Workspace Awareness in Application Development

Roger M. Ripley, Ryan Y. Yasui, Anita Sarma, André van der Hoek
University of California, Irvine
Donald Bren School of Information and Computer Sciences
Department of Informatics
Irvine, CA 92697-3425 USA
{rripley, ryasui, asarma, andre}@ics.uci.edu

Abstract

Coordination of development activities is often the most crucial success factor in a software development team. Typically, teams rely on con guration management (CM) systems for coordination purposes. CM systems manage concurrent development by isolating workspaces. As a result of this isolation, developers are aware of concurrent changes only when they interact with the repository and when changes have already been completed. Breaking this isolation would enable developers to detect potential con icts as they occur in the workspaces and proactively take steps to avoid them. Our Eclipse plug-in, Palant´r, supports such a workstyle by showing which artifacts have been changed by which developers and by how much.

1 Introduction

Typical software development is a multi-team effort with con guration management (CM) systems being the most commonly used coordination tool. Almost all modern development environments are integrated with at least one CM system, thereby minimizing the learning curve required for users to take advantage of CM functionality.

CM systems provide coordination support by isolating developers in their workspaces. By design, developers are unaware of parallel development and only become aware of the existence of con icts when they either commit or synchronize with the repository. This pull-based architecture only affords for con icts to be detected when changes are already completed—often too late.

OOPSLA'04 Eclipse Technology eXchange (ETX) Workshop, Oct. 24-28, 2004, Vancouver, British Columbia, Canada.

Existing automated merge tools provide limited merge support, failing when changes overlap, requiring lengthy and tedious manual con ict resolution. Palant´r is based on the hypothesis that providing workspace awareness to users will enable them to detect potential con icts earlier, as they occur. Ideally developers can then proactively coordinate their actions to avoid those con icts. Palant´r speci cally shows which developers are changing which artifacts by how much [4].

In order to be effective, Palant´r must provide workspace awareness such that developers can monitor parallel workspace activity while working on their current task and *without having to mentally switch contexts*. Integrating Palant´r with an IDE was the obvious solution. We built a plug-in for Eclipse as it is a widely-used open-source IDE with a very high adoption rate. Its elegant plug-in architecture promotes the creation and deployment of plug-ins. Creating a plug-in for Eclipse gives us two bene ts: 1) the large community that uses Eclipse allows us to deploy Palant´r to selected groups as a test bed to evaluate its effectiveness; and 2) the plug-in would be an excellent tool to teach students collaborative development.

The rest of the paper is organized as follows. Section 2 gives a brief description of Palant´r. In Section 3, we discuss our implementation of Palant´r as a plug-in. Next, we discuss our experience in creating the plug-in in Section 4, concluding in Section 5 with an outlook at future work.

2 Palant´r

Palant´r provides workspace awareness by building on top of existing CM facilities and concentrates on the collection, distribution, organization, and

presentation of relevant workspace information. Palant´r's architecture gives it independence from the underlying CM system and presents only relevant information to avoid cognition overload [4].

*Workspace Wrapper*s collect and translate CM-speci c activities to Palant´r events, since different CM systems have different interfaces. We have identi ed a set of eleven events that encompass typical CM-related activities, such as check out (POPULATED), edit (CHANGESINPROGRESS), and commit (CHANGESCOMMITTED) [4]. Separating the speci c CM system commands from Palant´r events allows for easily plugging in different CM systems, since the only requirement is the creation of a new *Workspace Wrapper*.

Informing a developer of all parallel activities in all workspaces can potentially be overwhelming, thereby being more detrimental than helpful. It, in fact, is also unnecessary. We leverage the event l-tering mechanism of *SIENA* (a generic event noti- cation service) [1] to inform developers only of relevant activities in other workspaces, that is, all activities in other workspaces relating to the artifacts in the local workspace.

Developers may wish to further reduce the information presented to them. Most of the time, a developer would be interested in parallel activity on certain artifacts or by certain developers. Each of Palant´r's visualizations presents different amounts of information in different formats. Developers can choose the visualization that best meets their preferences, or choose to use more than one in parallel. The visualizations can also be con gured to extract and present a subset of desired events from the entire set. The visualizations are discussed in detail in our previous work [5].

The visualization components also display the severity (magnitude) of changes in other workspaces. This allows developers to quickly gauge the severity of parallel work and decide whether to take immediate action or to defer. Currently, the severity of changes is calculated as the percentage of lines of code that have changed from the checked out version.

3 Implementation

Although the architectures of Palant´r and Eclipse are complex, integration of Palant´r into Eclipse was relatively straightforward, but with some caveats. In this section we talk about how the existing system was modi ed to be used as a plug-in. The *Workspace Wrapper* was modi ed to intercept the activities of the Eclipse CVS repository and local changes in the workspace. A new visualization was created for Eclipse and the existing visualizations were integrated. Additionally, existing Eclipse facilities were leveraged to add functionality to Palant´r.

3.1 Workspace Wrapper Integration

The starting point of our implementation was modifying the *Workspace Wrapper*. The modi cations to the *Workspace Wrapper* required three substeps. The rst sub-step was to start Palant´r from the Eclipse workbench with the appropriate artifacts. In the standalone version of Palant´r, Palant´r was invoked when the developer rst checked out les into their workspace and killed when the CM client was closed. There was no persistent storage of the event log and a fresh check out was necessary on restarting the CM client. In most development environments however, including Eclipse, developers do not check out artifacts frequently. Instead, once they check out the project into their workspace they enter into a cycle of local modi- cations, updates and commits.

In order to address this issue, Palant´r was modi ed to start with the active les in the workbench that were under version control. If there were no active les under version control, then a check out would invoke Palant´r. An event database persistently stores all the CM-related actions as events.

The second sub-step addressed the issue of intercepting the activities pertaining to the CM repository. In the standalone implementation, the *Workspace Wrapper* wrapped the command line of the CM client, reading the input and output to and from the repository. Thus, to integrate Palant´r with a speci c CM system all that was needed was a new *Workspace Wrapper*. Eclipse, however, does not use a command line interface to the repository and all CM-related functionalities are provided by a Team plug-in. This plug-in provides access to the repository and all relevant commands through the Eclipse workbench. Different CM systems extend this Team plug-in to create their CM clients.

Even though a generic Team API was slated for Eclipse 3.0, using the Workspace Versioning and Con guration Management (WVCM) API being developed in JSR-147, a decision was made to postpone that as no reference implementation was yet available [2]. Since Eclipse does not provide a

generic team/repository API, we could not make an extension to the Team plug-in to emit Palant´r related events. Modifying the source code of the speci c CM plug-in was clearly not desirable. We therefore used the set of event listeners provided by and speci c to the particular CM plug-in that trigger upon activity with the repository. Since the event listeners are not generic but speci c to a particular CM plug-in, different integrations would require creating implementations for different sets of event listeners.

Finally, the last sub-step of the *Workspace Wrapper* modi cations was to include facilities for detecting local edits in the workspace, in addition to publishing the standard Palant´r events. The previous version of Palant´r would generate events only on CM-related actions. Developers would therefore remain unaware of changes taking place in remote workspaces until the artifacts were checked back into the repository. In the case of lengthy check out/check in cycles, Palant´r would fail to provide an up-to-date picture of the changes taking place in remote workspaces.

To address this problem, a *Workspace Monitor* was created. The *Workspace Monitor* implements the Eclipse workspace resource change event listener (`IResourceChangeListener`) and intercepts the local "edits" and "saves" in the workspace. The monitor immediately publishes an event when a resource is rst edited ("dirtied"), but queues and consolidates resource changes, only publishing events related to those activities every thirty seconds to prevent storms of events during heavy artifact editing. The *Workspace Monitor* is thus responsible for providing noti cations when a developer starts to edit an artifact and calculating the severity of the changes made during editing.

Figure 1 illustrates the new architecture of the Palant´r/Eclipse integration. The arrows represent the ow of information. All the components of the Palant´r architecture and the CM system are intact. However, the interface between Palant´r and the CM system has changed to Eclipse components. The *Workspace Wrapper* no longer wraps the CM client as it did in the standalone architecture [4].

3.2 Visualization Integration

The next step in creating the Eclipse plug-in was to integrate the set of Palant´r visualizations that are responsible for displaying information about parallel activity. Unfortunately, integrating the visualizations into the Eclipse IDE was not a trivial effort. The user interface components that Eclipse and Palant´r used were mismatched as they use different toolkits for their widgets: the Eclipse UI is implemented in SWT, while the Palant´r visualizations were implemented in Swing/AWT. Moreover, at that time, there was no direct translation between the two toolkits or robust tools that supported migration between the toolkits. Integrating the visualization components into Eclipse thus needed modi cations to the source code, which was clearly not a desirable option.

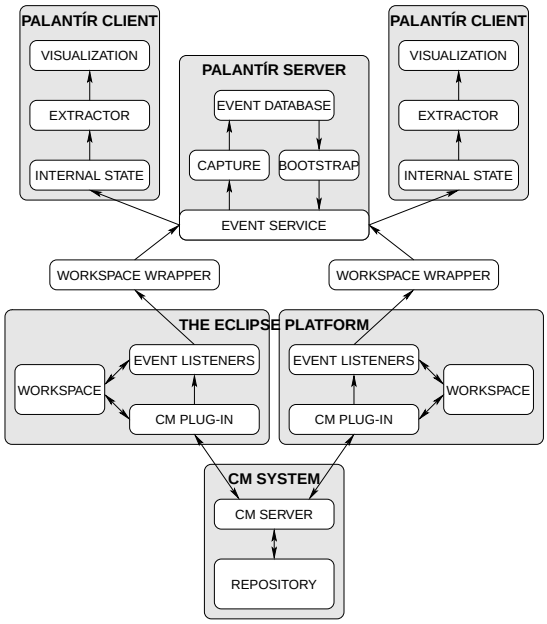
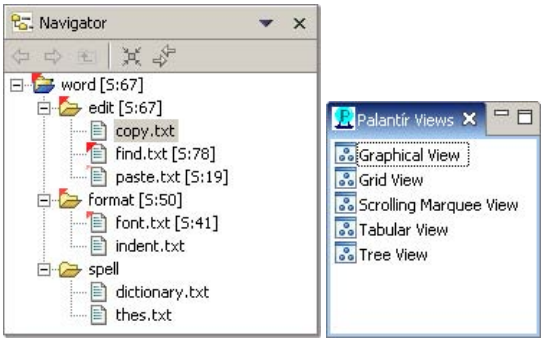


Figure 1. Palant´r/Eclipse Integration Architecture

We created a short term solution by creating an Eclipse view that listed the set of visualizations, as shown in Figure 2(b). When activated from the Eclipse view the Palant´r visualizations open outside the Eclipse workbench in a separate window. The release version of Eclipse 3.0 allows for integration of Swing/AWT components within SWT; we intend to leverage this functionality to integrate the visualizations directly into Eclipse.

To be effective, workspace awareness should be provided as peripheral information as developers do not want to be distracted while they work [3]. In our current integration, there was a de nite mental context switch required to view the Palant´r visualizations. A new Eclipse visualization was therefore created to complement the existing set such that the developer could monitor activities in the workspace within the IDE.



(a) Severity Triangles (b) Views View

Figure 2. Palant´r Plug-In Screenshots

We extended the *Navigator* and *Package Explorer* views in Eclipse by annotating the resources in the view with decorators. These decorators are in the form of graphical triangles and numeric text, as shown in Figure 2(a). When artifacts are changed in remote workspaces, the severity (magnitude) of the change is calculated by Palant´r. The triangles in the view (red in color, but not seen in the black and white picture) depict the severity of the change. As the severity of the changes increases so does the size and darkness of the triangles; the triangles become redder and bigger as the severity increases. Because we cannot display multiple (say 10–20) triangles at once for one artifact, only the average value across all workspaces in which an artifact is modi ed (not including the local workspace) is shown. Thus by monitoring the size and color of the triangles, the developer can easily gauge concurrent activities in others' workspaces at a glance. The numeric severity is shown textually to the right of the resource name as well. To investigate a particular change or activities in a remote workspace, the developer can refer to the other, more detailed, Palant´r visualizations [5].

3.3 Directory Severity

One of the key features of Palant´r is that it does not just display which artifacts are being changed by which developers, but also calculates the severity of those changes. In order to get an overall picture of the state of the workspace, the individual severities of the artifacts have to be communicated up the directory tree. Directory severity is thus a compilation of the severities of the child artifacts contained within a directory. In our previous version, the directory severity was calculated as $\frac{\#\ of\ artifacts\ with\ severity > 0}{total\ \#\ of\ artifacts}$.

Unfortunately this measure does not yield satisfactory results for Java artifacts as Java specifies that each component of a package name corresponds to a directory. Thus, for package `edu.uci.isr.palantir`, the directory structure would be `/edu/uci/isr/palantir`. If this Java package contained two artifacts, of which one was modified, the severity of the `palantir` directory would be $1/2$. This presents an accurate description of the directory state. However, the parent of `palantir`, `isr`, would have a severity of $1/1$ as its only child artifact, `palantir`, has changed and this pattern will continue up the directory tree. Thus, once the severity becomes $1/1$, it degenerates into a flag indicating that something has changed somewhere down the directory structure. While in and of itself not necessarily bad, we believe we can do better: display a flag that indicates the severity of the changes inside the directory.

We created a new directory severity measure, such that it will notify the developer of any significant changes occurring in an artifact contained within the directory, while simultaneously correlating to the severities of those artifacts. This new directory severity is calculated as $\frac{\sum Actual\ artifact\ severities}{\sum Possible\ artifact\ severities}$. In this measure the directory severity corresponds to the contained severities as every artifact has an equal weight.

Returning to our Java example, the severity of the `palantir` directory would still be $1/2$. But the parent of `palantir`, `isr`, would now have a severity of $1/2$. This severity propagates up the directory tree. The directory severity measure now not only shows that there have been changes, but also gives a measure of the magnitude of those changes.

4 Reflections

Since most developers depend on a development environment, we decided to integrate Palant´r with an IDE. This integration avoids the mental context switch that the developer would have to make otherwise. Eclipse was the ideal choice as it provides advanced features and has a wide user base. As Eclipse is plug-in based, creating a plug-in for Palant´r was relatively straight forward. Eclipse provides a set of event listeners that can be tapped into to intercept CM and workspace activities.

By creating our plug-in for Eclipse we were able to fulfill a number of objectives: 1) Eclipse supports different types of CM clients as plug-ins so we can easily create Palant´r integrations for the

different CM clients; 2) The Eclipse Navigator and Package Explorer are well-understood and widely-used features. Adding decorators to these views helped us create the most simple and yet effective peripheral workspace awareness widget; 3) Since Eclipse is open source it has a strong community that is open to trying out new ideas. We thus have a much higher chance of nding teams willing to try Palant´r; 4) Further enhancements are planned for Palant´r, such as change impact and additional visualizations. Since Eclipse allows for incremental development of features that can be easily deployed via an Eclipse update site, it will be easy to deploy those new features as they become available; and nally 5) Eclipse is now being widely used by the student community as the IDE of choice. The Palant´r plug-in can be used teach students about collaborative development.

While creating a plug-in for Eclipse has been, for the most part, straightforward and successful, we would also like to bring to attention some of the problems that we faced while creating the plug-in. We were surprised to nd that there was no generic repository API and that each CM plug-in (e.g., CVS and Subversion) had a different API. In order to integrate with a different CM client we will have to create an entirely new wrapper for it.

The decorator facility for Eclipse has certain restrictions: only unformatted text can be added to the label (we would have liked to add a graphical bar that would grow and shrink, or been able to change the color). This severely constrained the amount and types of information we could provide.

Finally, before Eclipse 3.0 there was no reliable way to integrate Swing/AWT code within SWT, so in order to integrate the Palant´r visualizations they would have to be rewritten using the SWT toolkit.

5 Conclusion and Future Work

Palant´r is based on the hypothesis that enhancing CM systems with awareness allows developers to have an improved insight into potentially con icting parallel activities. The hope is that developers will use this insight to self-coordinate, in effect detecting con icts earlier, thereby reducing the amount of effort involved in addressing them. We have created an Eclipse plug-in for Palant´r so that developers do not have to switch contexts between their IDE and Palant´r.

There are several avenues we are pursuing with future Palant´r development to take advantage of Eclipse. Even though Palant´r informs developers which artifacts are being changed by which developer and the severity (magnitude) of the changes, the severity of a set of changes is inherently of limited utility; we need to calculate what the actual impact of those changes will be on a developer's current work. We plan to use Eclipse's APIs for parsing and analyzing Java source code to add a change impact metric to Palant´r. A possible impact metric could be the percentage of method signatures that have changed in a le, even tempering that with how often that method is referred to in the workspace. We also intend to integrate Palant´r with the Eclipse Subversion client, Subclipse.

Acknowledgments

Effort partially funded by the National Science Foundation, grant numbers CCR-0093489 and IIS-0205724, and an Eclipse Innovation Grant, 2004.

About the Authors

André van der Hoek is an assistant professor at UC Irvine, Anita Sarma is a Ph.D. student under his supervision, and Ryan Yasui and Roger Ripley were Masters students at UC Irvine who were the primary authors of the work presented here.

References

[1] Antonio Carzaniga, David S. Rosenblum, and Alexander L. Wolf. Design and evaluation of a wide-area event noti cation service. *ACM Transactions on Computer Systems*, 19(3):332–383, August 2001.

[2] Eclipse Foundation. Eclipse 3.0—Team API Plan Item. http://dev.eclipse.org/viewcvs/index.cgi/~checkout~/platform-vcm-home/docs/online/team3.0/team_api.html, 2003.

[3] Christian Heath and Paul Luff. Collaborative activity and technological design: Task coordination in London underground control rooms. In *Proceedings of the Second European Conference on Computer-Supported Cooperative Work*, September 1991.

[4] Anita Sarma, Zahra Noroozi, and André van der Hoek. Palant´r: Raising awareness among con guration management workspaces. In *Proceedings of the Twenty-Fifth International Conference on Software Engineering*, pages 444–454, May 2003.

[5] Anita Sarma and André van der Hoek. Visualizing parallel workspace activities. In *Proceedings of IASTED International Conference on Software Engineering and Applications (SEA 2003)*, pages 435–440, November 2003.