Evolving a Text-Based Conferencing System: An Experience Report

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

Abstract—**In this paper we describe the evolution of eConference, a text-based conferencing system that has turned into a collaborative platform. We draw the lessons learned from the evolution process, as first we changed the underlying communication framework, from the JXTA P2P platform to the XMPP client/server protocol, and then its overall architecture, from traditional plugin to pure-plugin system, built on top of the Eclipse Rich Client Platform.**

Keywords-distributed meeting system, Jabber/XMPP, JXTA, communication frameworks, Eclipse RCP, plugin systems;

I. Introduction

Working across distances has become commonplace today. Nevertheless, distance poses hard challenges due to substantial reduction in frequency and richness of communication [12].

In this paper we describe the evolution of eConference, a text-based conferencing system developed at the University of Bari. The tool is part of a broader research effort that aims at better supporting interaction of nimble, ad hoc teams (i.e., goal-oriented workgroups with no history and future, such as distributed inspection teams), which need low-cost administration infrastructure just to complete the task at hand. As an instance of the broader category of distributed meeting systems, complexity and usability are major problems. People need to focus on the content of their meeting, not on the meeting tool itself, and thus, features have to be chosen carefully to maximize the tool effectiveness while minimizing complexity. However, the scope of a distributed meeting system goes beyond supporting users' activities during the meeting itself, but also to facilitating the arrangement and set up of meetings.

Here we also discuss the lessons learned from the evolution of the eConference tool. Our prototype has evolved through the years, first changing the underlying communication framework, from the JXTA P2P platform to the XMPP client/server protocol, which has proved to be a more robust and reliable solution to develop an extensible tool for distributed meetings. Then, in the latest version, eConference has evolved from a conferencing system to a pure-plugin collaborative framework, built on top of the Eclipse Rich Client Platform.

The remainder of this paper is structured as follows. In Section 2 we briefly describe the tool. In Section 3 problems encountered during the development with JXTA are discussed. Section 4 explains XMPP and the motivation for its adoption. Section 5 first illustrates the pilot study and then discusses the results, which have been used to further improve the tool. In Section 6 we discuss the lessons learned. Finally, in Section 7 we draw conclusions and provide directions for future work.

II. Tool Description

eConference is a text-based distributed meeting system. 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, edit panel, and presence panel (see Fig. 1). The *agenda* indicates the status of the meeting (“started”, “stopped”), 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. The *edit panel* is used to synthesize a summary of the discussion. The *presence panel* shows participants currently logged in and the role they play. Finally, the *hand raise panel* mimics the hand-raise social protocol that people use during real meetings to coordinate discussion and turn-taking. Compared to the real-life social protocol, the hand raise feature of eConference also gives to the moderator the ability to preview queued questions, showing a tooltip when hovering the mouse pointer over them (see Fig. 2).

The organization of a meeting in eConference follows a protocol inspired by the eWorkshop tool [2]. The *meeting organizer* is guided by a wizard through a few steps in order to 1) define the main topic and the agenda of the meeting, 2) specify participants invited and their roles, and, finally, 3), schedule the conference and training sessions, if necessary.

When inviting participants, the meeting organizer has to select who will act as moderator and scribe. The *moderator* is supposed to facilitate the meeting and has control over participants, whereas the *scribe* captures and summarizes the discussion in the edit panel. Thus, the content of the panel becomes the first draft of the meeting minutes. The role of scribe is flexible in that the participant who is selected as scribe can change over time and there can be more than one scribe at a time. Finally, some participants may also be invited as *observers*, in that they will attend the meeting, but they will not be able to actively contribute to the discussion.

III. P2PConference: Problems with JXTA

The first version of our tool, also known as P2PConference [4], was developed using the Java binding of JXTA [13]. Project JXTA is an open-source effort led by Sun Microsystems, which provides a general purpose, language independent middleware for building P2P applications. It defines an XML-based suite of protocols that build a virtual overlay network on top of the existing physical network, with its own addressing and routing mechanisms. The building blocks of the JXTA network are rendezvous and relay peers, also referred to as super peers, which deal respectively with the resources discovery and message routing.

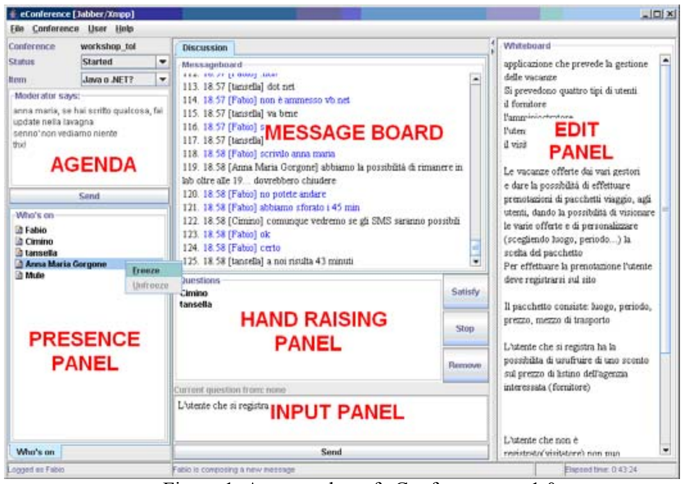
The development of P2PConference started in March 2002 using the Java binding of JXTA. The first useable version of P2PConference was released at the end of 2002. The project was active during the year 2003, when file sharing and co-browsing features were added, but it was completely discontinued in 2004. Eight different releases of the platform were used for the development of P2PConference.

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 eConference). No central server to maintain and no single point of failure is what the platform promised. JXTA did not deliver on all of its promises though.[1]

A. Low level API & End User Complexity

One of the main disadvantages of JXTA was its overly low-level API, which made developers subject to frequent changes. A low level API was probably considered as a means to build a general purpose middleware and grant flexibility to developers, but it ended up adding considerable amount of extra code and complexity. Furthermore, JXTA was not only complex for developers, but even for end users who had to know about its internals because, the first time a JXTA peer was started and each time network configuration changed, the platform had to be manually set up through a complex configurator.

[1] All the experiences reported and judgments expressed here refer to versions of the platform previous to JXTA 2.3.



workshop_tol
Started
Java o .NET?

anna maria, se hai scritto qualcosa, fai update nella lavagna
senno' non vediamo niente
thx

Fabio
Cimino
tansella
Anna Maria Gorgone
Mule

113. 18.57 [tansella] dot net
114. 18.57 [Fabio] non è ammesso vb net
115. 18.57 [tansella] va bene
116. 18.57 [Fabio] s
117. 18.57 [tansella]
118. 18.58 [Fabio] scrivilo anna maria
119. 18.58 [Anna Maria Gorgone] abbiamo la possibilità di rimanere in lab oltre alle 19… dovrebbero chiudere
120. 18.58 [Fabio] no potete andare
121. 18.58 [Fabio] abbiamo sforato i 45 min
122. 18.58 [Cimino] comunque vedremo se gli SMS saranno possibili
123. 18.58 [Fabio] ok
124. 18.58 [Fabio] certo
125. 18.58 [tansella] a noi risulta 43 minuti

Cimino
tansella
Current question from: none
L'utente che si registra

applicazione che prevede la gestione delle vacanze
Si prevedono quattro tipi di utenti
il fornitore
l'amministratore
l'uten
il visi

Le vacanze offerte dai vari gestori e dare la possibilità di effettuare prenotazioni di pacchetti viaggio, agli utenti, dando la possibilità di visionare le varie offerte e di personalizzare (scegliendo luogo, periodo...) la scelta del pacchetto
Per effettuare la prenotazione l'utente deve registrarsi sul sito

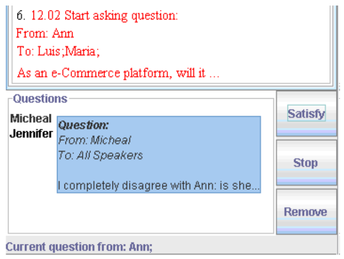
Il pacchetto consiste: luogo, periodo, prezzo, mezzo di trasporto

L'utente che si registra ha la possibilita di usufruire di uno sconto sul prezzo di listino dell'agenzia interessata (fornitore)

L'utente che non è

Logged as Fabio
Fabio is composing a new message
Elapsed time: 0:43:24

Figure 1. A screenshot of eConference ver 1.0



6. 12.02 Start asking question:
From: Ann
To: Luis;Maria;
As an e-Commerce platform, will it ...

Micheal
Jennifer
Question:
From: Micheal
To: All Speakers
I completely disagree with Ann: is she...

Current question from: Ann;

Figure 2. Preview of a question as tooltip

B. Lack of reliable messaging mechanisms

The main issue that forced us to abandon the P2P platform was the inadequateness of the JXTA messaging service. In JXTA the fundamental abstraction used for inter-peer communication is the pipe, a virtual channel that consists of an input and an output end. JXTA offered different alternatives to implement group communication in our prototype. Since the release of JXTA 1.0, the JXTA core protocol specification defines three kinds of core pipes: unicast, secure, and propagate pipes. We also considered non-core pipe services, namely bidirectional pipes and JXTA Sockets, which were available only since JXTA 2.0.

We chose to use the propagate pipe service because, its one-to-many communication mode was the most apt for implementing group communication in a decentralized system. Despite the fact that reliability was not ensured, propagate pipe was actually the only practical solution, as all the other communication services were meant for point-to-point communication. Indeed, the use of any one-to-one service would have entailed the need to set up in the peer group a super peer that behaved very similar to a server (i.e., receive a message from a peer, then route it to all other known peers). This solution would have defeated any motivation for experimenting a P2P approach, as it would have been equivalent to using a traditional client/server solution, but on a P2P platform, and with much more complexity. Unfortunately, in our experience propagate pipes and discovery on rendezvous peers proved to be too much unreliable, unless all the peers were in the same subnet using multicast. Instead, when peers were dispersed over the Internet, results were discouraging, with high message drop rate and low resource discovery recall. Although we did not collect data from formal tests or benchmarks, other research studies have somewhat confirmed the problems of the JXTA messaging architecture in general [1],[11].

IV. Rebuilding over Jabber/XMPP

JXTA was released in 2001. After having developed with it for over a year and a half, our feeling was that it had been released in a yet too-early stage, not mature enough, probably just on the heels of the growing popularity and hype of P2P. In addition, the JXTA messaging service proved to be inadequate for developing a fully decentralized meeting system. Considered 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.

The Jabber project started in 1999 to create an open alternative to closed instant messaging (IM) and presence services. In 2002 the Jabber Software Foundation contributed the Jabber core XML streaming protocols to the IETF as XMPP, eXtensible Messaging and Presence Protocol. XMPP was finally approved in early 2004 (RFC 3920–3923) [17] and now it is being used to build not only a large and open IM network, but also and mostly to develop a wide range of XML-based applications, from network-management systems to online gaming networks, content syndication, and remote instrument monitoring [15].

Compared to JXTA, XMPP offered us three clear advantages. First, XMPP provides by design a reliable and, extensible architecture conceived for near real-time presence, messaging and structured data exchange. The second advantage is simplicity. XMPP has been conceived to delegate complexities to the servers as much as possible, so that developers can keep focused on the application logic, and the clients can stay lightweight and simple. Furthermore, the intrinsic extensibility of XMPP allows leveraging the existing services (e.g., multi-user chat) and also adding extra features (e.g. agenda, hand raise). Third, the IETF standardization of the core XMPP protocols has generated a plethora of high level XMPP APIs, available for a number of programming languages. XMPP programmers do not even need to know the protocol details, as all the raw XML exchanges are hidden by the use of any of these APIs.

At a first glance, compared to our previous P2P solution, choosing XMPP might look somewhat contradictory. However its architecture is not purely client/server, but a hybrid, very similar to email. XMPP entities are identified by a unique Jabber ID, which is all that is needed in order to exchange messages. The XMPP network is formed by hundreds public servers, which are all interconnected to form the XMPP federation. Although running an XMPP server which is not part of the federation is still possible for a corporate LAN, from our perspective, using the XMPP federation was preferable because it allowed us to develop a client/server meeting system, without abandoning the goal of keeping at minimum the infrastructure costs (i.e., again no central server to install and administer, and no infrastructure costs, as in the case of P2P).

We refactored P2PConference to make the tool independent of the underlying communication protocol. The implementation that used XMPP as network backend was called eConference (ver. 2.0) [7]. Unfortunately, co-browsing and file sharing features could not be easily migrated to work with XMPP, as they needed to be rewritten from scratch. These were not features related to communication though, and so we chose to run a pilot study without them anyway.

V. eConference as a Collaborative Platform

When we ported our tool from JXTA to XMPP, we lost some features (namely file sharing and web-browser sharing), because they could not be easily adapted, but needed to be rewritten from scratch. From this idea we realized that we wished to avoid all the effort spent in adapting the tool to support another communication platform in the future. Furthermore, we conducted a pilot study at the University of Bari, from which we collected many useful requests of feature extensions. Nevertheless, it is overly challenging to foresee all the possible features needed to make a meeting system flexible enough to be apt for all contexts. These concerns led us to think about evolving eConference from a simple collaborative application to a collaborative platform. Our intention was to have a platform that offered as core functionality a reliable, extensible, and scalable messaging framework, on the top of which new collaborative features could be added as plugins. We also wanted to support multiple communication protocols through pluggable network backends, so as to have the possibility to add a new one at any time by writing only the specialized code for its integration.

To support the composition of a larger system that is not pre-structured, or to extend it in ways that cannot be foreseen, an architecture that fully supports extensibility is needed. We decided to build another prototype exploiting the Eclipse Rich Client Platform (RCP) [14]. Since the release of version 3.0, Eclipse has evolved to become an open and fully extensible framework for developing rich client applications. While mostly known as a powerful Java IDE, now Eclipse is actually a universal plug-in 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 plug-ins for missing functions. In traditional plugin architectures plugins are mere add-ons that extend the functionality of a host application, i.e., binary components not compiled into the application, but linked via well-defined interfaces and callbacks. Instead, in pure-plugin systems plugins become the building blocks of the architecture, as almost everything is a plugin and, consequently, the host application becomes a runtime engine with no inherent end-user functionality. Instead, every application behavior is provided by a federation of plugins orchestrated by the engine [3].

The latest version of eConference is a *rich client application*, built upon Eclipse RCP. Besides all the benefits that come from using native widgets, our tool has inherited all the capabilities of the RCP, in terms of extensibility and classical concepts from the Eclipse world, like views (i.e., UI widgets) and perspectives (i.e., the particular arrangements of views in the application windows). The experience gained in developing the first two versions of our prototype has helped us in identifying the basic features that a communication protocol must provide to work with our tool. Thus, in our rich client application we have developed an abstract network layer that exposes the core communication features, which have to be mapped onto concrete network backends. If the mapping cannot be completed for a given protocol, it means that the protocol does not guarantee the minimal requirements needed. At the moment the only network backend supported is XMPP. The new eConference has been developed incrementally, using a story-driven agile process. In the following we describe some of the epics, i.e., the high-level, long stories that have then been split into smaller, testable user stories.

1) *Epic 1: A user can see presence status of contacts and send instant messages*

We started building a feature (i.e., a collection of plugins in Eclipse terminology) to provide instant messaging and presence awareness capabilities, which are both at the core of XMPP and, thus, the mapping was almost effortless.

2) *Epic 2: A user can create and join a chat room*

We extended the existing feature to implement multi-user chat for reliable group communication. Unlike presence and instant messaging, multi-user chat is not a core functionality of XMPP. Instead, it is available as a XMPP Extension Proposal (XEP). The Jabber Software Foundation develops extensions to XMPP through a standards process centered on XEPs. The Multi-User Chat XEP [16] is the protocol extension proposed for managing chat rooms. Though not in the final stage yet, this draft is already supported by all the hundreds public servers belonging to the XMPP federation. One limit we found with the multi-user chat extension was that it did not handle typing awareness. We tackled this problem leveraging the intrinsic extensibility of XMPP and creating a custom typing notification, sent whenever a participant in the room starts to type.

3) *Epic 3: A user can create and join an eConference*

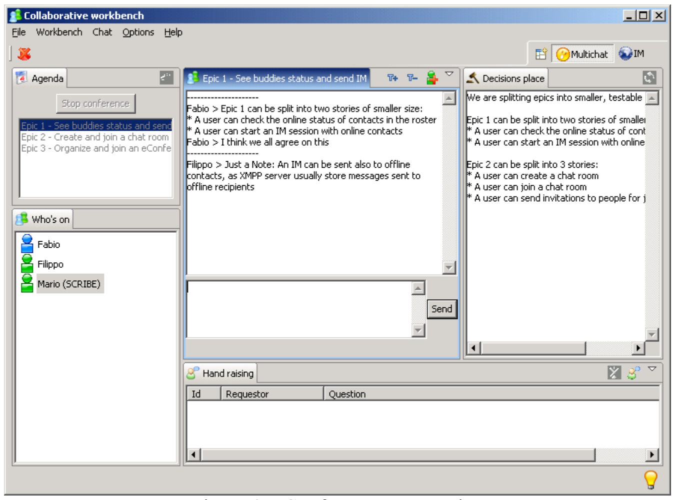
Finally, leveraging the functionality already provided by the multi-user chat feature, we developed new plugins for each view needed, namely the agenda, edit panel and hand raising, so as to obtain the overall "eConference feature" (see Fig. 3). Indeed, rather than an application, eConference is now just a feature of our rich client application, with its own perspective. Similarly, when developing new features for web-browser and presentation sharing, we will build onto the existing features and plugins, and create new perspectives to optimize the arrangements of the UI views. To complete the implementation of the eConference feature, we also added support for one-to-one private messaging and we implemented the item-based discussion threads, so that all the utterances related to an item are grouped together.

VI. Lessons Learned

In this section we draw three lessons, which may be of help when making architectural decisions that have the potential to affect the evolution process.

A. *Stability as a key aspect*

Our experience with JXTA was not positive. Although it aimed at addressing a real problem (i.e., the fragmentation and redundancy of services offered by the plethora of existing P2P



Epic 1 - See buddies status and send
Epic 2 - Create and join a chat room
Epic 3 - Organize and join an eConfe

Fabio > Epic 1 can be split into two stories of smaller size:
* A user can check the online status of contacts in the roster
* A user can start an IM session with online contacts
Fabio > I think we all agree on this

Filippo > Just a Note: An IM can be sent also to offline contacts, as XMPP server usually store messages sent to offline recipients

We are splitting epics into smaller, testable

Epic 1 can be split into two stories of smalle
* A user can check the online status of cont
* A user can start an IM session with online

Epic 2 can be split into 3 stories:
* A user can create a chat room
* A user can join a chat room
* A user can send invitations to people for j

Figure 3. eConference perspective

systems), JXTA 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 spike 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 spike, however, would have never spotted the platform API instability issues, which probably derived from being too low level.

When building a new application from existing components you make implicit assumptions or have expectations, which often turn out to be wrong or just do not match the actual environments [10]. Stability is a key aspect of any API to guarantee the promised independence between API producers (i.e., software developers who write the API implementation) and API consumers (i.e., software developers who write code with method calls to the API). Changes in the API itself require changes in the API consumers' code because this code use services provided by the API [6]. As API consumers, we did not expect the JXTA API to change often and we assumed the platform not to have backward compatibility issues as well.

B. *Complexity on server side Vs. Extensibility on client side*

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. 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 multi-user chat capability, adding the extra functionality we needed to build eConference.

Obviously, also developing with XMPP was not without problems, which mostly stemmed from the limitations of the current multi-user chat extension proposal draft. A drawback of the latest implementation of our tool is related to the synchronization that occurs in case of unintentional disconnections of clients, or when there are latecomers. The multi-user chat extension ensures persistency, delegating to servers the tasks of history logging and dispatching. Thus, in both cases, all the events are sent back to clients in order.

However, all the custom notifications we added, such as agenda items selection and edit panel updates, are logged in the history as if they were participants' utterances. That is, during synchronization of clients, servers do not send the current content of the edit panel or agenda all at once, instead each and every change made is sent in chronological order. Furthermore, synchronization also includes useless notifications, like speaking requests and typing awareness. As a quick fix, on the client side we could prevent these events to be saved in the history, thus limiting the size of history to be stored and propagated. However, this solution, although easy to implement, would only alleviate the issue. Instead, according to the XMPP philosophy (i.e., to move the complexity away from the client), to completely overcome it, we have to tackle the synchronization problem from the server side. The goodness of an XMPP server is measured by the percentage of extensions supported. Hence, to accomplish a comprehensive solution we should either submit an extension proposal for the existing multi-user chat extension, or rather write on top of it a new extension proposal for a "structured multi-user chat" that handles history synchronization at lower level. Writing a new extension proposal is a neat solution, in line with the XMPP philosophy, although it has a drawback in terms of time required. To be accepted, any new extension proposal has to go through the XEP standards process, which involves discussion on mailing list, formal review, voting by the Jabber Council, and, eventually, the approval as protocol extension. Thus, in the worst case, a new extension proposal submitted can be rejected at the end of the process, otherwise, in the best case, it will take several months and revisions before the draft becomes mature enough for public servers to implement it.

C. Bloated Rich Client Application

Eclipse RCP is a platform for building other platforms. With a little more coding, this excellent framework offers to an application all the benefits seen in Eclipse (e.g., pure-plugin architecture, perspectives, update manager, help system). The only, but negligible, problem we found was the final size of the product itself, which gets bloated because of all the Eclipse RCP libraries to be included, even if not all of its services are utilized. The size of the product for our prototype is almost 9 Megabytes, when the custom plugins developed, plus all the other third-party libraries we used, account for only 980 Kilobytes. This limitation is already known [9] and the Eclipse community is now working to reduce the minimal set of libraries needed.

VII. CONCLUSIONS & FUTURE WORK

In this paper we have described the development of eConference, a text-based electronic meeting system. We have also drawn the lessons learned from the change of the network paradigm for the underlying communication framework used, from JXTA (P2P) to XMPP (client/server). Furthermore, using the insights gained from a pilot study, we have described the redevelopment of our prototype as a pure-plugin system, built on top of the Eclipse RCP.

Recently, we have used our tool at the University of Victoria, Canada, to run a controlled experiment on the comparison between F2F and synchronous text-based interaction, in the context of distributed RE [5].

The next generation of eConference will be built on the basis of the Eclipse Communication Framework (ECF), a set of plugins for developing Eclipse-based applications that require to abstract from the underlying communication services [8]. We also aim to implement new features and plugins, such as freehand drawing, web-browsing and presentation sharing, so as to increase the support to distributed collaboration. This upcoming project has received the IBM Eclipse Innovation Award in the 2006 competition.

ACKNOWLEDGMENT

This work has been partially supported by the MiUR-Italy under grant 2006 "METAMORPHOS."

REFERENCES

[1] G. Antoniu, P. Hatcher, M. Jan, and D.A. Noblet, "Performance Evaluation of JXTA Communication Layers", 5th Int'l Workshop on Global and Peer-to-Peer Computing (GP2PC '05), Cardiff, UK, May 2005.

[2] V. Basili, R. Tesoriero, P. Costa, M. Lindvall, I. Rus, F. Shull, and M. Zelkowitz, Building an Experience Base for Software Engineering: A Report on the First CeBASE eWorkshop, in Product Focused Software Process Improvement, PROFES 2001, LNCS, vol. 2188, Springer Berlin/Heidelberg, 2001.

[3] D. Birsan, "On Plug-ins and Extensible Architectures", *Queue,* ACM, vol. 3, n. 2, March 2005, pp. 40-46.

[4] F. Calefato, F., Lanubile, and T. Mallardo, "Peer-to-Peer Remote Conferencing", 3rd Int'l Workshop on Global Software Development (GSD '04), Edinburgh, Scotland, UK, IEE Publishing, May 2004, pp. 34-38.

[5] F. Calefato, D. Damian, and F. Lanubile, "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.

[6] C.R.B. de Souza, D. Redmiles, D. Millen, and J. Patterson, "Sometimes you need to see through walls: a field study of application programming interfaces", Int'l Conf. on Computer Supported Cooperative Work (CSCW '04), Chicago, Illinois, USA, November 6-10, 2004, pp. 63-71.

[7] eConference Project Wiki, http://cdg.di.uniba.it/index.php?n=Research.EConference

[8] Eclipse Communication Framework (ECF), http://www.eclipse.org/ecf

[9] Eclipse RCP size bug, https://bugs.eclipse.org/bugs/show_bug.cgi?id=53338

[10] D. Garlan, R. Allen, and J. Ockerbloom, "Architectural mismatch: why reuse is so hard", *Software*, IEEE, vol. 12, n. 6, Nov. 1995, pp. 17-26.

[11] E. Halepovic, and R. Deters, "The Cost of Using JXTA", 3rd Int'l Conf. on Peer-to-Peer Computing (P2P '03). Linköping, Sweden: IEEE Computer Society, Sept. 2003, pp. 160-167.

[12] J.D. Herbsleb, A. Mockus, T.A. Finholt, and R.E. Grinter, "Distance, Dependencies, and Delay in a Global Collaboration", Int'l Conf. on Computer-Supported Cooperative Work (CSCW '00), Philadelphia, PA, USA, Dec. 16-20, 2000, pp.319-328.

[13] JXTA dev portal, http://jxta.dev.java.org

[14] J. McAffer, and J-M. Lemieux, Eclipse Rich Client Platform: Designing, Coding, and Packaging Java™ Applications, Addison Wesley Professional, 2005.

[15] P. St. Andre, "Streaming XML with Jabber/XMPP", *Internet Computing,* IEEE, vol. 9, n. 5, Sept.-Oct. 2005, pp. 82-89.

[16] XMPP Multi-User Chat (MUC) XEP, http://www.jabber.org/xeps/xep-0045.html

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