The Evolution of the eConference Project

F. Calefato, F. Lanubile, M. Scalas

Abstract
In this paper we describe both the evolution of eConference, a text-based conferencing system that has turned into a collaborative platform, and how we used our tool in an empirical study that assessed the use of synchronous text-based communication in distributed requirements workshops, as compared to face-to-face interaction.

1. Introduction

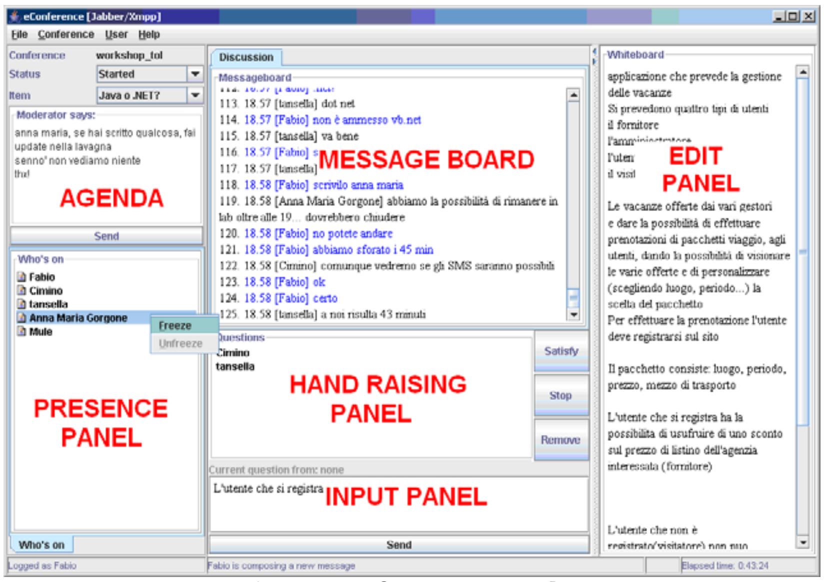
eConference is a text-based distributed meeting system. The primary functionality provided by the tool is a closed group chat, augmented with agenda, meeting minutes editing and typing awareness capabilities. Around this basic functionality, other features have been built to help organizers control the 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 (Fig.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. The edit panel is used to synthesize a summary of the discussion. Finally, the presence panel shows participants currently logged in and the role they play.

Among the participants invited, 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. 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.

During meetings, the interaction of active participants is driven by the use of the hand raise feature. This feature mimics the hand-raise social protocol that people use during real meetings to coordinate discussion and turn-taking. It is a duty of the moderator to manage the queue of the questions asked by participants during presentations and panels. Compared to the real-life social protocol, the hand raise feature of eConference also gives to the moderator the ability to preview questions. 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.

In the following, from Section 2 to Section 5, we first discuss in detail each of the four generations and the motivations for the changes.



Conference workshop_tol
Status Started
Item Java o .NET?

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

Who's on
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

Questions
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 visit

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
registrato(visitatore) non può

Logged as Fabio
Fabio is composing a new message
Elapsed time: 0.43.24

Fig.1 - eConference screenshot

In Section 6 we briefly report on an empirical study conducted with eConference. Finally, in Section 7 we draw the conclusions.

2. P2PConference (ver. 1.0)

The first version of our tool, named P2PConference, was developed using the Java binding of JXTA [1]. 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 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). 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]

3.1. Low level API & End User Complexity

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 (Tab.I). 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. Our initial feeling about the low level and complexity of the API was later confirmed by the creation of the JXTA Abstraction Layer, a community project launched at the end of 2002 with the goal of providing an immutable, high-level API for all the most commonly used JXTA primitives.

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

Tab.I - Impact of JXTA platform changes

Version	Release date	Impact (compared to previous release)
1.0 build 49b	2002/02/08	Low
1.0 build 65e	2002/07/08	None
1.0 final	2002/09/24	None
2.0	2003/03/01	High
2.1	2003/06/09	Low
2.1.1	2003/09/16	None
2.2	2003/12/15	Medium
2.2.1	2004/03/15	Medium

None= No changes to API, bug fixes, other improvements
Low = New APIs
Medium = New APIs, APIs changes (deprecations, methods/classes removed, signature changes)
High = New APIs, APIs and Protocol changes (no backward compatibility)

Until the release of the first useable version of P2PConference, there were two major update releases of JXTA (version 1.0 build 65e, and version 1.0 final in Tab.I), with only API additions or bug fixes that did not break our code. Since then, however, the release of JXTA 2.0 caused a high impact on our code because of large protocol changes. Afterwards, three of the following four releases had significant changes and a considerable impact on developers. Impact assessed in the release-announce emails sent on the JXTA mailing list were occasionally optimistic. Sometimes, as in the case of release 2.1, although the impact on developers was assessed as low, there were some platform incompatibilities that actually obliged us to update the tool. Indeed, as soon as the super peers that build the overlay network were updated to the latest release, we used to experience erratic behaviors (e.g., failure of resource discoveries, high rate of lost messages). Thus, not upgrading to the latest release meant a lack of interoperability, i.e., we could not properly use fundamental services like routing or discovery, and run our system over the Internet, in a truly distributed mode, but only in our subnet, using IP multicast. JXTA was not only complex for developers, but even for end users. The first time a JXTA peer was started and each time network configuration changed, a user had to manually set up the platform through the JXTA configurator, which was overwhelmingly complex because a plethora of settings were provided, not only about the network configuration (e.g., behind a firewall or not), but also about the JXTA network itself (e.g., the peer is an edge, rendezvous or relay). Furthermore, it did not try to make any automatic setup (e.g., use of HTTP tunneling rather than TCP, behind a firewall/NAT). However, since JXTA 2.0, the community felt the need to bypass the manual configuration and make it fully automatic. Until JXTA 2.2.1, automatic configuration was not sophisticated, as it simply tried to skip manual configuration using template configuration files (e.g. HTTP firewalled edge peer, TCP rendezvous peer) and it did not always work well without manual tuning.

3.2. 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 (Tab.II). Since the release of JXTA 1.0, the JXTA core protocol specification defines three kinds of core pipes: unicast, secure, and propagate pipes. Unicast and secure pipes serve for one-to-one communication, connecting two peers in unicast mode. Propagate pipes, instead, operate in one-to-many mode, leveraging either IP multicast on the subnet, or rendezvous peers. All types of core pipes are not reliable by definition and thus, they cannot guarantee ordered message delivery. We also considered non-core pipe services, namely bidirectional pipes and JXTA Sockets, whose purpose is to provide bidirectional communication. Bidirectional pipes were available since JXTA 1.0, but became reliable only since the release of JXTA 2.3, when we had already discontinued the prototype development and maintenance. JXTA sockets, available only since JXTA 2.0, are fundamentally a reimplementation of the standard Java socket API upon the underlying JXTA pipe infrastructure and, thus, reliable by design. We chose to use the propagate pipe service in our prototype because its one-to-many communication mode was the most apt for implementing group communication in our

Tab.II - Alternative JXTA pipe services evaluated

Pipe service	Since	Type	Needs a server for group communication	Reliability ensured
Unicast	v 1.0	1-to-1	Yes	No
Secure	v 1.0	1-to-1	Yes	No
Propagate	v 1.0	1-to-M	No	No
Bidirectional	v 1.0	1-to-1	Yes	Yes (v 2.3+)
JXTA Socket	v 2.0	1-to-1	Yes	Yes

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 have not collected data from formal tests or benchmarks, other research studies have somewhat confirmed the problems of the JXTA messaging architecture in general. Benchmarking JXTA is a hard challenge and test results show a high variance because of the several platform releases, and the very many different network settings and peer configurations to take into account (e.g., using multicast or rendezvous discovery, relay peers or direct connection, TCP, UDP or HTTP). In their analysis of pipe services performance in versions 1.0 build 49b and 1.0 build 65e, Seigneur et al. found that unicast pipes behaved reliably only using TCP in local/LAN test scenario, whereas an extremely high message-drop rate was found when using HTTP [2]. Halepovic & Deters tested performances of core and non-core pipe services for three JXTA releases (1.0, 2.0 and 2.2) in both LAN and WAN [3][4]. Results reported in these studies are positive in terms of scalability both in LAN and WAN, also for propagate pipes, though authors say they perform best on the LAN when UDP multicast is available. However, these tests are performed considering only one sender and an increasing number of receiver peers (1, 2, 4, and 8). Hence, these tests on propagate pipe scalability did not take into account the realistic case of multiple senders and receivers in a large peer group over the Internet, messaging through relays and performing discovery on rendezvous. Finally, Antoniou et al. concluded that throughput of JXTA socket is similar to plain socket throughput on LAN, whereas latency values are higher, due to the verbosity of XML messages [5].

3.3. Lack of a presence awareness mechanisms

Another issue with JXTA was that the platform did not come with any built-in presence awareness mechanism. In collaborative applications, presence awareness plays a key role for coping with the lack of physical proximity and improving distributed work. Thus, we decided to develop from scratch a simple presence-broadcasting feature that propagated a custom presence notification to all known peers. Despite the importance of a presence awareness mechanism for a collaborative tool, we decided not to develop a more sophisticated custom service because the community had already started a project to develop a framework for presence management. Unfortunately, the project did not make any progress until we discontinued the development of P2PConfenrence, and, to date, it has released no files yet. Though not a major issue, we felt the lack of a reliable and sophisticated presence awareness mechanism, as we believe it is a very basic service for a general purpose middleware.

3. eConference (ver. 2.0)

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. Although it aimed at addressing a real problem (i.e., the fragmentation and redundancy of services offered by the plethora of existing P2P 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. In addition, we did not expect the JXTA API to change often and to have backward compatibility issues as well.

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 [6]. 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) 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.

Compared to JXTA, XMPP offered us three clear advantages. First, XMPP provides by design a robust, extensible, secure and scalable architecture 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 [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.

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.

4. eConference RCA (ver. 3.0)

When developing our prototype, we initially focused on basic features for supporting smooth discussion and facilitating meeting creation and execution, so as to maximize the tool effectiveness while minimizing complexity.

The first time we used eConference was to organize and run sixteen distributed requirements workshops, with the main intent of testing the tool itself. The participants were master students in computer science, attending a web engineering course at the University of Bari. As final course assignment they were required to work in groups of three to five people and develop an enterprise application, including both analysis and design documentation. The minutes edited by the scribe were the main outcome of the workshops. They contained a general description of the application to develop, a high-level list of the features to implement, all the decisions taken, and the constraints, both technical and functional, imposed by customers. Afterwards, the minutes were used by the developers to edit a full requirements specification document. We analyzed information from multiple sources to collect experience results, namely direct observations of the meetings, conversational logs, and questionnaires, which were then used to evolve the tool. Direct observation helped us to spot design flaws in the implementation of the hand raising feature, also confirmed by the log analysis, whereas the feedback from the participants allowed us to obtain mainly feature suggestions and enhancements. The most common feature requests were about being able to add/edit/remove agenda items, draw UML diagrams in the edit panel, and send private messages to a single stakeholder or the whole group (i.e., developers or customers). We believed that editing agenda when the meeting is going on would be useful for granting a greater flexibility. Drawing UML diagrams is certainly useful for some technical meetings, but useless for others. This feature was considered a serious candidate for being developed as a plugin. Instead, we were skeptical about the usefulness of enabling private messaging. Though students motivated their request ("*sometimes there were some points we wanted to make, but not in public*"), we were worried that this feature, if implemented, could be abused to the detriment of the discussion itself, especially in the case of private group communication. Thus, we decided for a tradeoff, and accepted only to implement one-to-one private messaging.

The only technical problem that some students reported about in questionnaires was related to the scrolling of the message board panel when a new message was received. Talking informally to students about this annoyance allowed us to spot and deepen another issue that had not been revealed by questionnaires. Students perceived that the item based discussion helped to stay focused on the item currently at hand, but, sometimes, they needed to switch back to another one previously discussed. In such cases, students found awkward to scroll up, looking for the lines about that item. Moreover, the message board automatically scrolled down again as soon as a new message was received. To some extent, this issue should have been mitigated by having always at hand the minutes draft in the edit panel. However, our course was not on requirements engineering techniques and, hence, it is likely that students designated as scribes lacked training, and that the draft did not reported all the information needed. Nevertheless, the feedback on this issue allowed us to understand that, to ease communication flow in eConference, we needed to have separate threads of discussion for each item available in the agenda. Such a feature would avoid having a cluttered message board, with utterances about items interleaved with each other. 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. Furthermore, from the pilot study we collected many useful requests of feature extensions, although specific for the requirements engineering context. 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 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) [8]. 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 functionalities, each of which are provided by a federation of plugins and orchestrated by the engine [9].

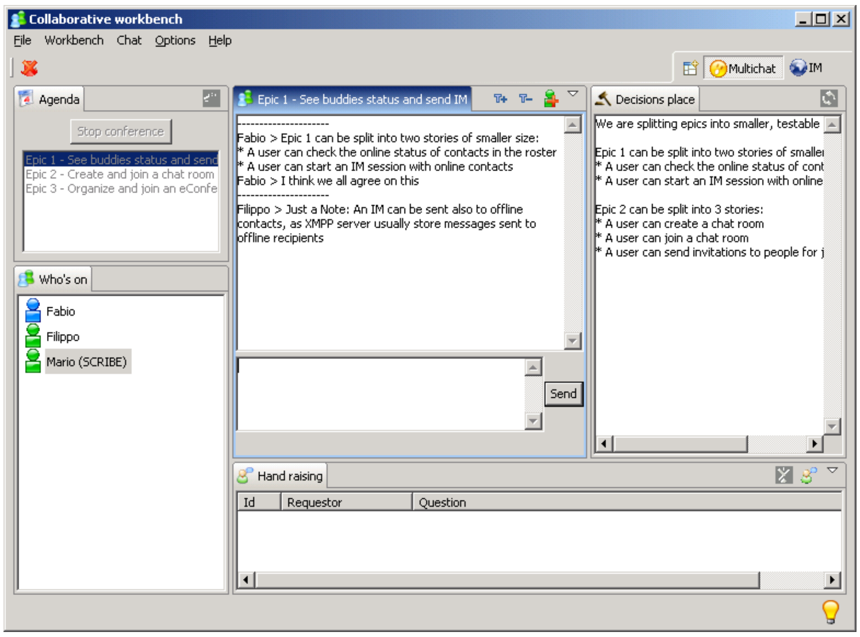
The latest version of eConference is a *rich client application*, built upon Eclipse RCP. Besides all the benefits that come from using native widgets, our tool has inherited all the capabilities of the RCP, in terms of extensibility and classical concepts from the Eclipse world, like views and perspectives. It 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 is the protocol extension proposed for managing chat rooms [10]. 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.2). 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 implement the eConference feature, we took into account the feedback and suggestions gathered from the pilot study. Thus, we made the agenda editable by the moderator, when the meeting is already started, and added support for one-to-one private messaging. Finally, we also implemented the item-based discussion threads, so that all the utterances related to an item are grouped together. As soon as the moderator selects the first item in the agenda, say 'Epic1', the meeting topic is changed accordingly (see the tab name in Fig.3a). When it is time to move to the next item (say 'Epic 2'), the moderator selects it in the agenda and all the utterances about the previous item ('Epic 1') are hidden away from the message board, so as to show only the newly-entered utterances about the item at hand (Fig.3b). Suppose, for instance, that a note has to be added to 'Epic 1'. As soon as the moderator selects it back in the agenda, all the utterances previously exchanged


Collaborative workbench
File Workbench Chat Options Help
Multichat
IM
Agenda
Stop conference
Epic 1 - See buddies status and send
Epic 2 - Create and join a chat room
Epic 3 - Organize and join an eConfe
Who's on
Fabio
Filippo
Mario (SCRIBE)
Epic 1 - See buddies status and send IM
Send
Decisions place
Hand raising
Id
Requestor
Question

```
--------------------
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
```

Fig.2 - eConference perspective

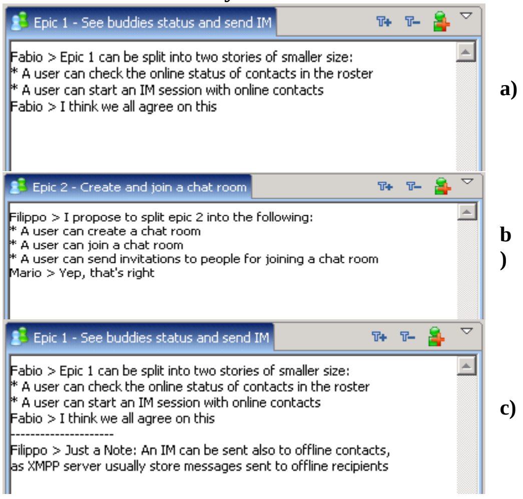
will appear in the message board again. The dashes indicate the new session in the conversation (Fig.3c).

Our experience with Eclipse RCP was positive: With a little extra coding, this 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 encountered was the final size of the product itself, since the final application gets bloated because of all the Eclipse RCP libraries to be included, even if not all of its services are utilized. This limitation is already known and the Eclipse community is now working to reduce the minimal set of libraries needed [11].

4. eConference over ECF (ver. 4.0)

Although designed to be independent from the network protocol and implemented using a pure plug-in 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, expansive to maintain on our own; 2) a burdensome publish/subscribe subsystem, not taking advantage of the Eclipse internals for the dispatching of events in a dynamic plug-in environment; 3) the use of components statically-wired together, which limits the testability of single components and the chance to effectively work with a test-driven approach.

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


a)
b
)
c)
Epic 1 - See buddies status and send IM
Epic 2 - Create and join a chat room
Epic 1 - See buddies status and send IM

```
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 > I propose to split epic 2 into the following:
* A user can create a chat room
* A user can join a chat room
* A user can send invitations to people for joining a chat room
Mario > Yep, that's right
```

```
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
```

Fig.3 - The item-based discussion threads

addition to the application itself, while we wanted to concentrate efforts on the eConference domain components, instead.

The Eclipse Communication Framework (ECF) [12][13] 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, 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 plug-ins 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 (such as, XMPP, Skype, MSN, and Yahoo). Besides, support to new network protocols can be added to ECF at any time, by defining and implementing additional providers.

ECF, however, does not come only with a set of non-GUI interfaces. Instead, it includes several out-of-the box widgets, such as, contacts roster, chat editors, and user account management, which can be embedded in any Eclipse-based application.

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.

With respect to the functionalities, Tab.III summarizes the major features available in eConference 3 and their support available out-of-the-box in ECF. The table shows that ECF can replace most of the features available in eConference 3. Nevertheless, the most specific plugins (i.e., hand raising, 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 plug-ins either available out-of-the-box in ECF, or developed ad hoc upon its abstraction framework (Fig.4).

Tab.III - 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
Chat*	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

5. The empirical investigation

The goal of the empirical investigation described in the remainder of this paper was to evaluate (1) the use of synchronous, text-based communication in distributed requirements workshops, as compared to F2F, and (2) the effects of CMC with respects to the different tasks of distributed requirements elicitation and negotiation.

Requirements engineering is an appropriate domain for this study for a couple of reasons. First, it involves a complex set of communication-intensive tasks. Requirements elicitations and negotiations are among the most challenging and communication-intensive practice in software engineering [14]. Further, requirements elicitation and negotiation are complex tasks that require a constant interplay between idea generation, decision making, and conflict resolution activities, although in different measure (elicitation is more a generative task, whereas negotiation is more oriented to decision making). Secondly, recent research in the field has compared to F2F both audio and video links [15][16], but it has not yet given same attention to synchronous, text-based communication.

5.1. Experimental setting & design

We conducted an empirical study of six academic groups, playing the role of stakeholders involved in requirements engineering activities. The six groups observed (Gr1-6) were attending a Requirements Engineering course held at the University of Victoria in 2006. The study subjects were forty undergraduate students who volunteered to take part in the experimentation, after giving informed consent. Each group was composed of five to eight randomly-selected students (the terms students, stakeholders, and study participants are used interchangeably henceforth).

The goal of each project team was to develop a Requirements Specification (RS) document as a negotiated software contract between the developer team and the client team.

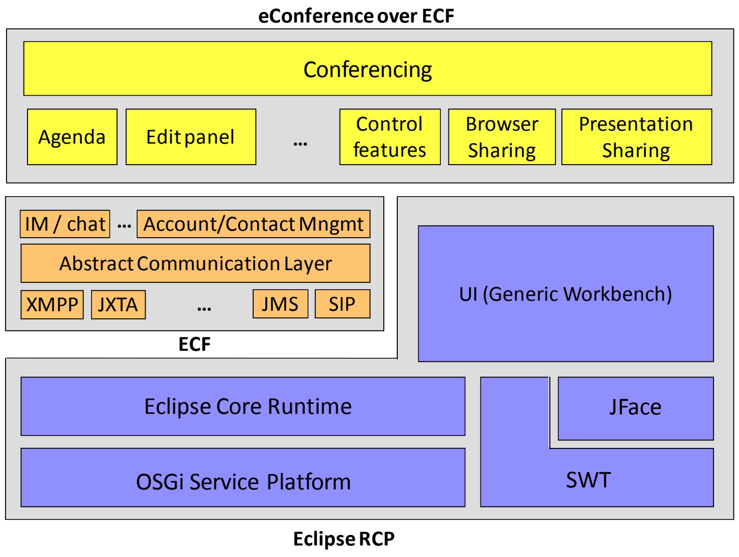


Fig.4 - The fourth generation of eConference is a rich client application composed by ad hoc and ECF-native plug-ins

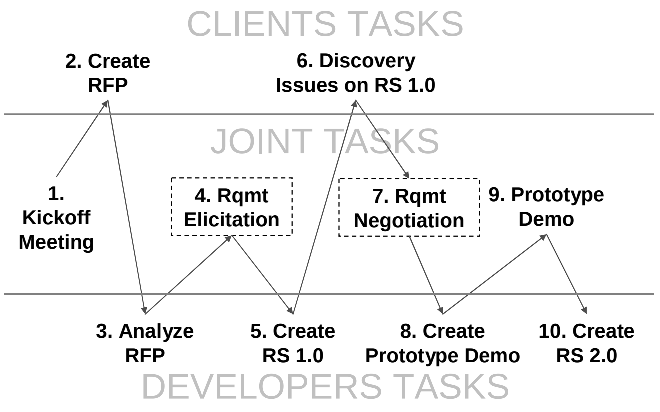


Fig.5 - Workflow for the development process of the RS documents

The project work did not contemplate the writing of any code for the developer groups. Fig.5 illustrates the workflow of the requirements development process, over a period of about ten weeks. It comprises ten phases of continuous requirements discovery and validation, through which the understanding and documentation of requirements was improved. Each of these phases consists of tasks for either one of the client/developer groups, or both groups (project tasks). The developers, together with the clients, created several versions of the Requirements Specification document, while applying techniques of requirements elicitation and negotiation. The deliverables on which students were graded in the course are the RS 1.0 and 2.0, reflecting the shared understanding of the project that the clients and the developers built over the requirements elicitation and negotiation workshops.

The experiment required to compare CMC and F2F communication mode in requirements elicitation and negotiations workshops and, thus, the experimental plan corresponds to a 2^3 factorial design [17]. The three factors, each having two levels, are:

1. communication mode (levels: F2F and CMC);
2. requirements workshop (levels: elicitation and negotiation);
3. role (levels: client and developer).

The requirements workshop sessions were instructed so that all the workshops could be held in parallel and be completed within an hour. F2F workshops (both elicitations and negotiations) were held in parallel, in the same classroom. Also the CMC workshops were all held in parallel, but the students interacted from three different laboratories, so as to simulate geographical dispersion. CMC workshops were run using the eConference tool. To let participants gain familiarity with the tool, a one hour demo was given at class time. In addition, a user manual was made publicly available on the course web site. Furthermore, to reduce the risks of technical problems, a training session was instructed one week before each CMC workshop session, during which the students installed the tool and got acquainted with it.

5.2. Data Analysis

The data sources for the experiment are the post-elicitation and post-negotiation questionnaires, which were administered to the students about one week after each requirements workshop session.

For the sake of space, we briefly report here the results from the analysis applied to data collected from the subjects who got exposure to all the four workshop/medium combinations. Further details can be found in [18]. The box plot in Fig.6 shows F2F negotiation to exhibit the highest, or best, mean rank (3.5) followed by F2F elicitation (2.75). CMC elicitation and CMC negotiation have the lowest average ranks (2.15 and 1.6, respectively).

Given the results of nonparametric test for differences, we can conclude that study subjects perceived F2F negotiations as the best-fitting task/technology match in terms of the extent to

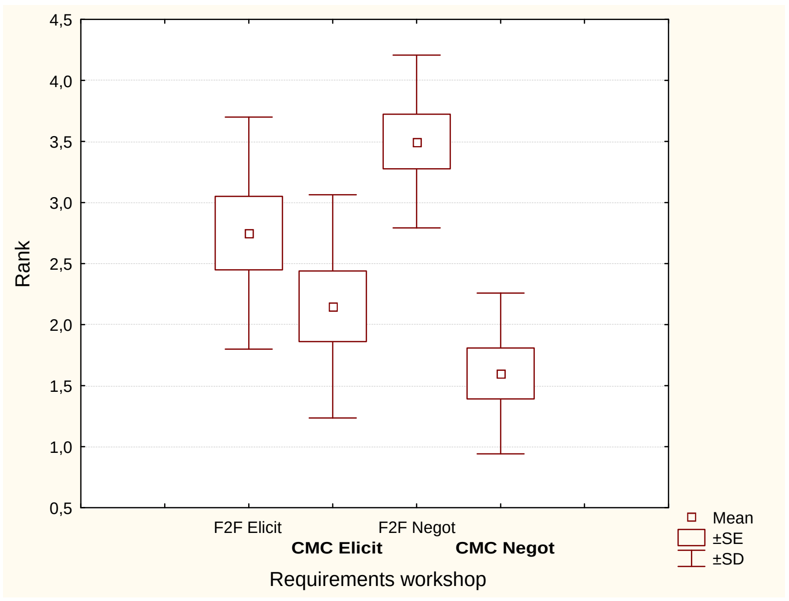


Fig.6 - Ranks based on subjects' evaluation of satisfaction with performance (the higher the rank, the better the workshop/medium fit)

which discussion was consensus-based and the information generated not missed. GSS research has shown that groups interacting on text-based channels have often outperformed collocated groups in task of idea generation because of the possibility to input ideas in parallel. Conversely, collocated groups have usually outperformed distributed groups in executing tasks that involve problem solving, decision making, and conflict resolution [19]. Neither the use of rich media, like video or F2F communication, has been shown to positively affect the performance quality of the work when it involves negotiation [20][21]. Thus, consistently with these findings, we expect that synchronous, text-based elicitation represents a better task/technology fit than synchronous, text-based negotiation. The box plot in Fig.5 shows a large and statistically significant difference between subjects' satisfaction with performance during F2F and CMC negotiations, perceived as the best and worst fit, respectively. In contrast, the difference between F2F and CMC elicitation is not statistically significant. These results, on the one hand, confirm that in terms of satisfaction with performance CMC elicitation is a better task/technology fit than CMC negotiation, and, on the other hand, suggest that the general preference for F2F requirements workshops is due to the strong preference for the F2F negotiation fit over the CMC counterpart.

6. Conclusions

In this paper we have described the development of eConference throughout four major versions, from the initial prototype based on JXTA until the work-in-progress prototype based on the Eclipse Communication Framework for rich-client applications that provides transparent support to the most used communication protocols. This upcoming project has received the IBM Eclipse Innovation Award in the 2006 competition.

We have used our tool at the University of Victoria, Canada, to run a controlled experiment to assess the differences between F2F and text-based, as perceived by stakeholders during both elicitation and negotiation workshops. The findings from the first analyses of the experimental data have confirmed CMC elicitation is a better task/technology fit than CMC negotiation.

References

[1] Project JXTA, https://jxta.dev.java.net/

[2] J-M. Seigneur, *"Jxta Pipes Performance,"* 2002.

[3] 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.

[4] E. Halepovic, and R. Deters, *"JXTA Messaging: Analysis of Feature-Performance Tradeoffs"*, 2005, http://bosna.usask.ca/pub/JXTAMessagingPerf-toReview.pdf

[5] 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.

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

[7] The eConference Project, http://cdg.di.uniba.it/projects/econference

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

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

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

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

[12] Aniszczyk, C. and Safabakhsh, B. "*Getting Started With The Eclipse Communication Framework,*" http://www-128.ibm.com/developerworks/opensource/library/os-ecl-commfwk/, last visit: July 24th 2007

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

[14] Macaulay, L.A. *Requirements Engineering.* Springer-Verlag Telos, 1996.

[15] Lloyd., W.J., Rosson, M.B., and Arthur, J.D. *"Effectiveness of Elicitation Techniques in Distributed Requirements Engineering."* Proc. IEEE Int'l Conf. on Requirements Engineering (RE '02), Essen, Germany, 9-13 September 2002, pp. 311- 318.

[16] Damian, D., and Zowghi, D. *"Requirements Engineering Challenges in Multi-Site Software Development Organizations."* Requirements Engineering Journal, Vol. 8, 2003, pp. 149-160.

[17] Montgomery, D.C. *Design and Analysis of Experiments.* J. Wiley & Sons, New York, 1996.

[18] Calefato, F., Damian, D., and Lanubile, F. "*An Empirical Investigation on Text-Based Communication in Distributed Requirements Engineering", Proc.* of the 2nd International Conference on Global Software Engineering (ICGSE 2007), IEEE Computer Society, pp.3-11.

[19] Murthy, U.S., and Kerr, D.S. *"Task/Technology Fit and The Effectiveness of Group Support Systems: Evidence in The Context of Tasks Requiring Domain Specific Knowledge."* Proc. 33rd Hawaii Int'l Conf. on System Sciences (HICSS-33), 2000, Vol. 2, pp. 1-10.

[20] Finn, K.E., Sellen, A.J., and Wilbur, S.B. *Video-Mediated Communication. Lawrence Erlbaum Assoc.* Inc., Hillsdale, NJ, 1997.

[21] Olson, J.S., Olson, G.M., and Meader, D. *"Face-To-Face Group Work Compared to Remote Group Work with and without Video."* In Finn, K., Sellen, A., and Wilbur, S. (eds.), Video Mediated Communication, Hillsdale, NJ: Lawrence Erlbaum Associates, 1997.