Design and implementation of a tailorable awareness framework

Verena Fastenbauer Jakob Hummes* Bernard Merialdo
Institut EURECOM – Sophia Antipolis, FRANCE
{fastenba,hummes,merialdo}@eurecom.fr

Abstract

This paper describes an approach for tailoring that combines frameworks and components. The approach is validated by implementing a general awareness service, which can be adapted to different problem domains. Using an object-oriented framework approach together with component based programming, the presented awareness framework remains invariant, but cooperates with its extending components. Furthermore, the framework offers the end-users a filtering mechanism, which allows them to tailor the behavior of the awareness service during the cooperative session at run-time to their current needs.

Keywords: Tailoring, CSCW, awareness, component model, Java Beans

1 Introduction

Awareness is a key requirement for cooperation [4]. Within groupware systems, spatially dispersed cooperating users must be notified about actions of others. Apart from a notification mechanism, an awareness service should offer a filtering mechanism. Since in a personalized setting, different users want to be notified in different ways, each user may tailor the filter to his personal needs. For efficient cooperation the notification settings must be tailorable at any time.

As example, users perceive awareness information in the background or in the foreground [1]. A tailorable awareness service allows each user to weight the importance for specific notification types. While the user is working on an individual task, most notifications from others are recorded in the background, but certain from the user selected notifications trigger immediately a pop-up window.

This paper introduces a tailorable groupware framework, which offers a general awareness service. This distributed framework uses the notion of awareness events, which are passed by producing components to consuming components. The end-user tailors the appearance and behavior of the framework with the help of filter components. Filters provide the means to select at run-time the components, which compute and present the awareness data. Although we focus on the development of an awareness framework, the presented approach is valid for a range of tailorable groupware systems.

Mørch [7] distinguishes three levels of tailoring. The levels are classified by the design distance which is experienced by the end-user during tailoring. Generally speaking, with an increasing level the tailoring possibilities for a user increase, but also become more complex. This paper focuses on the part of the awareness framework that enables integration

*Contact author. Tel: (33) 04 93 00 26 70, Fax: -27; email: hummes@eurecom.fr. The described work is part of the ACOST research project, which is funded by the research institute CNET Lannion of France Telecom.

of new components and behavioral changes at run-time – second level tailoring following Mørch's taxonomy.

In section 2, frameworks and components are described as enabling technology for tailoring. In section 3, we describe the design of the awareness framework. Section 4 presents implementation issues, which highlight the benefits of using frameworks together with a standardized component model. Section 5 illustrates the tailoring activities of both the application developer and the end-user.

2 Frameworks and Components

Both component models and frameworks for object-oriented languages emerged of software engineering research to reduce development costs and to improve the overall quality of software.

A component is an independent "unit of software that encapsulates its design and implementation and offers interfaces to the outside, by which it may be composed with other components to form a larger whole" [2]. A component is self-descriptive and can such be analyzed automatically at run-time. Therefore, a standardized component model supports tailoring on different composition levels [8].

A framework is defined as "a reusable, 'semi-complete' application, that can be specialized to produce custom applications" [3]. Frameworks provide a reusable context for components [6].

Frameworks and components are different, but cooperating technologies. The main distinction between the component-based development and the framework approach is that frameworks offer an even higher degree of reusability. Components focus on code-reuse. Apart from code-reuse, frameworks provide design-reuse [6].

Regardless of their specific scope and aim, frameworks can be classified by the used technique for extension [3]. The specific peculiarity ranges between the two extremes of black-box and white-box frameworks. The extension mechanism of white-box frameworks is based on object-oriented inheritance and overriding of certain hook-methods. Black-box frameworks support extensibility by delegation. They define interfaces for components. Interface-conform components can be plugged into the framework.

In our approach, the framework is the invariant which is extended by application specific components. The extensibility of black-box frameworks and the reflective capabilities of components enable the tailoring of the system's behavior, which is described in the next sections.

3 Design Issues

We have developed an application framework which provides a **general awareness service.** This framework is described as **Awareness Service Framework** or short **AS Framework.** The main goals of the AS Framework are to support awareness in distributed systems, to offer run-time tailoring and to be independent of any specific problem domain.

Awareness in groupware systems means that the group members gain the needed information about the others to perform their work. We use the notion of awareness events, which are produced by components at the source and consumed by other components at the receiver. The main intent of the AS Framework is to offer the enabling infrastructure to support awareness among a distributed group of persons.

A high-level use case illustrates the requirements of the AS Framework (figure 1(a)). Two different actors are distinguished: Sources and Receivers. Components at the sources

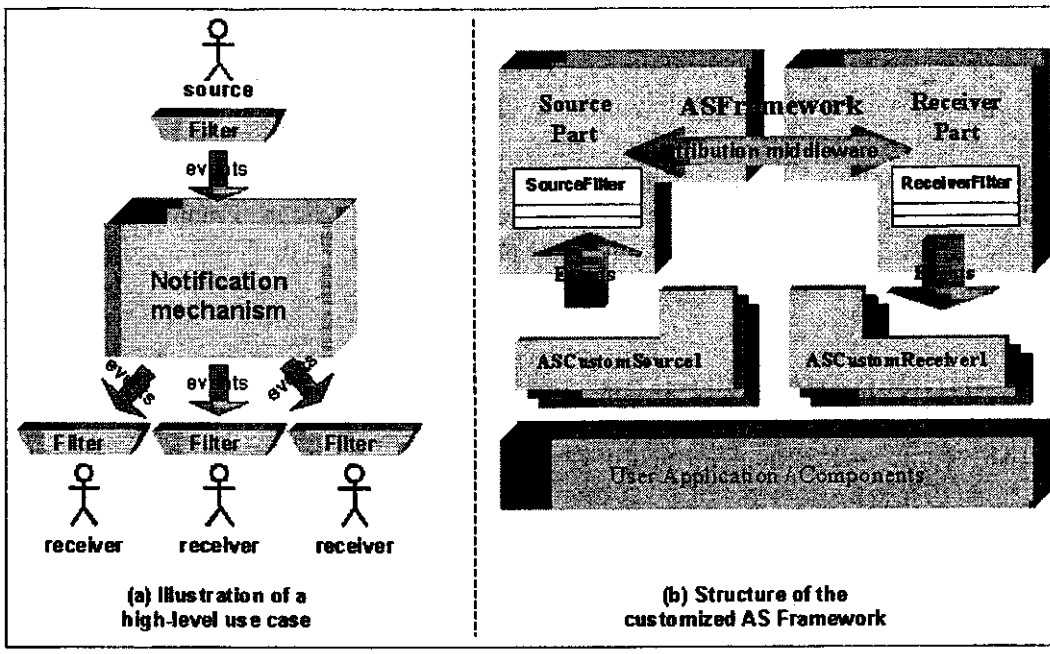

Figure 1: Overview of the AS Framework

emit awareness events. Filters can be applied by the user to ensure privacy. The events are then distributed to all receivers. Users set filters to receive only those events, they are interested in. The users specify the components which finally process the events. So, different presentations are achieved by end-user tailoring. The AS Framework itself is independent of the actual transmitted events. It does not have to be changed for different scenarios.

The end-user tailors the application by changing the settings of the filters. Should they feel uncomfortable with the publication of certain events, the users suppress them with source filters. The users, who receive awareness events, apply receiver filters to select those components that interpret the events. If no component handles a specific event, it is discarded.

Since the AS Framework offers a general awareness service, future event types cannot be anticipated. Each specific problem domain will have its own awareness events. Components which support those events can be plugged into the framework without changing the AS Framework itself. The application developer uses the offered interfaces to create new event types.

Figure 1(b) depicts the event flow. After the registration of the customized source components to the AS Framework, awareness events are accepted by the AS Framework. Using a distribution middleware, the framework passes the events to every receiver part of the AS Framework. At the receiver part, the events are passed to the customized receiver components with regard to the filter's setting.

Regardless to the level of tailoring, design-time and run-time tailoring are distinguished. The main difference between the two forms is that design-time tailoring is applied only once by the application developer or system integrator. In opposition, run-time tailoring is performed by every end-user frequently.

Design-time tailoring covers the creation of the problem specific event and listener types. In addition, the source and receiver components of those events are created. These components are plugged into the AS Framework and integrated into the user application.

The components may be integrated into the user application either at design-time or at run-time. With the help of an IDE, even the end-user can customize the application at design-time. Although not shown in this paper, the customized source and receiver components can be integrated also at run-time [5].

Run-time tailoring allows to change the event processing. The end-users apply the filtering mechanism at both the source and the receiver side to tailor the AS Framework to their specific needs. To facilitate this task, the AS Framework offers a default implementation, which already provides a graphical user interface.

4 Implementation Issues

The **AS Framework** is implemented in Java and uses the component model JavaBeans. The *standardized event model* of JavaBeans provides a foundation for the communication between components. In order to apply the event model in distributed systems, we reused prior developed group communication beans [5].

JavaBeans offers introspection to analyze components at run-time. In addition, the reflection API of Java enables very late binding of method calls by looking up interfaces and methods at run-time [2]. This allows to keep the framework invariant from the plugged domain specific components.

As outlined in section 3, the events may be filtered at the source and at the receiver side. The filtering mechanism at the source side guarantees the assurance of privacy needs. The filtering mechanism at the receiver side covers the ability to select the events and to change the processing of the events. In the following, we will concentrate on the filtering mechanism at the receiver side.

The standardized event model in conjunction with the introspection mechanism of JavaBeans allows to offer the filtering mechanism for events without knowing their specific type at design-time of the framework. The filter dynamically manages the connections between the interested listeners and the AS Framework. In addition, the filter ensures that the end-user is not overwhelmed with all events that may occur. Only those awareness events which may be emitted by the sources are considered. Using the filtering mechanism, the user decides, how the events are processed during run-time.

The filter stores the by introspection discovered event types and the interested receivers in a dictionary. One entry in the dictionary consists of a key, which is the event type, and a vector containing the references to all interested receivers. The main task of the filtering component is to manage the entries of the dictionary. Each entry reflects one filter setting. Every change to the filter settings causes an update of the receiver vector. Furthermore, the filter vetoes the attempt of incorrect settings.

Apart from the management of the filters settings, the filter is responsible to process the awareness events properly. Whenever an awareness event arrives the filter queries its dictionary and forwards the event to all interested receivers. Although the receiver components differ, they work with the same filtering mechanism.

In order to ensure that the filter considers only those events which can be emitted by a source component, the filtering mechanism must be supplied with all types of potential awareness events. Therefore, the event types of the source components are extracted during their registration process. This data are transmitted to the receivers. Referring to latecomers, a trade-off has been made. To ensure the tailorability of the filter, the information about the event types is queried. However, the late comers are not supplied with the events that have already been distributed.

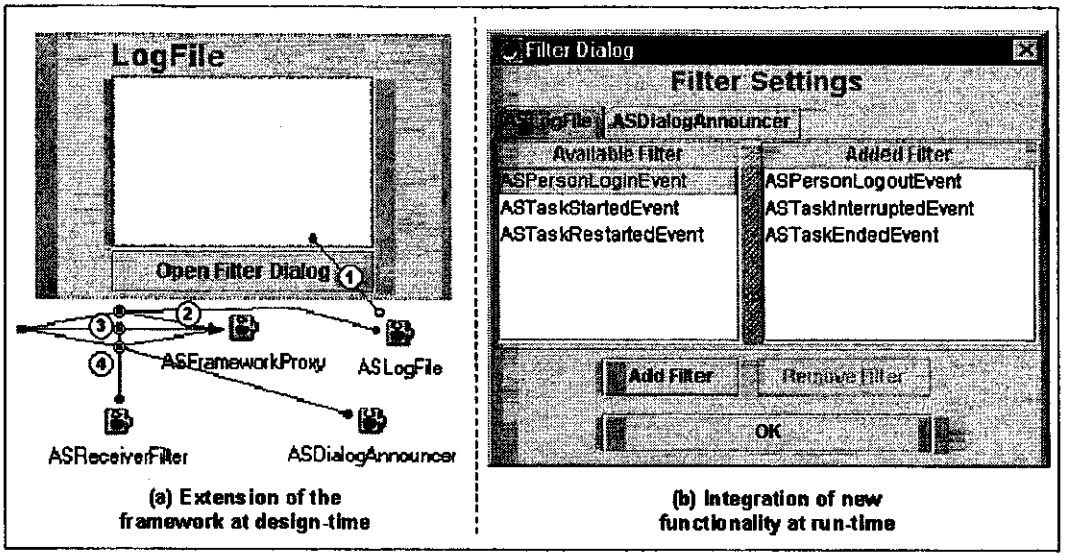

Figure 2: Illustration of the tailoring activities

5 Tailoring Example

This section introduces a small example to highlight the tailoring activities by both the application developer and the end-user.

Figure 2(a) shows, how a developer extends the AS Framework at design time. This example assumes that the receiver components are already developed and the event types are defined. The developer drops in a visual composition editor for JavaBeans the proxy for the framework (*ASFrameworkProxy*), a filter (*ASReceiverFilter*), two receiver components (*ASLogFile* and *ASDialogAnnouncer*) and the viewer for the log file. The model of the log file is attached to the corresponding view (1) and plugged into the AS Framework (2). In order to offer the facility to be informed immediately of certain events, additionally a dialog announcer is attached to the AS Framework (3). The receiver filter is plugged into the the AS Framework (4) to provide selection at run-time. These activities suffice to add two new receiver components among their filter to the AS Framework at design-time.

In the running application, the end-user can now tailor the such extended AS Framework. Pressing the button from the log file view, the user opens the default dialog of the filter component (figure 2(b)). The dialog offers two control buttons to change the settings: an "Add" and a "Remove" filter button. Corresponding to the selection of the list items the buttons are enabled. The filter discovers the receiver components at run-time. Each receiver is represented with a tab, which allows the user to easily select a receiver component. Corresponding to the selected component, the current filter settings are displayed.

The list of available filters contains the potential event types for the selected receiver. The event types of the list of added filters represent the enabled events, which are forwarded to the component. In addition, the filtering mechanism of the AS Framework ensures, that the user is not overwhelmed. Only those events that may be emitted by the sources and the receiver is able to handle are shown. Whenever new event types are introduced by a source component, the filter reflects this new state by adding the potential events to the list of available filters.

Figure 2(b) depicts a scenario, where the user has already tailored the filters settings of the receiver components. In this example, there are two receiver components, *ASLogFile* and *ASDialogAnnouncer*. The log file adds the events into a list and the dialog announcer

pops up a confirm dialog, whenever a selected event arrives.

In the example illustrated in figure 2(b), the log file is triggered each time a *ASPerson-Logout*, *ASTaskInterrupted* or *ASTaskEnded* event occurs. However, the user may also get informed if a new person logs in. Since the user has already selected the *ASPersonLogin* event in the available filter list, the "Add" button is available; when it is pressed, the selected filter is added and the view is updated

We have chosen these two receiver components to highlight the capabilities of supporting different styles of awareness by tailoring. The user perception of occurred events differ drastically for these two components. The log file works silently in the background, while the dialog announcer pops into foreground, when a selected event occurs. With the help of tailoring, the user can easily switch between different perception modes, which fulfills a major demand of awareness systems [1].

6 Conclusion

This paper has demonstrated how software engineering concepts such as components and frameworks are used to support tailorability. In conjunction with introspection, we have shown how to implement a reusable, tailorable, and general awareness groupware framework. The example of the awareness system validates our approach.

In further work, we will concentrate on the ability of adding and concatenating more sophisticated filters. We have already a first implementation of extension filter components, which analyze the contents of awareness events before forwarding an event.

References

[1] B. Buxton. Integrating the Periphery and Context: A New Taxonomy of Telematics. In *Proceedings of Graphics Interface'95*, pages 239–246, 1995. http://www.dgp.toronto.edu/people/rroom/research/papers/bg_fg/bg_fg.html.

[2] D. F. D'Souza and A. C. Wills. *Objects, Components and Frameworks With Uml: The Catalysis Approach.* Object Technology Series. Addison-Wesley, October 1998.

[3] M. E. Fayad and D. C. Schmidt. Object–oriented application frameworks. *Communications of the ACM*, 40(10):32–38, October 1997.

[4] S. Greenberg, C. Gutwin, and A. Cockburn. Using distortion-oriented displays to support workspace awareness. Research report 96/581/01, Department of Computer Science, University of Calgary, Calgary, Canada, November 1996.

[5] J. Hummes and B. Merialdo. Design of extensible component-based groupware. *Computer Supported Cooperative Work – An International Journal*, 1998. accepted for publication.

[6] R. E. Johnson. Frameworks = (components + patterns). *Communications of the ACM*, 40(10):39–42, October 1997.

[7] A. Mørch. Three levels of end-user tailoring: Customization, integration, and extension. In M. Kyng and L. Mathiassen, editors, *Computers and Design in Context*, chapter 3, pages 51–76. The MIT Press, Cambridge, MA, 1997.

[8] O. Stiemerling and A. B. Cremers. Tailorable component architectures for cscw-systems. In *Proceedings of the 6th Euromicro Workshop on Parallel and Distributed Programming*, pages 302–308, Madrid, Spain, January 1998. IEEE Press. http://www.cs.uni-bonn.de/ os/.