Palantír: Increasing Awareness in Distributed Software Development

Anita Sarma André van der Hoek

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

ABSTRACT

Distributed software development, just like regular software development, typically involves developers working in parallel on the same set of artifacts. Unlike regular software development, however, distributed software development is limited since developers are unable to easily coordinate their efforts in person due to the presence of physical boundaries. While configuration management systems provide some automated coordination support in the form of locking and merging, the high cost of conflict resolution in distributed software development requires even higher levels of support to ensure as few integration problems as possible. In this paper, we introduce Palantír, a system that complements existing configuration management systems by providing distributed awareness of project progress. In particular, Palantír provides each developer with a graphical display that not only shows which remote artifacts are changing, but also presents them with a measure of both the severity and the impact of the changes. As a result, developers are provided with an increased level of awareness that allows them to detect and resolve problems much earlier.

1. INTRODUCTION

A large number of software development organizations have subsidiaries or branches located in different geographical locations. In these settings, work tends to be carefully partitioned to minimize the number of conflicts. Nonetheless, it is often unavoidable that, at times, developers have to simultaneously change the same set of artifacts. As a result of such parallel development, conflicts [1] arise both directly and indirectly. Directly, parallel changes to the same artifact may lead to conflicts if the changes involve modifications to the same parts of the artifact. Indirectly, mutually exclusive changes can individually be working properly, but combined lead to a non-working version of the system under development.

Consider, for example, a situation in which one change in the program logic leads to a different value of a global variable and a second change modifies the behavior of the main control loop of the program. While each change individually does not lead to problems, the fact that the behavior of the main loop of the program depends on the value of the global variable may lead to an overall failure of the program. Thus, while 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.

Typically, a configuration management (CM) system [2] is used to help coordinate the activities of developers such that the number of conflicts resulting from parallel change is reduced. 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 conflicts [3]. Nonetheless, merge conflicts do occur and often must be manually resolved, a time-consuming and sometimes difficult task. Furthermore, the issue of indirect conflicts remains unaddressed by CM systems [4]. At best, they provide a historical trace of changes. While useful in identifying what changes may have caused a problem and which developers may have been involved, no further information is provided and the problem has to be manually resolved. Especially when the developers involved reside in geographically different locations, this may be a rather unpleasant exercise.

As a complement to existing configuration management systems, we are developing Palantír, a system that is specifically designed to provide distributed project awareness targeted at reducing both direct and indirect conflicts. Palantír is based on the observation that it is not only pertinent for developers to know what artifacts are being changed by other developers, but also what the severity and impact of those changes are with respect to the specific task at hand. Palantír provides each developer with a graphical display that shows the relevant set of artifacts, illustrates which artifacts have already been and are being modified by other developers, and, based on an analysis of the changes, calculates and highlights the severity and impact of each change. In doing so, Palantír deliberately but non-intrusively breaks the isolation that currently exists when a developer is performing a task in their CM workspace. As a result, a developer can proactively resolve potential problems and thereby reduce the number, and significance, of direct and indirect conflicts.

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

2. Conceptual Architecture

Figure 1 shows the conceptual architecture of Palantír, which consists of four different types of components interlinked via a generic event notification service [5]. 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. A check-out represents the beginnings of a change activity. Since it is important for developers to be aware of which other developers are changing which artifacts, Palantír records this fact and displays it in its visualizations. A check-in signals the end of a change activity (or, at least, a checkpoint that is being made by a developer). Again, it is important for other developers to be aware of this fact and Palantír updates its visualizations accordingly. In addition, it 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 conflict according to some measure between the change and the artifacts that another developer has in their workspace). Whereas the severity can be calculated once for all developers, it is important to note that the impact is calculated per developer since each developer may have a different set of artifacts in their workspace and the impact of the change can vary accordingly.

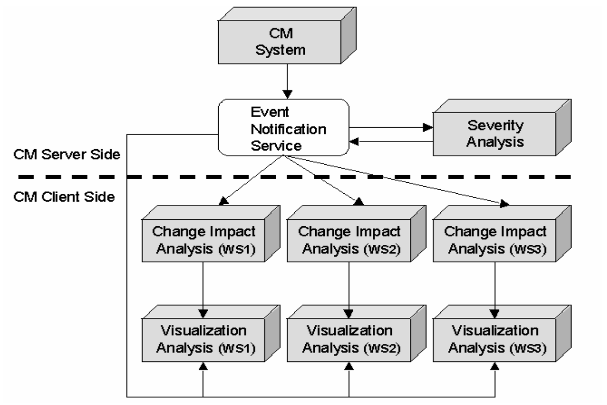

CM System
Event Notification Service
Severity Analysis
CM Server Side
CM Client Side
Change Impact Analysis (WS1)
Change Impact Analysis (WS2)
Change Impact Analysis (WS3)
Visualization Analysis (WS1)
Visualization Analysis (WS2)
Visualization Analysis (WS3)

Figure 1. Conceptual Architecture of Palantír.

It is important to note that the architecture is purposefully generic. Not only do we want Palantír to interoperate with many different CM systems, we also want to experiment and understand the effectiveness of different severity and change impact analysis algorithms as well as of different visualizations. It is also important to note that the architecture is inherently distributed. Different components can (and sometimes must) execute in different locations since they are all connected via a distributed event notification service.

The heart of Palantír is formed by the severity analysis component, the change impact analysis component, and the visualization component. 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. Different measures can be useful at different times. Some of the measures that we intend to explore are the following.

- **Relative number of lines of code that has been changed.** Although a remarkably trivial measure, it can be important to understand how many lines of code actually have been altered by a developer. The rationale is simple: the more lines of code that have been touched, the more faults that could have been introduced [6]. There are two drawbacks to using this measure. First, an action such as renaming a variable throughout a file may indicate a severe change whereas in reality it is a rather simple change that deserves little to no attention. The other drawback is that the measure is language dependent, and means much more for, for example, a Cobol program than a Prolog program.

- **Relative number of token-based differences.** Some algorithms used to detect and report similarities between two files are token-based and replace each keyword and variable name throughout a file with a unique token before applying a differencing algorithm. The advantage of this kind of measure is that it takes into account such simple changes as renaming a variable. The disadvantage is that the token replacement algorithm is dependent on the implementation language and requires specific language plug-ins.

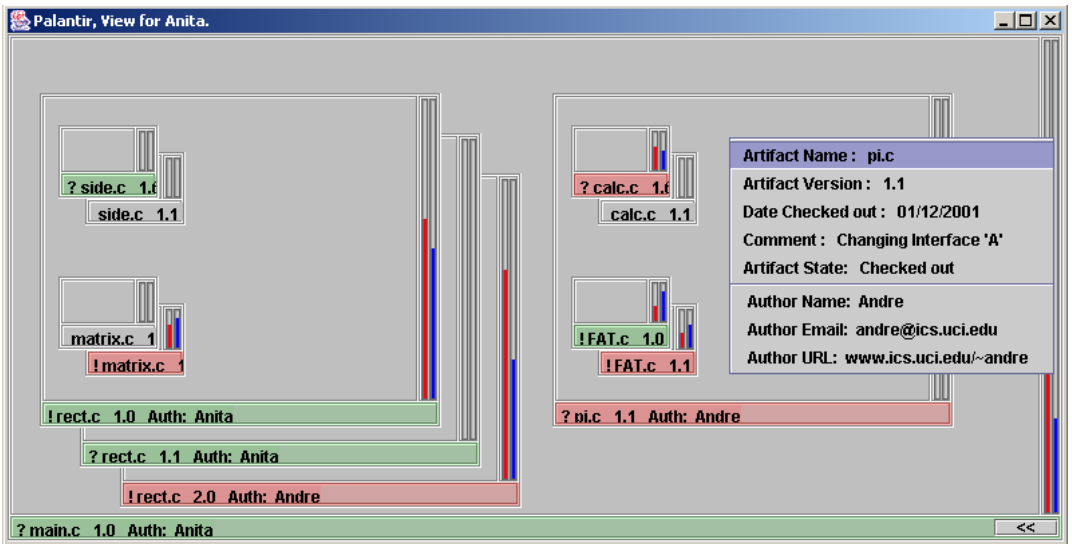
- **Relative number of changes in the abstract syntax tree.** It is possible to analyze the differences between the abstract syntax trees of the two different versions of an artifact. This has the distinct advantage that the measure is as closely as possible related to the implementation language, which results in a rather accurate assessment of the severity of the change. At the same time, however, this approach is extremely language dependent and requires the construction of abstract syntax trees before the measure can be calculated.

While a severity analysis can be performed solely based on artifacts in the CM repository, the change impact analysis is dependent on the artifacts that a developer has in their workspace. The measures that we intend to explore, therefore, are different from those used in the severity analysis.

- **Relative number of overlapping lines of code that have changed.** A crude measure of impact, this measure basically attempt to assess the impact by calculating the number of lines of code that have changed in the artifact and are present in the workspace of a developer. The more overlap exists, the more impact the change is likely to have.

- **Relative number of interfaces that have changed.** This measure is based on the assumption that changes that do not change the interface of an artifact have no further influence beyond the realm of the artifact itself. While certainly accurate if the information hiding principle is strictly followed, the measure certainly does not apply to all software artifacts.

- **Relative size of dependency analysis graph.** This measure adapts established dependence analysis techniques [7] to basically calculate the “reach” of a change (e.g., how much of the code in the workspace



Artifact Name : pi.c
Artifact Version : 1.1
Date Checked out : 01/12/2001
Comment : Changing Interface 'A'
Artifact State: Checked out
Author Name: Andre
Author Email: andre@ics.uci.edu
Author URL: www.ics.uci.edu/~andre

Figure 2. Palantír Main Visualization.

might be affected). The larger the reach, the higher the impact. A drawback of this method is that computationally it can be very expensive.

It should be noted that none of the severity or change impact measures is completely accurate. Rather, most provide a conservative estimate of severity or impact. This is by design since it is neither necessary nor possible to provide completely accurate severity or change impact analyses. Nonetheless, it is useful for any developer to have a "feel" for the severity or impact of a change, allowing them to make a judgment call and undertake preventive action as they see fit.

It should also be noted that these algorithms should not only operate on single artifacts, but also have to be adapted to operate on compound artifacts. Many a time, Palantír will show a set of compound rather than individual artifacts. The severity and change impact of these compound artifacts has to be communicated to the developer as much as those of individual artifacts. In fact, often Palantír will be used in a setting in which a developer monitors several compound artifacts and only when the severity or change impact of one of those rises above a certain threshold will the developer examine the individual artifacts that make up that compound artifact to understand the reason for the high severity or change impact.

The last component in the Palantír architecture is the visualization component. This component is responsible for presenting to the developers, the information that is generated by the CM system and the analyzers. At a minimum, the visualization component will show which artifacts are being checked-out and checked-in and the severity and impact of each change. In doing so, developers are presented with an increased level of awareness of other developers' activities.

We intend to develop different visualizations that each strikes a different balance among the amount of information that is displayed, usability of the interface, and intrusiveness of the interface. Clearly, large graphical displays can convey more information, but are much more intrusive than, for example, a small ticker tape. The ticker tape, on the other hand, can only convey a minimal amount of information. We intend to develop a range of visualizations from which each developer can choose the one they prefer.

3. Implementation Details

We are currently implementing Palantír. Our focus, thus far, has been on creating one of the main visualization components. Shown in Figure 2, the visualization presents a developer with a hierarchical view of a compound artifact and its constituent artifacts. Each constituent artifact itself may consist of other artifacts, and each artifact in the view may consist of multiple versions (as indicated by a stack of artifacts). For example, in the figure, developer Anita is viewing the artifact "par.c" version 1.0, which consists of the artifacts "foo.c" and "bar.c". Since Anita started monitoring, the artifact "bar.c" has been checked out by Andre (as indicated by the question mark). The artifact "foo.c" version 1.0 has been checked in by Anita (as indicated by the exclamation mark), version 1.1 is currently checked out by Anita, and version 2.0 has been checked in by Andre. The artifacts are labeled such that changes made by other developers are discernable from those made by the user of the workspace. Color coding (green for changes made by the user of the workspace and red for others) further enhances this separation (though not visible in this black and white print).

The two vertical bars indicate severity and change impact. Note that atomic artifacts (artifacts that are not partitioned into smaller artifacts) that have not been checked in yet do not have any severity or change impact, since the potential changes are still "hidden" in the workspace.[1] Compound artifacts that are not checked in yet, on the other hand, will

[1] Typically, however, a developer will periodically create snapshots, which will lead to new versions that exhibit severity and change impact.

have severity and change impact since their constituent artifacts already may have been changed and checked in.

The visualization allows for zooming in and zooming out. Double-clicking on a particular version of an artifact makes that the primary artifact being viewed. A back button allows a developer to go back up the hierarchy to view the artifacts from a higher point of view. Note also that more detailed information on an artifact can be requested and that "hidden" artifacts can be brought to the foreground with a simple click.

The next step of our implementation effort involves creating several of the severity and change impact components and linking those to the visualization component using the Siena event notification service [5]. We have chosen to use Siena, because it operates in a distributed setting and allows filtering of notifications. Specifically, a visualization component can register the artifacts they are interested in, and Siena will only deliver those events that are of pertinence to those artifacts, thereby optimizing much of the network traffic.

4. Related Work

Most configuration management systems ignore awareness altogether [2, 8]. At best, developers can query the CM system for new changes and synchronize their workspace with those changes. Other than that, a developer is completely isolated from the changes being performed by others. One exception is Coven [9], which requires developers to specify beforehand which artifacts they will be modifying. Unfortunately, only direct conflicts are avoided this way and, furthermore, developers usually do not know beforehand the complete set of artifacts they will be changing.

The area of software visualization has produced a number of visualizations, including matrix views [6], 3-D colored graphs [10], bar graphs [11], and ticker tapes [12], that provide insight in the way a software system evolves over time. As compared to Palantír, these systems tend to focus on project management rather than awareness among developers. As a result, the visualizations are not dynamic (e.g., they only provide a view of the system evolution at one particular moment in time) and focus on severity, not change impact.

Hill et al. [13] created an extension to the Zmacs editor that shows how often a particular section in a document has been read or modified. Similar to Palantír, this extension provides developers with an increased level of awareness. Unfortunately, the visualization only provides an indication of change severity, not change impact. Moreover, use of the system requires developers to know a-priori who is changing which artifact and, more importantly, requires continuous network connectivity, something that is simply not feasible in a geographically distributed setting. TUKAN [14], another collaborative editor that augments the editor's interface with an indication of changes being performed by other developers, suffers from similar drawbacks.

5. Conclusion

Palantír is a system that we are currently developing to bring project awareness to developers. Palantír is explicitly designed to operate in a distributed setting and is aimed at reducing the number of occurrences of both direct and indirect conflicts. As such, it represents a departure from current philosophy in configuration management in breaking the traditional notion of isolated workspaces. Although changes can still be made in isolation and developers are not immediately influenced by other changes, they most certainly must be aware of the changes in order to avoid problems when they check in their artifacts. Especially in a geographically distributed setting, such awareness allows them to avoid and resolve conflicts early, which has the potential of saving a tremendous amount of cost and effort.

Clearly, 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 a case study involving an Open Source project to understand the effectiveness of a tool like Palantír in supporting distributed software development. We specifically want to determine which analyses and visualizations are most effective in which situations. In addition, we would like to explore the potential role of Palantír in project management. For that purpose, we are looking into the possibility of extending Palantír with a graphical view of all artifacts (rather than the small subset it currently supports) and using a movie-like capability in which we replay events from a CM archive to visually show project progress.

6. Acknowledgements

This research is supported by the National Science Foundation under grant number CCR-0093489. The authors would like to thank David Redmiles and Paul Dourish for their valuable input that shaped the project in its early phases.

7. References

[1] Easterbrook, S.M., et al., *A Survey of Empirical Studies,* in *S. M. Easterbrook (ed) CSCW: Cooperation or Conflict?* 1993: London,Springer-Verlag. p. pp. 1-68.

[2] Conradi, R. and B. Westfechtel, *Version Models for Software Configuration Management.* ACM Computing Surveys, 1998. 30(2): p. 232-282.

[3] Perry, D.E., H.P. Siy, and L.G. Votta. *Parallel Changes in Large Scale Software Development: An Observational Case Study*. in *Proceedings of the 1998 International Conference on Software Engineering*. 1998.

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

[5] 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.

[6] 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.

[7] 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.

[8] Burrows, C. and I. Wesley, *Ovum Evaluates Configuration Management*. 1998, Burlington, Massachusetts: Ovum Ltd.

[9] Chu-Carroll, M.C. *Supporting Distributed Collaboration through Multidimensional Software Configuration Management*. in *Proceedings of the Tenth International Workshop on Software Configuration Management*. 2001.

[10] 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.

[11] Baker, M.J., and Eick,S.G., *Space-Filling Software Visualization. Journal of Visual Languages and Computing*, 1995. 6: p. 119-133.

[12] 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.

[13] Hill, W.C., Hollan, J. D., Wroblewski, D., and McCandless,. *Edit wear and read wear*. in *Proceedings of the 1992 ACM Conference on Human Factors in Computer Systems.* 1992.

[14] 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.