From PRIMI to PRIMIFaces: Technical Concepts for Selective Information Disclosure

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

Instant messaging platforms facilitate coordination in workgroups by providing users with mutual information on their presence and availability, allowing for ad-hoc conversations with little disturbance. For this purpose they typically capture, process, and present information from and to the users involved. Thereby, they have to address a fundamental trade-off: on the one hand users want and need up-to-the-minute information about each other, yet on the other hand the users whose information is captured have a legitimate right for privacy and the users whom the information is presented have a legitimate need for avoiding frequent disruptions. In this paper we present advanced technical concepts for selective information disclosure. We introduce the concept of face-based instant messaging platforms that allow end-users on one side to flexibly integrate static and dynamic data sources, and to easily specify their preferences as to the data to be shared as well as the granularity and the timing of the sharing, and on the other side to easily specify their preferences for notification.

1 Introduction

Instant messaging platforms support the cooperation in distributed workgroups. They facilitate coordination in workgroups by providing users with mutual information on their presence and availability, allowing for ad-hoc conversations with little disturbance. For this purpose they typically capture, process, and present information from and to the users involved. Thereby, they have to address a fundamental trade-off: on the one hand users want and need up-to-the-minute information about each other, yet on the other hand the users whose information is captured have a legitimate right for privacy and the users whom the information is presented have a legitimate need for avoiding frequent disruptions.

The following little scenario should illustrate the advantages and trade-offs of providing the members of distributed workgroups with mutual information: imagine a workgroup in which all the members are spread in various offices in the same office building. Whenever, member M1 needs to spontaneously contact colleague M2 (e.g., because M1 has a question for M2, or because M1 needs to quickly discuss an open issue with M2) M1 can glance into M2's office to check if it is a good time to disrupt, if not M1 can return later. Imagine another workgroup in which people from several organisations and locations are involved. Spontaneous contact that fits member M1 and M2 likewise is more difficult here. Member M1 can call M2, but it is unclear if M2 is present and available; member M1 can send an email to M2, but email might involve more effort for discussing the open issues and coming to a conclusion.

Technological support for the second workgroup could provide member M1 with information on M2's presence and availability, and so forth, and would have a great potential to make it much easier to coordinate spontaneous social interactions. However, this technology should also address new challenges that might arise: M2 should be able to have privacy and consequently have information on which information is captured and means to influence which information is captured and who will be allowed to receive this information; and M1 should be able to avoid unwanted disruption and to specify which information to receive, and when and how.

This selective information disclosure is a base requirement for social interaction—both in face-to-face interaction and in technologically mediated interaction. As Palen and Dourish [11] point out: 'participation in the social world also requires selective disclosure of personal information. [...] We seek to maintain not just a personal life, but also a public face. Managing privacy means paying attention to both of these desires.'.

In this paper we present the instant messaging platform PRIMIFaces, which provides advanced technical concepts for selective information disclosure. In the next section we outline the detailed requirements for the PRIMIFaces platform. We, then, describe the technical concepts of PRIMIFaces and how they were applied to augment the PRIMI platform. Finally, we draw conclusions and outline future work.

2 Related Work

Traditional instant messaging platforms support privacy by allowing users to explicitly specify their online state (typically many systems also provide an ‘invisible’ state that allows users to disappear completely); and support disruption avoidance by providing a broad choice of notification mechanisms (typically from intense audio and visual notifications to subtle cues) [18].

Traditional instant messaging is either supported in consumer instant messaging tools, or in enterprise instant messaging platforms and message-oriented middleware.

Consumer instant messaging tools range from single-protocol clients supporting one specific communication protocol (e.g., ICQ, AOL Instant Messenger, Yahoo! Messenger) to multi-protocol clients (e.g., Gaim, Trillian, Adium) supporting multiple protocols in parallel [cf. 18 for an overview]. These tools are typically highly usable, simple, and widespread. Yet, they are mostly only available as clients to centralised servers that are run by the inventors of the respective protocol, and they are not radically customisable.

Enterprise instant messaging platforms and message-oriented middleware infrastructures (e.g., IBM Lotus Instant Messaging and Web Conferencing, Microsoft Office Live Communications, Novel GroupWise) are typically technically more advanced. They are comprised of clients and servers and provide support for security, reliability, scalability, directory integration, and so forth in complex work environments. Message-oriented middleware [12] infrastructures and frameworks provide basic support for program-to-program connectivity by message passing that can be used to build reliable and scalable high-performance distributed applications (e.g., the IBM MQ Series [17], the Java Messaging Service [14], JXTA [16]). Both—enterprise instant messaging platforms and message-oriented middleware infrastructures—are technically advanced, based on client-server or peer-to-peer architectures, and highly customisable. However, the customisation is typically complex and can only be done by system administrators, and not by end-users in their day-to-day interaction with the system.

Personality-based instant messaging platforms introduce concepts for selective information disclosure. They allow end-users to easily create various online personalities and to share and receive specific information for each personality. For instance, users can have a private personality where they disclose private data such as the number of their private mobile phone, and a work personality where they disclose work-related data such as the office location and their meeting schedule. There are only a few platforms available: The PRIMInality platform allows users to create personalities and to assign static data that they want to share [7]. The GrapeVine system introduced eCards with a similar concept allowing users to create distinct online business cards and assign specific assess rights to individual or groups of colleagues [13]. While these platforms offer more flexibility and customisability to the end-user, they still have their limitations—especially with respect to easy handling for end-users.

Face-based instant messaging platforms are a novel concept. They allow end-users on one side to integrate static and dynamic information sources that capture data that might help other users to get a better impression of their current presence and availability, and to easily specify their preferences as to the data to be shared as well as the granularity and the timing of the sharing. On the other side they allow the potential authorised recipients to easily specify their preferences for notification and presentation of incoming information.

3 Requirements

Instant messaging platforms should provide advanced technical concepts for human-centred computing while preserving well-established strengths of existing technology.

Advanced technical concepts for instant messaging platforms should include information from the electronic world and from the real world. They should provide a balance for the trade-off between the one hand providing mutual information and the other hand protecting privacy and avoiding interruption. This trade-off was already identified in computer-mediated communication [2] and in computer support for mutual awareness information [9]. Bellotti and Sellen [1] point out that these challenges are aggravated when applications become more and more ubiquitous—that is, when they increasingly capture and provide data from real-world sensors, not only from electronic sensors. They point out that: mediated interaction between people via technology is ‘prone to breakdowns due to inadequate feedback about what information one is broadcasting and an inability to control one’s accessibility to others’ [1, p. 78]. They conclude that the following two requirements are vital for adequate mechanisms. Systems should:

- provide *feedback* through adequate information to the users whose data are captured about both the information captured and the users having access or receiving the data
- support *control* through adequate assistance for the users to specify the information they want to share and the recipients to share their information with

Several strengths are well established in recent instant messaging platforms. In particular, instant messaging platforms should (cf. [6]): be open, extensible, and flexible; allow easy and efficient extension; offer powerful logging capabilities; be compliant with up-to-date standards; should support various deployment modes; and be independent of the underlying operating system.

So, concepts and platforms are needed, which provide advanced support and novel solutions to new challenges as well as preserve the strengths of existing approaches.

4 Towards Selective Information Disclosure

In this section we describe advanced concepts for and the implementation of selective information disclosure. We first discuss the concept of selective information disclosure and faces from Goffman and then introduce the concepts and implementation of the PRIMIFaces platform and client.

4.1 Faces

The concept of selective information disclosure and faces is grounded in real world sociology and social psychology. Here, people usually control implicitly what information to expose. According to Goffman [4] we construct social identities or faces representing a subset of characteristics and information about ourselves, which is to be revealed to a certain audience—that is, we selectively disclose and disseminate information depending on the current temporal, spatial, and interpersonal context. For a human-centred approach we sought to adopt this mechanism since people are expected to be familiar with it. Thus the concept becomes easier to comprehend and users should have less learning effort.

Hence our approach seeks appropriate means of feedback and control regarding Goffman's faces for the disclosure of personal information. In order to realise our approach we defined the following concepts:

- *Faces* define specific fronts of a user and contain information sources and recipients who are allowed to get the information; users can mute faces—that is, temporarily interrupt the information supply.
- *Contacts* are the recipients of the information; each user can be member of any number of faces (e.g., user A can be user B's working colleague and at the same time his tennis partner) and any number of contacts can be added to a face
- *Information sources* refer to the origins of the information that is to be shown in each face
- *Bridges* are a user's user-face combinations in relation to another user's user-face combination. For instance, user A assigns user B to his face 1 while user B assigned user A to his face 2 (this can be described by the bridge A-1:B-2). Bridges can be symmetrical or asymmetrical. For instance, while user B is in user A's face best friends, user A does not have to be in user B's face best friends. Also the type and number of sources on either side of a bridge can be symmetrical or asymmetrical.

Faces, contacts, information sources, and bridges represent means of control since they impact the flow of information in selective disclosure. Additionally we designed feedback items in the form of a three level (contact, face, user) information exchange visualisation and a tickertape to provide users with feedback about their current information sharing status.

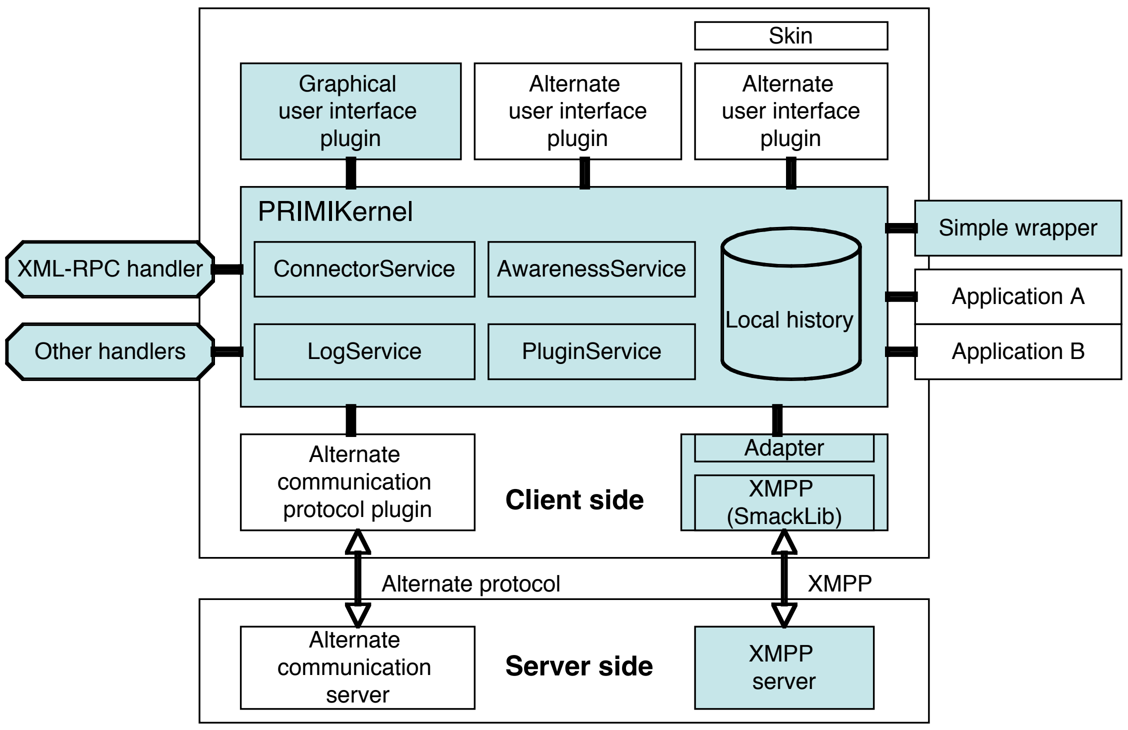

Figure 1. Overall PRIMI software architecture. The grey-shaded parts are implemented as part of the PRIMIBase reference implementation of the PRIMI platform (cf. also [6]).

4.2 PRIMIFaces

The PRIMIFaces instant messaging platform implements the requirements presented above.

In order to meet the *general requirements* for instant messaging platforms we based PRIMIFaces on the PRIMI platform. The PRIMI platform was presented elsewhere [6]; here we just quickly sketch the main concepts that need to be known in order to understand the concepts and implementation of PRIMIFaces.

Figure 1 shows the basic software architecture of PRIMI. Generally, the PRIMI platform covers three core issues: data handling, communication protocols, and application integration. These aspects are considered on the client and server side: graphical user interfaces and especially awareness displays are research issues on the client side, while on the server side event histories and other mechanisms (e.g., heuristics and algorithms to generate prospective information from accumulated data) are used to enhance communication and collaboration. The platform is used to build clients and servers that support existing and custom protocols for communication. User interfaces and communication protocols are deployed in the Java programming language [15] as plugins [19] as part of a central infrastructure that offers all necessary services, especially for plugin handling and logging. Most communication inside PRIMI and with other applications and environments outside of PRIMI is done via XML-RPC [20]. A great strength of XML-RPC is that it can be combined with Web-servers and then can work via HTTP on port 80 and does not interfere with firewalls. PRIMIBase constitutes the PRIMI reference implementation of a plugin-based infrastructure.

In order to provide *advanced technical concepts* the PRIMIFaces platform *faces* need to be introduced. Every user has at least one face, the default or public face. Faces get assigned contacts and sources. *Bridges* need to be stored; a storage format has to be found. Additionally, *information sources* need to be introduced. We distinguish:

- Sources that expose information that is manually provided by users (static sources)
- Sources that capture information using sensors (either software sensors capturing information from computer operating systems and applications, and hardware sensors capturing information from the real-world environment)
- Sources that create and expose information based on algorithms without any user input or sensing (dynamic sources).

Sources either provide their information in an active or passive operational mode (push or pull sources). Furthermore, information can come from an internal source (either user input or results from algorithms), or from an external source (a sensor outside the platform). Users may use an arbitrary number of sources. Source handling, data storage destinations, and formats need to be found in order to define where and how to store the data. Table 1 provides an overview of source types and operational modes.

Sources are implemented as part of the core platform. Like user interfaces and connectors, sensor sources are realised as external plugins that can be dynamically added to the platform. Sources with user or algorithmic input are implemented as internal classes of the platform.

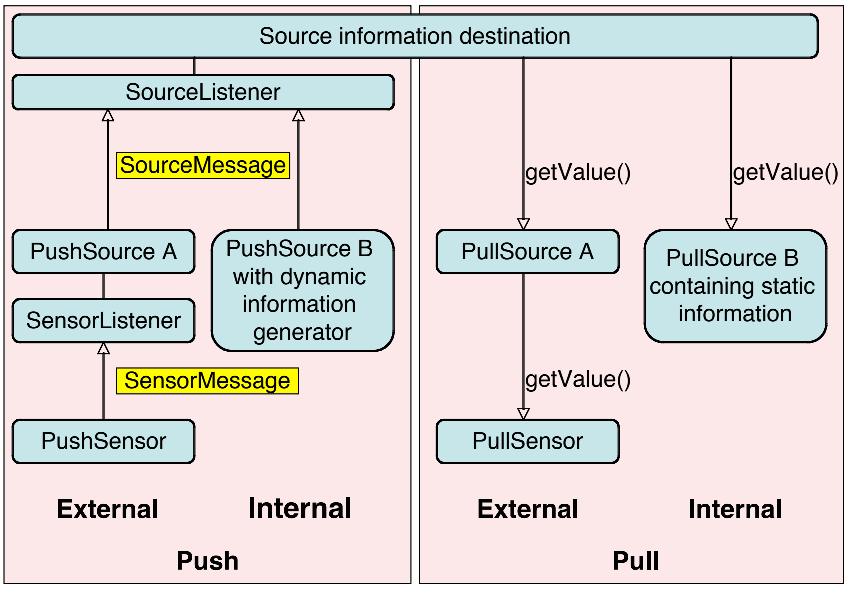

Figure 2. Overview of internal and external push and pull sources.

		Information type	
		Internal	External (i.e., sensor plugin)
Operational mode	Pull	Static information (e.g., an email address or telephone number)	CPU sensor, application sensor, nearby-people sensor
	Push	Dynamic information (algorithm-based information generators)	Mouse idle sensor

Table 1. Source types and operational modes.

Pull sources need to be asked for their values, while push sources inform registered listening components via a callback interface (`SourceListener` interface) in case of any change to their values. Push sources can be based on dynamic information generators that create their values based on algorithms, or on push sensors that deliver their values via another call-back interface (`SensorListener` interface). Sensor components are wrapped into source objects in order to provide a common interface for source handling. Figure 2 depicts an overview of the workings of internal and external push and pull sources.

Overall, there are six implementations of the source interface depicted in Figure 3. The `SourceSensor-Impl` class provides a wrapper for push and pull sensors (external sources). Static information sources mentioned above are implemented as `SourceInternalStatic-InfoImpl`, dynamic sources are realised as `Source-`

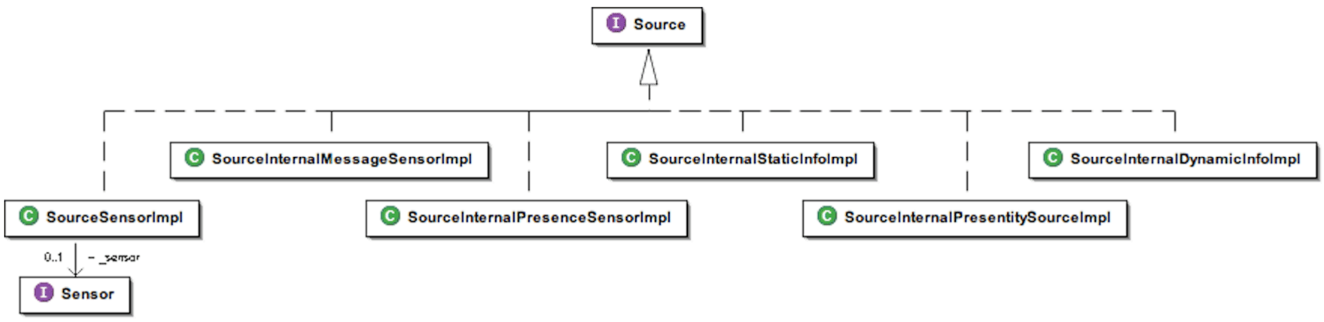
Source
SourceInternalMessageSensorImpl
SourceInternalStaticInfoImpl
SourceInternalDynamicInfoImpl
SourceSensorImpl
SourceInternalPresenceSensorImpl
SourceInternalPresentitySourceImpl
0..1
- _sensor
Sensor

Figure 3. Source interface and implementing classes.

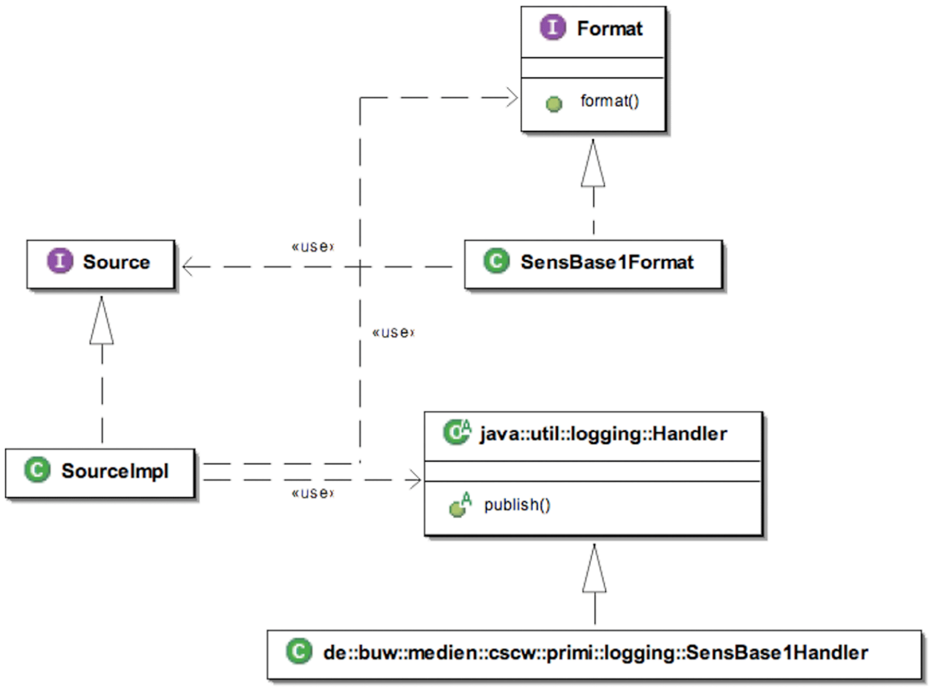
Format
format()
Source
«use»
SensBase1Format
«use»
SourceImpl
«use»
java::util::logging::Handler
publish()
de::buw::medien::cscw::primi::logging::SensBase1Handler

Figure 4. Sources are assigned handlers and formats in order to define the destination and appearance of their data.

`InternalDynamicInfoImpl`. Figure 3 also shows further internal sources:

- `SourceInternalMessageSensorImpl`: message data sent via instant messaging can be perceived as source data as well; that is why we implemented this class that provides this information as source
- `SourceInternalPresenceSensorImpl`: as for instant messages the same accounts for instant messaging data on the users' online states; this class provides this information as source
- `SourceInternalPresentitySourceImpl`: this implementation of the `Source` interface represents the sources of the other users (the contacts or presentities)

The source classes also take care of logging their data. In the case of sensor sources the source class acts as a proxy, the underlying sensor class does not know anything about the logging mechanisms. This approach is also used for the connector plugins where communication and status data are logged transparently, yet for a different purpose. Sources can be assigned handlers and formats in order to define the logging data's destination and appearance. Thus, the concept of sources becomes generic enough to be used in a multitude of scenarios as part of the PRIMIFaces platform.

Figure 4 shows how `Format` and `Handler` classes are assigned to sources. The formatting follows the command software pattern [3]. Yet, the invoker is the `SourceImpl` itself: when its `getValue()` method (pull case) or a push event occurs (push case) the `SourceImpl` class calls its `Format` class' `format()` method to output the values as desired. The output is then handed over to the `Handler` class' `publish` method. As it can be seen in Figure 4 the `Handler` classes and the entire logging is based on Java's Logging API [8].

In order to adapt sources for different settings simply new `Handler` and `Format` classes need to be provided. The conceptually required muting is realised by switching off loggers inside the `SourceImpl` classes.

For using sources within PRIMIFaces we finally set up a service, the `Source` service. It acts as factory creating or acquiring source instances. `Handler` and `Format` classes need to be provided here as parameters. The service belongs to PRIMIFaces' set of high-level services and is initialised when PRIMIFaces starts.

In PRIMIFaces faces are stored as a compound of username and face name with a dot in between (e.g., `userA.face1`). Sources and the data they produce are stored in a sensor-based environment called SensBase that was develop with the Sens-ation platform [5] in XML format. SensBase provides a generic infrastructure mostly written in the Java programming language. It facilitates the registration and management of hardware and software sensors, the capturing and storing of sensor values via these sensors, and the retrieval of information on sensors and present and past sensor values via various gateways (e.g., Web Services, XML-RPC, Sockets, CGI). Basic building blocks for data storage are sensors and locations. The latter basically tag a collection of sensors that have something in common, usually a physical location.

We developed a handler for sending data to SensBase and a format, which took care of the data's layout in XML. Both are provided to source class instances retrieved from PRIMIFaces' source service. Additionally, for the way back, a class was implemented that converts data back from flat XML retrieved from SensBase into objects.

Bridges are realised using dictionaries that are specialised SensBase sensors containing data of the format shown in Figure 5. A dictionary holds a user's settings about how contacts are assigned to faces. The example in Figure 5 describes two faces containing two users each. In order to build a complete bridge the application has to read the current user's dictionary. For each contact found, it has to look up the current user's name as contacts in their respective dictionaries. This way the application also finds out about the faces' name in order to retrieve all sources from. These dictionaries are created and updated automatically, based on the users' preferences. The users can create faces, and add contacts directly in the PRIMIFaces client (cf. Figure 6 below). It is also the users' choice which information sources to add and to check the sources and their combination is of help to the

```
<dictionary>
 <face>
   <name> UserA.Face1</name>
   <presentity>UserB</presentity>
   <presentity>UserC</presentity>
 </face>
 <face>
   <name> UserA.Face2</name>
   <presentity>UserD</presentity>
   <presentity>UserE</presentity>
   </face>
</dictionary>
```

User A

```
<dictionary>
 <face>
   <name> UserB.Face1</name>
   <presentity>UserF</presentity>
   <presentity>UserC</presentity>
 </face>
 <face>
   <name> UserB.Face2</name>
   <presentity>UserA</presentity>
   <presentity>UserE</presentity>
   </face>
</dictionary>
```

User B

Figure 5. The bridge between user A and user B (UserA.Face1:UserB.Face2).

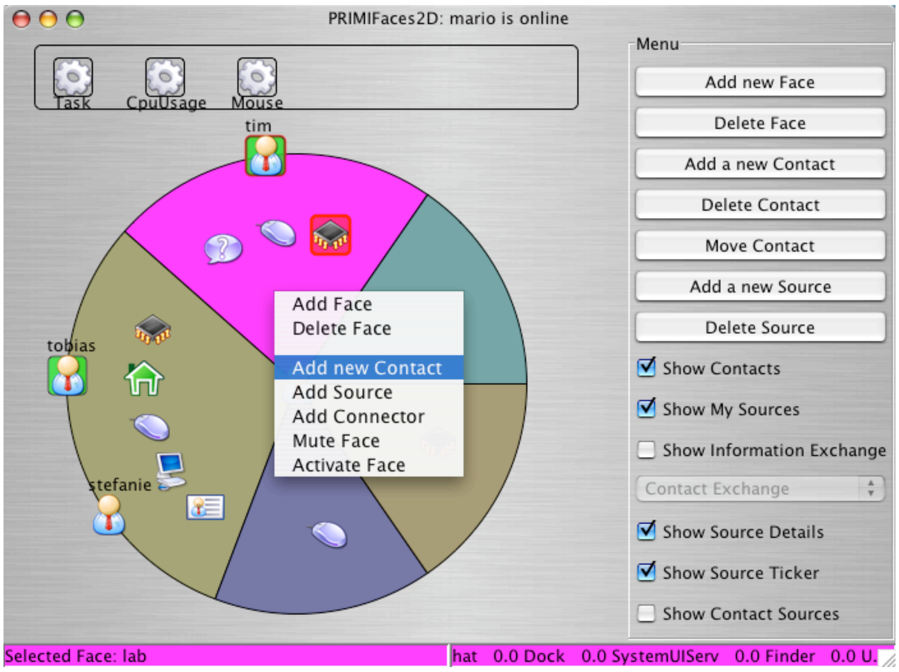

Figure 6. The PRIMIFaces user interface.

contacts who receive this information.

With sources and bridges implemented at the level of the core platform we still need to focus on the client and its user interface. On the client side faces are a central user interface issue, since easy and intuitive user interface mechanisms for feedback and control are a challenge. Figure 6 shows a screenshot of the PRIMIFaces client's user interface. A pie chart serves as visualisation for faces. Contacts and sources can be added simply by using the buttons on the right side or via the menu bar.

So far, only a fat client for the computer is available. While a mobile client would certainly be interesting to experiment with, it also entails several new challenges (e.g., how to provide good interaction design for creating and maintaining faces, adding and removing contact, adding and removing information sources).

5 Conclusions and Future Work

More than merely status information is needed to overcome the two major problems in computer-mediated communication, disembodiment and dissociation. For addressing these challenges we introduced the concept of faces from Goffman and presented the concept and implementation of advanced selective information disclosure in PRIMIFaces. This advanced selective information disclosure provides easy means to the users to decide what to share with whom. This goes beyond other approaches that simply provide some additional information to contact initiators (e.g., ContextContacts is a mobile phone system that nicely provides initiators of phone calls with information on current location and other information on the call recipient's phone, yet it is also limited to these types of information and only allows general disclosure without faces [10]).

With the setup presented we are able to provide more information, which is to be disclosed selectively. We additionally meet the requirements of asymmetry: Based on dictionaries user may assign other users in an asymmetric manner; sources may be added in an arbitrary manner on either side.

Due to the use of formats and handlers with sources we allow to easily switch and use different storage infrastructures where we used SensBase as the default. Developers simply need to create a new handler, which defines the new destination and calls its provided methods to store values. The format only has to be adapted in the format class in order to be used with another infrastructure.

However, the concepts presented have some limitations in their current state. For example, users may switch the computer they use. Yet, a different set of sensor plugins might be installed on either location—that is, a user might provide more, less, or even different information depending on the equipment used. This issue has not been covered yet.

Another issue is the handling of variations of precision of sensor values. For numeric values this might be an easy case to increase or decrease precision (e.g., a 4.5 becomes 5). Using more semantic values the issue becomes increasingly difficult. For instance, for a user who wants to share location information, but in a coarse-gained fashion, generalising the current location from Weimar to Germany can become a challenge.

Further, in our approach we provided dynamic sources that generate their information based on algorithms. It is not completely clear what this can be used for. A possible scenario could be the generation of fake data in order to pretend a certain situation (e.g., I appear to very busy though I am not). This needs to be explored.

Last but not least, the way back from the storing infrastructure remains an issue: we realised it as part of the user interface plugin since it is the instance that knows about the destination and format.

From a social science perspective the consequences of asymmetrical information provision is an open question. In the current platform and its clients user get feedback on the information they are sharing and have control over the selection and granularity of the information to share. The users do not have feedback on the overall situation of the other users—that is, they only receive the information that is shared in faces. They might receive all information, or they might miss information. In the latter case the information might be missing, because the other users cannot provide it (e.g., simply because they do not have the software and hardware infrastructure to capture and process the data when they are online, but travelling), or because the other users have deliberately excluded information sources from the faces.

Acknowledgments

We would like to thank our colleague Tareg Egla and the Cooperative Media Lab (CML) students—especially Tim Gollub, Tobias Langlotz, Mario Urbina, and Stefanie Zollmann—for contributing to the concepts and implementation of this work. We also thank Thilo Paul-Stueve and the anonymous reviewers for providing stimulating comments on earlier versions of this paper.

References

1. Bellotti, V. and Sellen, A. Design for Privacy in Ubiquitous Computing Environments. In *Proceedings of the Third European Conference on Computer-Supported Cooperative Work - ECSCW'93* (Sept. 13-17, Milan, Italy). Kluwer Academic Publishers, Dortrecht, NL, 1993. pp. 77-92.
2. Coutaz, J., Crowley, J.L. and Berard, F. Eigen-Space Coding as a Means to Support Privacy in Computer-Mediated Communication. In *IFIP TC13 International Conference on Human-Computer Interaction - INTERACT'97* (July 14-18, Sydney, Australia). Chapman & Hall, London, UK, 1997. pp. 532-538.
3. Gamma, E., Helm, R., Johnson, R. and Vlissides, J. *Design Patterns: Elements of Reusable Object-Oriented Software*. Addison-Wesley, Reading, MA, 1994.
4. Goffman, E. *The Presentation of Self in Everyday Life*. Doubleday Anchor Books, N.Y., 1959.
5. Gross, T., Egla, T. and Marquardt, N. Sens-ation: A Service-Oriented Platform for Developing Sensor-Based Infrastructures. *International Journal of Internet Protocol Technology (IJIPT) 1*, 3 (2006). pp. 159-167.
6. Gross, T. and Oemig, C. PRIMI: An Open Platform for the Rapid and Easy Development of Instant Messaging Infrastructures. In *Proceedings of the 31st EUROMICRO Conference on Software Engineering and Advanced Applications - SEAA 2005* (Aug. 30-Sept. 3, Oporto, Portugal). IEEE Computer Society Press, Los Alamitos, CA, 2005. pp. 460-467.
7. Gross, T. and Oemig, C. PRIMInality: Towards Human-Centred Instant Messaging Infrastructures. In *Mensch & Computer - 5. Fachuebergreifende Konferenz - M&C 2005* (Sept. 4-7, Linz, Austria). Oldenbourg, Linz, 2005. pp. 71-80.
8. Hamilton, G. *Java Logging APIs*. Sun Microsystems Inc., http://jcp.org/aboutJava/communityprocess/review/jsr047/spec.pdf, 2000. (Accessed 14/3/2006).
9. Hudson, S.E. and Smith, I. Techniques for Addressing Fundamental Privacy and Disruption Tradeoffs in Awareness Support Systems. In *Proceedings of the ACM 1996 Conference on Computer-Supported Cooperative Work - CSCW'96* (Nov. 16-20, Boston, MA). ACM, N.Y., 1996. pp. 248-257.
10. Oulasvirta, A., Raento, M. and Tiitta, S. ContextContacts: Re-Designing SmartPhone's Contact Book to Support Mobile Awareness and Collaboration. In *Seventh International Symposium on Human-Computer Interaction with Mobile Devices and Services - MobileHCI 2005* (Sept. 19-22, Salzburg, Austria). ACM, N.Y., 2005. pp. 167-174.
11. Palen, L. and Dourish, P. Unpacking 'Privacy' for a Networked World. In *Proceedings of the Conference on Human Factors in Computing Systems - CHI 2003* (Apr. 5-10, Minneapolis, Minnesota). ACM, N.Y., 2003. pp. 129-136.
12. Rao, B.R. Making the Most of Middleware. *Data Communications International 24*, 12 (1995). pp. 89-96.
13. Richards, J.T. and Christiansen, J. People in Our Software. *ACM Queue 1*, 10 (Feb. 2004). pp. 80-86.
14. Sun. *Java Message Service (JMS)*. Sun Microsystems Inc., http://java.sun.com/products/jms/, 2005. (Accessed 15/3/2006).
15. Sun. *Java Technology*. Sun Microsystems Inc., http://java.sun.com/, 2006. (Accessed 15/3/2006).
16. Sun. *JXTA Technology*. Sun Microsystems Inc., http://www.sun.com/software/jxta/, 2006. (Accessed 15/3/2006).
17. Wackerow, D. *MQSeries Primer*. http://www.redbooks.ibm.com/redpapers/pdfs/redp0021.pdf, 1999. (Accessed 15/3/2006).
18. Wikipedia. *Instant Messaging - Wikipedia, the Free Encyclopedia*. http://en.wikipedia.org/wiki/Instant_messaging, 2005. (Accessed 9/3/2005).
19. Wikipedia *Plugin - Wikipedia, the Free Encyclopedia*. http://en.wikipedia.org/wiki/Plugin, 2006. (Accessed 15/3/2006).
20. Winer, D. *XML-RPC Specification*. http://xmlrpc.scripting.com/spec, 1999. (Accessed 15/3/2006).