PRIMI—An Open Platform for the Rapid and Easy Development of Instant Messaging Infrastructures

Tom Gross, Christoph Oemig
Faculty of Media
Bauhaus-University Weimar
Bauhausstr. 11
99423 Weimar, Germany
{tom.gross, christoph.oemig}(at)medien.uni-weimar.de

Abstract

Many concepts and systems for the technical support of communication, cooperation, and coordination of workgroups have been developed in the research field of computer-supported cooperative work over the last two decades. Yet, spontaneous coordination with remote parties can still be a challenge. Instant messaging tools improve the situation by providing users with presence and availability information about other users, and by supporting spontaneous text chat among co-present users. In this paper we present PRIMI (Platform for Research on Instant Messaging Infrastructures)—an open and flexible platform that provides software developers with powerful, yet easy to use support for the implementation of novel concepts for instant messaging infrastructures. It offers the core functionalities for the development of instant messaging infrastructures; and it uses a sophisticated plug-in mechanism for adding user interfaces and communication protocols, as well as logging mechanisms.

1 Introduction

The primary goal of the research field of computer-supported cooperative work (CSCW) is to develop concepts and tools for the easy and flexible support of social interaction in groups, which are often dispersed in time and space [8]. Over more than two decades since the emergence of CSCW many concepts and tools have been developed enhancing communication, cooperation, and coordination [3].

Yet, spontaneous coordination with remote parties remains a challenge [10]. There are various approaches, however, most of them have some unwanted side-effects: having pre-arranged meetings can be too inflexible for real-life spontaneous actions and reactions; sending emails can leave it unclear to the sender when to expect a reply; reaching somebody by phone, even if the person has a mobile phone, strongly depends on the availability and interruptibility of the callee; and sending short messages via mobile phone can be challenging to the recipient, because of the potentially high number of messages. Thus besides various technical means, there is a tension between the need of a caller to spontaneously approach a callee over distance and the wish of the callee to avoid inappropriate disruption.

Instant messaging tools improve the situation by providing users with presence and availability information about their counterparts, and by supporting spontaneous text chats among co-present users. Since the invention of the first general instant messaging tool (ICQ, cf. [13]), these tools have spread tremendously, for both private and professional purposes [11]. Most tools provide the same basic functionality for registering, logging in, logging out, manually setting the online status (e.g., 'Online', 'Away', 'Do-Not-Disturb', etc.), entering additional information (e.g., a personal profile, or an away message with information explaining what a user is currently doing), file transfer, as well as real-time text chats among online users and leaving messages for offline users.

Besides these features and strengths instant messaging tools still suffer from considerable shortcomings:

- The online status has to be adapted manually; or the online status is dependent on pure keyboard and mouse events, assuming that a user who is not active has left the computer (which is not always the case).
- The clients typically have common default alerts for incoming messages, which can easily distract users and interfere with their work.
- The alerts are often identical for all incoming messages by default.
- Many users have a tendency towards sending many messages, even in a single conversational turn [1].

We have designed and developed PRIMI (Platform for Research on Instant Messaging Infrastructures)—a flexible and open platform that provides software developers with powerful, yet easy to use, support for the implementation of novel concepts in the area of instant messaging. It offers the core functionality for the development of instant messaging infrastructures; and it uses a sophisticated plug-in mechanism for the deployment of user interfaces and communication protocols, as well as extensive logging mechanisms.

In the remainder of this paper we present the concept of the PRIMI platform and its reference implementation PRIMIBase. We first discuss related work. Then we describe a scenario explaining how an instantiation of PRIMI—named PRIMInality—is used to implement an instant messaging infrastructure for research purposes. Finally, we draw some conclusions and provide glances on future work.

2 Related Work

In this section we present work that is either similar from a functionality's perspective, or from an implementation's perspective.

From a functionality's perspective consumer instant messaging tools, enterprise instant messaging platforms, and instant messaging research prototypes are similar to PRIMI.

Many consumer instant messaging tools have emerged: single protocol clients such as ICQ, AOL Instant Messenger, Yahoo! Messenger, or MSN Messenger typically only support their native protocol for the communication between clients and server; and multi-protocol clients such as Gaim, Trillian, Miranda, Adium, Fire, or Proteus typically support several of the above mentioned protocols in parallel [cf. 25 for an overview]. For most of these tools servers are typically not available, and only a few clients provide application programming interfaces for tailoring.

Enterprise instant messaging platforms like IBM Lotus Instant Messaging and Web Conferencing, Microsoft Office Live Communications, Novell GroupWise or Sun Java System Instant Messaging have spread as business critical application leveraging the communication between co-workers. These infrastructures especially focus on security, reliability, scalability, directory integration, service access and availability, archive capabilities, and branch specific legal issues (e.g., the Sarbane-Oxley Act). They are comprised of server and client applications configurable to work in highly complex settings usually inside a company (including remote workers) but not beyond (i.e., for internal communication). They offer application programming interfaces, they are not flexible enough for addressing the above-mentioned shortcomings.

Instant messaging research prototypes specifically address some of the challenges mentioned in the introduction. The QnA instant messaging client extension aims to help users to identify the urgency on incoming instant messages, and to notify users accordingly [1]. The activities of users of the Awarenex instant messaging client have been analysed and presented in Actograms in order to make users aware of daily and weekly presence and absence rhythms, through which the reachability of remote users should be increased [2]. In the BusyBody system the disruption of remote users was reduced by automatically asking users from time to time about their interruptibility, and by trying to avoid disruption in cases of low interruptibility [12]. All these approaches have interesting concepts and ideas. Yet, their technical implementations are highly proprietary and mostly lack interfaces to other applications.

From an implementation's perspective message-oriented middleware (MOM) has concepts that are similar to PRIMI.

In message-oriented middleware [16] infrastructures and frameworks realise program-to-program connectivity by message passing, which can be leveraged for instant messaging (e.g., Grapevine [17]). They usually provide an application programming interface hiding the complexity of protocols and platform-specific details allowing to build reliable and scalable high-performance distributed application networks across diverse platforms. Most message-oriented middleware is implemented as client-server architecture with queued message store-and-forward capability, like IBM's MQ Series [24]. The Java Message Service (JMS), an integral part of the J2EE platform, makes MQ services available to any of its applications [21]. Leaving client-server architectures aside JXTA [22] offers a peer-to-peer approach to distributing messages among peer applications. On a whole it provides highly generic technical support for any type of messaging. It is very flexible, however at the cost of cumbersome low-level programming that is necessary to build instant messaging tools on top of it.

3 The Concept

Conducting research with instant messaging systems usually requires the following steps: First, a concept for a chat system comprised of client and server or peer-to-peer components has to be developed and implemented. Then, experiments can be performed by manipulating the functionality of specific components of the system, and studying the effects of the manipulation on its users.

This approach entails the following major drawbacks. The developed systems are often proprietary: they hardly utilise any standards. Only their developers are familiar with their functionality. And theses systems are usually not used outside the lab. Consequently, these systems are hard to integrate and to extend. Each system is usually developed from scratch: this entails a considerable effort, and requires good knowledge from several areas of computer science (e.g., user interfaces, communication protocols). Due to this effort, these instant messaging systems often remain rather simple.

Based on these findings from existing approaches, we describe the concept of PRIMI by first specifying requirements for this novel platform, and then we present its software architecture.

3.1 Requirements

The PRIMI platform was designed and developed as a generic platform to build client and server infrastructures that support existing and custom protocols for communication. The platform has to be flexible enough to be integrated into other applications (e.g., as instant messaging support module in a text editor) or to integrate

other applications into PRIMI (e.g., directory services such as LDAP). It should serve as a platform for research on user interfaces, especially awareness displays and indicators on the client side, as well as event histories and respective mechanisms (e.g., heuristics and algorithms to generate prospective information from accumulated data) on the server side. For these purposes, an instant messaging platform has to meet the following requirements:

- The platform should be *open, extensible, and flexible*. Communication protocols and user interfaces are considered to be components, which can easily be integrated into an existing configuration, even at runtime.
- The platform's extension should be *easy and efficient*. The use of components propagates the separation of concerns—that is, research work on user interfaces does not require work on neither communication nor protocol issues.
- The platform should offer *powerful logging* capabilities. To gain as much information as possible a flexible and pervasive logging facility is need covering all aspects of the infrastructure. Simple local user history files will not suffice.
- The platform should be compliant with up-to-date *standards*. Standards serve as a common language among domain developers. They facilitate rapid development and deployment and help to reduce misunderstandings in naming and requirements. For instance, the requests for comments (RFC) 2778 [5] and 2779 [4] provide guidance concerning basic requirements for instant messaging systems. Other specific technical aspect of instant messaging are covered by further RFCs contributed by the IMPP Charter [cf. 14]).
- The platform should support *various deployment modes* of the client: end-user modes with a pre-configured start-up as well as expert-modes with free configuration.
- The platform should be *independent* of the underlying operating system in order to be of use to many developers and end-users.

These requirements were the corner stones of the subsequently described software architecture of PRIMI.

3.2 Software Architecture

The software architecture of PRIMI has been optimised to particularly provide support for the following development issues of instant messaging infrastructures: communication (i.e., protocols and communication components needed for instant messaging); data handling (i.e., gathering, processing, and presentation of data relevant to instant messaging); and application integration (i.e., aspects on how to integrate other applications with the infrastructure and vice versa). Table 1 provides an overview of PRIMI features on the client and on the server side.

In order to optimally provide these features, the PRIMI platform is using a component-based approach, which is based on plugins.

Figure 1 provides an overview of PRIMI's software architecture, which is subsequently explained. The PRIMI platform is comprised of a central kernel (the plugin application) and surrounding plugins (i.e., plugin packages). There are two types of plugins: user interface and communication plugins. The *plugin-based communication* among the individual components is a vital part of the PRIMI architecture.

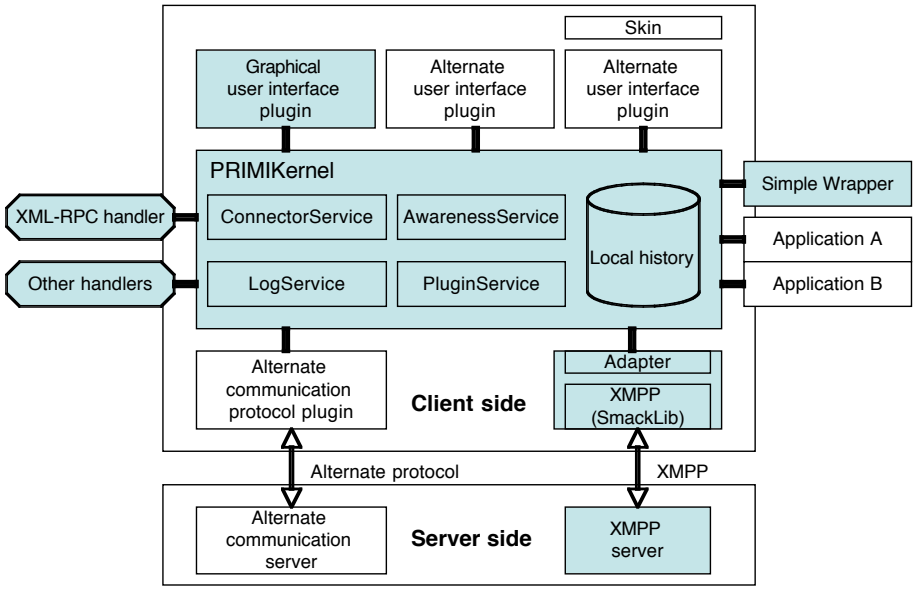

Figure 1. Overview of the PRIMI software architecture. The grey-shaded parts are implemented as part of the PRIMIBase reference implementation of the PRIMI platform.

	Client side	Server side
Communication	Communication facilities for protocols like XMPP, ICQ, TOC etc.	Server side components for protocols like XMPP, ICQ, TOC etc.
Data handling	User interfaces, awareness displays and indicators, sensor integration	Data integration, event histories, heuristics and algorithms
Application integration	Integration of instant messaging functionality into other applications	Integration of other application and services into servers

Table 1. Features of the PRIMI platform.

Therefore, we first describe the plugin mechanism, and then introduce the individual components. Generally, in a plugin architecture a central plugin application is extended by components—the plugins—which are delivered as part of plugin packages containing the plugin and further resources needed. A plugin is a piece of code usually dynamically loaded at runtime. However, it cannot run by itself and has to interact with the plugin application to provide a specific set of functionality [26]. Some well-known examples of plugin-based systems range from simple plugins in Web browsers [23] to complex plugins in the development environment Eclipse.

In PRIMI, the plugin contract determines the installation, registration, lifecycle, domain of use, interfaces, and services offered by the plugin application itself (cf. Figure 2).

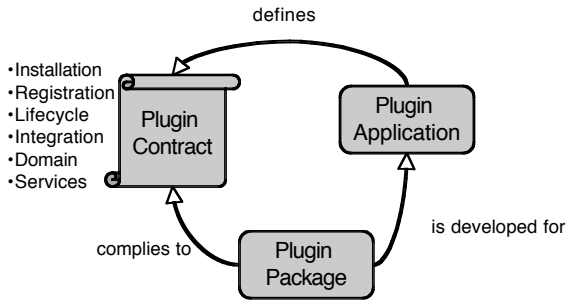

Figure 2. Plugin contract between plugin package and plugin application

The *PRIMIKernel* provides the core of the PRIMI platform; it consists of low-level services (the PluginService and the LogService), and high-level services (the ConnectorService and the AwarenessService). The low-level services provide basic functionality for the high-level services.

The *PluginService* realises the (un-)loading and deployment of all plugins. Before accepting a plugin, its interface compliance is validated and its functionality is probed. Plugins are sorted by their type and are provided to the platform and its services.

The *LogService* takes care of the platform's logging. Each class within the PRIMI platform has its own logger, which is part of the platform's logger hierarchy. Loggers are extended by one or more handlers that output log data to various destinations in specified formats. The LogService supports multi-level logging—that is, the platform is assigned a certain logging level and all log statements of equal or higher level are processed. The log level can be changed at runtime. While the kernel classes access their loggers directly, plugins are using proxy objects according to the Proxy pattern [6]—that is, plugin components are wrapped into proxy objects, which comply to the same kind of super interface. Log statements are executed by these proxy objects. Thus plugin developers do not have to be concerned about the platform's logging details.

The *ConnectorService* is a higher-level service on top of the PluginService responsible for communication protocol plugins (i.e., the Connectors). It allows configuring a subset of active Connector plugins to be seen and used by the user interface in the current session. Thus it establishes a simple plugin access control mechanism. Asked for a Connector (e.g., by type or name) the ConnectorService returns a Connector, which is actually a Connector implementation, wrapped inside a ConnectorProxy object.

The *AwarenessService* is another high-level service on top of the PluginService and LogService taking care of awareness information. Current communication protocols merely integrate presence as awareness information. The AwarenessService allows utilising a broader set of awareness information and its communication (for an example see the scenario below).

All of these services are initialised at application start-up. Besides them the plugin contract defines three API's outlining a plugin's required functionality. As mentioned before there are two types of plugins: plugins for user interfaces and communication protocols (Connectors). Each plugin complies with two interfaces, the generic Plugin interface, and a specific interface (either the GUI or Connector interface) (see Figure 3). The Plugin interface offers a factory-like method returning a class instance conforming to one of the specific interfaces.

PRIMI plugins are loaded at application start-up or at runtime via a network connection. There are a great number of possibilities for plugins. Connector plugins may use third-party libraries, which are adapted to the corresponding PRIMI interface using the Adapter pattern [6]. Or one may implement the required methods entirely inside a monolithic type of plugin. GUI plugins may provide simple or complex user interfaces. The platform assures they seamlessly work with the other components the platform is configured with. Due to the reuse and separate development of components, PRIMI represents a generic platform for rapidly building research type specific instant messaging solutions.

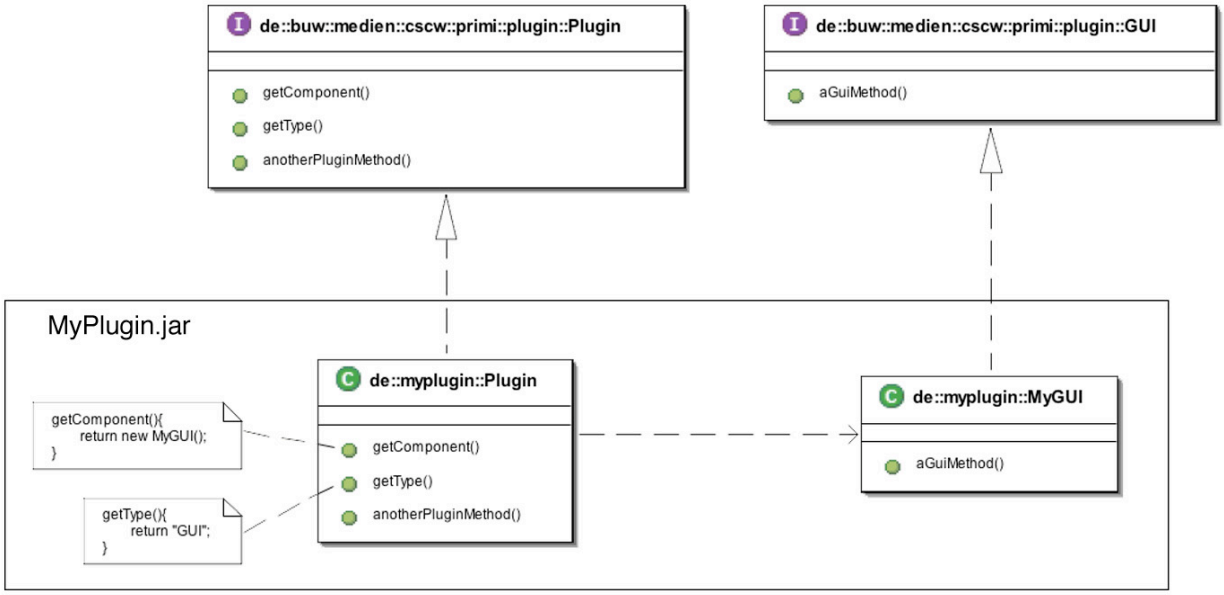

Figure 3. Example of a GUI plugin implementing two interfaces: the generic Plugin interface and the GUI interface.

4 Implementation

The PRIMI platform is written in the Java programming language [20]. This decision was not only due to platform independence, but also due to the availability of numerous Java's APIs, which account for eased extensibility and numerous possibilities of integration. One of the Java APIs used in PRIMI is the Logging API [19]. It offers multi-level logging, which was extended by the concept of proxy objects as described above. Additionally we use our own custom levels and handlers (cf. Figure 1).

Java also provides powerful mechanism for the acquisition and deployment of component code at runtime (e.g., network class loading using a custom class loader), and packaging. Plugins are delivered inside Java archive files (.jar files). For plugin deployment simple conventions, which are part of the plugin contract, have to be met:

- The plugin's main class implementing the Plugin interface has to be named Plugin (cf. Figure 3).
- Plugins are placed within PRIMI's plugin directory (see Figure 4). Here, the .jar files are placed in subdirectories named according to the package name of the specific Plugin class (e.g., the .jar file of the class de.myplugin.Plugin is placed inside the de.myplugin directory).
- Everything belonging to the plugin package is included in a single .jar file.

The PRIMI platform offers three types of deployment modes. Software developers can choose one of the following for the clients of their infrastructure:

- End-user's mode. A simple wrapper allows defining a fixed configuration, which is simply started.
- Application-integrated mode. Allows integrating instant messaging functionality into other applications (e.g., Web portals, email clients). These applications then act as complex wrapper pre-configuring the PRIMIKernel.
- Expert mode. Launches the Configuration Frame, which allows administrating the available plugins, and setting the application's log level (cf. Figure 5).

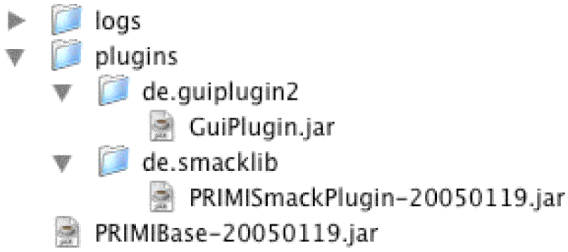

Figure 4. PRIMI's plugin directory structure. For instance, it can be inferred from the directory name de.smacklib that the main plugin class is de.smacklib.Plugin.

PRIMIBase is a reference infrastructure of the PRIMI platform. Its architecture contains two reference plugins: a user interface and a Connector plugin using the Extensible Messaging and Presence Protocol (XMPP) [18], formerly known as the Jabber protocol. The first features a multi-protocol capable user interface supporting multiple accounts and tabbed chat windows. The latter implemented an adapter on top of the SmackLib, a Java-based open source XMPP library [15].

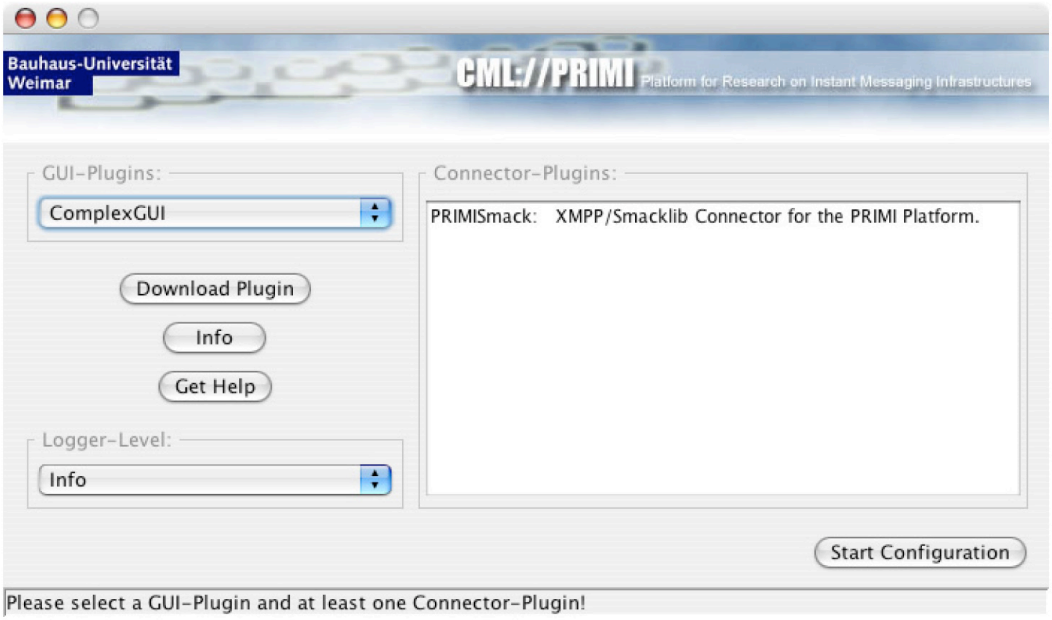

Figure 5. The Configuration Frame shows two available plugins, a GUI plugin named ComplexGUI and a Connector plugin named PRIMISmack.

Finally, PRIMIBase's LogService is equipped with two custom handlers: a custom file handler responsible for writing local user chat histories in flat files formatted in XML, and a XML-RPC [27] handler for communicating awareness information (i.e., log data) as part of the AwarenessService (the next section describes where this handler is used).

5 An Example Scenario

After the presentation of the concepts and implementation of the PRIMI platform as well as the PRIMIBase infrastructure, we now describe a little scenario of another PRIMI-based infrastructure called PRIMInality. First, we want to briefly outline some of the scenario's central research questions and then we explain how PRIMInality addresses the aforementioned research problem.

5.1 Beyond Existing Instant Messaging Systems

Current instant messaging tools support presence awareness using online states like 'Available', 'Away' or 'Do-Not-Disturb'. Yet, the accuracy of these states proves to be insufficient. Being 'Away' may result from keyboard or mouse inactivity, although the user may still be sitting at her desk. We used the PRIMI platform to implement three new concepts for instant messaging:

The first concept extended the way of measuring a user's presence status to obtain a more precise picture. We wanted to use a conglomerate of software and hardware sensors yielding not only more accurate and reliable information of the user's current availability but also further and different presence status incarnations.

In the second concept we applied this new information in two ways: First, we used presence awareness features like a common presentity list to display the availability of other users applying our just mentioned new presence states. Second, the PRIMInality client was not only aware of the (new) availability of others, but also about the user's own status. We used the latter for appropriate user interface adaptations such as when a user is not at her desk, loud notification signals alert for incoming messages while more silent ones are used when the user works at her computer (please note that this configuration aims to support users who have a personal office, not disturbing others with their loud notifications).

The third concept added personalities. In reality we expose only certain details of our identity, the ones that are appropriate to our current situation [7]. Personalities allow to selectively disseminate information about the online states to other users—that is, a user can create several personalities and specify for each personality the information that is published as well as the information's recipient (e.g., a private personality can show private phone numbers to friends; a working personality can reveal the office phone number to business colleagues). Handling personalities in several ways goes beyond existing features such as the multi-account capability of clients. For instance, one core challenge is the adequate adaptation of the client's user interface, and of notification in a situation where a user has multiple concurrent personalities. Here, a solution is to adapt the user interface and notification to the user's most recently active personality.

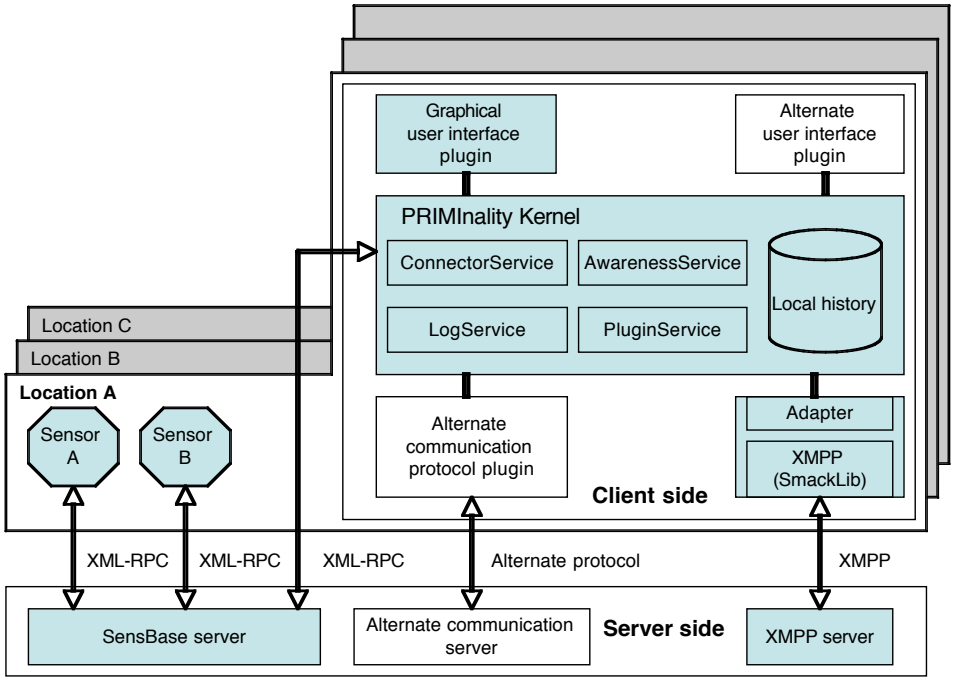

Figure 6. The scenario's architecture setup. We used three locations that were prepared with two hardware sensors and the PRIMInality client.

5.2 Rapid and Easy Development

In order to realise these concepts, PRIMIBase just needed some rapid and easy adaptations to become PRIMInality—the GUI reference plugin was replaced with a new GUI plugin that handles the three new concepts, and that communicates with the SensBase infrastructure.

SensBase [9] provides a generic infrastructure that facilitates the registration and management of hardware sensors, the capturing and storing of sensor values via these sensors, and the retrieving of information on sensors and present and past sensor values via various gateways (e.g., Web-Service, XML-RPC, Sockets, CGI). The SensBase infrastructure was equipped with two hardware sensors, which provide movement information to the PRIMInality's AwarenessService via XML-RPC.

The communication with the SensBase infrastructure is needed to retrieve additional sensor data about the user's physical presence in her office.

In order to accommodate the new concepts, the new GUI plugin contains the logical settings (a simple table) calculating the user's online status from the information of two hardware sensors (movement at the user's desk and movement in the rest of the user's room) and two software sensors indicating whether the user is currently logged in with the PRIMInality client and whether there are open chat windows in the PRIMInality client. The latter information is actually retrieved from PRIMInality's loggers inside the connector plugin's proxy object. The GUI plugin then disseminates the information according to the user's personalities.

This configuration was deployed and tested in three separate offices. Thus, for a start three users had a chance to use a PRIMInality client in their offices.

The PRIMIBase reference Connector plugin realised the communication. Figure 6 depicts our research scenario.

6 Conclusions

In this paper we introduced PRIMI, a platform for building instant messaging infrastructures. We presented its current architecture and its reference implementation PRIMIBase. Finally, we showed its deployment and usage in a sample scenario with PRIMInality. Here, PRIMI proved its conceptual strength: it allowed rapid deployment due to the separation of concerns. The communication plugin from the reference implementation was used; therefore, our work could be focussed on the development of the user interface.

The communication plugin is fully compliant with the XMPP standard, allowing the message exchange with both standard clients and servers—that is, the PRIMIBase clients can be used to communicate with standard Jabber servers, and standard Jabber clients can be used to communicate to PRIMIBase servers. Other protocols such as ICQ and so forth are not yet supported, but could easily be added in a respective connector plugin.

We are convinced that there are various other research scenarios where PRIMI would prove to be useful to quickly build instant messaging infrastructures.

Yet, the platform shows some points for improvement and future work. For instance, the AwarenessService can be rearranged to use plugins itself. These awareness

plugins could provide awareness information about operating systems events, other applications, and other hardware events. Adding these components on demand would improve the accuracy of awareness data. Additionally, log handlers could be implemented as plugins, as well, contributing to the platform's flexibility when the logging output is needed at other destinations.

Further additions can be made in the area of persistence and synchronisation. In future versions we want to add a local embedded database replacing and extending the local file based history. Adding persistence and synchronisation entails several new challenges concerning scalability. So far, scalability has not been a challenge since the PRIMIBase and PRIMInality infrastructures were only used in typical CSCW settings of up to 15 users.

Finally, we aim to complement the PRIMI platform and the PRIMIBase infrastructure, with a comprehensive collection of plugins. Some of them are already under development.

Acknowledgements

We would like to thank our colleague Tareg Egla and the Cooperative Media Lab (CML) students A. Kunert, K. Riege, R. Gerling, Y. Ai, A. Lahn, N. Marquardt, C. Semisch, and M. Pfaff. We also thank the anonymous reviewers for providing stimulating comments.

References

1. Avrahami, D. and Hudson, S.E. QnA: Augmenting an Instant Messaging Client to Balance User Responsiveness and Performance. In Proceedings of the ACM 2004 Conference on Computer-Supported Cooperative Work - CSCW 2004 (Nov. 6-10, Chicago, IL). ACM, N.Y., 2004. pp. 515-518.
2. Begole, J.B., Tang, J.C., Smith, R.B. and Yankelovich, N. Work Rhythms Analysing Visualisations of Awareness Histories of Distributed Groups. In Proceedings of the ACM 2002 Conference on Computer-Supported Cooperative Work - CSCW 2002 (Nov. 16-20, New Orleans, LO). ACM, N.Y., 2002. pp. 334-343.
3. Borghoff, U.M. and Schlichter, J.H. Computer-Supported Cooperative Work: Introduction to Distributed Applications. Springer-Verlag, Heidelberg, 2000.
4. Day, M., Aggarwal, S., Mohr, G., Vincent, J. Instant Messaging / Presence Protocol Requirements (RFC 2779). http://www.ietf.org/rfc/rfc2779.txt, 2000. (Accessed 20/2/2005).
5. Day, M., Rosenberg, J., Sugano, H. A Model for Presence and Instant Messaging (RFC 2778). http://www.ietf.org/rfc/rfc2778.txt, 2000. (Accessed 20/2/2005).
6. Gamma, E., Helm, R., Johnson, R. and Vlissides, J. Design Patterns: Elements of Reusable Object-Oriented Software. Addison-Wesley, Reading, MA, 1994.
7. Goffman, E. The Presentation of Self in Everyday Life. Anchor, N.Y., 1956.
8. Greif, I., ed. Computer-Supported Cooperative Work: A Book of Readings. Computer and People Series. Morgan Kaufmann Publishers, San Mateo, CA, 1988.
9. Gross, T., Egla, T. and Oemig, C. Sens-ation. Cooperative Media Lab, http://cml.medien.uni-weimar.de/sens-ation, 2005. (Accessed 10/5/2005).
10. Gross, T. and Prinz, W. Modelling Shared Contexts in Cooperative Environments: Concept, Implementation, and Evaluation. Computer Supported Cooperative Work: The Journal of Collaborative Computing 13, 3-4 (Aug. 2004). pp. 283-303.
11. Handel, M. and Herbsleb, J.D. What Is Chat Doing in the Workplace? In Proceedings of the ACM 2002 Conference on Computer-Supported Cooperative Work - CSCW 2002 (Nov. 16-20, New Orleans, LO). ACM, N.Y., 2002. pp. 1-10.
12. Horvitz, E., Koch, P. and Apacible, J. BusyBody: Creating and Fielding Personalised Models of the Cost of Interruption. In Proceedings of the ACM 2004 Conference on Computer-Supported Cooperative Work - CSCW 2004 (Nov. 6-10, Chicago, IL). ACM, N.Y., 2004. pp. 507-510.
13. ICQ Inc. ICQ.com - Community, People Search, and Messaging Service! http://www.icq.com/, 2002. (Accessed 18/2/2005).
14. IETF. Instant Messaging and Presence Protocol (IMPP) Charter. http://www.ietf.org/html.charters/OLD/impp-charter.html, 2002. (Accessed 20/2/2005).
15. Jive Software. Smack API. http://www.jivesoftware.org/smack/, 2005. (Accessed 20/2/2005).
16. Rao, B.R. Making the Most of Middleware. Data Communications International 24, 12 (1995). pp. 89-96.
17. Richards, J. and Christensen, J. People in Our Software Queue 1 10 (2004). pp. 80-86
18. Saint-Andre, P. Extensible Messaging and Presence Protocol (XMPP): Core (RFC 3920). http://www.ietf.org/rfc/rfc3920.txt, 2004. (Accessed 20/2/2005).
19. Sun Microsystems. Java Logging APIs. http://java.sun.com/j2se/1.4.2/docs/guide/util/logging/, 2002. (Accessed 20/2/2005).
20. Sun Microsystems. Java Technology. http://java.sun.com, 2005. (Accessed 20/2/2005).
21. Sun Microsystems Inc. Java Message Service (JMS). http://java.sun.com/products/jms/, 2005. (Accessed 20/2/2005).
22. Sun Microsystems Inc. JXTA Technology. http://www.sun.com/software/jxta/, 2005. (Accessed 20/2/2005).
23. The Mozilla Organization. Mozilla Plugins. http://www.mozilla.org/projects/plugins/, 2005. (Accessed 20/2/2005).
24. Wackerow, D. MQSeries Primer. http://publib-b.boulder.ibm.com/Redbooks.nsf/9445fa5b416f6e32852569ae006bb65f/cdce1eb0c77b2105852569900072a3f4?OpenDocument&Highlight=0,mqseries, 1999. (Accessed 20/2/2005).
25. Wikipedia. Instant Messenger - Wikipedia, the Free Encyclopedia. http://en.wikipedia.org/wiki/Instant_messaging, 2005. (Accessed 18/2/2005).
26. Wikipedia Plugin - Wikipedia, the Free Encyclopedia. http://en.wikipedia.org/wiki/Plugin, 2005. (Accessed 20/2/2005).
27. Winer, D. XML-RPC Specification. http://www.xmlrpc.com/spec, 1999. (Accessed 20/2/2005).