# A Conflict Detected Earlier is a Conflict Resolved Easier

Anita Sarma and André van der Hoek
*Department of Informatics*
*University of California, Irvine*
*Irvine, CA 92697-3425 USA*
*{asarma, andre}ics.uci.edu*

## Abstract

*Open Source development is highly distributed and parallel in nature. There are no definite boundaries, either for people or from where they work. This high level of parallel, distributed development leads to conflicting changes made concurrently by different developers. Because OSS developers lack the kinds of informal coordination opportunities that collocation offers, OSS developers must rely on mailing lists, discussion groups, and tools such as CM and bug tracking systems to try to manage their parallel efforts such that conflicts do not occur. Unfortunately, these coordination mechanisms are not adequate: it still regularly happens that parallel changes interfere, either via direct overlap or indirect, semantic conflicts. In this paper, we build upon our previous work in raising awareness as a mechanism to support better coordination among developers, and introduce a new integration of our Palantír tool with Eclipse as well as a new visualization of parallel work that we believe is especially useful in Open Source settings.*

## 1. Introduction

Open Source development is highly distributed and by its very nature a multi-person, team effort [1, 2]. One of the key aspects of any Open Source project is that there are no definite boundaries for people – who participates – and for location – from where they participate. Although there is typically a small group of core developers, other developers enter and leave the project on will, and may or may not contribute significant changes. Usually there are no set release dates; the software is released when the core group thinks that it is time. This development style is very different from the traditional corporate development style in which a predefined group of developers follows a carefully-defined process to develop a software system according to a predefined timeline. We make two observations regarding the style of OSS development as compared to traditional software development.

1. It has been shown that collocation helps developers in getting an idea about their co-developers work [3, 4] Coffee hour talks or just casual conversations provide the context that helps developers gauge such things as what problems their co-developers are addressing and the schedules under which they are operating. This context helps developers in informally coordinating their efforts and code changes [5].
2. It has also been shown that parallel development leads to conflicting changes [6]. Some changes can be resolved with the help of automated merge tools [7], but when changes overlap or, worse, have semantic aspects, they have to be manually resolved, often a time consuming and costly effort.

From these two observations we can infer that Open Source development, by its very nature of being distributed and involving parallel work, will lead to the presence of conflicting changes that cannot be caught by the same kinds of informal interactions as present in traditional software development. OSS, in a way, is the epitome of parallel development: anyone who is interested in a project can work on anything that interests them at any point in time.

To compensate, most OSS developers extensively use such tools as e-mail, discussion lists, bug tracking tools, and CM tools to "informally" coordinate their work with ongoing and past activities. This clearly helps, and many problems are avoided. However, it also is somewhat of an undesirable situation as it is labor intensive, requires manual action and interpretation, and typically is based on "old" data – data about events that have already happened. Take CM systems, for instance. These systems manage parallel work by partitioning work into isolated workspaces. Developers make their changes in isolation and periodically resynchronize their work with the contents of the repository. This is done either by checking in the files that they have modified, or by synchronizing the files in a workspace with newer versions that other developers have committed to the repository. Conflicting changes, thus, are discovered only *after they have been introduced*. Some of these conflicts can be merged with the help of automated tools [8], but in the event of over-

lapping changes (either directly, as in changes to the same lines of code, or indirectly, as in changes that are in different parts of the code but semantically interfere) merge tools fail or produce false results and the conflicts have to be manually resolved [7].

The problem is that in current CM systems it is not possible to get an overall picture of the extent of parallel changes in different workspaces and how those changes relate to each other. The isolation enforced by the CM systems limits developers in knowing "what is going on", and thus takes away the information upon which they could decide to have informal "online hallway conversations".

To address this problem, we built Palantír, a CM workspace awareness tool that deliberately breaks the isolation of workspaces. Palantír operates by providing developers with an at all times up to date picture of any relevant changes in any of the parallel workspaces. The availability of this information gives developers a chance to be aware of "problematic" concurrent parallel activities, and coordinate with each other or even adjust ones own activity based on this information. Developers can detect potential conflicts *as they are occurring*.

In our previous work [9], we showed that Palantír can easily be used to instrument existing CM systems with awareness (we integrated Palantír with RCS [10], CVS [11], and Subversion [12]; each integration was less than 1000 lines of code). Moreover, we concentrated on building a series of visualizations that each have different properties of how conflicting changes are presented to the developer [13]. In this paper, we concentrate on the applicability of Palantír in the Open Source domain. In particular, we introduce a new view (which we call the full project view) that presents a project-wide overview of all workspaces and introduce our integration of Palantír with Eclipse. The first helps in giving the core set of developers of an Open Source project an idea of all individual efforts currently underway, the second demonstrates that Palantír can be integrated into a representative development environment and helps those developers using Eclipse with a minimally obtrusive awareness mechanism.

The remainder of the paper is organized as follows. Section 2 discusses our previous work and the high-level approach that Palantír follows. Next, we present the new full project view in Section 3. Section 4 introduces the integration of Palantír and Eclipse. We discuss related work in Section 5, and conclude in Section 6 with an outlook at our future work.

## 2. Palantír to Date

Palantír is based on the hypothesis that conflicts in parallel development can be considerably reduced, both in magnitude and number, by providing developers with an insight into ongoing changes in parallel workspaces. Instead of solely relying on the coordination mechanism of CM systems (i.e., the process of checking in and checking out artifacts, with or without a necessary merge step), Palantír provides developers with information so they can detect potential conflicts as they start occurring, and can self-coordinate their activities to avoid them or resolve them before they grow too large.

Palantír is not another CM system, but is a workspace awareness tool that builds on top of existing CM facilities and concentrates on the collection, distribution, organization and presentation of relevant workspace information. Palantír does not change the way developers interact with the CM system (i.e., the commands with which they interact with the CM system), but instead silently intercepts activities and shares relevant information regarding those activities.

We briefly discuss the components of Palantír below (a full discussion of Palantír, its architecture, visualizations, and integration with several existing CM systems can be found in [9]).

A *workspace wrapper* collects configuration management activities and translates those to Palantír events. These events are then distributed by the Siena event notification service [14], which we have adopted as our mechanism of transmitting information from workspace to workspace. Since different CM systems have different access mechanisms, a workspace wrapper is designed specifically for a particular CM system and must interface with the existing mechanism of user interaction (for instance, through wrapping existing commands or leveraging the trigger capabilities present in some CM systems).

The *internal state* stores and organizes all events, the relevant set of which are then extracted by the *extractor* before being displayed by one or more *visualization* components. Currently we have implemented four different visualizations (ticker tape, tabular view, explorer view and the fully graphical visualization), two of which are presented in Figure 1. Each visualization shows, in its particular form, specifically which artifacts are being changed by which other developers. In addition, they present a measure of severity of those changes: for each change, Palantír calculates the percentage of the artifact that has changed as a rough measure indicating "how much" potentially interfering activity is occurring in another workspace.

In the *explorer* view, shown in the left hand side of Figure 1, the artifacts in the local workspace are shown organized in the familiar form of an expandable tree. This tree is enhanced with vertical bars indicating the severity of ongoing and committed changes: the longer the bar, the higher the severity of the change. Although not visible in this black and white view, changes are color coded to distinguish changes in a local workspace from changes in other, "remote" workspaces.

The fully graphical visualization, shown on the right hand side of Figure 1, presents a developer with a hierarchical "3-dimensional" stackable view of an artifact and its constituents (in this case version 1.1 of the folder home/word"). Each constituent artifact may itself contain other artifacts and each artifact in the view may be present and being changed in multiple workspaces (as indicated by stacks of artifacts). Color-coding separates different workspaces. For instance, the stack for the artifact "/home/word/edit" indicates that Ellen, Pete, and Mike each have a version of the artifact in their workspace. Pete and Mike each have version 1.0 in their workspace, and their changes are still in progress as indicated by the question mark. Ellen, on the other hand, already has checked in a new version of the artifact (as indicated by the exclamation mark), resulting in her having version 1.1 in her workspace. The severity of ongoing and committed changes is shown by the vertical blue lines: the taller the bar, the higher the severity.

## 3. Full Project View

Thus far, the visualizations of Palantír have been geared towards individual developers. Each visualization only shows concurrent changes that relate to artifacts that are in the "local" workspace. We realize that only having such self-centered views is a limitation and that, benefits exist to having an overarching view of all parallel activities. In particular, we believe project managers in regular corporate settings as well as core programmers in Open Source settings can benefit from having a view in which all concurrent workspaces are shown and potential conflicts among workspaces are displayed. The needs of project managers are different: they need both a high level and detailed view of parallel activity to better handle task management. Similarly, core developers in open source projects can benefit from knowing who is working on what, what changes may be forthcoming for future integration cycles, and generally steer developers who are working on related projects towards each other.

We are currently building a new visualization for seeing the entire project at a glance. This visualization, an early prototype of which is shown in Figure 2, provides an overview of changes in all the workspaces. In particular, it lays out all the workspaces in the project in a circle. If there is any overlap of work in any pair of workspaces (as defined by changes to the same files), this overlap is indicated with the presence of a line between the two workspaces. The higher the overlap, the thicker the lines appear. In order to avoid cluttering the visualization, the manager can set a threshold for viewing the overlaps. For example, one may want to only see overlaps that are greater than 40%.

Anyone using the full project view can zoom into any of the workspaces to view the changes in that workspace. Specifically, zooming brings up the Palantír explorer visualization discussed in the previous section. Double clicking on a developers name in the visualization fully opens that developers' workspace in a separate window. This allows easy exploration of conflicts, by showing workspaces side by side. Managers can guide a project; core developers can point novices to things they may have missed or are doing wrong. In turn, novices can use the view as a rudimentary kind of "expertise" browser [15], in the sense that they can see who is working on

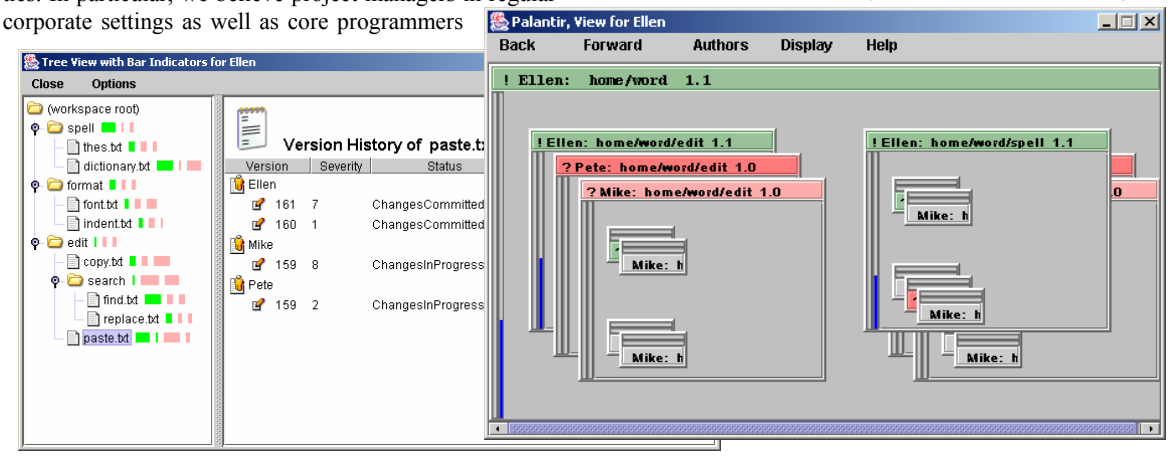


**Figure 1. Explorer Visualization and Fully Graphical Visualization.**

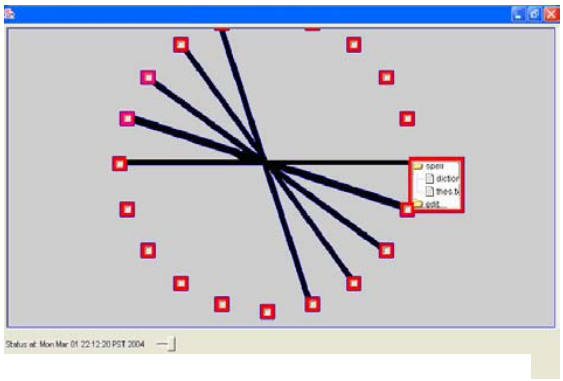

**Figure 2. Project View Visualization.**

what parts of the code.

A particularly interesting feature of the full project visualization is that it allows the user to go "back in time". By using the "time" slider at the bottom of the window, it is possible to view the level of conflict at regular past intervals. This helps managers and any other party using the visualization to gain an understanding of how a project evolves and the level of parallel conflicts during that evolution.

We are currently still in the building stages of the project management view, but have high hopes that it will be one of the most informative views of Palantír yet – it can be used by anyone, and provides interesting information at both a high and a very detailed level.

## 4. Eclipse Integration

So far Palantír has been independent of any development environment and runs as a separate application. On the one hand, integration with a particular development environment is not good, as different developers have different preferences and integration with a particular editor limits the use of Palantír. On the other hand, when a developer is deep into programming they might not switch to any other applications until their task is done to avoid any kind of distraction. Thus, in the case of Palantír, the developer might be missing out on important information during that time. To avoid this context switch between the editor and Palantír, we have integrated Palantír with Eclipse, a Java development environment. We are aware that different developers have different preferences and Eclipse may not be the editor of choice of all developers. The purpose of our integration is therefore to demonstrate that it is possible to easily integrate Palantír into a development environment; we hope other integrations can be as simple as this one.

We decided to integrate Palantír with Eclipse specifically because Eclipse is itself an Open Source product geared towards OSS development. Since its initial release Eclipse has become increasingly popular as the Java editor of choice. Additionally, our choice was guided by the fact that Eclipse has been designed such that adding new functionality is relatively easy; it's plug-in facility is very powerful and the number of extension points is great.

Our integration is provided through a new Palantír perspective in Eclipse, similar to the already existing Java perspective. Shown in Figure 3, the perspective lays out all the artifacts present in the local workspace in an expandable tree view. Each artifact is annotated on the left hand side with a small red triangle denoting the cumulative severity of changes made by other developers: the higher the cumulative severity, the larger and redder the triangle. On the right hand side of the artifact name, the numerical value of the cumulative severity is listed in text as well. A similar mirror image of a blue triangle is used to denote the impact of the changes.[1] This view presents only cumulative values, developers should use the other Palantír visualization for in-depth investigations.

Note that the Palantír-Eclipse integration recalculates the severity of changes every time a file is saved: this allows Palantír to share up-to-the-minute details regarding activities in other workspaces.

## 5. Related Work

To date, CM systems have very limited awareness capabilities. Existing CM tools usually require explicit manual action to break the isolation of workspaces, and even then information that is provided is old – not current and up-to-date directly from the workspace. Other than Palantír, few tools attempt to bridge CM and Computer-Supported Collaborative Work in raising awareness for users of CM systems. Jazz [16] is a collaborative tool built within the Eclipse IDE that shows the status of artifacts in other workspaces (artifact has been checked out and/or committed). Night watch [17] builds on the CVS watch facility, and notifies developers by email when an artifact in which they are interested has been changed by others. State Tree Maps [18] is an awareness widget that shows which artifacts have been modified locally or remotely and which artifacts have been checked-in into the repository. Compared to these tools, Palantír has two advantages: it provides a measure of severity, and it provides the full project view.

[1] We intend to build not just a measure of severity, but also a measure of change impact – one to measure "how large" a change is, the other to measure "how much I should care" about the change.

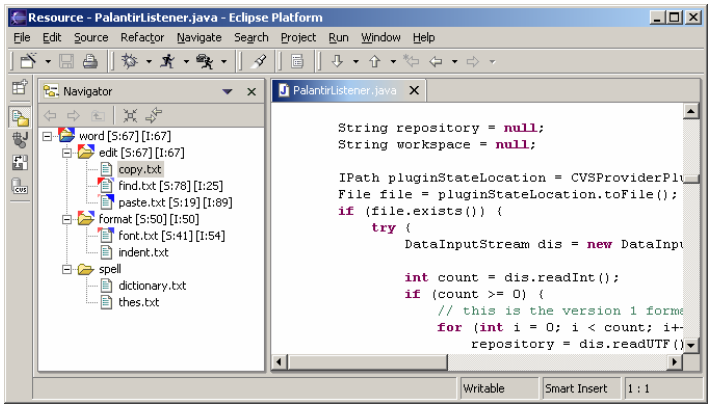


**Figure 3. Eclipse Integration.**

## 6. Conclusions

Palantír is based on the hypothesis that enhancing CM systems with awareness allows developers to have an improved insight into potentially conflicting parallel activities. The hope is that developers will use this insight to self-coordinate, in effect making earlier the point at which a conflict is detected and thereby reducing the amount of effort involved in addressing it.

We are currently finishing up the Eclipse integration and full project view. Our next steps in the project are to put Palantír in use, first in a class project in which we will have a large group of students all be part of a single team; and then in actual real-world settings as we release and deploy Palantír to the Open Source community itself.

## 7. Acknowledgements

We thank Ryan Yasui, Roger Ripley and Ksatria Williams for their contributions to Palantír.

Effort funded by the National Science Foundation under grant numbers CCR-0093489 and IIS-0205724.

## 8. References

[1] Raymond, E.S., *The Cathedral & the Bazaar: Musings on Linux and Open Source by an Accidental Revolutionary*. 2001: O'Reilly,.

[2] Erenkrantz, J.R. and R.N. Taylor. *Supporting Distributed and Decentralized Projects: Drawing Lessons from the Open Source Community*. in *Proceedings of 1st Workshop on Open Source in an Industrial Context*. Oct, 2003. Anaheim, California.

[3] Dourish, P. and V. Bellotti. *Awareness and Coordination in Shared Workspaces*. in *ACM Conference on Computer-Supported Cooperative Work*. 1992.

[4] Gutwin, C. and S. Greenberg. *Workspace Awareness for Groupware*. in *CHI'96 Conference Companion on Human Factors in Computing Systems*. 1996.

[5] Grinter, R.E., *Supporting Articulation Work Using Software Configuration Management Systems.* Computer Supported Cooperative Work, 1996. **5**(4): p. 447-465.

[6] Perry, D.E., H.P. Siy, and L.G. Votta, *Parallel Changes in Large-Scale Software Development: An Observational Case Study.* ACM Transactions on Software Engineering and Methodology, 2001. **10**(3): p. 308-337.

[7] Mens, T., *A State-of-the-Art Survey on Software Merging.* IEEE Transactions on Software Engineering, 2002. **28**(5): p. 449-462.

[8] IBM, *XML Diff and Merge Tool*. 2002.

[9] Sarma, A., Z. Noroozi, and A. van der Hoek. *Palantír: Raising Awareness among Configuration Management Workspaces*. in *Twentyfifth International Conference on software Engineering*. 2003. Portland, Oregon, USA.

[10] Tichy, W.F., *RCS, A System for Version Control.* Software - Practice and Experience, 1985. **15**(7): p. 637-654.

[11] Berliner, B. *CVS II: Parallelizing Software Development*. in *USENIX Winter 1990 Technical Conference*. 1990.

[12] Tigris.org, *Subversion*.

[13] Sarma, A. and A. van der Hoek. *Visualizing Parallel Workspace Activities*. in *IASTED International Conference on Software Engineering and Applications (SEA)*. 2003. Marina Del Rey, California.

[14] Carzaniga, A., D.S. Rosenblum, and A.L. Wolf, *Design and Evaluation of a Wide-Area Event Notification Service.* ACM Transactions on Computer Systems, 2001.

[15] Mockus, A. and J. Herbsleb. *Expertise Browser: A Quantitative Approach to Identifying Expertise*. in *2002 International Conference on Software Engineering*. 2002.

[16] Cheng, L.-T., et al. *Jazzing up Eclipse with Collaborative Tools*. in *18th Annual ACM SIGPLAN Conference on Object-Oriented Programming, Systems, Languages, and Applications / Eclipse Technology Exchange Workshop*. 2003. Anaheim, CA.

[17] O'Reilly, C., P. Morrow, and D. Bustard. *Improving Conflict Detection in Optimistic Concurrency Control Models*. in *Proceedings of the Eleventh International Workshop on Software Configuration Management*. 2003. Portland, Oregon.

[18] Molli, P., H. Skaf-Molli, and C. Bouthier. *State Treemap: an Awareness Widget for Multi-Synchronous Groupware*. in *Seventh International Workshop on Groupware*. 2001.