Porting a Distributed Meeting System to the Eclipse Communication Framework

Fabio Calefato, Filippo Lanubile, Mario Scalas
Dipartimento di Informatica
University of Bari
via E. Orabona, 4 - 70125
Bari, Italy
{calefato | lanubile | scalas}@di.uniba.it

ABSTRACT
eConference is a text-based conferencing tool that supports distributed teams in need for synchronous communication and structured discussion services. Other than offering communication services, it integrates an agenda and minutes editor, plus other control and coordination features, like hand raising and threaded discussion. The current version of the tool is based on Eclipse RCP and uses the eXtensible Messaging and Presence Protocol (XMPP) as the only communication infrastructure. The goal of this paper is presenting a work-in-progress to port the eConference tool on the Eclipse Communication Framework (ECF), which will enables us to abstract from the underlying communication protocol, provide better decoupling among components, and build additional team-support services.

Keywords
Computer-Mediated Communication, Eclipse RCP, Eclipse Communication Framework, XMPP.

1. INTRODUCTION
eConference is a text-based conferencing tool that supports distributed teams in need for synchronous communication and structured discussion services. Other than offering communication services, it integrates an agenda and minutes editor, plus other control and coordination features like hand raising. Our tool has been successfully used at the University of Bari and at the University of Victoria, both as a training aid and in laboratory experimentations conducted to validate our hypothesis about the adequateness of synchronous lean communication media for distributed requirements workshops [1][2].

The primary functionality provided by the tool is a closed group chat, augmented with agenda, meeting minutes editing and typing awareness capabilities. Around this basic functionality, other features have been built to help organizers control the discussion during distribute meetings. Indeed, eConference is structured to accommodate the needs of a meeting without becoming an unconstrained on-line chat discussion. The inceptive idea behind the eConference is to reduce the need for face-to-face meetings, using a simple collaboration tool that minimizes potential technical problems and decreases the time it would take to learn it.

The tool screen has six main areas: agenda, input panel, message board, hand raising panel, minutes editor, and presence panel (see Figure 1). The *agenda* indicates the status of the meeting, as well as the current item under discussion. The *input panel* enables participants to type and send statements during the discussion. The *message board* is the area where the meeting discussion takes place. The *hand raising panel* is used to enable turn-based discussions, by mimicking the hand-raise social protocol. The *minutes editor* is used to synthesize a summary of the discussion. Finally, the *presence panel* shows participants currently logged in and the role they play.

The present version of the tool is built on top of the Eclipse Rich Client Platform (RCP) [5] and currently uses the eXtensible Messaging and Presence Protocol (XMPP) [10] as the only communication infrastructure. The goal of this paper is presenting a work-in-progress to port the eConference tool on the Eclipse Communication Framework (ECF) [3], which will enable us to abstract from the underlying communication protocol and build additional team support services. ECF provides, indeed, a general component model for creating plugins, tools, or fully distributed Eclipse-RCP applications that require messaging and communication services. Porting the eConference tool on ECF will help us to provide distributed teams with a lightweight means to conduct synchronous, structured communication, leveraging any supported communication protocol and/or infrastructure that members will choose to best fit their needs in term of ease of use, costs, privacy and security. Besides, ECF comes with libraries of reusable components that can be used for creating new extensions, as new collaboration needs emerge.

The remainder of the paper is structured as follows. In Section 2 we briefly describe the features of eConference, and sum up the history and evolution of the tool. In Section 3 we motivate the switch to a new architecture and discuss the ECF, whereas, in Section 4, we describe the porting of eConference to the new framework, discussing the major architectural changes and how we are further evolving our tool. Finally, Section 5 presents conclusions and outlines future work.

2. THE EVOLUTION OF ECONFERENCE
eConference has evolved over three generations, as first we changed the underlying communication framework, from the JXTA P2P platform (1st version) to the XMPP client/server protocol (2nd version), and then its overall architecture, from traditional plugin to pure-plugin system, built on top of the Eclipse Rich Client Platform (3rd version). In the following we briefly sum up the history and evolution of our tool.

The first version of the tool, named P2PConference, started in March 2002, using the Java binding of JXTA [8], and was completely discontinued in 2004. The choice of adopting a fully-decentralized, P2P approach stemmed from our intent of building a distributed meeting system easy to use and set up, with administration cost kept at minimum. JXTA seemed a promising technology because, by exploiting its virtual network, we aimed at using existing resources that live on the edge of the Internet infrastructure (e.g., bandwidth, storage space of the PCs running

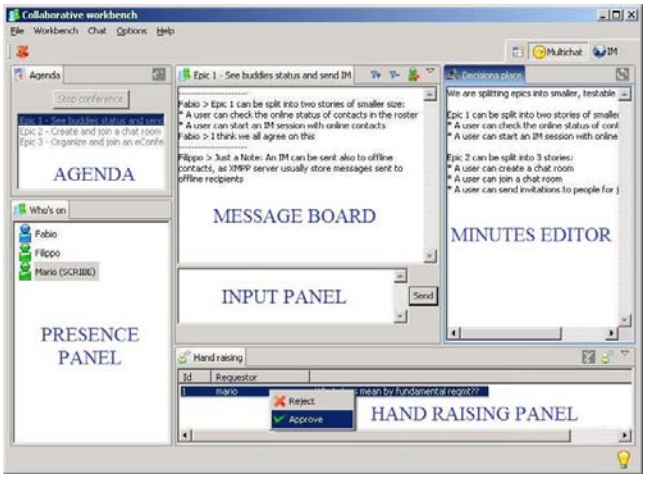

Figure 1. A screenshot of eConference

eConference). No central server to maintain and no single point of failure is what the platform promised, but JXTA did not deliver on all of its promises. We developed with eight consecutive releases of the platform (from ver. 1.0-build49b to ver. 2.2.1).

Our experience with JXTA was not positive mainly because it failed at delivering a robust, general-purpose platform that can serve as the building blocks for P2P communication-intensive applications. Paradoxically, its messaging framework proved inappropriate for implementing group communication without using a client/server-like approach. Developing a brief proof-of-concept experiment would have probably shown that JXTA pipe services were not suitable for many-to-many communication in pure P2P approach, and that the platform API was too low level and complex. The proof-of-concept, however, would have never spotted the platform API instability issues, which probably derived from being too low level. Stability is a key aspect of any API to guarantee the promised independence between API producers and API consumers.

Considering the several issues we encountered during the development of P2PConference, we decided to port the tool onto a different communication platform. Our choice fell onto Jabber/XMPP (in short XMPP, hereafter). In the end of 2004, we refactored our tool to use XMPP, and this porting was named eConference. In our experience XMPP proved to be more stable, easy-to-use, and reliable than JXTA. Our preference for XMPP over JXTA is not based on a preference for the client/server paradigm over P2P, though. On the whole, XMPP is a good choice for applications that need an extensible messaging framework. Indeed, its intrinsic extensibility has allowed us to easily expand the native multi-user chat capability, adding the extra functionality we needed to build eConference (e.g., agenda, minutes editor, hand raising). Obviously, also developing with XMPP was not without problems, mostly stemming from the limitations of the current multi-user chat service, related to the synchronization of chat content, which occurs in case of unintentional disconnections of clients or when there are latecomers.

Finally, in the beginning of 2006 we started to consider evolving the tool architecture in a way that supported the composition of a larger system that is not pre-structured and the extensibility in ways that cannot be foreseen. We decided to build another version of our prototype exploiting the Eclipse Rich Client Platform (RCP). [5]. Eclipse RCP is based on Equinox, the Eclipse implementation of the OSGi specs, which define a Java-based dynamic component model [9]. While mostly known as a powerful Java IDE, now Eclipse is actually a universal plugin platform for creating other platforms. Eclipse RCP is a pure-plugin system and, hence, fully extensible by architectural design. This new modular architecture looked very attractive to us because it promised to help us in developing with a focus on modular functionality and writing new plugins for missing functions. Our experience with Eclipse RCP was very positive. With a little more coding, the present version of our tool offers all the benefits seen in Eclipse (e.g., pure-plugin architecture, perspectives, update manager, help system). The only, but negligible, problems we encountered were some erratic behaviors during the process of product export (to make the application executable outside of the Eclipse workbench), and the final size of the product itself (which grew up to almost 9 Megabytes, while the size of the our own code, plus all the other third-party libraries used, account for only 980 Kilobytes). This is a limitation that is already being worked on by the Eclipse community [6].

3. THE NEED FOR A NEW ARCHITECTURE

Although designed to be independent from the network protocol and implemented using a pure-plugin architecture, the present version of eConference suffers from some architectural drawbacks. Among these limitations, the major ones include 1) a low-level, abstract network layer, too expensive to maintain on our own; 2) a burdensome publish/subscribe subsystem, in which every bundle implement the Observer pattern without taking advantage of the Eclipse/OSGi internals for the dispatching of events in a dynamic pure-plugin environment.

Although we were working only with XMPP, for the third generation of the eConference tool, an abstract network infrastructure layer was designed and implemented to allow the use of other communication protocols in the future, without a severe impact on the code base. Consequently, all the domain-specific features were built on that API. As a side effect, the low-level network layer had to be maintained in addition to the application itself, while we wanted to concentrate efforts on the eConference domain components.

The Eclipse Communication Framework (ECF) [3] provides RCP-based applications with an abstract communication layer that not only replaces the whole network infrastructure layer of eConference, but also provides some of the collaborative features available in our tool, either in terms of API or visual components. Thus, ECF can be employed to replace the communication layer and some domain-specific parts of our tool, with the promise of relieving us from the burden of maintaining an abstract network layer to cope with future evolutions.

ECF is a set of reusable components, which introduce, within the Eclipse platform, typical collaborative services and features (e.g., instant messaging, white-boarding), bundled as standard plugins that can be reused in whatever context (e.g., the JDT, as well as any rich-client application, built on top of Eclipse RCP). Such components include core API definitions, graphical user interface widgets, and interfaces for specific network protocols. The ECF core includes an extensible framework, the SharedObject API which is of critical importance for distributed applications built

using the MVC pattern (like a distributed meeting system), since they need to share and synchronize the model(s) across network. Thus, the SharedObject API provides a way for sharing data at application-level, without having to bother with protocol-specific details. The other notable components, available in ECF, include the Presence API, which handles the presence events, the File Transfer API, for sharing content between remote users, and the Remote Services API, which provides a RPC-like mechanism for remote procedure calls.

All these APIs provide a high-level abstraction layer that enables ECF-based applications to support multiple protocols wholesale, ignoring any implementation detail, which is transparently handled by the underlying framework. ECF, in fact, already provides the implementations (called *providers*) of abstract interfaces for the most used communication protocols (e.g., XMPP, MSN, and Skype). Besides, support to new network protocols can be added to ECF at any time, by defining and implementing additional providers. We observed that the best supported protocol is XMPP, probably because it is an open IETF standard. Usually, new features are first implemented for it and then adapted to other providers. For this reason, we are going to use just the richer XMPP provider. As soon as the porting is completed, we will include other providers and verify that the promise of multi-protocol abstraction has been kept.

For the sake of space, we are not describing the use of the Dependency Injection pattern [7], to solve the architectural concern of coupling, and the reimplementation of the conferencing service on top of the standard OSGi Event Administration Service [9], to improve the publish/subscribe mechanism in eConference.

4. PORTING ECONFERENCE TO ECF

While it is not possible to illustrate all the architectural changes, in this section we show at least the major ones and how we are using ECF to build the new version of eConference. We also introduce the *Automatic Conference Assistant*, a remote agent that will assist users during the arrangement and execution of conferencing sessions.

4.1 Replacement of the Communication Layer

The porting of eConference to ECF was not a straightforward task, as one might expect. Indeed, between eConference 3 and ECF there was a large overlapping of both the whole network infrastructure layer and the features provided, either in terms of API and visual components.

The main design similarity between the communication infrastructures in eConference 3 and the ECF regarded the separation of functionalities from their implementation, realized by a complex core set of interfaces. Thus, both architectures provided the basic interfaces for protocol abstraction and, then, the adoption of ECF, suggested a whole rewrite of the application, with only a limited portion of the existing GUI code reused. With the development of eConference 3 we realized that we were not able to sustain the cost of maintaining an abstract communication network infrastructure on our own. Hence, the cost of rewriting the application almost from scratch was justified by our intension of employing a standard network technology, maintained separately from our tool, by a larger community than that of eConference will ever be, given its more restricted audience. In addition, consistently with the Eclipse RCP goal, also the ECF architecture is designed for extensibility. This means that adding new features in a second time (i.e. shared web browsing) won't break our existing code.

As stated earlier, ECF does not come only with a set of non-GUI interfaces or network services. In fact, it includes several out-of-the box graphical components, such as contacts roster, chat editors, and user account management, which can be embedded in any Eclipse-based application. Table 1 shows that ECF can replace most of the features available in eConference 3. Nevertheless, the most specific plugins (i.e., hand raising, threaded message board, event manager) had to be redeveloped using the API of ECF, because they were too dependent on the previous design to be just ported. Hence, the fourth generation of eConference is being developed as a rich-client application that uses plugins either available out-of-the-box in ECF, or developed ad hoc upon its abstraction framework (see Figure 2).

In eConference 3 we used the Model-View-Controller (MVC) architectural pattern both within each plugin and across their whole set. MVC is an architectural pattern that separates the data (model), from presentation (view) and logic (controller): the controller changes the model; the model notifies the view about change events; the view updates accordingly and notifies controllers about user input. For instance, in eConference 3, the conferencing session was designed as a master controller that organized the work of each bundle and, thus, had to know in advance (i.e., statically) about the views to be activated. This approach, which we adopted due to our inexperience with pure-plugin architectures, did not fully take advantage of the dynamic environment provided by the Eclipse platform. In the new version we have addressed this problem by properly using an extension point based event subsystem.

In addition, in MVC-based distributed applications, like a distributed meeting system, the model needs to be synchronized across the network between different peers through the exchange of messages containing state change primitives. In eConference 3, a network service wrapped up the model and provided the facilities for notifying change events to remote peers while listening for incoming (XMPP) messages to update the model accordingly.

Table 1. Components required by eConference 3 and their support in ECF (only the major components are listed)

Available in eConference 3	Provided by ECF
Contacts management	Yes
Message board*	Partially
Roster View	Yes
Extension Points API	Yes
Hand Raising	No
White board	Yes
Conferencing Events Manager (invitations, reminders, …)	No
Account creation / Login Manager	Yes

* Does not support multiple discussion threads

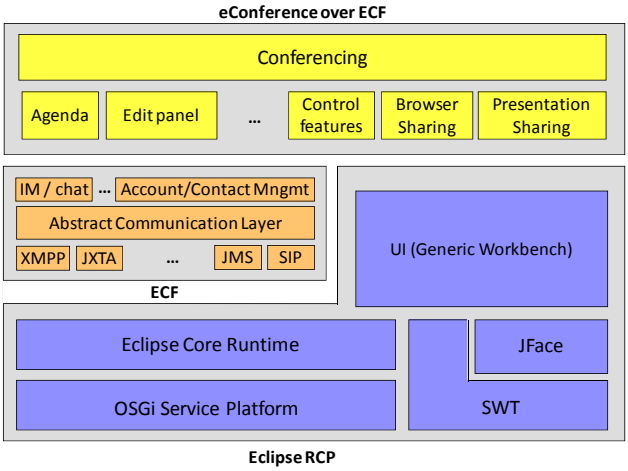

Figure 2. The 4th generation of eConference is a rich client application composed by ad hoc and ECF-native plugins

However, while the communication bus was composed by an extensible infrastructure capable of supporting additional requirements, services and models were tightly coupled: the controller explicitly knew about the service and used it for accessing the model and perform changes. The service captured changes to the model and notified remote hosts. On the other side, the service received remote messages and updated the model, triggering new events to update views and other observers. This communication bus was the main architectural block we replace by plugging ECF in eConference 4. The general MVC design scheme is still applied (see Figure 3).

Although the change may seem somewhat trivial, with ECF, however, we are actually relieved from the burden of continuously create abstract network messages to wrap and handle low-level protocol-specific messages.

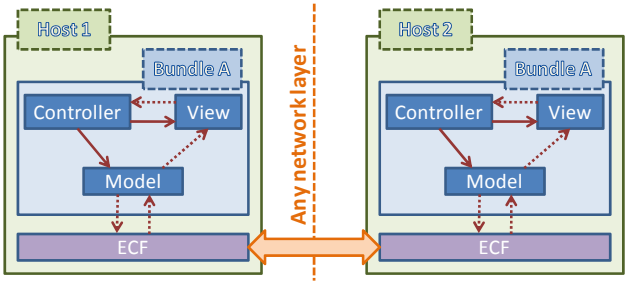

Figure 3. ECF as communication bus in eConference 4

4.2 Extending eConference with the Conference Assistant

In the fourth generation of eConference, from the user point of view, the most visible change during conferencing sessions will be the explicit presence of the *Automatic Conference Assistant*, a software agent, built upon the ECF Bot framework [4]. The conference assistant is deployed as an independent agent permanently logged on some network server, appearing just as a contact in user's roster and waiting for some call from users. As of this writing, the agent is only assisting in basic tasks, like sending notifications to invitees. However, in the future releases, we are going to extend it with additional capabilities for assisting users during conversations, thanks to the extensibility of the ECF Robot API. According to our vision, when a user wants to set-up a new conference, he/she will just have to contact the assistant, which will set up the conference (e.g., allocate the room on some network server, set permissions, send notifications to all invitees) on the behalf of the user. It will also log into the room and wait for further requests.

5. CONCLUSIONS & FUTURE WORK

In this paper we have presented the porting of eConference, a distributed meeting system built upon Eclipse RCP and XMPP, to ECF, a communication framework for rich-client application that provides transparent support to the most used communication protocols. Although the porting is still in progress, we have already taken advantage of ECF also for enhancing the feature set of our tool: Using the ECF Robot API, we have built a software agent that assists users in the tasks of arranging and running remote conferencing sessions.

ACKNOWLEDGMENTS

This work has been partially supported by the 2006 Eclipse Innovation Award and the MiUR-Italy under grant 2006 "METAMORPHOS". We wish to thank Carla Milani and Cristina Cannone of IBM University Relations, and Doug Tidwell of IBM's Software Group Strategy organization, for their support and encouragement.

REFERENCES

[1] Calefato, F., and Lanubile, F.. *Using The Econference Tool for Synchronous Distributed Requirements Workshops,* Proc. 1st Int'l Workshop on Distributed Software Development (DiSD 2005, August 2005

[2] Calefato, F., Damian, D., and Lanubile, F. *An Empirical Investigation on Text-Based Communication in Distributed Requirements Engineering*, Proc. 2nd Int'l Conf. Global Software Engineering (ICGSE '07), Munich, Germany, 27-30 August, 2007.

[3] Eclipse Communication Framework homepage, http://www.eclipse.org/ecf/

[4] ECF Bot Framework, http://wiki.eclipse.org/index.php/Bot_Framework

[5] Eclipse Rich Client Platform, http://www.eclipse.org/rcp, last visited: July 1st 2007

[6] Eclipse RCP product size enhancement, https://bugs.eclipse.org/bugs/show_bug.cgi?id=53338, last visited: July 1st 2007

[7] Fowler, M. *Inversion of Control Containers and the Dependency Injection pattern*, http://martinfowler.com/articles/injection.html

[8] JXTA homepage, https://jxta.dev.java.net/

[9] The OSGi Alliance, OSGi Service Platform Service Compendium, Release 4, Version 4.1, April 2007

[10] XMPP protocol specifications, 2004, http://www.xmpp.org/specs/