Palantír: Coordinating Distributed Workspaces

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

Abstract

Distributed software development suffers from limited collaboration capabilities, as developers are unable to easily coordinate their efforts across physical boundaries. Different fields, such as CSCW and groupware, have attempted to bridge this gap, but few of the approaches developed so far have been incorporated in current software development environments. Configuration Management (CM) systems are vital to any software development process, support distributed development, and are in widespread use. Unfortunately, they have only limited support for distributed collaboration. In this paper we describe Palantír, a system that is aimed at bringing collaborative capabilities to distributed development. Palantír builds upon existing CM systems to introduce project awareness to the developer workspace. In particular, Palantír supports close collaboration among developers by visualizing concurrent changes and showing, in real time, the severity and impact of those changes on the developer's workspace.

1. Introduction

Distributed software development and virtual organizations involve developers who are geographically separated. In these settings, work tends to be carefully partitioned to minimize the number of conflicts. Nonetheless, tasks sometimes overlap and as a result of such parallel development direct and indirect conflicts arise. Direct conflicts arise when parallel changes to an artifact involve modification to the same part of that artifact. Indirect conflicts arise when mutually exclusive changes individually compile and execute correctly, but combined in a single system lead to a non-working version of the system.

The disciplines of CSCW and Groupware have long concerned themselves with providing facilities in which distributed teams can coordinate their efforts. Systems such as BSCW [1], TUKAN [2], DIVA [3], SUITE [4], and MMM [5], for example, help in collaborative editing of artifacts. Unfortunately, although successes certainly exist, few of these systems and approaches have been incorporated in current software development practices.

Configuration management (CM) systems [6] are vital to any software development project. Typically, a CM system is used to help coordinate the activities of developers. By providing capabilities that either avoid parallel development altogether (e.g., locking) or assist in resolving conflicts (e.g., merging), these systems certainly have succeeded in reducing the number of coordination problems occurring in a typical software development project. Unfortunately, however, workspaces remain isolated and changes made by different developers are not visible until check-in time, thereby significantly delaying the discovery of potential problems.

As a complement to existing CM systems, and in order to address this problem, we are developing Palantír, a system that provides a developer with information about concurrent changes in a software development project. In particular, Palantír informs developers of check-outs and check-ins being performed by other developers in their workspaces. In doing so, Palantír deliberately but non-intrusively breaks the isolation that is currently provided by traditional configuration management workspaces.

Palantír is based on the observation that it is not only pertinent for developers to have an idea of their own set of changes, but to also be continuously informed about the changes being made by other developers. Specifically, Palantír allows developers to better coordinate their activities by providing each developer with a graphical display that shows the set of artifacts they are modifying, meta-data about those artifacts (e.g., artifact name, version number, author information, etc.), and the severity and impact of the modifications being made in parallel by other developers. Knowing this information allows developers to better assess the ongoing activities and accordingly coordinate their activities amongst each other.

In the remainder of this paper, we discuss the details of Palantír. We first introduce a small example in Section 2. We then describe the architecture of Palantír in Section 3. Section 4 discusses our implementation thus far. Section 5 discusses related work and we conclude in Section 6 with an outlook at our future work.

2. Example Scenario

Figure 1 depicts a scenario in which four developers use their workspaces (numbered W1 through W4) to make some changes to a particular software system. Two of those developers, Ellen (W1) and Pete (W2), inadvertently make some conflicting changes. Specifically, Ellen makes some changes in the program logic of "spell.c", which upon program execution leads to a different value

of a global variable in “bar.c”. Meanwhile, Pete modifies the behavior of the main control loop in “bar.c”. While each of these two changes individually does not lead to problems and results in a correct system, the fact that the behavior of the main loop of “bar.c” depends on the value of the global variable in “spell.c” leads to an overall failure of the program when the two changes are integrated. Thus, even though both changes are syntactically and semantically correct in their own right, they indirectly conflict by influencing the overall semantics of the program in an undesirable manner. These kinds of conflicts are not visible to either of the developers until both changes are checked in and an overall system test is performed.

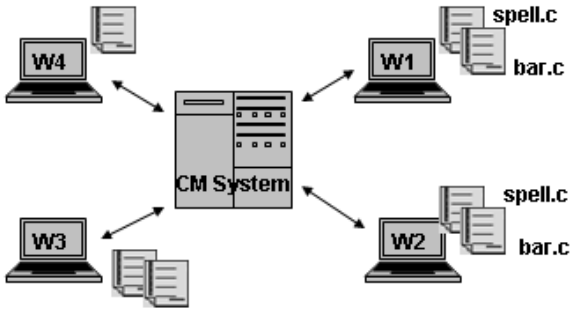

Figure 1. Example Scenario.

3. Conceptual Architecture

Figure 2 shows the conceptual architecture of Palantír, which consists of four different types of components interlinked via a generic event notification service [7]. The first component is the CM system, which serves as the source of all information used by Palantír. To drive updates to its visualizations, Palantír assumes that the CM system sends out notifications upon check-out or check-in of an artifact.

We use an event notification service for communication between the CM system and the other components of Palantír, because Palantír operates in a real time, distributed setting. Moreover, these kinds of event services provide routing algorithms that can be used to only deliver those events that are of interested to a particular workspace user [7]. This optimizes the use of the network and reduces the number of events that are sent back-and-forth significantly.

Once a modified artifact has been checked in, Palantír calculates and shows the severity of the change (e.g., the difference according to some measure between the old version and the new version of the artifact) and the impact of the change (e.g., the potential impact according to some measure of the change on the artifacts that another developer has in their workspace).

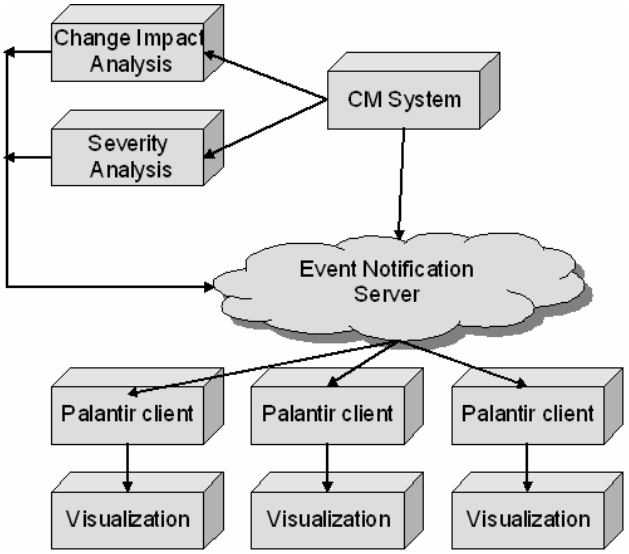

Figure 2. Conceptual Architecture of Palantír.

Based upon this information, we anticipate that developers can better coordinate their activities such that there are fewer conflicts.

The severity analysis component basically attempts to answer the question of how much has changed between the newly checked-in and previous version of an artifact. This allows a developer to be cognizant of the magnitude of the changes taking place. Different measures can be useful at different times. Initially, we intend to explore the following measures for severity calculation with respect to the checked out version.

- *Lines of code*. Calculate the relative number of lines of code that has been changed.
- *Token based difference*. Detect similarities on the basis of tokens and replace each keyword and variable throughout a file with a unique token before applying a differencing algorithm [8].
- *Abstract syntax tree*. Analyze the differences between the abstract syntax trees of the two different versions.

The impact analysis provides the developer with a synoptic view of how the changes made by others in the project affect their current work. The measures that we intend to explore are the following.

- *Lines of code*. Calculate the relative number of overlapping lines of code that have changed.
- *Interface*. Assuming information hiding, calculate the relative number of interfaces that have changed.
- *Dependency analysis graph*. Calculate the relative size of the dependency analysis graphs [9] to determine the “reach” of a change (e.g., how much of the code in the workspace might be affected on a semantic basis).

Of note is that both the severity analysis and the change impact analysis can be performed based on the information stored in the CM repository. This is possible since the analysis is carried out between the checked-in

and the checked-out versions. Nonetheless, more accurate and up-to-date information can be gathered if the analyses are performed based upon the actual contents of a developer workspace. To do so would require significant amounts of code to be shipped back and forth. We intend to explore the resulting tradeoff between accuracy and network usage to determine which solution would be more favorable.

The Palantír Client side intercepts the severity and change impact events and translates them into an internal representation that is subsequently used by the last component in the Palantír architecture, the visualization. At a minimum, the visualization component will show which artifacts are being checked-out and checked-in and the severity and impact of each of the changes. In doing so, developers are presented with an increased level of awareness of other developers' activities. We intend to develop a range of visualizations from which each developer can choose the one they prefer. Each of these visualizations will be based on a different balance among the amount of information displayed, interface usability, and intrusiveness of the interface (e.g., a ticker tape [10], a tabular view, small visual clues, or even a large, fully graphical visualization).

4. Implementation Details

In implementing Palantír, our focus thus far has been on creating one of the main visualization components. Shown in Figure 3, the visualization presents a developer with a hierarchical view of an artifact (in this case main.c, version 1.0) and its constituent artifacts. Each constituent artifact may itself consist of other artifacts, and each artifact in the view may exhibit multiple versions (as indicated by a stack of artifacts).

While not visible in the black-and-white screenshot, color coding separates users. Changes made by the workspace owner are highlighted in green, whereas artifacts being manipulated in other workspaces by other developers are shown in red.

Potential conflicts among workspaces are shown in a pair-wise fashion. Rather than having one visualization with all conflicts, which would quickly become very cluttered, Palantír shows the conflicts per workspace pair from the perspective of the workspace with which a visualization is associated (e.g., Ellen and Pete, Ellen and Jennifer, Ellen and Karl). As a consequence, each red box only shows the conflicts with the artifacts being manipulated by the owner of "the other" workspace

The Palantír visualization highlights artifacts that are being changed with a "?" to indicate that the exact nature of the change is not known yet. Once changes are committed to the CM system, Palantír highlights the artifact with an "!" to indicate the changes are stable.

The two vertical bars that associate artifacts that are stored in the CM system indicate the severity and change impact of the changes made to that artifact. The severity and change impact are shown both for individual, atomic artifacts and compound artifacts. Severity and change impact of compound artifacts summarize the severity and change impact of their contained artifacts. This allows a developer to monitor other workspace at a high level, only examining changes in constituent artifacts if the severity of change impact of a compound artifacts indicates a problem.

Note that atomic artifacts that have not been stored in the repository do not have any severity or change impact associated with them, since the potential changes are still "hidden" in the workspace. Compound artifacts that are not checked in yet, on the other hand, will have severity and change impact since their constituent artifacts already may have been changed and checked in.

The visualization component allows for zooming in and zooming out to understand changes at a fine-grained level of control. Double-clicking on a particular version of an artifact makes that the primary artifact being viewed and shows all the constituent of that artifact version. This process can be repeated until an artifact has no constituent artifact. A back button allows a developer to go "back up"

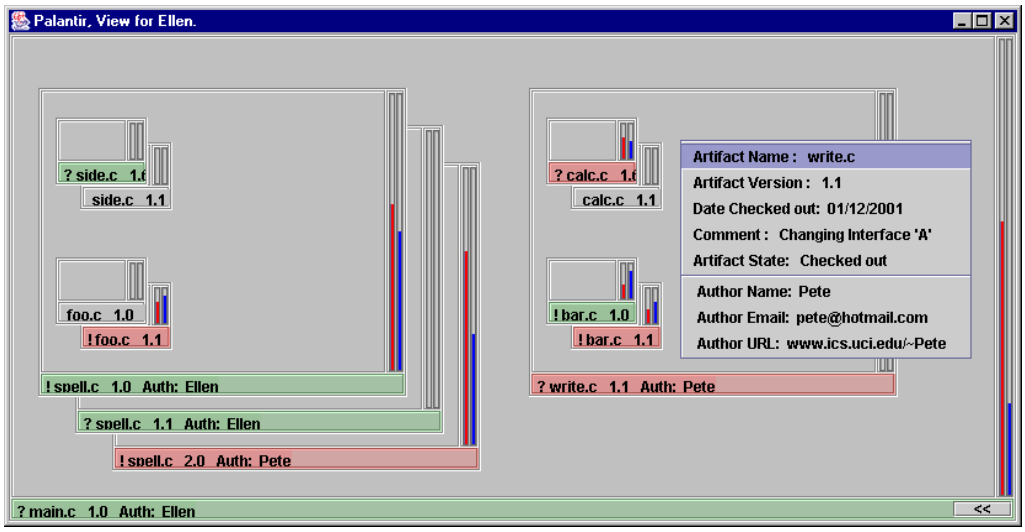

Figure 3. Palantír Visualization Component.

the chain to view the changes from a higher point of view.

Note also that, as shown in Figure 3, more detailed information, such as change comments and author names, can be requested. Also, individual, older versions that are displayed "lower" in a stack can be brought to the foreground with a simple mouse click.

5. Related Work

Our work draws from and combines contributions in three areas: computer-supported collaborative work (CSCW), configuration management, and software visualization. We briefly discuss related work in each of these areas below.

Most CSCW systems enable coordination between developers either by allowing multiple users to explicitly and immediately share their work (via shared workspaces or shared editors). Multi-user interfaces, like MMM[5] or Suite[4], provide excellent examples in allowing multiple, geographically distributed authors to make changes to a document concurrently and synchronously. Unlike Palantír, which aims to preserve isolation of authorship, the focus of these systems is on tight, immediate collaboration via sharing changes directly at real time.

TUKAN [2] is a collaborative editor that is similar to Palantír in its objective of preserving isolation of changes. To do so, it augments the editor's interface with an indication of the changes being performed by other developers. This is very similar to Palantír, but has the drawback that it only highlights the severity of changes as a simple frequency. Additionally, developers have to decide upon the group of collaborating authors a-priori.

BSCW [1] allows asynchronous collaboration and is suited for collaborations over the web. Shared workspaces are used for coordinating efforts between different authors. Among other features, BSCW provides some versioning and also annotates artifacts with an indication of the latest activities through which it was modified.

Most configuration management systems ignore awareness altogether [6,11] and developers work in their isolated workspaces. When a direct or indirect conflict occurs, developers have to manually resolve the changes. Although automated merging algorithms are provided that simplify this task, direct conflicts must still be resolved by hand—a time consuming and tedious task.

One exception is Coven [12], which requires developers to specify beforehand which artifacts they will be modifying. Based upon this information, Coven keeps track of the parallel changes being made and informs developers of potential conflicts. Unfortunately, this mechanism can only avoid direct conflicts. Furthermore, developers often do not know beforehand the complete set of artifacts they will be changing.

Visualization systems are used to present developers with useful visualizations that highlight certain aspects of the software under development. With respect to change, a number of different visualization systems have been developed that each provide a different mechanism of visualizing the evolution of the system over time. Some of these visualizations provide matrix views [13], others 3-D colored graphs [14] or bar graphs [15]. Yet others take a minimalistic approach. Ticker tapes [10], for example, only notify developers about recent check ins and check outs. Palantír builds upon all of these and distinguishes itself in two ways: it integrates severity and change impact information in the visualization display and is specifically geared towards avoiding direct and indirect conflict.

6. Conclusions

Palantír is a system that we are currently developing to bring workspace awareness to developers. Palantír is explicitly designed to operate in a distributed setting and aims to help in coordinating the activities that take place in each workspace. While not all changes that are made in isolation influence other developers, it is desirable that developers be quickly aware of all the pertinent changes happening in parallel. In a distributed setting, such awareness allows developers to coordinate their activities to avoid and resolve conflicts early, which has the potential of saving a significant amount of cost and effort that would otherwise have been involved in resolving the conflict at a later stage.

We are not finished developing Palantír. Now that we have nearly completed the visualization component, our focus is shifting to implementing the severity and change impact analysis components. Once those components are completed, we intend to experiment significantly in case studies involving both an open source project and a "regular" software development project. We will carry out the case studies to understand the effectiveness of a tool like Palantír in supporting coordination in distributed software development. In particular, we want to both understand whether Palantír is viewed as useful and informative by developers as well as whether the use of Palantír actually reduces the occurrence of direct and indirect conflicts.

7. Acknowledgments

This research is supported by the National Science Foundation under grant number CCR-0093489.

8. References

1. Appelt, W. *WWW Based Collaboration with the BSCW System,*. in *Proceedings of the Conference on Current Trends in Theory and Informatics,*. 1999.
2. Schümmer, T., and J.M. Haake. *Supporting Distributed Software Development by Modes of Collaboration*. in *Proceedings of the Seventh European Conference on Computer Supported Cooperative Work.* 2001.
3. Sohlenkamp, M. and G. Chwelos. *Integrating Communication, Cooperation, and Awareness: The DIVA Virtual Office Environment*. in *In Proceedings ACM 1994 Conference on computer-Supported Cooperative Work: Transcending Boundaries*. October 1994. Chapel Hill, NC,.

4. Dewan, P. and R. Choudhary. *A High-Level and Flexible Framework for Implementing Multi-user Interfaces.* in *ACM Transactions on Information Systems,*. 1992.
5. Bier, E.A. and Freeman. *MMM: A User Interface Architecture for Shared Editors on a Single Screen.* in *Proceedings of the ACM Symposium on User Interface Software and Technology,*. 1991.
6. Conradi, R. and B. Westfechtel, *Version Models for Software Configuration Management.* ACM Computing Surveys, 1998. **30**(2): p. 232-282.
7. 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.
8. A. Aiken, *MOSS - Measure of Software Similarity*. 2002.
9. Dwyer, M. and L.A. Clarke, *A Flexible Architecture for Building Data Flow Analyzers*, in *Proceedings of the Eighteenth International Conference on Software Engineering*. 1996, ACM. p. 554-564.
10. Fitzpatrick, G., et al. *Instrumenting and Augmenting the Workaday World with a Generic Notification Service called Elvin*. in *Proceedings of the Sixth European Conference on Computer Supported Cooperative Work.* 1999. Copenhagen, Denmark.
11. Burrows, C. and I. Wesley, *Ovum Evaluates Configuration Management*. 1998, Burlington, Massachusetts: Ovum Ltd.
12. Chu-Carroll, M.C. *Supporting Distributed Collaboration through Multidimensional Software Configuration Management.* in *Proceedings of the Tenth International Workshop on Software Configuration Management.* 2001.
13. Lanza, M. *The Evolution Matrix: Recovering Software Evolution using Software Visualization Techniques*. in *Proceedings of the Fourth International Symposium on the Principles of Software Evolution*. 2001.
14. Jazayeri, M., C. Riva, and H. Gall. *Visualizing Software Release Histories: The Use of Color and Third Dimension*. in *Proceedings of the International Conference on Software Maintenance.* 1999: IEEE Computer Society.
15. Baker, M.J., and Eick,S.G., *Space-Filling Software Visualization. Journal of Visual Languages and Computing*, 1995. **6**: p. 119-133.