A Collaborative Writing Mode for Avoiding Blind Modifications

Claudia Ignat, Gérald Oster, Pascal Molli, Hala Skaf-Molli

A Collaborative Writing Mode for Avoiding Blind Modifications

Claudia-Lavinia Ignat
INRIA Nancy Grand Est
Campus Scientifique, BP 239
F-54506 Vandoeuvre-lès-Nancy, France
ignatcla@loria.fr

Gérald Oster, Pascal Molli and Hala Skaf-Molli
Nancy-Université, LORIA-INRIA Lorraine
Campus Scientifique, BP 239
F-54506 Vandoeuvre-lès-Nancy, France
{oster,molli,skaf}@loria.fr

ABSTRACT
Existing collaborative environments such as Wiki or version control systems allow users to work in isolation, but they do not prevent blind modifications. For instance, users may concurrently perform the same task or they might work on obsolete versions of shared documents. We propose a novel writing mode for avoiding blind modifications by providing real-time information about group activities. Changes performed concurrently are filtered according to user privacy preferences and depicted in their local documents.

Author Keywords
CSCW, Collaborative writing, Blind modifications, Awareness, Synchronous interaction, Asynchronous interaction

ACM Classification Keywords
C.2.4 Computer-Communication Networks: Distributed Systems—*Distributed applications*; D.2.2 Software Engineering: Design Tools and Techniques—*User interfaces*; H.5.3 Information Interfaces and Presentation: Group and Organization Interfaces—*Asynchronous interaction, Synchronous interaction, Computer-supported cooperative work*

INTRODUCTION
Collaborative writing is becoming increasingly common, often compulsory in academic and corporate work. The collaboration can be on real-time in the case that members work closely together and give immediate feedback to changes of other users or asynchronously when users work separately without quickly reacting to group contributions.

In asynchronous collaboration users perform changes in isolation and publish these changes at a later time. Working in isolation may generate blind modifications. Users perform blind modifications when they modify a document without being aware of concurrent changes. Blind modifications can lead to useless or redundant work. For example, useless work can occur if a user updates a section of a document while another user concurrently deletes this section. Two users perform redundant work if they concurrently perform an identical task.

In this paper, we propose a new interaction work mode for avoiding blind modifications while allowing users to work asynchronously in isolation. This mode provides information in real-time about group members activities. Changes are extracted from activities performed by users working in isolation. Then, details of changes could be filtered according to user preferences in order to preserve their privacy. The information is sent within the group for computing awareness information that is then displayed to users. Depending on details of filtered changes, users will be more or less aware of concurrent changes, and in this way, prevented from performing blind modifications in the document. For example, if a user chooses a low privacy level, all details of his modifications are sent to other users. Therefore, these concurrent modifications can be precisely localised and displayed to other users. Consequently, these users are aware of others' modifications and can avoid working in concurrently modified areas of the document. On the contrary, if a user chooses a high privacy level for his modifications, the only provided information is the name of the document currently modified by that user. In this paper we present the main concepts of our approach and use some mockups to give an idea of the visualisation interface of a system based on our approach.

Our paper is organized as follows. We start by motivating our approach and illustrating problems regarding blind modifications while working in isolation. We then describe our approach that addresses the issues of blind modifications during collaborative work by introducing the concept of *ghost operations*. We continue by revisiting our motivating example to show how blind modifications could be prevented. Next we compare our approach with related approaches. We end the paper with some concluding remarks and directions of future work.

MOTIVATING EXAMPLE
In this section we are going to present collaborative scenarios illustrating some of the problems regarding blind modifications while working in isolation.

Consider a scenario involving three software engineers collaborating on the source code of the same project as summarised in the table 1. Although at the beginning they divide their work according to predefined tasks, their modifications will overlap later on during their isolated work since their tasks involve some common classes.

In step 1 of the scenario, the first developer decides to remove the method isReal() from the class Integer illustrated

Step	Actions of developer 1	Actions of developer 2	Actions of developer 3
1	op_1: removes method isReal() from class Integer	op_2: updates method isReal() from class Integer	op_3: creates test class IntegerTest to check methods of class Integer
2	COMMIT		
3		UPDATE *(a conflict is detected between operations op_1 and op_2)*	UPDATE *(the test class IntegerTest does not compile)*
4		solves conflict between op_1 and op_2 by re-inserting new method isReal()	removes test for method isReal() as this method was removed
5		COMMIT	
6			UPDATE & COMMIT *(no test for the method isReal())*
7	UPDATE	UPDATE	

Table 1. Summary of scenario

in figure 1. Concurrently, the second user updates the method isReal() from class Integer such that it returns false instead of real. The third user tests the class Integer by creating the test class IntegerTest. One of the added methods in that class is the test method for isReal(). Steps 2, 3, 4, 5, 6 and 7 describe the sequence of actions performed by each user.

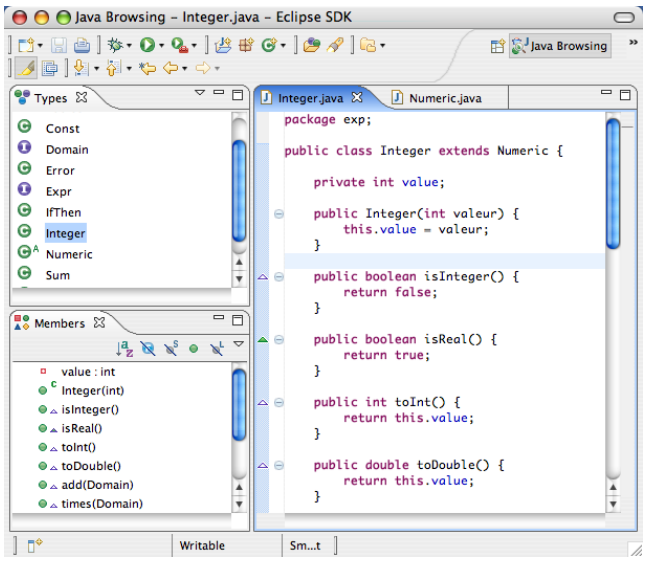

Figure 1. Initial document state

Due to blind modifications performed by users while working in isolation, the following undesired situations of useless and redundant work occurred:

- User 1 deleted the method isReal() which was finally re-inserted by user 2. His work was useless and produced side-effects for the tasks of other users.
- User 2 modified the method isReal() but due to its removal by user 1 he needed to re-perform his initial change.
- User 3 wrote the test for method isReal() and was obliged to remove it. Therefore, he performed "useless" work and finally he did not realise that his task is incompleted.

Therefore, we see the need of a notification mechanism that informs in real-time users about concurrent activities performed in isolation.

PREVENTING BLIND MODIFICATIONS BY MEANS OF GHOST OPERATIONS

In this section, we present our approach for preventing blind modifications in asynchronous collaborative writing that might occur while working in isolation.

We consider an asynchronous collaborative editing system as being composed of n user sites and a server acting as a central repository. Each user associated with a site works on his copy of shared documents. A user can perform the following actions:

- modifies a document by generating operations that are immediately applied on his local copy,
- makes available his local changes to other users by committing his local operations to the repository. The user is not allowed to commit his local operations if his local copy is not up-to-date, i.e. if some non-integrated remote operations are available on the repository.
- updates his local copy of a document by integrating remote operations from the repository.

Operations can model any changes targeting a document such as insertion, deletion and update of lines in that document or concerning a document file system such as moving a file from a directory to another one.

An operation is composed by a type and a list of parameters *operation=<type,(parameter)*>*. The type of an operation can be *INSERT*, *DELETE*, *UPDATE*, *MOVE*, etc. Each parameter is a tuple composed by the name of the parameter, its type and its value.

For instance, an operation of insertion of line 4 with the content *"Preventing blind modifications"* inside the document file.txt generated by $User_2$ will have the form: op_2=*<insert, [(initiator,integer,2), (file,string,"file.txt"),(line,integer,4), (content, string,"Preventing blind modifications")]>*.

For the sake of simplicity, throughout this paper we denote operations by specifying their type, target and content ignoring other parameters. For instance, the operation *op=insert(file.txt,4,"Preventing blind modifications")* denotes the insertion of line 4 into the file called file.txt, the content of the inserted text being *"Preventing blind modifications"*.

Under the assumption that a user is continuously connected, it is possible that she receives in real-time non-committed parallel modifications in order to annotate his local copy of the document. This presumes that users agree to send in real-time their local non-committed operations. Unfortunately, this assumption might violate user privacy as users may not agree to send draft changes of their work.

There is a trade-off between privacy and the usefulness of awareness: if users agree to have less privacy, other group members are provided with rich awareness. We provide a filtering mechanism that deals with this trade-off. We filter non-committed local operations before sending them to other users by masking some operation parameters. We call these operations *ghost operations*.

A *ghost operation* is the result operation obtained by filtering an original operation according to user privacy preferences $g(operation) =< filter(type), (filter(parameter))* >$. In the rest of the paper, in order to distinguish between original form and ghost form of an operation, we will refer to them as *real operation* and *ghost operation* respectively.

The type of a real operation might be masked in the corresponding ghost operation. The list of parameters of a real operation might be masked in the corresponding ghost operation. The ghost operation might not contain a parameter belonging to the real operation, it might contain it in the original form or it might filter it. A filtered parameter is formed by the filtered name, the filtered type and the filtered value of the original parameter.

For instance, for the original operation of insertion of line 4 with the content *"Preventing blind modifications"* inside the document file.txt generated by *User*$_2$ *op*$_2$=<*insert, [(initiator,integer,2),(file,string,"file.txt"),(line,integer,4),(content, string,"Preventing blind modifications")]*>, the following ghost operations might be generated:

- *g(op*$_2$*)=<insert, [(initiator,integer,2), (file,string,"file.txt"),(line,integer,4),(contentSize, integer, 30)]*>. In this case the ghost operation masks the content of the inserted line, by specifying the file name and the line where the insertion takes place as well as the size of the inserted content.

- *g(op*$_2$*)=<edit, [(file,string,"file.txt"),(line,integer,4)]*>. In this case the ghost operation masks the identity of the user that generated the operation, the type of the operation and the content of the inserted line. It just indicates that a modification has been performed by a certain user in the document file.txt at line 4.

In the rest of the paper we are going to use a simplified form for representing ghost operations in the same way we represent real operations.

Ghost operations are not designed to be integrated in the local copy of the document, but rather "to annotate" the document. Depending on the carried data, various annotation forms can be computed. If the carried data are sufficient for locating the concurrent changes, annotations can form an overlay model that is presented to users over their document view. Even if all the parameters of the real operation were filtered in the ghost operations, it is possible to count the number of concurrent operations performed by group members. On the contrary, if the form of ghost operation equals the real operation, i.e. no information is filtered, future committed changes and their consequences can be predicted and previewed in real-time.

REVISITING MOTIVATING EXAMPLE

In this section we revisit the motivating example by showing how blind modifications can be prevented by applying our approach.

Consider that the three developers have performed their actions described in step 1 of table 1. The first user removes method isReal() by generating the operation *op*$_1$=*delete(User*$_1$*,Integer.java,15-18)*. This operation removes the lines 15 to 18 describing the definition of the method isReal() from the file Integer.java. The second user modifies method isReal() by updating the content of the line 16 with the content return false;. Therefore, operation *op*$_2$=*update(User*$_2$*, Integer.java,16,"return false;")* is generated. The third user creates the file IntegerTest.java and inserts the test methods for the class Integer. For the sake of simplicity, we do not define the form of the generated operations. In the what follows, these operations will be referred as operation op_3.

Further, suppose that users decide to send ghost operations describing their activity while working in isolation. The first user decides to apply the privacy policy allowing to send the full content of his modifications as ghost operations. Therefore, he sends the following ghost operation: $g(op_1)$=*delete(User*$_1$*,Integer.java,15-18)*. In order to make other users aware about his modification, the second user decides to apply the privacy policy that hides the content of his changes but shares their location. Therefore, the form of the generated ghost operation is $g(op_2)$= *update(User*$_2$*,Integer. java,16)* signifying that line 16 is under modification. The third user performing testing decides not to send any ghost operations regarding the changes he performs by applying the strongest privacy policy. The real and their corresponding ghost operations are summarised in table 2.

In the following we show how ghost operations can make users aware about concurrent changes and avoid undesired situations. Let us analyse what happens at the site of the first user who deletes the method isReal(). After the reception of the ghost operation sent by the second user $g(op_2)$, awareness information concerning activity of the second user can be presented as depicted in the figure 2. Since the ghost operation $g(op_2)$ contains information about the target file, it is possible to indicate by means of a marker that the class Integer is concurrently modified as shown on the top left hand side window of the interface. In the right hand side window an annotation marker will indicate that a line was concurrently modified by another user. The position of the modified line is computed by using the line number indicated by the ghost operation.

Generated operations	Privacy filter	Shared ghost operations
op_1=*delete(User$_1$,Integer.java,15-18)*	do not filter	$g(op_1)$=*delete(User$_1$,Integer.java,15-18)*
op_2=*update(User$_2$,Integer.java,16,"return false;")*	filter content	$g(op_2)$=*update(User$_2$,Integer.java,16,-)*
op_3	do not send ghost	-

Table 2. Summary of operations.

An annotation can be associated with a marker in order to provide additional information regarding concurrent changes. Assume the editor is capable of finding the methods associated with a certain line range. In this way, the user can be informed that there is a conflict between his local change and the remote ones. For instance, the associated annotation in figure 2 informs the user that the method isReal() locally deleted was modified by another user. In this manner, the user can decide to contact the other user or not to delete the method.

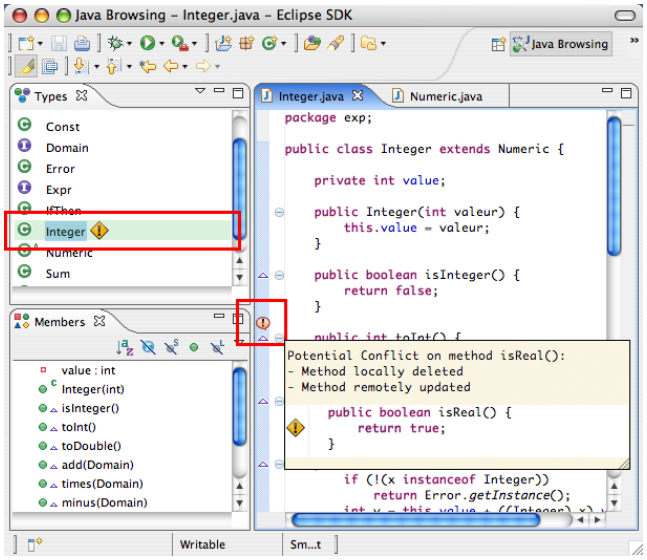

Figure 2. Interface at the first site after integration of ghost operations

Let us analyse what happens at the site of the second user. After modifying the method isReal(), the ghost operation of the first user arrives at the site and awareness information will be displayed as shown in Figure 3. In the right hand side window the user will be notified that the method isReal() is deleted by annotating the lines composing this method. The left hand side windows displaying the class hierarchy and the methods belonging to class Integer will highlight the fact that class Integer was concurrently modified and method isReal() was deleted.

At the site of the third user, after the ghost operations $g(op_2)$ and $g(op_1)$ arrive, awareness information regarding modification of class Integer is presented as shown in the top left hand side window of the interface shown in the Figure 4. In this way, the user could examine the concurrent modifications performed in class Integer and be presented with almost the same view as the second user. The third user could also initiate a communication with the other users that performed concurrent changes.

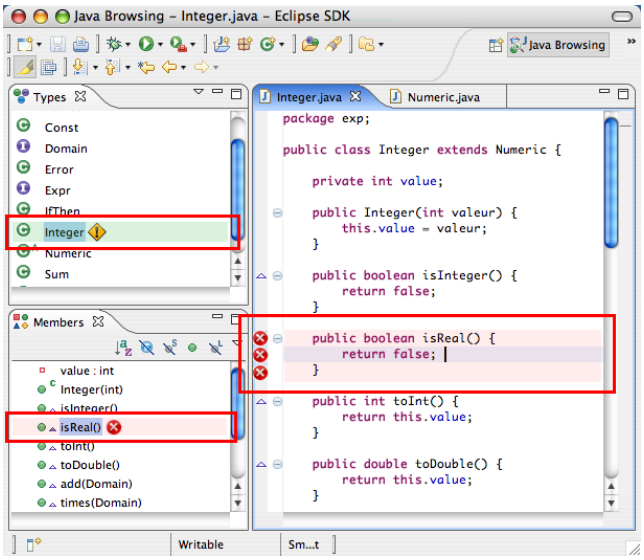

Figure 3. Interface at the second site after integration of ghost operations

The undesired situations produced by blind modifications illustrated in our motivating example do not occur:

- User 1 will not validate his removal of the method isReal() as he is informed that another user is currently modifying it.
- User 2 will notice that another user wants to remove the method he is currently modifying . He can initiate a communication with that user.
- User 3 is informed that the class Integer is currently modified by two users and therefore decide to postpone testing this class at a later time.

As shown in this section, undesired effects of isolated work such as useless or redundant work can be avoided by providing awareness information in real-time.

RELATED WORK

Many approaches in the literature were dedicated to providing various awareness mechanisms while working in a collaborative environment. But most of these awareness approaches were developed for the communication on real-time in order to help users to coordinate their group work. For instance, multi-user scrollbars represent the relative location of each user in a large document by means of a coloured bar layered beside the conventional scrollbar [2], telepointers indicate where users are pointing [13] and radar views [5] display miniatures of user workspaces which might contain user pointers. However, these approaches cannot be used for

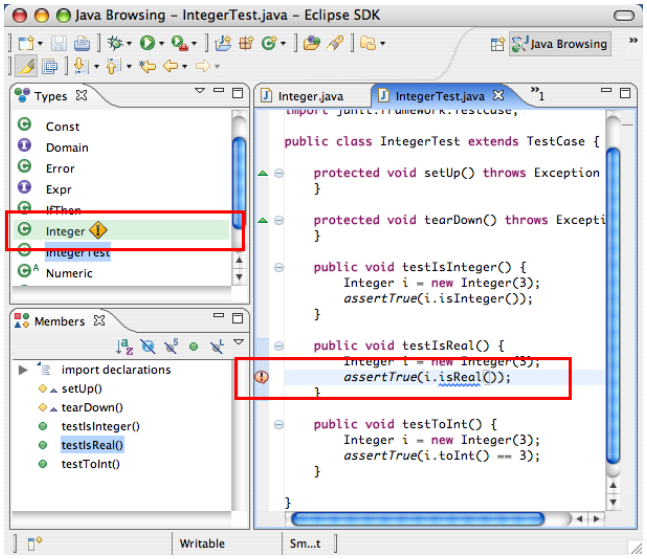

Figure 4. Interface at the third site after integration of ghost operations

avoiding blind modifications in asynchronous communication.

Few awareness mechanisms were proposed for the asynchronous mode. In the field of configuration management systems, the oldest mechanism for avoiding blind modifications is the CVS watches [3]. Watches allow developers to specify the artifacts they want to monitor. A developer can announce his intent of modification of an artifact by invoking a certain command. This command triggers notifications by means of emails to developers that registered for the change of those artifacts. However, this approach offers limited awareness information by means of emails and does not provide a presentation mechanism.

Most awareness approaches for the asynchronous communication concentrated mainly on change awareness. These approaches highlight changes made by other participants over time to an artifact such as a document or workspace. An initial framework on change awareness was proposed in [4] and then refined in [12]. These approaches maintain a user aware about changes that were performed and published while he was working in isolation. They do not present changes that are concurrently performed and not yet published and therefore, these approaches do not prevent blind modifications.

In [9] the authors proposed an editing profile that counts operations performed by users on different parts of the document, such as paragraphs, sentences and words in the case of textual documents. The editing profile provides an awareness mechanism regarding the hot areas of the document with the highest number of changes. Unfortunately, it is computed only after users perform an update of their copy of the document. Therefore this mechanism is dedicated for change awareness and does not help avoiding blind modifications. The editing profiles proposed in [9] can be computed in our approach by using the committed operations. Additionally, in our approach the same profile can be computed in real-time before changes are published. In this way blind modifications might be prevented.

The State Treemap [7] is an awareness widget designed to inform users about states of the shared documents. Different states are defined for a document such as LOCALLYMODIFIED, POTENTIALLYCONFLICT – when two copies of the document are modified and none of the changes are published yet – or WILLCONFLICT – when a document copy is modified locally and some changes on that document have been committed. However, the granularity of the awareness information is the document and therefore it is impossible to locate concurrent modifications within the document. Moreover, no measure for the divergence between two copies is provided.

Palantìr [10] provides awareness information about concurrent modifications performed in isolation in the context of configuration management systems. It is based on the same principle as State Treemap, the main difference being that a severity information that computes the amount of changes performed among documents was added. Unfortunately, the granularity of provided information is still the document. Moreover, the severity metrics does not provide enough information to infer changes that could cause potential conflicts at the merging phase.

Concerning quantitative measurement of divergence between document copies, the approach proposed in [8] provides divergence metrics. Contrary to Palantìr and State Treemap approaches, metrics are computed using operations modeling concurrent changes and not regarding events triggered by document state transitions. Merging of these concurrent operations is simulated in real-time on each site making possible the computation of various metrics. For instance, it is possible to compute the amount of changes performed on each document as in Palantìr, but also an amount of conflicting/overlapping changes. However this approach provides only metrics for each shared document but does not localise changes within documents.

Our approach is localising exactly the changes in a document and additionally deals with ghost operations that maintain user privacy while providing group awareness. Our approach can be seen as a more general approach than Palantìr, State Treemap and divergence metrics approaches in the sense that it can simulate the functionality of these three approaches. If ghost operations carry data about location of the targeted artifacts, it is possible to compute the information requested by State Treemap approach. If additionally the amount of changes is included in ghost operations, then severity measure proposed in Palantìr can be included. Furthermore, if ghost operations contain precise information about the location and size of changes within the documents, then the divergence metrics proposed in [8] can be evaluated.

The NICE [11] approach provides a notification mechanism for both real-time and asynchronous collaboration. Various notification policies can be defined such as system-triggered (instant or scheduled) or user controlled. If appropriate set-

tings for the notification policies are set, the system could be used to avoid blind modifications. However, a user can be aware of other user changes only when he decides to integrate them with his own as reception of operations at one site implies their immediate integration. The approach does not deal with ghost operations that offer support for maintaining user privacy and the possibility of being aware about group changes without the integration of these changes.

CONCLUSIONS AND FUTURE WORK

In this paper we presented a novel collaborative writing mode for avoiding *blind modifications* that occur during isolated work. We illustrated by means of scenarios the undesired consequences of blind modifications such as redundant or useless works. Based on the assumption that users are most of the time connected including work in isolation, we proposed to exploit their connectivity by continuously providing them with awareness information about group activity. We introduced the concept of *ghost operations* that carry information about performed operations while preserving user privacy preferences. As an example, we showed how awareness information provided by ghost operations could be represented in a software engineering development environment. We expect that provided awareness information will generate group communication and auto-coordination between users in order to prevent conflicts.

In this paper we presented the main ideas of our approach. We plan to develop a model for our proposed interaction mode including the issues of maintaining consistency in the presence of both real and ghost operations. Further, we want to investigate suitable interfaces for allowing users the possibility to filter operations according to their privacy preferences. Next, we plan to implement the awareness mechanism described in this paper for collaborative authoring of file systems containing text and software engineering code source documents. For this we will implement the awareness mechanism over the Eclipse plugin of Libresource [1], a version control system using the operational transformation approach [6]. Another direction of future work is to investigate the usability and the benefits of our approach by performing user studies.

REFERENCES

1. SO6, an operational transformation-based synchronizer, 2007. http://www.libresource.org/.

2. R. M. Baecker, D. Nastos, I. R. Posner, and K. L. Mawby. The User-centered Iterative Design of Collaborative Writing Software. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems - CHI'93*, pages 399–405, Amsterdam, Netherlands, 1993. ACM Press.

3. B. Berliner. CVS II: Parallelizing Software Development. In *Proceedings of the USENIX Winter Technical Conference*, pages 341–352, Washington, D. C., USA, January 1990. USENIX Association.

4. C. Gutwin. *Workspace Awareness in Real-time Groupware Environments*. Ph.D. Thesis, Department of Computer Science, University of Calgary, Calgary, Canada, 1997.

5. C. Gutwin, S. Greenberg, and M. Roseman. Workspace Awareness Support with Radar Views. In *Conference Companion on Human Factors in Computing Systems - CHI'96*, pages 210–211. ACM Press, April 1996.

6. P. Molli, G. Oster, H. Skaf-Molli, and A. Imine. Using the Transformational Approach to Build a Safe and Generic Data Synchronizer. In *Proceedings of the ACM SIGGROUP Conference on Supporting Group Work - GROUP 2003*, pages 212–220, Sanibel Island, Florida, USA, November 2003. ACM Press.

7. P. Molli, H. Skaf-Molli, and C. Bouthier. State Treemap: an Awareness Widget for Multi-Synchronous Groupware. In *Proceedings of the Seventh International Workshop on Groupware - CRIWG 2001*, pages 106–114, Darmstadt, Germany, September 2001. IEEE Computer Society.

8. P. Molli, H. Skaf-Molli, and G. Oster. Divergence Awareness for Virtual Team Through the Web. In *Proceedings of World Conference on the Integrated Design and Process Technology - IDPT 2002*, Pasadena, California, USA, June 2002. Society for Design and Process Science.

9. S. Papadopoulou, C.-L. Ignat, G. Oster, and M. Norrie. Increasing Awareness in Collaborative Authoring through Edit Profiling. In *Proceedings of the IEEE Conference on Collaborative Computing: Networking, Applications and Worksharing - CollaborateCom 2006*, pages 1–10, Atlanta, Georgia, USA, November 2006. IEEE Computer Society.

10. A. Sarma, Z. Noroozi, and A. van der Hoek. Palantìr: Raising Awareness among Configuration Management Worspaces. In *Proceedings of the International Conference on Software Engineering - ICSE 2003*, pages 444–454, Portland, Oregon, USA, May 2003. IEEE Computer Society.

11. H. Shen and C. Sun. Flexible Notification for Collaborative Systems. In *Proceedings of the ACM Conference on Computer-Supported Cooperative Work - CSCW'02*, pages 77–86, New Orleans, Louisiana, USA, November 2002. ACM Press.

12. J. Tam and S. Greenberg. A Framework for Asynchronous Change Awareness in Collaborative Documents and Workspaces. *International Journal of Human-Computer Studies - IJHCS*, 64(7):583–598, July 2006.

13. S. Xia, D. Sun, C. Sun, and D. Chen. Collaborative object grouping in graphics editing systems. In *Proceedings of IEEE 2005 International Conference in Collaborative Computing (CollaborateCom'05)*, San Jose, California, USA, December 2005.