An Instant Message-Driven User Interface Framework for Thin Client Applications

Matthias Book, Volker Gruhn, Gerald Mücke
Deutsche Telekom Chair of Applied Telematics / e-Business, University of Leipzig
Klostergasse 3, 04109 Leipzig, Germany; Tel. +49-341-97-32337, Fax +49-341-97-32339
{book, gruhn}@ebus.informatik.uni-leipzig.de, gerald@xuhia.org

Abstract

Today, thin client applications often rely on the infrastructure of the WWW to deliver their user interfaces (UIs) to clients. While this approach does not require the deployment of application logic on the client, web-based UIs typically do not provide the same level of usability as window-based UIs. We therefore present a UI framework that combines the flexibility of a thin presentation logic with the usability of a full-featured UI: Our approach uses an XMPP-based instant messaging infrastructure to exchange XUL interface descriptions and events between the application logic on the server and a generic UI rendering engine on the client.

1. Introduction

Many information systems are implemented as thin client applications today for reasons such as reduced infrastructure cost, reduced deployment and maintenance effort, and increased user flexibility. Often, these thin client applications take the form of web-based applications whose hypertext user interface (UI) can be easily rendered by web browsers, while all application logic remains on a central server [1].

However, for complex applications, web-based UIs typically cannot offer the same level of usability that window-based applications do, since they do not provide intrinsic support for complex widgets (e.g. tree views, type-ahead combo boxes etc.) or interaction patterns (e.g. direct manipulation of objects, multiple window panes, context menus etc.) [8]. While some of these constructs can be simulated in HTML, they are not natively supported by web browsers and thus require additional presentation logic (e.g. JavaScript embedded in web pages) or may work in unusual ways (e.g. using single instead of double mouse clicks). This increases the development effort unnecessarily while leaving users with an unsatisfactory user experience.

Besides the limitations of HTML, the underlying request-response communication pattern imposed by HTTP restricts web-based UIs further, since any interaction must be initiated by the user – the server cannot update the UI on its own by "pushing" data to the client [12]. Recently, a combination of web technologies collectively termed Ajax (Asynchronous JavaScript and XML) [2] is being touted as the solution to these problems. The approach uses JavaScript to manipulate one web page's Document Object Model (DOM) at run-time according to XML-based update instructions requested from the server. However, while Ajax provides a smoother user experience, largely due to the elimination of page jumps, it still relies on the same web technologies and therefore requires quite a bit of implementation and communication overhead in order to simulate a more sophisticated user interface.

While the implementation overhead may be ameliorated through the use of suitable frameworks, the communication overhead can be a critical factor when the thin client application shall be used on mobile networks, which are often characterized by low bandwith and high transmission costs.

In order to retain the usability of window-based applications without giving up the flexibility of a thin client architecture, we present a user interface framework that combines their benefits in this paper. Our framework enables developers to build thin client applications with window-based UIs, where clients communicate with the server using an instant messaging (IM) protocol. By exchanging user interface descriptions and events over public IM networks, clients and servers can communicate very flexibly, and the use of a powerful user interface description language allows developers to let sophisticated GUIs be rendered on the clients without the high implementation effort required for reconstructing such GUIs in a web browser.

In the following sections, we introduce the infrastructure and features of our IM framework, give an overview of the related work and conclude with an outlook on further research opportunities.

2. Instant Messaging Infrastructure

For highly interactive thin client UIs, a more flexible communication scheme than HTTP's one-sided, synchronous pull mechanism would be desirable. Ideally, both the client and the server should be able to transmit data anytime on their own initiative (i.e. push messages), and without having to wait for a response (i.e. asynchronously). These messages should require low bandwidth in order to save costs on mobile networks, and be delivered quickly in order to avoid the sluggish response often experienced in web-based applications. Existing IM protocols such as the Extensible Messaging and Presence Protocol (XMPP) [10] fulfill these requirements: Typically employed for chat networks, they are tailored to the asynchronous, nearly instantaneous transmission of small messages between peers, and thus provide an ideal medium for communicating UI events (i.e. user gestures and interface reactions) between a thin client and its application server.

In order to keep the client as thin as possible, we do not want to deploy any application-specific code on the client, but only require the presence of an IM client for handling communications, and a generic GUI engine for rendering the interface. Much like a web browser interpreting and rendering HTML code, this GUI engine interprets and renders a window-based interface described in the XML User Interface Language (XUL) [6] or a comparable format.

Figure 1 gives a coarse overview of this instant messaging infrastructure for thin client applications: The mobile or stationary client devices are running an IM client that shows the typical "buddy list" of other users on the IM network. In addition to showing online users, however, it also shows the applications that are currently accessible via the IM network. When the user selects an application from the list, the GUI engine will download and render the respective UI description markup. While the user works with the application, all client-server interactions are transmitted as instant messages to the application server, which is connected to the central IM server just like any other peer. Since both thin clients and application servers are equal peers on the IM network, they can communicate freely over the IM infrastructure and use its special features, e.g. peers' presence information, as described in the following section.

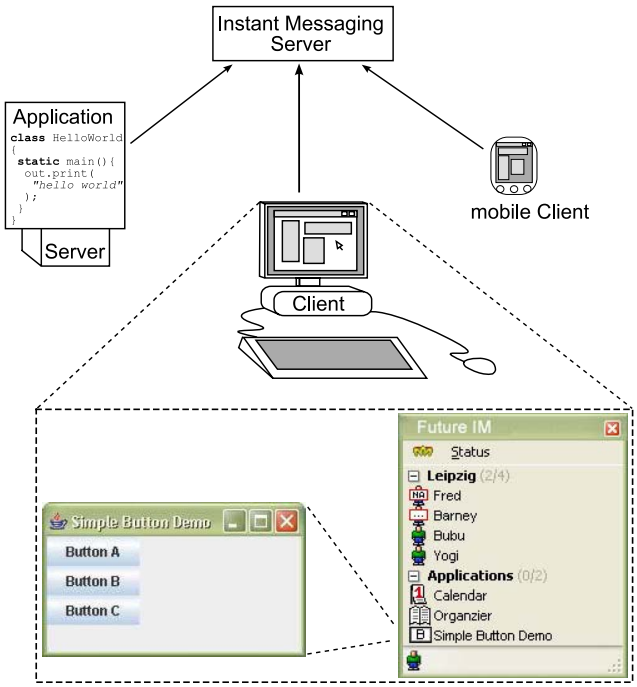

Figure 1. IM-based thin client infrastructure

3. Framework Features

To support the development of applications that can be accessed by IM-based thin clients, we developed a framework that encapsulates the technical aspects of the IM-based communication and provides high-level services like session management to the application. Our framework implementation is independent of a concrete user interface description language and IM transport protocol, but developers will typically want to use the popular XUL language and the standardized XMPP transport protocol when developing applications.

In order to reduce bandwidth usage, the framework transmits the complete UI description (i.e. the specification of all windows and their widgets) to each client only once. Since the UI structure typically remains static over time, the client can cache and re-use this description the next time the user works with the same application, without having to receive it again. After the UI description has been transmitted, the GUI engine can render the interface and let the user work with the application. From now on, only user and application events need to be transmitted over the IM network. Each event is encapsulated in a single instant message that contains all required parameters (e.g. type of the user gesture, entered or selected data, etc.), and is sent to the application server through the IM network. If necessary, the application server may then react by sending back an instant message that contains instructions for updating the view rendered on the client. The application server may also send such an event on his own initiative (e.g. triggered by the expiry of a timer, the completion of a time-intensive transaction, the fulfilment of a certain condition etc.) to modify the UI without previous user interaction. Based on instructions for modifying the DOM of the UI description, the GUI engine can then display or hide certain windows, enable or gray out certain widgets, fill content areas with application data etc.

In contrast to web-based applications, an IM-based thin client infrastructure offers a number of special characteristics that application developers can benefit from:

- **Single Sign On:** A major advantage of IM-based vs. web-based thin client infrastructures is that all users must authenticate themselves to the IM server in order to use the IM network. Consequently, IM-based applications do not need to implement their own authentication mechanisms, but can rely on the IM server to take care of checking the users' credentials. This is also convenient for users, since they need to provide their credentials only once to the IM server as the single sign-on point, and can then access all applications without having to enter their credentials again.

- **Session Management:** Since all users on the network must log in to the IM server, there are no anonymous messages – each instant message is unambiguously associated with a certain sender and receiver. Therefore, IM-based applications do not have to implement the cumbersome session identification and expiry mechanisms known from web-based applications, but can simply associate all data with users by their name.
  In addition, our framework contains additional session management logic to distinguish "sub-sessions" that are spawned when the same user simultaneously works with several instances of the same application, i.e. when he has openend several windows of the same application (a situation that is virtually impossible to detect or handle in web-based applications, where users may clone browser windows).
  Our framework can also handle persistent sessions: Since the user's application data and UI state are maintained on the server, users can close the thin client window and open it again at a later time in order to resume working where they left off. This way, users can suspend work with the thin client application (both voluntarily and involuntarily, e.g. due to a network outage) without loss of data.

- **Presence Information:** An additional feature that is unique to IM networks is the use of so-called presence information (PI) to convey status information about clients and applications. In peer-to-peer chat applications, the PI typically conveys user information such as "online", "offline", "away", "do not disturb" etc., but can also be used for application-specific status information: Using the PI mechanism, application servers can publish information about their program version or server load. Transparently for the users, the client can then update its UI description if an application's PI indicates that a new version has been deployed. Or, if multiple instances of the same application are available on the IM network, clients can connect to the application instance that publishes the lowest server load in its PI. In combination with positioning systems like GPS, clients could publish their current geographical coordinates in the PI to enable applications to provide them with location-based content and services [7].

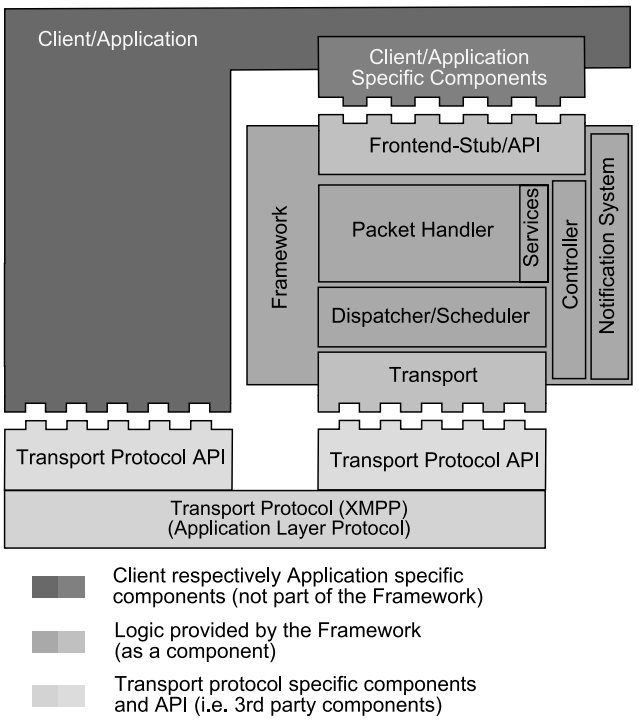

Figure 2. Framework architecture

Our framework comprises a set of components that implement the client-server communication and serve as an interface between the IM transport protocol and the application or client logic, as shown in Fig. 2. The framework is connected to the transport layer using a suitable third-party transport API, such as the open-source Smack API for the XMPP protocol [3]. Since our framework only serves as a middleware for the presentation logic, it does not require exclusive access to the transport API. Rather, if the application or client need to communicate other domain-specific data over the IM network (e.g. chat messages), they can access the transport API directly to do this, without having to go through our framework. In order to allow thin clients' UIs to remain responsive while data is exchanged with the application server, all messages are prioritized according to their type (e.g. UI updates vs. PI information) by the dispatcher/scheduler and processed asynchronously by the packet handler. While the packet processing component is the same for clients and application servers, the frontend works differently depending on whether the framework is deployed in a client or application server context. On the

client side, it serves as an API for the GUI rendering engine, while on the application side, it serves as an API for the application logic.

4. Related Work

Providing a complex GUI on a thin client has been a challenge for many years. Among the first proposed solutions was the X Window System [11], a network-transparent window manager following the client-server paradigm. In comparison to our approach, the X protocol offers more graphical flexibility since it works on a lower level, however, the low-level implementation also requires more constant and high-bandwidth network connectivity than our framework does, where the user can perform basic interactions with the user interface (e.g. sizing windows, filling form fields) without having to communicate with the server.

Originally derived from the X Window System, Virtual Network Computing (VNC) [9] is another approach that works on the frame buffer level to transmit any graphical interface to a graphics-enabled client and communicate keystrokes and mouse gestures back to the server. However, even though the VNC protocol implements a number of compression algorithms, the transmission of pure pixel data is quite data-intensive, so a high-bandwidth connection is necessary for fluent work.

The Remote Java Foundation Classes (RJFC) approach [4] is an extension to the existing JFC model, where all communication between the interface and application logic is performed by Remote Method Invocation (RMI), which keeps the bandwidth requirements low. However, the UI cannot be adapted as flexibly to different client devices' capabilities as with the user interface description language employed in our approach, and firewalls typically preclude RMI communication over wide-area networks.

In the area of web-based solutions, the Ajax approach already mentioned in the introduction [2] and Macromedia's Flex framework [5] both strive to enhance the user experience beyond simple page-to-page navigation. However, both approaches rely on HTTP as the underlying protocol and thus are also limited to its request-response communication scheme, which does not allow the server to initiate interactions on its own.

5. Conclusion

In this paper, we presented a framework for developing thin client applications using a public IM infrastructure to transmit UI descriptions and events between the server-side application logic and a generic client-side GUI rendering engine. We propose that this approach can reduce the development effort for window-based thin client applications since it does not require simulating complex GUI features in HTML or deploying presentation logic on the client, and offers additional features such as inherent single-sign on, subsession management and presence information that would otherwise have to be implemented manually.

Our initial performance tests on a public IM network yielded response times between 160 and 175 ms, which were considered noticable, but not disturbing by testers. In our ongoing work, we are striving to perform more formal response time tests in various network environments, and gain more insights into the impact of the framework on the software development process. From these factors, we can then deduce the suitability of our instant message-driven user interface framework for developing thin client applications with complex window-based user interfaces.

References

[1] M. Gaedke, M. Beigl, H.-W. Gellersen, and C. Segor. Web content delivery to heterogeneous mobile platforms. In *ER Workshops*, pages 205–217, 1998.

[2] J. Garrett. Ajax: A new approach to web applications. http://www.adaptivepath.com/publications/essays/archives/000385.php, Feb 2005.

[3] Jive Software. Smack API. http://www.jivesoftware.org/smack

[4] S. Lok, S. Feiner, W. Chiong, and Y. Hirsch. A graphical user interface toolkit approach to thin-client computing. In *Proceedings of the 11th Intl Conf on the World Wide Web (WWW 2002)*, pages 718–725. ACM Press, 2002.

[5] Macromedia Inc. Macromedia Flex: The presentation tier solution for delivering enterprise rich internet applications. http://www.macromedia.com/software/flex/whitepapers/pdf/flex15_tech_wp.pdf, 2004.

[6] Mozilla.org. XUL. http://developer.mozilla.org/en/docs/XUL

[7] A. J. H. Peddemors, M. M. Lankhorst, and J. de Heer. Presence, location, and instant messaging in a context-aware application framework. In *MDM '03: Proceedings of the 4th International Conference on Mobile Data Management*, pages 325–330. Springer-Verlag, 2003.

[8] J. Rice, A. Farquhar, P. Piernot, and T. Gruber. Using the web instead of a window system. In *Proceedings of the ACM Conference on Human Factors in Computing Systems (CHI '96)*, pages 103–110, 1996.

[9] T. Richardson, Q. Stafford-Fraser, K. Wood, and A. Hopper. Virtual network computing. *IEEE Internet Computing*, 2(1):33–38, 1999.

[10] P. Saint-Andre. Extensible messaging and presence protocol (XMPP): Core. http://www.ietf.org/rfc/rfc3920.txt, Oct. 2004.

[11] X.Org Foundation. X window system. http://www.x.org.

[12] W. Zhao, D. Kearney, and G. Gioiosa. Architectures for web based applications. In *4th Australasian Workshop on Software and Systems Architectures (AWSA 2002)*, 2002.