RI06001, January 27, 2006 Computer Science

IBM Research Report

EGRET: A Collaborative Tool for Distributed Requirements Management

Vibha Sinha
IBM Research Division
IBM India Research Lab
Block I, I.I.T. Campus, Hauz Khas
New Delhi - 110016, India

Bikram Sengupta
IBM Research Division
IBM India Research Lab
Block I, I.I.T. Campus, Hauz Khas
New Delhi - 110016, India.

Satish Chandra
IBM T.J.Watson Research Center
Hawthorne, New York, USA

IBM Research Division
Almaden - Austin - Beijing - Delhi - Haifa - T.J. Watson - Tokyo - Zurich

It is now widely believed [1] that to reduce "friction" between stakeholders, collaboration needs to be a central aspect of software lifecycle tools. We feel this is particularly true for the collaboration-intensive phase of requirements management. Keeping today's distributed teams in mind, we have prototyped an Eclipse-based collaborative requirements tool called EGRET. This report provides the context in which EGRET has been developed, presents a detailed tool overview, and also discusses our plans for piloting EGRET. We conclude with a summary of related work, and directions for future research.

1 Introduction

During the last two decades, the software industry has witnessed a paradigm shift, whereby the management, development and maintenance of software have evolved from being concentrated at a single site to being geographically dispersed across the globe. This phenomenon is variously referred to as the "globalization of software" or "distributed / multi-site development". A number of business reasons have contributed to this trend. To start with, the global demand for software products and services beginning in the late 1980s led to a flood of mergers and acquisitions, as IT firms strived to penetrate new markets and complement their product lines. At the same time, companies increasingly chose to focus on core competencies and hand-off or "outsource" some of the other necessary activities to firms specializing in those areas. "Offshoring" brought in further benefits: low labor cost in developing countries, availability of a large pool of skilled labor, and the prospect of being able to do round-the-clock development. Of course, the process has also been aided by significant technological advances; most importantly, the explosive growth of the Internet, which often makes distances irrelevant, and has brought remote collaboration into the realm of possibility. Little wonder then, that a study in 2000 [2] revealed that 70% of US firms have outsourced some kind of business process, and 203 of US Fortune 500 companies engage in offshore outsourcing; or, that according to a Gartner Inc. estimate [3] in 2004, one of every 10 jobs in US tech companies would have moved to emerging markets by the end of the year.

Distributed/ multi-site software development is a natural consequence of these business drivers. However, the perceived benefits notwithstanding, multi-site development is also fraught with innumerable challenges. The critical issue is the inability to communicate effectively across distances, cultures and time-zones. This in turn, gives rise to other problems like lack of trust, lack of information sharing, and ultimately, lack of co-ordination. These and other challenges in distributed development have been well-documented in the literature [4] [5] [6]. Given that global software development appears to be an irreversible trend for now, ample motivations exist for exploring new methodologies and tools that make distributed software development more effective, particularly by facilitating collaboration and coordination between remote team members.

This paper presents a step in that direction. In particular, we describe a tool called EGRET (**E**clipse based **G**lobal **RE**quirements **T**ool) that has been designed to support one of the most collaboration-intensive activities in software development - that of requirements management – in a distributed setting. EGRET seamlessly weaves together a set of ad-hoc and process-driven collaboration services, along with rich awareness features, tailored to the needs of remote stakeholders working off a shared repository of requirements. EGRET may be looked upon as a natural adaptation of the emerging concept of Collaborative Development Environments [1] for the requirements space.

The rest of this paper is structured as follows. In the next section, we explain the challenges associated with multi-site requirements management, and motivate the need for collaborative requirements management tools. Section 3 then presents some sample use cases of such a tool, on the basis of which a high-level design is outlined. The following section presents a general overview of the tool, and then elaborates on the collaboration features. An overview of the plug-ins comprising EGRET and a description of its meta-model may be found in Section 5. Section 6 outlines considerations for empirical validation of the tool, while the rest of the paper discusses related work, and presents our conclusions and future directions.

2 Background

For more than a year, we have been extensively interacting with IBM practitioners engaged in distributed development, to understand their pain points, and propose solutions for the same. Our proximity to development teams in India who are heavily into offshore development, placed us in a favorable position to conduct such a study. We found that a very typical set-up in such a distributed project involves a customer-facing team (comprising managers, business analysts and senior architects) located somewhere in the US or Europe, and multiple development teams (comprising local managers or coordinators, system engineers, designers, programmers and testers) in remote locations like India, China and Brazil. It is the responsibility of the onsite team to closely interact with the customer and elicit high-level business requirements. The analysts then need to work with the system engineers in remote locations to create system requirements that would meet the business needs As the architecture of the system emerges, these may be refined further into component requirements, which would have to be communicated to programmers who implement the different modules that make up the system. Good requirements practices also necessitate that requirements be *testable*: so testers may also be roped in to write test cases or acceptance criteria for the various types of requirements. Moreover, it is common for development work in a project to be outsourced to multiple teams (even organizations) around the world; these teams have to collaborate and draw up interface agreements, which are requirements on how their modules will interact.

During our study, we spoke to around 30 practitioners in different roles, projects and locations; both onsite (US and Netherlands) and remote team members (based in India) were approached. The discussions with the India team members occurred through face-to-face meetings and phone calls. For the onsite teams, teleconferences and follow-up e-mails were used. Through these discussions, we discovered that there was a lot of concern regarding the efficacy of the requirements process in a distributed project. In particular, practitioners noted that two major challenges arise. Firstly, it becomes difficult for distributed teams to hold effective discussions around requirements. Since existing requirements management tools do not provide rich support for collaboration, teams typically use these tools only as a shared requirements repository, and hold all discussions *outside* of the tool in e-mails, chats or phone calls. This involves a significant amount of *context-switch* (as users have to continually move back and forth between the requirements and communication environments) and with requirements often numbering a few thousands, it is difficult to hold detailed discussions this way on individual requirements; the result is that development often proceeds on the basis of misinterpreted or incompletely understood requirements, leading to expensive re-work later on. It is also tedious to track and preserve discussions on requirements that are spread across several media; thus crucial knowledge about the rationale behind a particular requirement, or the reason why it was changed, degrades over time.

A second challenge in a distributed requirements process lies in the management of changes to requirements. We found that although such changes occur frequently, due to difficulties in cross-site communication the information is often not propagated to remote teams in a timely or effective manner; even when propagated it becomes difficult to track subsequent actions that may need to be taken at those sites. As a result, changes may not be consistently implemented, and gaps in understanding may creep in over time. A project manager summed up the overall situation as follows: “The root cause of problems is misinterpreted requirements. These later lead to changes in requirements, which are much more difficult to manage in a multi-site setting.” Of course, requirements also change with changing customer needs, further compounding the problem.

Note that these challenges are not specific to IBM. Other studies (e.g. [16]) have also reported on the difficulties remote stakeholders face in achieving a common understanding of requirements and have noted that the reaction to a requirements related issue is propagated much more quickly locally, through informal communication, than across sites. Thus the evidence is suggestive of a widely experienced problem: remote stakeholders in today's global projects are unable to collaborate effectively over requirements, using existing tools and methodologies.

We feel this motivates the need for a Collaborative Development Environment (CDE) [1] customized to the needs of geographically distributed stakeholders engaged in requirements management; as noted in [1], the purpose of a CDE is "to create a frictionless surface for development by eliminating or automating many of the daily, non-creative activities of the team and by providing mechanisms that encourage creative, healthy and high-bandwidth modes of communication among a project's stakeholders." Critical or regular activities involving requirements (e.g. propagation of change to remote stakeholders, tracking follow-up actions etc.) would benefit from some degree of automation to reduce unrealistic reliance on human conversation and memory. On the other hand, there also need to be mechanisms that support *ad-hoc* collaboration so that stakeholders are able to hold rich discussions around requirements whenever needed. While collaborative environments are being increasingly explored nowadays in support of distributed coding and bug tracking, as well as project planning and enactment, we could not find a comparable solution for distributed requirements management. The work described in the rest of the paper seeks to fill this gap, by exploring mechanisms that facilitate ad-hoc as well as process-driven collaboration between remote stakeholders working together on requirements.

3 Towards a Collaborative Requirements Management Tool

As a first step towards conceptualizing and implementing a collaborative requirements management tool, we documented the desired system functionality in the form of use cases. In continuation of our interactions with distributed development practitioners, these use cases were developed in close collaboration with them, so that we could support the information flows and needs that actually arise in practice. We provide some representative examples below.

3.1 Example Use Cases

1. **Resolving Ambiguous Requirements***:* Stakeholders should be able to resolve ambiguities in requirements and gain shared understanding through rich *contextual* discussions around individual requirements.
    a. A Test Analyst (TA) logs-in and reads a newly created system requirement for which an acceptance criterion needs to be created.
    b. The TA finds some ambiguity in the requirement and needs to discuss this with the System Engineer (SE) who created this requirement.
    c. The TA looks up the online status of the SE. If the SE is currently online, then the TA invites the SE to a chat; otherwise, the TA composes an e-mail explaining why the requirement may be ambiguous. Either way, the TA is able to embed a link to the requirement in the chat/e-mail message, so that the *context* of the discussion is clear and the SE can easily navigate to the actual requirement.
    d. A conversation on the requirement then ensues, through e-mail, chat or a mix of both, and a discussion thread is automatically created. Subsequently, the SE edits the requirement to make it more precise, and links this edit to the associated conversation to record the reason for change.
    e. The discussions and requirement change history are logged for future reference.

2. **Awareness of Ongoing Communication:** Stakeholders may be made aware of ongoing discussions on requirements, so that they may participate as and when necessary, and a high-bandwidth mode of communication is established.
    a. The Project Manager (PM), who was unaware of the above development, logs-in.
    b. A visual icon against the system requirement indicates to the PM that the requirement has been the subject of recent discussions.
    c. The PM opens the discussion log associated with the requirement and reads the conversation between the SE and TA.
    d. The PM has a further comment, which he adds to the discussion thread.

3. **Submitting and Processing Change Requests:** Stakeholders should be able to submit change requests on requirements, act on received requests, and be notified of changes to related requirements.
    a. A Business Analyst (BA) reads a system requirement, and sees that some additional features need to be incorporated in order to meet the customer needs.
    b. The BA submits a change request on the requirement, explaining the additions desired.
    c. The request is automatically routed to the SE in charge of the requirement. Also, to keep track of follow-up actions, a visual decorator is attached to the requirement to indicate that it has a pending request.
    d. The SE edits the requirement to incorporate the change, and links the edit to the received change request. The visual icon is reset, indicating that the request has been acted upon.
    e. Automatic change notifications are sent to owners of requirements that may be traced from the edited requirement, since these requirements may also be impacted.

4. **Processing Change Notifications:** Stakeholders, who are notified of a change in a related requirement, should be easily able to obtain the full context of the change before acting on the notification.
    a. A TA, who owns an acceptance criterion, logs-in and sees a message notifying a change in a related upstream system requirement. A visual decorator on the acceptance criterion indicates the pending notification.
    b. The TA clicks on an embedded link in the message to navigate to the appropriate version of the system requirement
    c. The TA also follows the message thread to look up the original change request, the other notification messages that have been sent, and the results of those that have been acted upon.
    d. The TA now has a broad understanding of the change, and realizes that the acceptance criterion will not be impacted by it, so the TA opens the notification message and selects “No Change Required”.
    e. The visual decorator is reset, since the TA has acted on the change notification.

5. **Running Search Queries:** Stakeholders should be able to quickly locate relevant information by running different kinds of search routines on the project corpus
    a. The SE logs in, and runs a search for all requirements that contain a particular keyword
    b. Search result returns links to all relevant requirements
    c. SE clicks on a link to go to the associated requirement

d. SE decides to refine search to only return requirements that were created during a specified period
e. The appropriate subset of the previous results is now displayed.

6. **Analyzing Impact:** Managers should be able to estimate the impact of proposed changes before committing to them
    a. PM receives a feature request from the customer, which requires change in some existing requirements
    b. Using a Google-like advanced search facility, the PM finds all previous requests that are similar, and requirements that were impacted by those. Based on this knowledge, the PM is able to estimate the "primary impact set".
    c. PM determines "secondary impact set" using traceability graph.
    d. The PM now submits change requests on all requirements that need to change. As above, the requests are all appropriately routed and tracked.

3.2 Design Considerations

From the above use cases, a preliminary design of a collaborative requirements management tool begins to emerge as a judicious mix of :

- *informal collaboration services* (use cases 1 and 2) to facilitate ad-hoc conversation around requirements as and when necessary. Since stakeholders may be in same or different time-zones, both synchronous and asynchronous communication facilities should be provided. Moreover, such collaboration should be *contextual*, so that users can easily navigate from the communication environment to the requirements under discussion (use case 1) or look-up conversations "rooted" to individual requirements (use case 2). This would also help address the practical difficulties (noted in Section 2) of frequent context switch and fragmentation of knowledge across several media, which occur when such discussions are conducted through external e-mail/chat.
- *formal collaboration services*, (use cases 3 and 4) designed to support the critical/regular processes in the requirements phase and reduce the need for human interaction. For example, observing the challenges remote stakeholders face in managing requirement changes, we decided to significantly automate the end-to-end processing of a change – from submission and routing of change request, to notification of changes and tracking follow-up actions
- *awareness features* (use cases 1,2,3 and 4) that facilitate the above collaboration mechanisms; examples are stakeholder online information, awareness of submitted change requests and pending notifications etc. Various visualization techniques have been explored in the literature [31, 32] for providing such awareness. We decided to adopt the technique of using visual decorators on requirements and stakeholder information to convey their status.
- *knowledge management techniques* (use cases 5 and 6) to navigate and make sense of project content. A basic necessity is to allow users to search for information. However, as we explain in Section 4, there are several opportunities for going beyond this and providing advanced analysis capabilities on project data.

The above services will run atop a persistence infrastructure, whose basic entities include people (project stakeholders), artifacts (requirements) and relationships (traceability, ownership). Project specific information e.g. roles, requirement types, project phases, modules etc. may also be defined here. We elaborate on infrastructure considerations below.

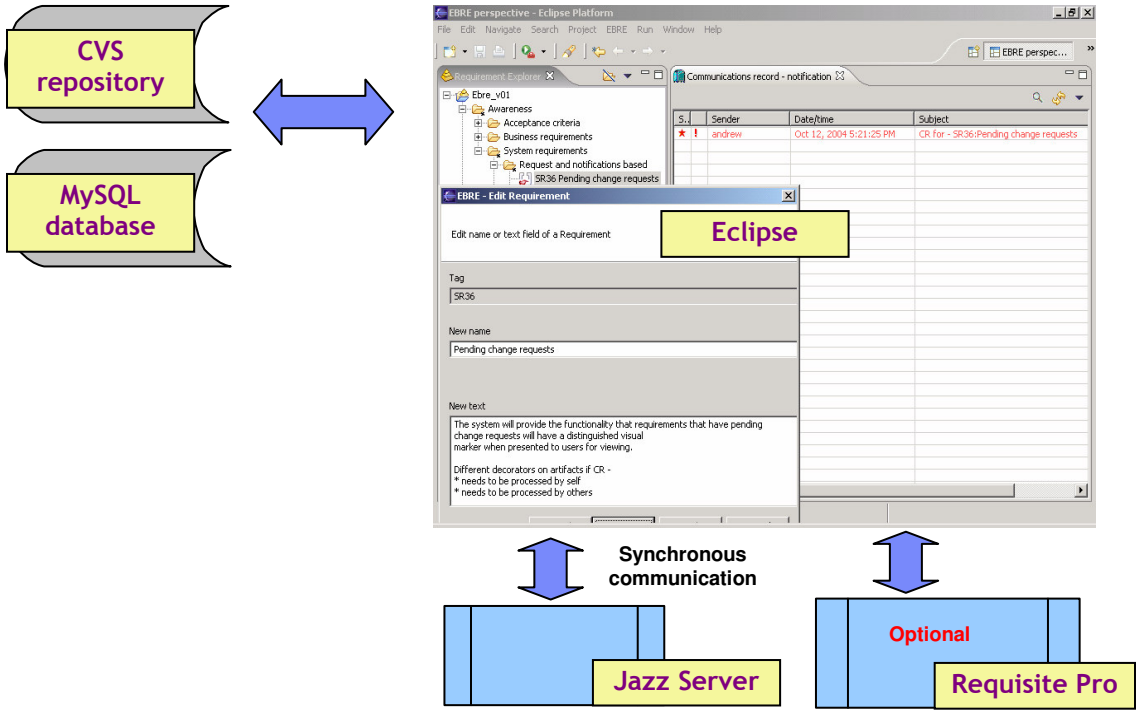

Fig. 1 – Infrastructure Overview

Infrastructure Issues: To start with, we had to decide on the platform on which the tool will run. We narrowed down the choices to making the tool either Web-based or Eclipse-based. The main advantage of the Web, as noted in [1], is that its very nature facilitates "the creation of virtual spaces that transcend the physical boundaries of its participants". Put simply, the Web being ubiquitous, a Web-based tool will be widely accessible and start-up overhead will be minimal. At the same time, Eclipse [36] also presented several advantages. For example, it offers very rich support for user interface design and is easily extensible through "plug-ins". Eclipse is now widely recognized as a universal tool platform and also as an integration point for tools. There are a growing number of successful Eclipse projects supporting various aspects of group collaboration e.g. Jazz [10], Sangam [25], CodeBeamer [19] etc. Moreover Eclipse has gained wide momentum within IBM and is an integral part of its overall tool strategy. All these factors tilted the balance in the favor of Eclipse as the client-side platform of choice for our proposed tool.

Next, we had to decide on the backend infrastructure. We needed a repository for storing requirements, stakeholder information, discussions and also change requests and notifications. We chose MySQL [34], a popular open source database, for this purpose. Requirements often come with associated figures, tables etc. or may be linked to lower-level design elements, so an appropriate repository for these was also needed. Since Eclipse comes with a built-in interface to CVS [35], we decided to use CVS as a common version-controlled repository for all such artifacts that may need to be linked to requirements. For synchronous communication, we use an experimental collaboration server that has been developed by our colleagues in the Jazz team in IBM Research. Jazz [10] provides rich instant messaging capabilities and is also Eclipse-based, making its integration with our tool relatively seamless.

Fig.1 shows how the infrastructure is set up, with an Eclipse front-end providing a set of views of backend data in MySQL and CVS repositories. To reduce network traffic in this client-server

setup, we decided to support "lazy loading" of artifacts: initially only the meta-data of artifacts is retrieved from the repositories, and more details are obtained on user demand. Note that both MySQL and CVS servers may be replicated to enhance performance in a multi-site setting.

Based on the use cases and design considerations described in this section, we have built EGRET, a collaborative tool for distributed requirements management.

4 EGRET: Bringing Collaboration into Distributed Requirements Management

4.1 General Overview

Like any Eclipse-based tool, EGRET (**E**clipse-based **G**lobal **RE**quiremenst **T**ool) consists of a set of views, as shown in the example snapshot in Fig. 2.

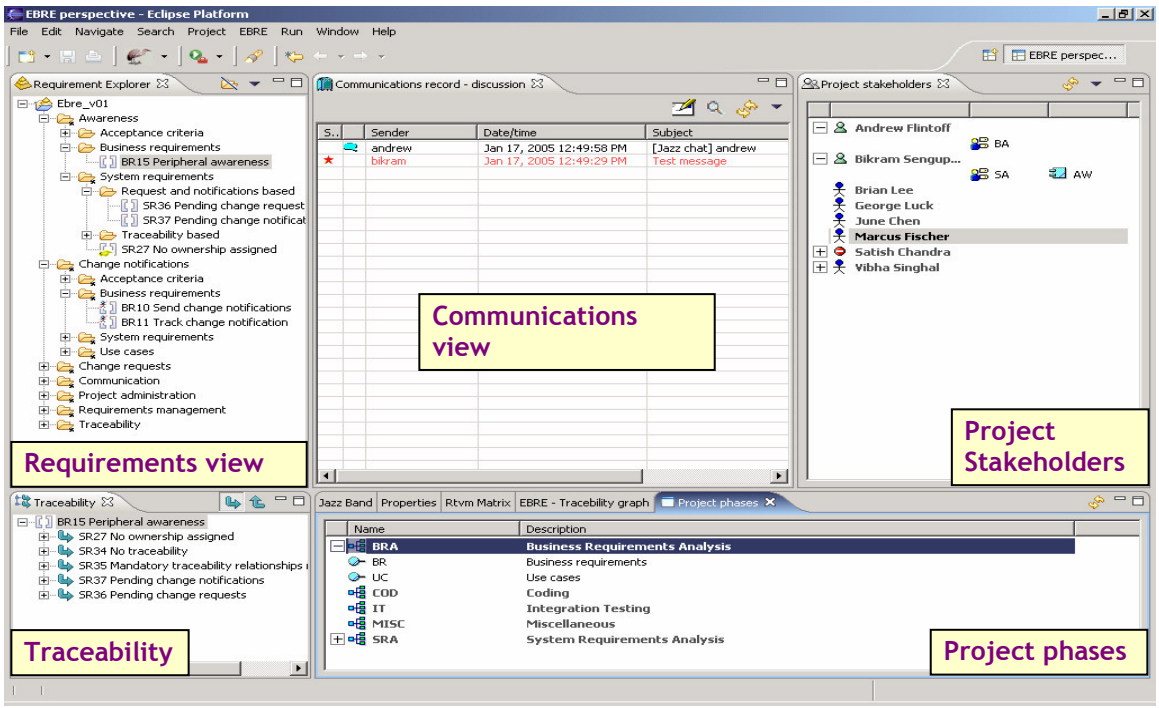

Fig. 2 – EGRET Overview

The main EGRET views are:

- **Requirements Explorer,** which shows the hierarchical structure of requirements that have been created for the project. In this case, the project under consideration is "EBRE_v01" shown as the root folder; there are several modules (e.g. Awareness, Change notifications, Change requests etc.) in this project, and each module contains different types of requirements (e.g. Business requirements, System requirements, Acceptance Criteria) that have been defined for this project. For example, "BR15 Peripheral awareness" is shown as a Business requirement for the Awareness module. Requirements may be both created and edited from within the Explorer; when a requirement is edited, its previous version is preserved and it is possible to compare different versions of a requirement. Users may also select a subset of the requirements into a "working set" of particular interest.

- **Communications Record** view, where the user can access all the synchronous/asynchronous discussions he/she has been a part of, receive change requests and also the automated messages (e.g. change notifications) generated by the system.
- The **Traceability** view shows the traceability relationships that exist for a requirement selected in the Requirements Explorer; thus, "BR15 Peripheral Awareness" can be traced to a set of system requirements with tags SR27, SR34, SR35 etc. The traceability information may also be viewed as a graph or a matrix.
- The **Project stakeholders** view lists all the stakeholders in this project, along with their roles in different modules, and their online status; thus, Bikram is shown as a System Analyst (SA) in the Awareness module (AW) and he is currently online; on the other hand, Satish, though online, should not be disturbed, while Vibha is offline.
- The **Project phases** view shows the various phases that have been defined for the project, and the deliverables e.g. Business Requirements (BR) and Use cases (UC) are the documents that need to be prepared for the Business Requirements Analysis (BRA) phase.

Other relevant views –

- **Eclipse Navigator view (resource view)** – this view shows the different project artifacts that exist in a shared CVS repository. All non text requirements and other documents for design, architecture, code etc can be shared through CVS. It is possible to link a requirement in the Requirements Explorer to related artifacts in CVS as shown in Fig 3.

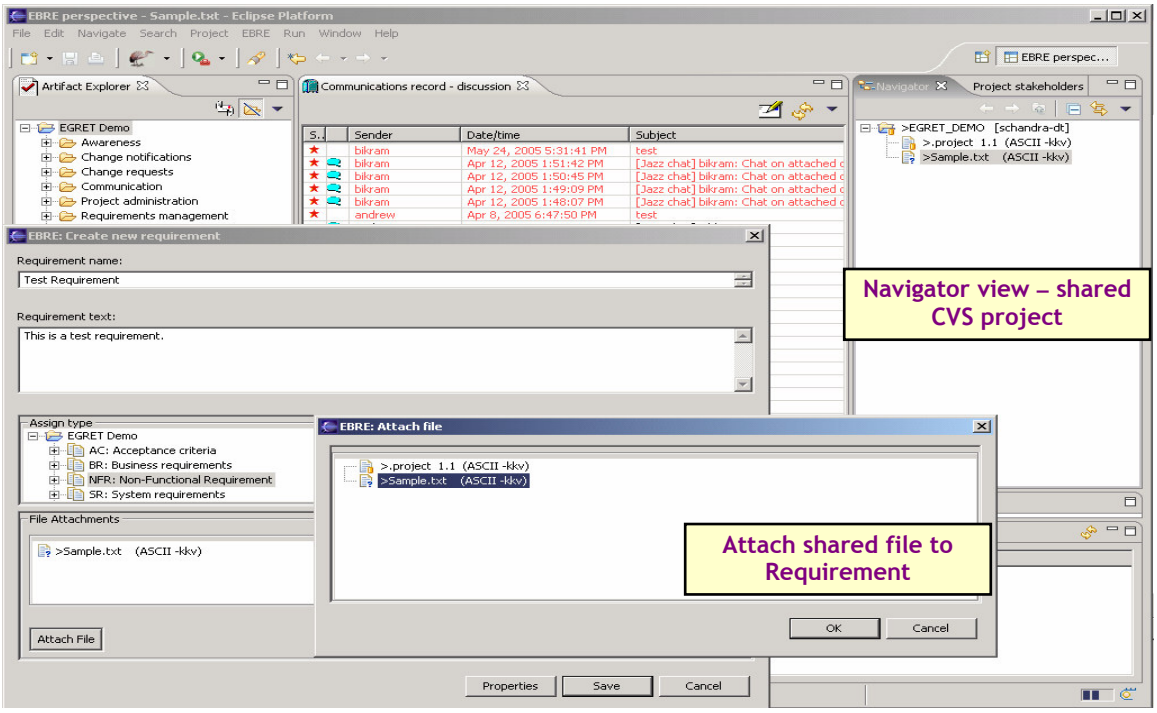

Fig 3 – Linking CVS shared documents to requirements

Setting up an EGRET project: During project initiation, an administrator will create an EGRET project, and define related information like types of requirements in this project, traceability rules for these types (e.g. a business requirement may trace to multiple system requirements) project phases and modules, stakeholder roles, repository locations etc. These may be defined using the tool itself, or imported through a template. User accounts will then be created and users will be assigned different roles. Subsequently, registered users will log in, an initial set of requirements will be created, and collaboration on requirements will begin.

We will now describe how EGRET supports informal as well as formal collaboration around requirements, and promotes awareness about ongoing/pending project activity among distributed stakeholders.

4.2 Informal Collaboration Using EGRET

Synchronous and asynchronous communication: EGRET supports both synchronous and asynchronous conversation around requirements. A stakeholder may compose an e-mail message in the Communications view, and attach links to some requirements in the Requirements Explorer through a simple drag-and-drop mechanism. Alternatively, the user may first select some requirements in the Explorer and choose to send a message related to these requirements, in which case links to the requirements are automatically embedded in the e-mail/chat message to capture the context. When a stakeholder receives such a message and clicks on the link, the appropriate requirement in the Explorer is highlighted. For example, in Fig.4, the "EBRE Read Discussion" window shows that Andrew has sent an e-mail message to Bikram, with a link to requirement "BR15 Peripheral Awareness" attached. Using this link, Bikram may easily navigate to the actual requirement, shown highlighted in the Requirements Explorer. All e-mails are persisted with in the communication repository.

The synchronous communication facility in EGRET has been adapted from the Jazz tool [10]. This facility has been seamlessly integrated with the asynchronous communication mechanism in EGRET; thus it becomes possible to chat on a received e-mail message, and separate discussion threads do not need to be maintained for related e-mails and chat. Continuing with the example in Fig.4, Bikram, on reading Andrew's e-mail, sees that Andrew is shown to be online in the Project Stakeholders view; so, instead of sending an e-mail "Reply" to Andrew, he clicks on the "Chat" button in the "EBRE Read Discussion" box. This opens up a chat window with two links automatically embedded: one pointing to the e-mail message that originated this chat and the other to the requirement attached to the e-mail message, since this requirement is presumably going to be the focus of the discussion. Andrew and Bikram may now engage in a chat, and when they are done, the chat transcript is saved and sent as an e-mail to the participants.

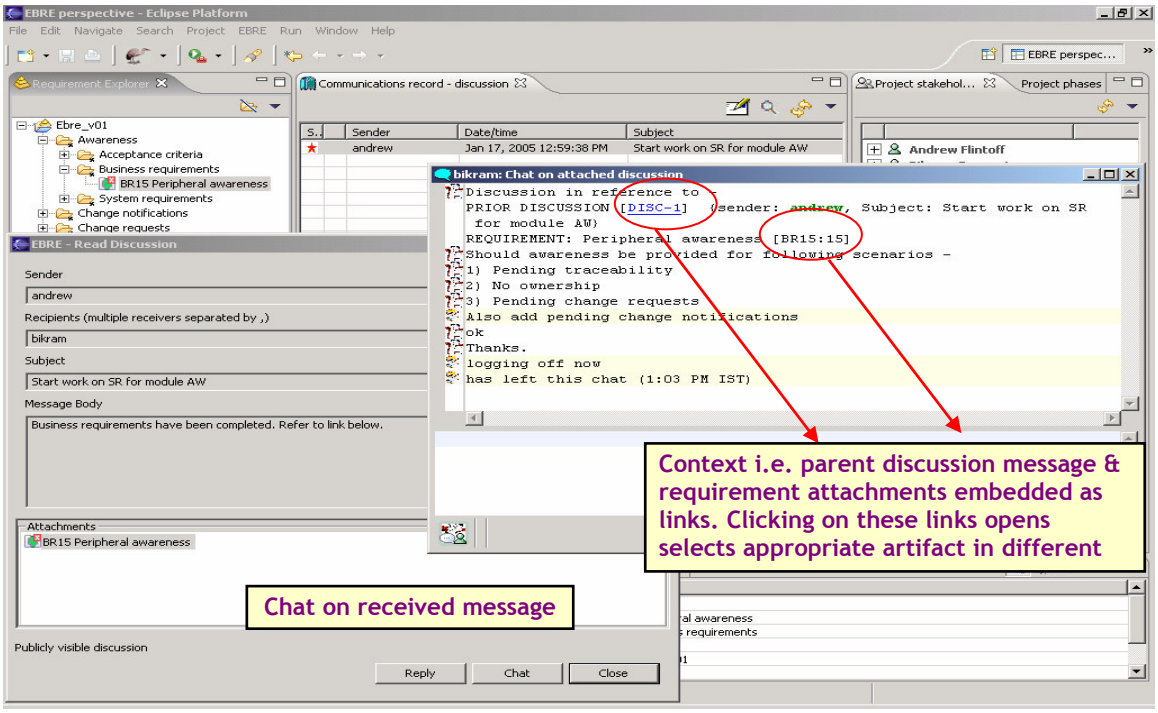

Fig 4: Contextual Communication in EGRET

The informal collaboration features in EGRET offers several benefits. By embedding artifact information within synchronous/asynchronous communication, EGRET supports "in-context" conversation around requirements. This allows easy navigation between the requirements and communication views: requirements embedded in e-mail/chat messages are just a mouse-click away, while one only has to select a requirement in the Requirements Explorer to access all related discussions (synchronous/asynchronous), which are presented in the form of a message thread e.g. Fig. 5 shows the message thread for the requirement "BR15 Peripheral Awareness". This helps address the difficulties associated with conducting such discussions through external media, as explained in Section 2.

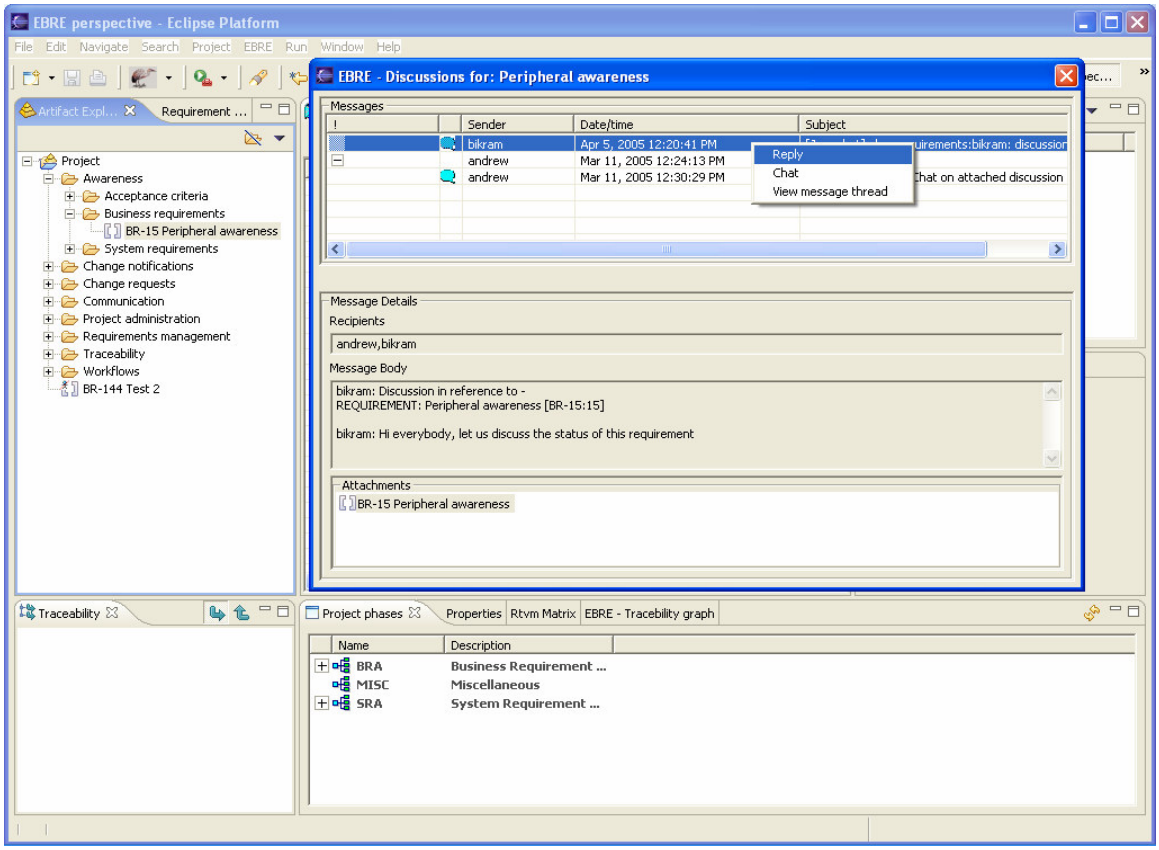

Fig. 5: Communication Log

4.3 Promoting Awareness

One of the challenges in distributed projects is a general lack of awareness about what is going on in other sites. A system engineer in a remote site may be unaware of an ongoing conversation between the business analyst and the project manager, which may have some bearing on the system requirements he/she owns; the project manager may be wondering if the system engineer has taken care of a change request that was submitted. A tester may not be notified in time of a change in a system requirement that traces to an acceptance criterion owned by the tester. It is widely believed that maintaining such awareness, or an "understanding of the activities of others, which provides a context for your own activity" [31], can improve productivity, especially for distributed teams [32].

In EGRET, such awareness information is provided in the UI, through a set of visual decorators that capture the status of requirements in the Explorer. An explanation of these decorators is provided in the "EBRE Artifact Status" box (Fig. 6). There are decorators that signify that a requirement has not been assigned an owner, a project phase or a module. EGRET lets users define what traceability relationships should exist between different requirement types and in case any of these rules have not been satisfied, the requirement is marked out, so that the error may be noted and fixed. In addition, decorators highlight requirements that have a pending change request or notification, as also requirements that have been the focus of recent discussions.

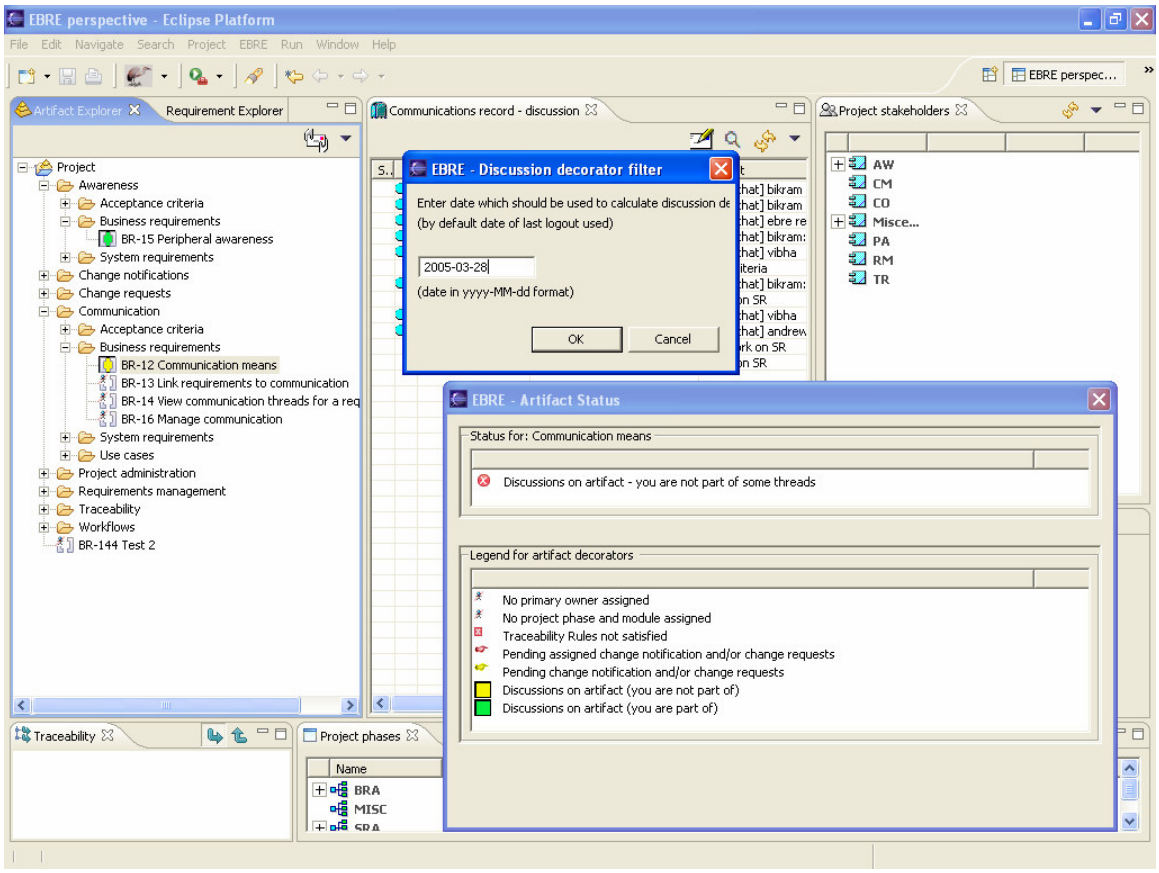

Fig 6 – Peripheral Awareness

The above information is visible to all project stakeholders; however, the exact view depends to some extent on who the individual is. Thus a pending change request/notification on a requirement will appear red in color in the Requirements Explorer view of the stakeholder who has been assigned to process it, while a yellow indicator will imply that although there is a pending request/notification on the requirement, someone else is responsible for it. Again, requirements that have been the topic of recent discussions are differentiated depending on whether the stakeholder has participated in all the recent discussions on the requirement (indicated by a green icon) or whether there have been discussions that the stakeholder has not been a part of (shown as an yellow icon). Thus team members can readily identify tasks that have been assigned to them, and start working on those, or look up unread discussions on requirements they are interested in. Such decorators are hence said to provide "peripheral awareness" to remote stakeholders, as if they were all part of one collocated team with members having shared knowledge of pending tasks and ongoing discussions.

4.4 Managing Changes through Formal Collaboration

In EGRET, the end-to-end processing of a change -- from the routing of change requests to the notification of changes and tracking of follow-up actions -- has been significantly automated. The tool also provides support for recording the context of a change for future reference.

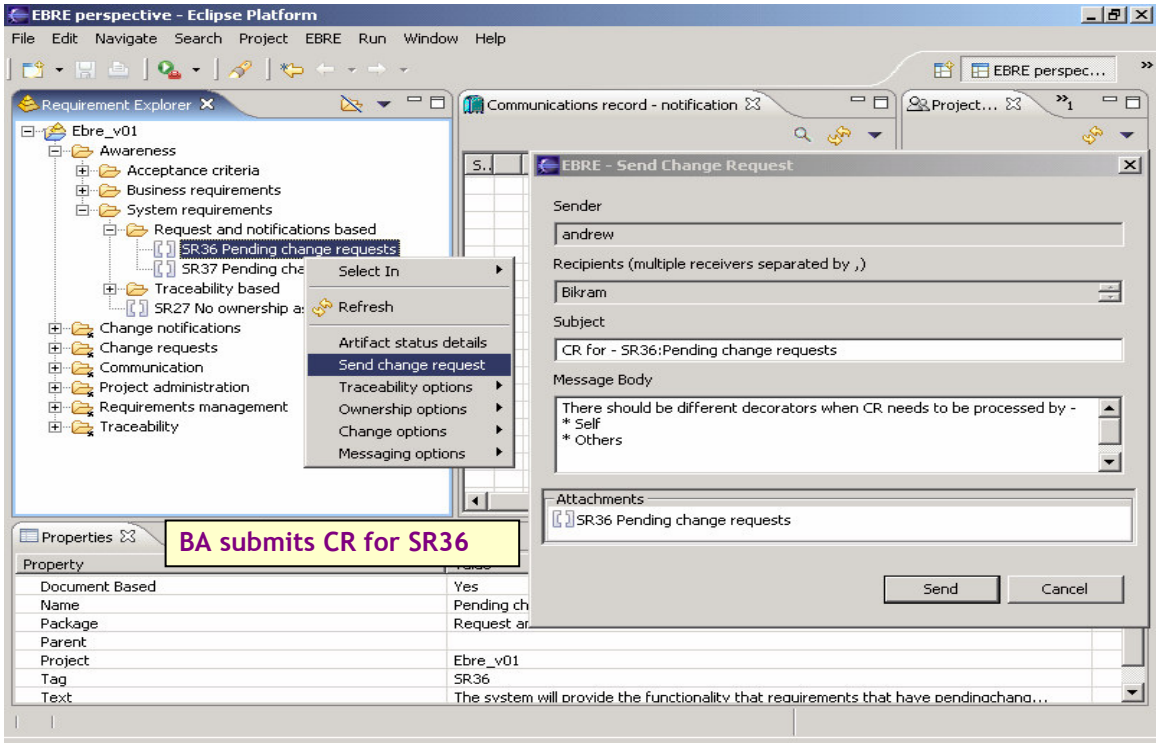

Fig. 7 – Submitting Change Request

Submitting Change Requests: Any user can select a requirement and submit a change request on it. This opens up a form as shown in Fig. 7, which is essentially like an EGRET e-mail message, but with the Recipients field automatically filled up with the name of the primary owner of the requirement (in this case, Bikram), and the Subject field indicating that this is a change request message on the selected requirement (in this case, "SR36 Pending Change Requests"), a link to which is also attached. The submitter of the request may use the "Message Body" section to explain the desired change. On clicking "Send", the request is delivered to the primary owner of the requirement, and the requirement is visually marked out to indicate a pending change request.

(In EGRET, each requirement has one primary owner, but multiple secondary owners. Only the primary owner can edit a requirement, and act on change requests/notifications on that requirement. Secondary owners are other stakeholders who have a reference interest in the requirement. Henceforth, if we simply say owner, we will mean the primary owner of the requirement).

Processing Change Requests: An owner of a requirement can either decide to reject a change request, or edit the requirement to incorporate the desired change. In the latter case, EGRET allows the owner to "link" the edit to a received change request/notification, so that the context of the change is recorded (Fig. 8). Subsequently, EGRET sends automatic notification messages to owners of related requirements, which might be impacted by this change. These related requirements are identified using the traceability graph. For example, in Fig.6, once the system requirement SR36 is edited, an automatic change notification message is sent to the owner of acceptance criterion AC52, since AC52 may be traced from SR36. FYI messages are also sent to the secondary owners of AC52, informing them of the change.

Once a change request has been processed (i.e. it has been rejected, or linked to a requirement edit), it is no longer "pending"; accordingly, the visual decorator is cleared.

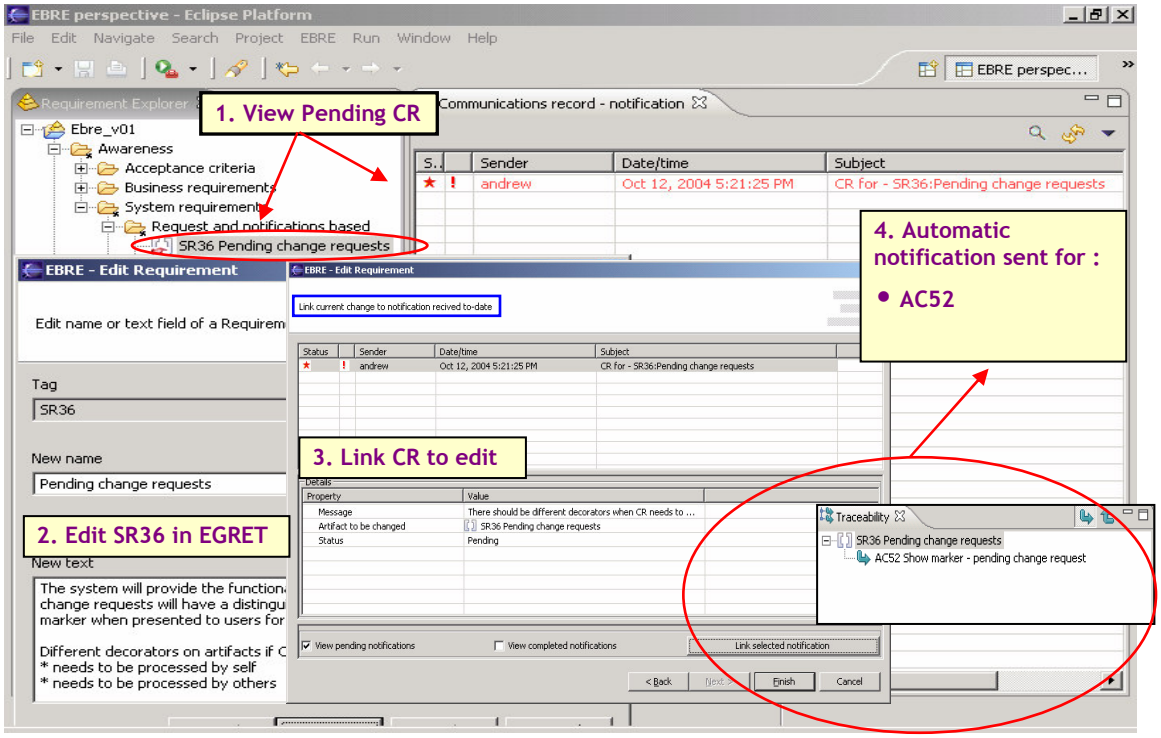

Fig. 8 – Processing Changes I

Acting on Change Notifications: A pending change notification, like a pending change request, is indicated by a red visual decorator, as shown in Fig. 9. The owner of the requirement also gets the notification details in an automated e-mail message generated by EGRET on behalf of the owner of the upstream requirement that changed. This message contains links to both the upstream requirement, as well as the downstream artifact that may need to change. A user may also choose to view the message thread associated with this notification, in which case the complete history of the change is presented (e.g. the original change request that triggered the change in some upstream artifact, and the sequence of changes that were performed subsequently, leading to this message). Based on this information, the owner may process the change notification in the same way as a pending change request: either reject it, or edit the requirement and link the edit to the received notification, in which case further notification messages may be sent out, depending on the traceability graph.

EGRET persists with all change information related to a requirement. Thus, for any requirement, users can view the list of change requests and notifications received till date, and their status. Users may also view the history of edits made to the requirement, which includes information about who edited the requirement, when the edit was made, reason for change, and version number. Moreover, by automating key parts of the change management process, EGRET spares users from the burden of having to send manual notification of changes, and allows project managers to keep abreast of pending actions without having to query project members for status and wait for their response. In other words, the approach reduces the need for ad-hoc communication in the management of requirement changes, and thereby promotes *formal collaboration* between stakeholders.

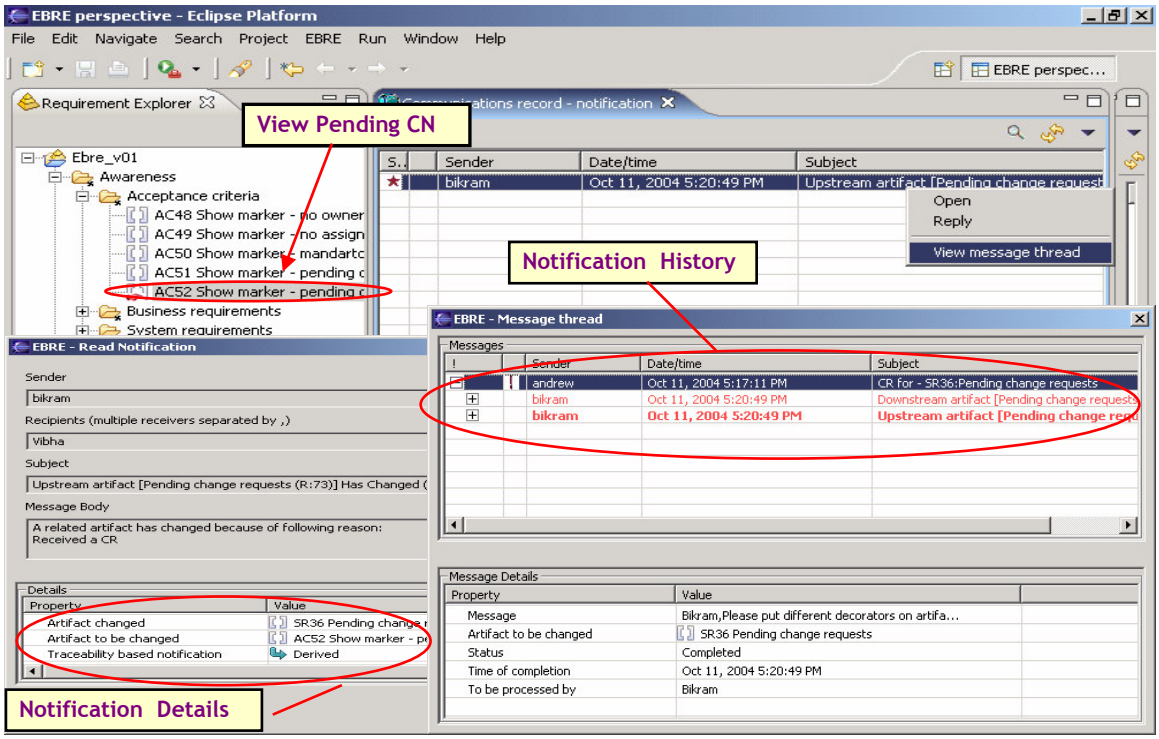

Fig 9 – Processing Change II

4.5 Knowledge Management

The broad goal of knowledge management is to help users navigate the data that builds up in the various EGRET repositories and manage and make sense of the same. Currently, EGRET supports simple search routines that let users query for artifacts (e.g. requirements, e-mail etc.) that contain a specified keyword. An example is shown in Fig.10. The search results are displayed in an "EBRE Search" view; the user can select any artifact returned by the search and view it.

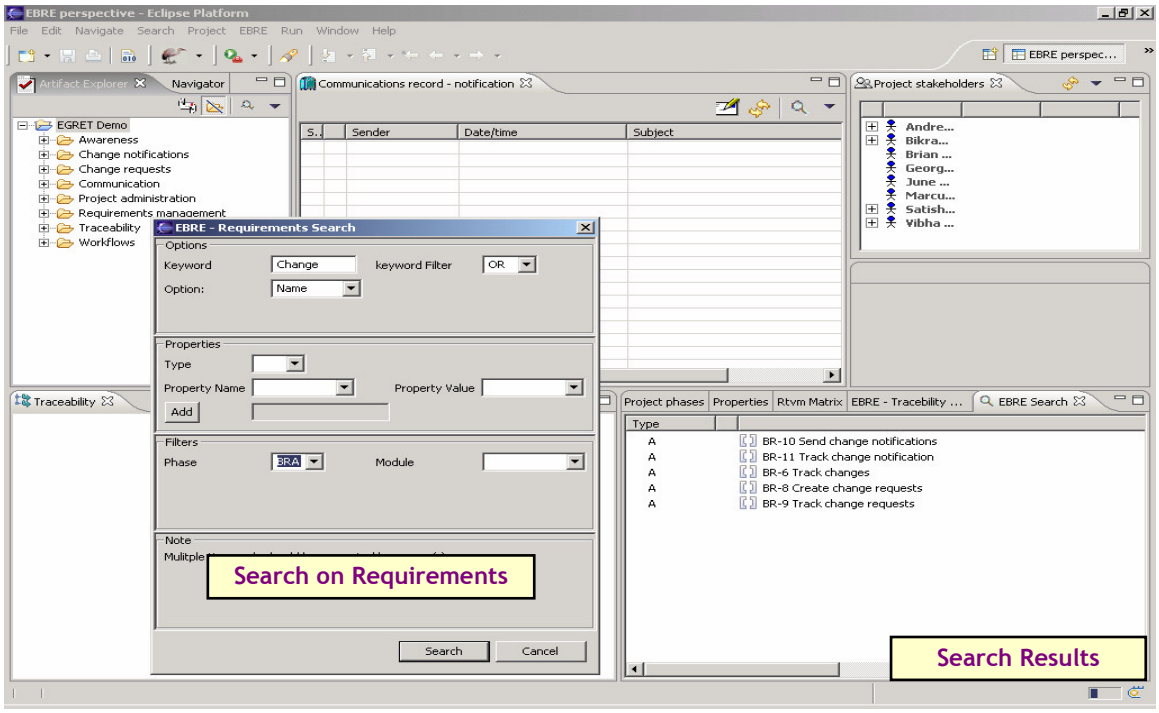

Fig 10 – Search on Requirements

However, there are several opportunities for going beyond this. For example, we plan to plug-in routines that will allow various "meta-data" analytics to be performed on project content. We elaborate on this in Section 5. It is also possible to adapt traditional knowledge management techniques like mining, text similarity, natural language processing etc. to provide advanced services like "Google"-like search, analyzing requirements for ambiguity, discovering hidden traceability links [33] etc. Moreover, all data generated through EGRET usage is currently persisted with, but as the volume of the data increases, this may become unmanageable. Hence we need to put in archiving/retiring mechanisms to let users control the data they need to store and "retire" those that are no longer relevant.

5 Implementation Overview

Eclipse Plug-ins: The functionality in EGRET has been implemented as a set of plug-ins in Eclipse. There are plug-ins for

- Artifact creation and editing
- Project information creation, deletion etc
- Ownership, access control
- Asynchronous communication
- Synchronous communication (from Jazz)
- Change logging and tracking
- Traceability rules, views generation
- Search
- Status checking – peripheral and presence awareness

To support synchronization with RequisitePro, some plug-ins developed by the RSM team in IBM Rational are used.

EGRET Meta-model: Data stored in EGRET can be classified in following categories (Fig 11)

1) Admin Data – this is mainly the project meta-data. Object types here are:

- Project phases – specifies different phases in project. Each phase has an id and name. For example – Business requirement analysis, integration testing
- Modules – different modules in project. Each module has an id and name. For example – backend module, UI module.
- Role – the different people roles in project. For example – System architect, integration tester
- Artifact type – the kind of artifacts that have to be produced in project. For example – business requirements, system requirements. Each type of artifact is produced in exactly one project phase. Each type of artifact can have properties defined (specified as name value pairs).
- Traceability guides – the type of mandatory traceability relationships that should or should not exist between different types of artifacts. For example – a traceability guide would be that every business requirement should derive to one or more system requirements.

2) Stakeholder Data – information on project stakeholders.

- Team member – information per stakeholder
- Responsibility – what role each stakeholder has in what module.

3) Artifact Data – this contains data about the actual work products i.e. artifacts in the project. For now it is mainly requirements in EGRET. Object types here are:

- Artifact – the main attributes here are id, name and text. If special types of data need to be put in the artifacts such as gifs, tables etc the information can be attached as file attachments. Other attributes an artifact object has is – artifact type, module it belongs to, it's primary owner (only one team member can be primary owner), optional secondary owners (multiple team members can be secondary owners – these have a reference only interest)

4) Traceability Data – this contains information on how different artifacts are related to each other.

- Trace Record – this is a traceability data record and attributes are source artifact, target artifact and the traceability type. EGRET currently supports 4 types of traceability – directed, undirected, derived and parent-child relationship.

5) Communication Data – this object contains all communication that happened in the EGRET project. The main type of communication objects here are –

- Discussion objects – these further can be chat or mail. Chat objects are saved transcripts of synchronous communication that happened, while mail objects contain asynchronous communication data. Discussion objects refer to artifacts in whose context the communication happened.
- Change request – team members can submit change requests on artifacts which are automatically routed to primary owner. This object contains data of change requests that are submitted and also contains reference of the artifact on which change is desired
- Change notification – when any artifact changes, primary owners of all related artifacts are sent notifications about the change. This object contains data for the notification i.e. what artifact was changed, what needs to change and a reference to actual change object.
- Information message – when any artifact changes, team members who have shown a reference only interest in it i.e. secondary owners are sent information messages. This object contains reference to object that was changes and also the actual change object.

- Action – in response to every change notification and change request the responsible team member needs to take an action. Action can be classified into – a) no action required, or b) can be an actual change which the user did in response.
- Change – this object contains information on what artifact was changed, when it was changed and also saves the older version of the artifact.

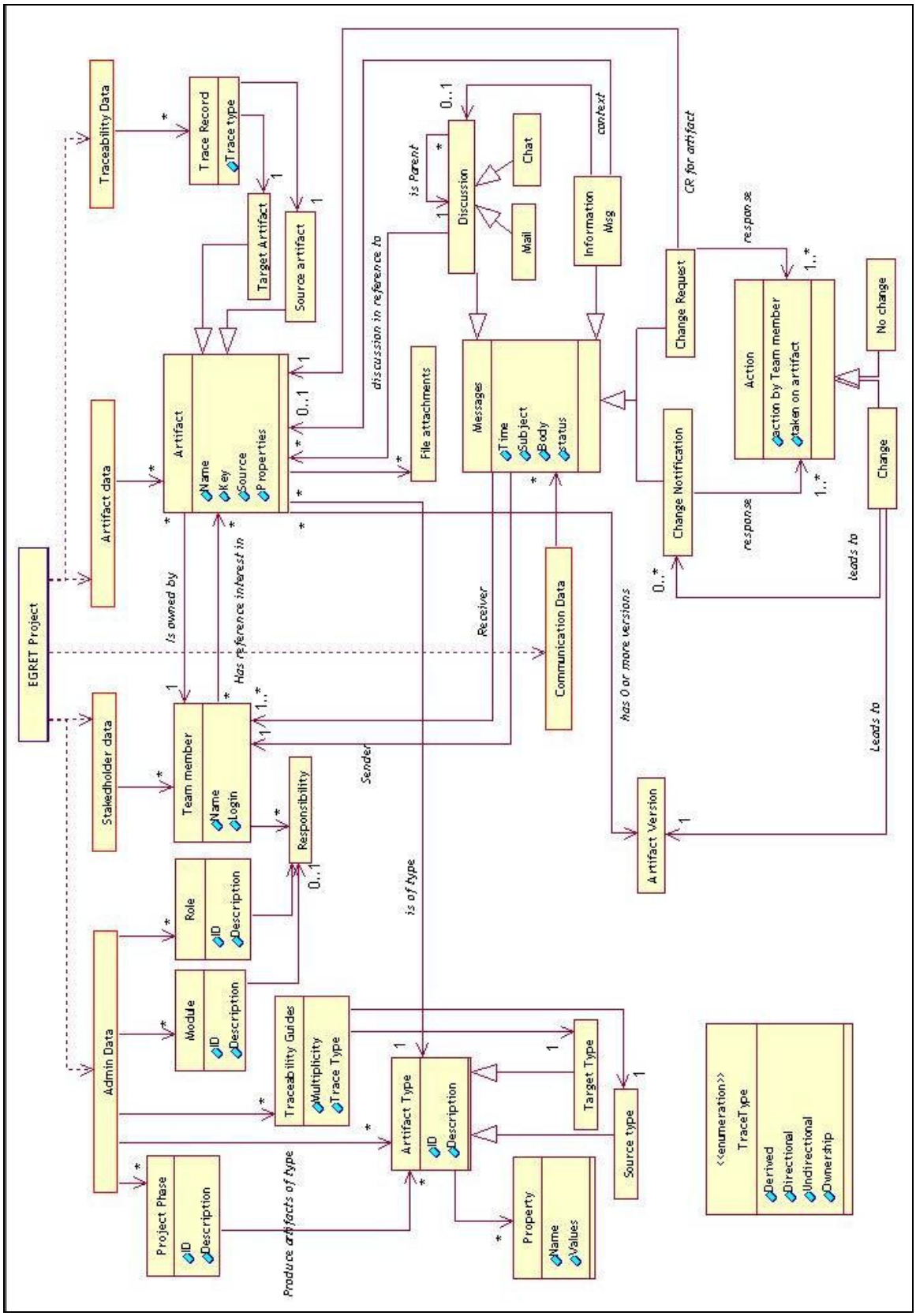

Fig 11 – EGRET Data Model

6 Piloting EGRET

We have developed EGRET based on the hypothesis that incorporating collaboration features into a requirements management tool would help distributed teams communicate and manage requirements across sites. To validate this hypothesis, we would like to pilot EGRET in a distributed project which requires cross-site collaboration over requirements. This will also provide us with an opportunity to gain an empirical understanding of multi-site requirements management e.g. patterns of communication across different stakeholders, reasons for communication, frequency/reason of requirement changes etc. Accordingly, we foresee two kinds of analysis that may be performed.

6.1 Usage Analysis

We would like to study how stakeholders use EGRET in practice. This would help us determine the usefulness of the different EGRET features e.g. peripheral awareness, in-context conversation etc. Accordingly, we have instrumented EGRET extensively to track sequences of user actions e.g. all EGRET menu options that are invoked, the pop-up of EGRET dialog boxes, the selection of an embedded link in an e-mail/chat window, and so on. When these user actions are tracked during a pilot, we would like to see for example, how often a requirement with a visual decorator is selected (which would give an indication of the usefulness of peripheral awareness); or, how often links are used to navigate between the requirements and communication views (i.e. how useful "in-context" conversations is) etc. Our instrumentation makes use of some plug-ins developed by the Jazz team and a few also come with Eclipse.

In addition to studying user behavior, we are also interested in the overall improvement that EGRET may bring to the development process. For example, are requirements better communicated using EGRET? An indication of this may come from how soon requirements are baselined, and also whether testing errors attributed to misunderstood requirements decrease. Also, are requirement changes better managed using EGRET? This may be reflected in how quickly requirement changes are processed. Of course, such a study would depend on the availability of past data from the same or a comparable project, in which EGRET was not used.

6.2 Meta-Data Analysis

In course of a pilot project, a significant volume of data (requirements, discussions, change request and notification etc.) will be collected in EGRET repositories. This would allow us to perform different kinds of analysis on the data, and give useful feedback to project stakeholders. For example, high volume of discussion on a requirement may indicate its criticality; there may be users who are slow in responding to queries (and thus, acting as a "bottleneck"), while there may be others who have to respond to too many queries (communication "overload"); then again, managers may be interested to know about the average-time taken for a requirement change to be "closed", or the frequency of change in requirements, and the main reasons for change (e.g. change in customer needs, misunderstood requirements etc.).

7 Related Work

Distributed software development has been active area of research for the past few years. There have been several studies of the outsourcing and offshoring trends in the software/IT industry and the opportunities they provide [2], [12]. Research has also increasingly reported on the difficulties

that arise in distributed development of software [13], [14], [15]. A comprehensive study of the challenges in multi-site development, best practices that have been developed, and directions for further research has been documented in [6]. In particular, it elaborates on the challenge of distributed requirements management, and motivates the development of a collaborative environment for remote practitioners working together on requirements. Similar observations have been made elsewhere e.g. [16] reports on the difficulties remote stakeholders face in achieving a common understanding of requirements and notes that the reaction to a requirement-related issue is propagated much quicker locally, through informal communication, than across sites. It concludes by proposing the development of an integrated RE tool environment that addresses the challenges of communication and knowledge management.

These observations are synergistic with recent research in the area of computer-supported collaboration. In [1], the authors present a vision for a Collaborative Development Environment (CDE) tailored to the needs of software practitioners, where a CDE is defined as "a virtual space wherein all the stakeholders of a project – even if distributed by time or distance – may negotiate, brainstorm, discuss, share knowledge, and generally labor together to carry out some task, most often to create an executable deliverable and its supporting artifacts." A number of research projects as well as open source efforts and commercial products are now bringing elements of collaboration into various software development activities. We refer to some of the notable ones below.

Commercially, *collab.net* [17] is a leading producer of such CDEs; its public face is *SourceForge* [18], a open-source CDE, which offers facilities for artifact storage, configuration management, bug tracking, task management and discussions. *CodeBeamer* [19], developed by Intland software, is another collaborative development platform with many similar features. Well-known configuration management systems like *ClearCase* [20] and *CVS* [21] now support code awareness by sending e-mail when specified files are changed. *Stellation* [22] [23] is another open source effort (led by IBM Research) that introduces "activity"-oriented fine-grained source control, to simplify collaboration and provide awareness of changes to team members. Another interesting research CDE is *MILOS* [24], also an open source effort that focuses primarily on software process workflow automation. *Sangam* [25] features a shared editor and chat for pair programming. *Jazz* [10] [11] is targeted towards a team of developers working in close proximity; it supports rich synchronous communication, and promotes mutual awareness of each other's coding activities in such a setting. There are also tools which help users identify artifacts and people pertinent to a given task. For example, *Hipikat* [29] recommends relevant software development artifacts (by searching code repositories, newsgroups, bug-reports etc.), based on the context in which a developer requests help. *Expertise Browser* [26] analyzes data in change management systems to locate people with desired expertise. Like EGRET, many of these tools e.g. [19, 22, 10, 25, 29], are Eclipse-based.

EGRET thus belongs to a growing family of collaborative tools for software development, many of which are being increasingly adapted by distributed teams; the novelty in EGRET, however, is its focus on collaborative *requirements management*. While ideas like awareness of activities, in-context conversation, tracking changes, workflows etc. have been explored in some of these other tools as well (which primarily focus on shared coding), EGRET's main contribution is the integration and customization of these features to facilitate collaboration among analysts, system engineers, testers and other stakeholders in the requirements space, particularly when they are geographically distributed.

8 Conclusions and Future Work

In this report, we first motivated the need for a collaborative requirements management tool for use by today's globally distributed teams. We then explored several example use cases of such a tool, on the basis of which a high-level functional specification was proposed. Next, we presented our prototype tool EGRET, a concrete implementation of this envisioned framework. Several actual tool snapshots were used to illustrate the support EGRET provides for both ad-hoc as well as formal collaboration in requirements management, and for promoting awareness among distributed stakeholders. The technical infrastructure underlying the tool was also outlined. We then discussed our plans for piloting EGRET to gain empirical understanding of multi-site requirements management and also to evaluate the practical utility of the tool. Finally we summarized related efforts in the area of distributed and collaborative software development, to put our work in the proper context.

There are several directions along which EGRET may be enhanced. For example, the notification scheme may be made more flexible through a publish-subscribe framework, based on customizable triggers [27]. Support for virtual requirements review sessions will be very useful. Knowledge management is another critical issue; in particular, archiving/retiring policies related to the management of informal communication (all of which is currently persisted), need to be put in place. Advanced analysis capabilities (e.g. based on natural language processing, information retrieval etc.) may be incorporated to provide better search facilities, detect ambiguities in requirements, perform impact analysis, or discover hidden traceability links. It would also be interesting to explore how the emerging notion of "activities" [28] may be supported in EGRET, and what additional benefits it may provide.

9 References

1) G.Booch and A.Brown. Collaborative Development Environments. Advances in Computers Vol. 59, Academic Press, August 2003.
2) E.Carmel, R.Agarwal: Offshore Sourcing of Information Technology Work by America's Largest Firms. Technical Report, Kogod School, American University, Washington D.C., November 2000.
3) Business Week Online, March 1, 2004 http://www.businessweek.com/magazine/content/04_09/b3872001_mz001.htm
4) J.D.Herbsleb, D.Moitra: Global software development : IEEE software March-April 2001
5) J.D.Herbsleb, A.Mockus, T.A.Finholt and R.E.Grinter . Distance, Dependencies and Delay in a Global Collaboration. CSCW 2000.
6) IBM India Research Laboratory: Software Engineering Issues in Global Outsourcing: Challenges, Best Practices and a Research Agenda (white paper)
7) IDC survey (http://www.idc.com) – an end user view of the collaborative software development market
8) B.Sengupta, V.Sinha, S.Chandra et al. Test-Driven Global Software Development. ICSE Workshop on Global Software Development, 2004
9) M.Fowler. Using an Agile Software Process with Offshore Development. http://www.martinfowler.com/articles/agileOffshore.html
10) L.Cheng, C.DeSouza, S.Hupfer, J.Patterson, S.Ross. Building Collaboration into IDEs. ACM Queue vol.1 no.9, 2004
11) L.Cheng, S.Hupfer, S.Ross, J.Patterson. Jazzing up Eclipse with Collaborative Tools. OOPSLA Workshop on Eclipse Technology eXchange, 2003

12) A.Arora, A.Gambardella. The Globalization of the Software Industry: Perspectives and Opportunities for Developed and Developing Countries. (June 2004). NBER Working Paper No. W10538. http://ssrn.com/abstract=556525
13) S.Krishna, S.Sahay and G.Walsham. Managing Cross-Cultural Issues in Global Software Outsourcing. Communications of the ACM. Volume 47, Number 4, April 2004.
14) R.Heeks, S. Krishna, B. Nicholson and S.Sahay. Synching or Sinking: Global Software Outsourcing Relationships. IEEE Software, March-April, 2000.
15) J.D.Herbsleb, A.Mockus, T.A.Finholt, R.E.Grinter. An Empirical Study of Global Software Development: Distance and Speed. ICSE 2001, pages 81-90.
16) Daniela E. Damian, Didar Zowghi: RE challenges in multi-site software development organisations. Requirements Engineering Journal 8(3): 149-160 (2003)
17) www.collab.net,
18) http://sourceforge.net/
19) http://www.intland.com
20) http://www-306.ibm.com/software/awdtools/clearcase/
21) www.gnu.org/software/**cvs**/
22) http://www.eclipse.org/stellation
23) M.Carroll, S.Sprenkle. Coven: Brewing Better Collaboration through Software Configuration Management. ACM SIGSOFT Foundations of Software Engineering, 2000
24) H.Holtz, A.Konnecker, F.Maurer. Task-Specific Knowledge Management in a Process Centered SEE". Workshop o Learning Software Organizations, LSO-2001, Springer 2001.
25) http://sangam.sourceforge.net
26) A.Mockus, J.Herbsleb. Expertise Browser: A Quantitative Approach to Identifying Expertise. ICSE 2002, ages 503-512
27) J.Huang, C.Chang. Event-Based Traceability for Managing Evolutionary Change. IEEE Transactions on Software Engineering vol.29, no.9, 2003
28) Unified Activity Management.
http://domino.research.ibm.com/cambridge/research.nsf/0/a5825415e2b50cf085256f87006a046a?OpenDocument
29) D.Cubranic, G.Murphy. Hipikat: Recommending Pertinent Software Development Artifacts, ICSE 2003, pages 403-418
30) A.Hutchings and S.Knox. Creating Products Customers Demand. Communications of the ACM, 38(5), 1995.
31) P.Dourish, V.Bellotti. Awareness and Coordination in Shared Workspaces. CSCW 1992.
32) P.Dourish, S.Bly. Portholes: Supporting Awareness in a Distributed Work Group. CHI 1992
33) J.Hayes, A.Dekhtyar and J.Osborne: Improving Requirements Tracing via Information Retrieval. International Conference on Requirements Engineering, 2003 (RE'03)
34) http://www.mysql.com/
35) https://www.cvshome.org/
36) www.eclipse.org