contains a stream packet
7          ...
8      }});
9  micStream.start();
```

Fig. 5. Example for accessing the microphone of a client device.

microphone and in line 4 we add a listener. To finally start the transmission we call `start` in line 9.

B. Multimedia Playback

In this section we describe the architecture of the multimedia playback.

1) Architecture of multimedia playback: Figure 6 shows the component-based implementation hidden behind our player API.

An application streams audio and video data to a client. Therefore, we use a custom widget in RAP to integrate Adobe Flash. Due to the MVC pattern, we need a duplicated state machine on the server and client side. The RTMP server Flazr bridges between the Java world on the server side and the Adobe Flash player on the client side. Controller events like `pause` are sent to the server via RAP.

2) Player API: Figure 7 shows an example of how multimedia is played back using our API.

In line 1 we create a Player through the factory. It is placed in the given container. We start the player in line 2 and pass incoming packets in line 4. Before these packets are played back, they are first passed to the RTMP server Flazr and transmitted to the custom Adobe Flash widget.

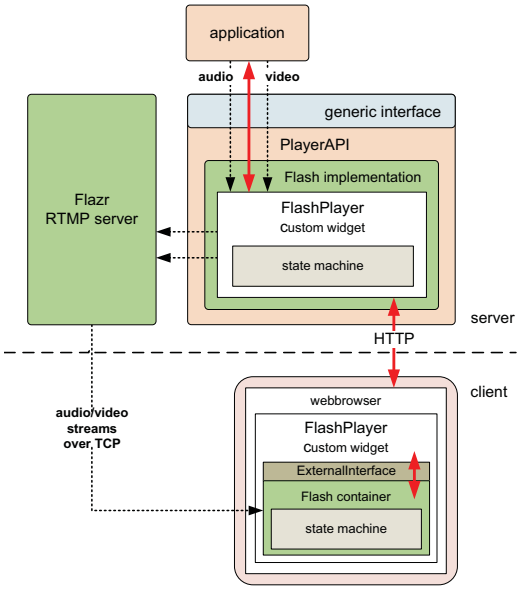

Fig. 6. Architecture of multimedia playback using Adobe Flash and Flazr.

```
1  Player p = PlayerFactory.createPlayer(container,
       container.getContentPane(), ...);
2  p.play();
3  ...
4  p.receivePacket(packet);
```

Fig. 7. Multimedia playback example.

C. Notification System

Applications for multimedia communication normally run in the background of the user until a communication request arrives. Then, a notification typically pops up to interrupt the user from the ongoing task. For example, an instant message (IM) application is a small icon in the taskbar until a window appears containing the incoming message.

1) Notification API: Figure 8 shows two possible notifications: The first is a status message at the bottom of a window. The second is a popup message in the lower right corner of the desktop.

```
1  // status
2  container.setStatus( "Connected to server." );
3
4  // popup
5  Notifier notifier = Notifier.getInstance();
6  Notification n = notifier.createNotification(
       NotificationLevel.CRITICAL, "Connection
       interrupted." );
7  notifier.notifyAll();
```

Fig. 8. Notification system example.

In line 2 we set the status message of a specific window. In line 5 we obtain the notifier as singleton. We create a critical notification in line 6 and send it to every web GUI attached to this model. A notification with a high priority is displayed longer and may displace others.

D. Generic resource list

Many multimedia applications use some sort of lists. Typically, an IM application offers a list of known buddies. Presence information is visualized by highlighting the appropriate entries. Even Video on Demand (VOD) applications need a kind of list. They list available movies. Thus, we integrated a generic resource list into our plug-in based RIA offering all kinds of possible usage. Following the MVC pattern, the application developer simply takes care of the model whereas the view is generated automatically according to the dynamically adjustable window management of our plug-in based RIA.

1) Resource list API: We demonstrate the application programmers view of our generic resource list for an IM scenario in Figure 9.

```
1 ResourceList list = new ResourceList();
2 list.add(new ListItem("user A", "192.168.178.20",
      ...));
3 list.add(new ListItem("user B", "192.168.178.23",
      ...));
```

Fig. 9. Generic resource list example for an IM scenario.

In line 1 we create a new resource list. Two users A and B are added in line 2 and 3. Additionally, their IP addresses are stored in the list. Optionally, an icon and tool tips can be passed as well. Our generic resource list supports the basic operations create, read, update and delete (CRUD).

V. Conclusion

In this paper we present RAPIX, a RIA supporting hot pluggable applications for multimedia communication. In contrast to related work, we focus on a web client with access to audio and video capture. We present a prototype implementation based on RAP enriched with Adobe Flash to gain access to devices and enable multimedia playback from the web browser. Furthermore, we offer windows management, a notification system and a generic resource list. This is transparent to the application programmer through a Java-only API hiding the complexity of web browser integration. This highly eases the development of multimedia applications. More than that, it enables worldwide multimedia communication on nearly any device with a web browser installed. In future work we would like to replace Adobe Flash with HTML5 and the W3C Capture API for device access on client devices. An evaluation of our API shall reveal conceptional problems and missing features.

References

[1] Y. Wang, M. Claypool, and Z. Zuo, "An empirical study of realvideo performance across the internet," in *IMW '01: Proceedings of the 1st ACM SIGCOMM Workshop on Internet Measurement*. New York, NY, USA: ACM, 2001, pp. 295–309.

[2] H. Verkasalo, "Empirical observations on the emergence of mobile multimedia services and applications in the US and Europe," in *Proceedings of the 5th international conference on Mobile and ubiquitous multimedia*. ACM, 2006, p. 3.

[3] "Rich Ajax Platform," http://www.eclipse.org/rap/, Eclipse Foundation, 2010.

[4] "OSGi Service Platform Core Specification," The OSGi Alliance, Juni 2009.

[5] I. J. Dave Raggett, Arnaud Le Hors, "HTML 4.01 Specification," http://www.w3.org/TR/html4/, W3C, Dezember 1999.

[6] B. Bos, T. elik, I. Hickson, and H. W. Lie, "Cascading Style Sheets Level 2 Revision 1 (CSS 2.1) Specification," http://www.w3.org/TR/CSS2/, W3C, September 2009.

[7] "Eclipse Rich Client Platform," http://www.eclipse.org/rcp/, Eclipse Foundation, 2010.

[8] A. Puder, "A cross-language framework for developing AJAX applications," in *Proceedings of the 5th international symposium on Principles and practice of programming in Java*. ACM, 2007, p. 112.

[9] "Google Web Toolkit," http://code.google.com/webtoolkit/, Google, 2010.

[10] "Adobe Flash Player," http://www.adobe.com/de/products/flashplayer/, Adobe Systems Inc., 2010.

[11] "Microsoft Silverlight," http://www.microsoft.com/silverlight/, Microsoft, 2010.

[12] I. K. Dzung D Tran, Ilkka Oksanen, "The Capture API," http://www.w3.org/TR/2010/WD-capture-api-20100401/, W3C, April 2010.

[13] I. Hickson and D. Hyatt, "HTML5 - A vocabulary and associated APIs for HTML and XHTML (Editor's Draft 13 November 2009)," http://www.w3.org/TR/html5/, W3C, November 2009.

[14] "BONDI - An open source industry collaboration for widget and web technologies," http://bondi.omtp.org/, OMTP, 2010.

[15] "PhoneGap - Web-enable native mobile device functionality," http://www.phonegap.com/, Nitobi Inc., 2010.

[16] "AIR. Adobe Integrated Runtime for Rich Internet Applications," http://www.adobe.com/de/products/air/, Adobe Systems Inc., 2010.

[17] "JavaFX," http://www.sun.com/software/javafx/, Oracle, 2010.

[18] "Google Gears," http://gears.google.com/, Google, 2010.

[19] T. Reenskaug, "Models-views-controllers," *Technical note, Xerox PARC*, Dezember 1979.

[20] M. Henning and H. Seeberger, "Einfhrung in den Extension Point-Mechanismus von Eclipse," *JavaSPEKTRUM*, vol. 1, pp. 19–24, 2008.

[21] "Eclipse.org," http://eclipse.org/, The Eclipse Foundation, 2010.

[22] N. Bartlett, "Eclipse extensions versus OSGi services," http://www.eclipsezone.com/articles/extensions-vs-services/, EclipseZone, Februar 2007.

[23] "OSGi Service Platform Service Compendium," The OSGi Alliance, August 2009.

[24] J.-P. Elsholz, H. Schmidt, S. Schober, F. J. Hauck, and A. J. Kassler, "Instant-X: Towards a Generic API for Multimedia Middleware," 12 2009.

[25] H. Schmidt, J.-P. Elsholz, and F. J. Hauck, "Instant-x: a component-based middleware architecture for a generic multimedia api," in *Companion '08: Proceedings of the ACM/IFIP/USENIX Middleware '08 Conference Companion*. New York, NY, USA: ACM, 2008, pp. 90–92.

[26] "SWT: The Standard Widget Toolkit," http://www.eclipse.org/swt/, Eclipse Foundation, Februar 2010.

[27] F. Lange, *Eclipse Rich Ajax Platform - Bringing Rich Clients to the Web*. Apress, 2008.

[28] J. J. Garret, "Ajax: A New Approach to Web Applications," http://adaptivepath.com/ideas/essays/archives/000385.php, adaptive path, Februar 2005.

[29] "Qooxdoo. Open Source Ajax Framework," 1&1 Internet AG, 2010.

[30] "Gmail: Google's approach to email," http://mail.google.com/mail/help/intl/en/about.html, Google, 2010.

[31] "RAP Developer Guide," http://help.eclipse.org/help33/nav/23, Eclipse Foundation, 2010.

[32] *Real-Time Messaging Protocol (RTMP) specification*, 1st ed., Adobe Systems Inc., April 2009.

[33] P. Thomas, "Flazr - A Java Implementation of Multimedia Streaming Protocols," http://www.flazr.com/, November 2009.

[34] "ExternalInterface - ActionScript 3.0 Language and Components Reference," http://livedocs.adobe.com/flash/9.0/ActionScriptLangRefV3/flash/external/ExternalInterface.html, Adobe, 2008.