Palantír: Raising Awareness among Configuration Management Workspaces

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

Abstract

Current configuration management systems promote workspaces that isolate developers from each other. This isolation is both good and bad. It is good, because developers make their changes without any interference from changes made concurrently by other developers. It is bad, because not knowing which artifacts are changing in parallel regularly leads to problems when changes are promoted from workspaces into a central configuration management repository. Overcoming the bad isolation, while retaining the good isolation, is a matter of raising awareness among developers, an issue traditionally ignored by the discipline of configuration management. To fill this void, we have developed Palantír, a novel workspace awareness tool that complements existing configuration management systems by providing developers with insight into other workspaces. In particular, the tool informs a developer of which other developers change which other artifacts, calculates a simple measure of severity of those changes, and graphically displays the information in a configurable and generally non-obtrusive manner. To illustrate the use of Palantír, we demonstrate how it integrates with two representative configuration management systems.

1. Introduction

One of the core functions of any configuration management system is to coordinate access to a common set of artifacts by multiple developers who are all working on the same project. While ideally project management assigns the developers tasks that are mutually exclusive, the reality is that changes made by one developer regularly affect another's work [18,19,28].

The various approaches that different configuration management systems take to address this situation can be distinguished into two classes: pessimistic and optimistic. In the pessimistic approach, a developer must lock artifacts before making any modifications. Such a lock prevents other developers from making concurrent modifications, and in essence serializes the set of changes to the artifacts. In the optimistic approach, multiple developers can change the same artifacts at the same time (perhaps on branches [34], or alternatively using independent change sets [36]). Conflicts may then arise, but semi-automated differencing and merging tools help in identifying and resolving them (albeit only a subset [23]).

Both approaches rely on workspaces to partition the work of developers. While these workspaces are essential to shield developers from the effects of other changes in other workspaces (good isolation), they have the unfortunate side effect of creating a barrier that prevents developers from knowing which other developers change which other artifacts in parallel (bad isolation). In a typical configuration management system, a developer becomes aware of other' activities only at three specific points in time: when they obtain artifacts from the repository and place them in their workspace; when they put modified artifacts back from their workspace into the repository; and when they explicitly query the repository. Even then, in the pessimistic approach they only know what artifacts other developers have locked for future changes and in the optimistic approach they only become aware of what changes already have occurred. From a coordination point of view, this is an undesirable and limited situation.

To alleviate this situation, we have built Palantír, a novel configuration management workspace awareness tool that deliberately breaks bad isolation while retaining good isolation. A crucial aspect of Palantír is that it inverts information flow from pull to push. Instead of informing developers of other efforts only when they themselves perform some configuration management operation (e.g., check in or check out), Palantír increases awareness by *continuously* sharing information regarding operations performed by all developers. Specifically, Palantír informs a developer of which other developers change which other artifacts, calculates a simple measure of severity of those changes, and graphically displays the information in a configurable and generally non-obtrusive manner.

Palantír architecturally separates specific workspace wrapper components from a generic visualization compo-

nent. Each workspace wrapper supports one particular configuration management system and intentionally has no further functionality than to emit events regarding the operations performed by different developers. The generic visualization component continuously collects and interprets the events, presenting a developer with an up-to-date picture of the workspace activities of others. Connecting the components is the Siena event notification service [7], whose distributed event filtering mechanism is leveraged to only deliver the necessary subset of events to each developer's visualization.

Constructing a system as Palantír raises many questions regarding, among others, which information should be shared, how to avoid overloading developers with information, scalability, and general effectiveness of the approach in helping developers coordinate their tasks. The aim of this paper is not to provide an absolute answer to all of the questions, but rather to serve as an initial investigation into the feasibility of increasing workspace awareness in configuration management systems. As such, the basic research question we seek to answer is whether Palantír can enhance existing configuration management systems with workspace awareness.

The remainder of this paper is organized as follows. In Section 2, we discuss background information regarding awareness in configuration management systems. Next, we discuss the overall approach and high-level architecture of Palantír in Section 3. Section 4 discusses the current implementation of Palantír. We demonstrate how Palantír can be integrated with two existing configuration management systems, namely RCS [32] and CVS [4], in Section 5. While old, these two systems represent the core of virtually every configuration management system, and cover both a pessimistic (RCS) and an optimistic (CVS) approach. We discuss related work in Section 6, and conclude in Section 7 with an outlook at future work.

2. Background

Awareness is characterized as "an understanding of the activities of others, which provides a context for your own activity" [13]. The kind of information needed to create awareness depends on the particular activity in which a group of persons is participating, but generally includes such information as who is part of the group, what tasks they perform, how active they are, what changes they make, and which objects they manipulate [20].

Awareness as a concept applies to many different activities, but within the discipline of computer science it is generally associated with the field of computer-supported collaborative work (CSCW). There, efforts have largely focused on the use of awareness for coordination purposes, for example in multi-user editors (e.g., MMM [5] and Suite [12]) or collaborative workspaces (e.g., BSCW [3] and TUKAN [29]). The basic underlying theory is that providing users with appropriate contextual information allows them to make much more sophisticated decisions in coordinating their individual and group activities than any automated approach ever could. A few studies on the use of awareness in particular domains confirm this hypothesis and discuss the benefits of awareness as a coordination mechanism [13,16,24].

At the heart of any configuration management system is the need to coordinate the changes made by different developers in different workspaces. It is therefore surprising that the use of awareness has not received much attention, especially considering the apparent success in other domains. In fact, the focus has largely been on eliminating the need for awareness altogether. The philosophy is that workspaces are sacred places in which a developer must be able to make their changes in complete isolation. Not only should the artifacts be shielded from potentially interfering changes made concurrently by other developers, developers themselves should not have to know who else makes changes, how active they are, which artifacts they modify, and so on [6,10,11].

Current configuration management systems provide extensive and automated support for maintaining this kind of total workspace isolation [10]. Unfortunately, current mechanisms are highly inadequate from a coordination point of view (see Table 1). In a pessimistic configuration management system, locking is used to coordinate activities and only one developer may change an artifact at a time. Conflicts in which two or more developers change the same artifact *(direct conflicts)* are avoided, but at the expense of project delays if one developer must access an artifact currently locked by another developer. Furthermore, conflicts in which changes by one developer to one artifact negatively affect changes by another developer to another artifact *(indirect conflicts)* cannot be avoided.

Optimistic configuration management systems support parallel work with either branches [34] or change sets [36], and in essence coordinate parallel activities via the use of merge tools that combine changes to an artifact by one developer with changes to the same artifact by another developer. Most merge tools can automatically resolve most direct conflicts, but unfortunately cannot handle overlapping changes, leading to a regular need for manual problem resolution. Indirect conflicts are not addressed.

Ideally, the above drawbacks of the pessimistic or optimistic approach never occur. Then, either approach will succeed in maintaining the image of workspace isolation by automatically coordinating the activities of individual developers. In reality, however, the ideal case cannot be enforced. More often than not the illusion of workspace isolation vanishes when complex direct or indirect conflicts arise that the automated procedures cannot handle [18,19,28].

Table 1. Different coordination mechanisms.

	Coordination mechanism	*Direct conflicts*	*Indirect conflicts*
Pessimistic	Locking before changes are made	Avoided, at the expense of project delays	Not addressed
Optimistic	Automated merging after changes are made	Resolved, except for overlapping changes	Not addressed

The root cause of why current configuration management systems perform poorly when it comes to coordination can be found in the following three observations:

- While coordination must be among workspace activities, current configuration management systems coordinate those activities based on information in the central repository only. Consequently, available information is restricted to which artifacts may potentially change (because they are locked) or which artifacts already have been changed by another developer (because they now must be merged).
- Coordination information is available to a developer only when they themselves: (1) attempt to lock an already locked artifact, (2) must merge an artifact, or (3) make an explicit request. As a consequence, information flow is irregular, limited to the particular artifact at hand, and typically out-of-date with respect to the actual state of the other workspaces.
- Coordination mechanisms focus on avoiding and resolving direct conflicts, but ignore indirect conflicts altogether. As a result, the conflicts that are perhaps the most difficult to discover and resolve remain elusive from a coordination point of view.

We are not alone in making the above observations and concluding that the use of awareness can make a difference in how configuration management systems are used in coordinating the activities of developers [18,24]. In fact, many developers already use some home-grown conventions that keep other developers up-to-date, for example via e-mails that are sent when some set of artifacts is checked out or checked in.

A few configuration management systems have started to include functionality for automatically creating awareness among developers (e.g., Coven [8], CVS [4], COOP/Orm [22]). Even some Open Source development portals, such as SourceCast [9] and SourceForge [30], provide some simple awareness mechanisms attached to their configuration management functionality. In general, however, these approaches have serious limitations in terms of what information is shared, when the information is shared, and how the information is presented to the developers (see Section 6). Perhaps most limiting is that all of the aforementioned systems inform developers only of direct conflicts concerning individual artifacts. An overall view of other developer's workspace activities is missing. Especially when compared with the successful awareness approaches developed in the field of CSCW, the potential for a principled, rich awareness mechanism that complements existing configuration management systems has not been realized as of yet.

3. Approach

To introduce awareness in current configuration management systems, we have developed Palantír, a novel configuration management workspace awareness tool that provides developers with insight into other workspaces. Palantír itself is not a configuration management system and does not provide any traditional configuration management functionality such as artifact storage, workspace management, differencing and merging, or locking. Instead, Palantír builds on top of existing configuration management facilities and concentrates on the *collection*, *distribution*, *organization*, and *presentation* of relevant workspace information.

The architecture of Palantír is shown in Figure 1. Arrows represent information flow. Grey ovals represent components traditionally found in configuration management systems; they are used unchanged. White ovals are Palantír components that incrementally implement its functionality. A WORKSPACE WRAPPER collects and subsequently emits relevant workspace events that the generic EVENT SERVICE distributes to other developers. The INTERNAL STATE receives and stores the events, which are extracted and organized by an EXTRACTOR before they are shown to a developer by a VISUALIZATION component.

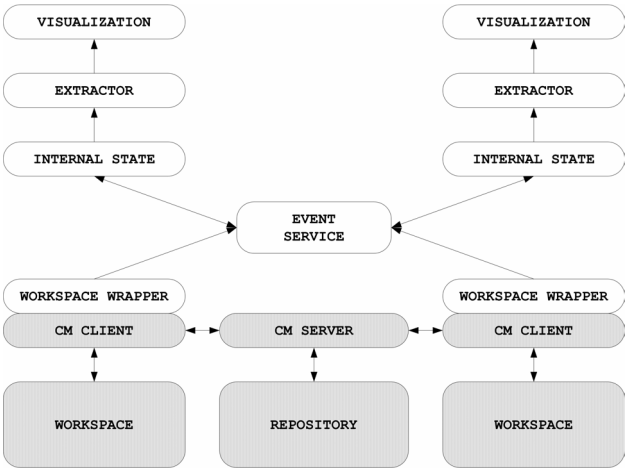

Figure 1. Palantír architecture.

The key observation underlying the Palantír architecture is that it inverts information flow from pull to push. Rather than informing developers of other efforts only when they themselves perform some configuration management operation, Palantír increases awareness by *continuously* sharing information regarding operations performed by all developers. As a result, Palantír not only frees developers from having to manually collect and interpret information from the configuration management repository, but in the process provides them with a more complete, accurate, and up-to-date picture concerning the activities in the other workspaces.

The architecture of Palantír is purposely constructed to address a number of concerns.

- **Non-obtrusiveness.** Developers should not have to change the way they interact with their particular configuration management system. Palantír, therefore, relies on simple workspace wrappers that, other than emitting events regarding the actions by the developers, have no further functionality. As discussed in Sections 4.2, these wrappers can normally be implemented with a minimal level of intrusiveness and without altering the state of a workspace.
- **Scalability.** Informing a developer of all activities in all workspaces overloads their cognitive senses and, in fact, is not necessary. Rather, Palantír uses the distributed event filtering mechanism provided by its event notification service of choice (Siena [7]) to only inform a developer of relevant activities in other workspaces. Relevant activities are defined as all activities pertaining to the artifacts in the local workspace, as performed in parallel by other developers in their remote workspaces (see Section 4.3).
- **Flexibility.** Not every configuration management system requires the same level of awareness. By separating internal data management from data extraction and information visualization, Palantír offers a flexible architecture in which different visualizations can be used. Currently, Palantír offers two such visualizations: a simple ticker tape similar to the one provided by Elvin [17] and a fully graphical visualization that presents a detailed overall view of the state of other workspaces. Other visualizations can easily be developed and added to the framework.
- **Configurability.** Not always will a developer want to be aware of all activities in all workspaces. Sometimes, it is desirable to select only a few developers or artifacts that one wants to monitor. The EXTRACTOR component of Palantír provides exactly this capability and can filter in a number of ways the events stored by the INTERNAL STATE component. Other selection mechanisms can easily be added.

4. Implementation

We have built a prototype implementation of Palantír on top of the generic Siena event notification service [7]. For each of the Palantír components identified in Figure 1, we highlight its design considerations and implementation details below. Given that Siena is used as an external service, we do not discuss its details. We do, however, illustrate how we leverage Siena for event filtering in order to achieve scalability within Palantír. We begin our discussion by introducing the events that Palantír uses internally.

4.1 Events

At the heart of Palantír are the events that describe the ongoing activities in each workspace. Of importance in the design of these events is the fact that Palantír must interoperate with different configuration management systems and cannot count on each of those systems to follow the same configuration management policy [33]. Rather than capturing actions (e.g., check in, check out, synchronize), events therefore represent particular *states* in which an artifact may be in a workspace. The resulting set of events is listed in Table 2, along with their interpretations and detailed data constituting each event.

Regardless of the particular configuration management system that is used, most artifacts will go through one of two cycles of events. Artifacts that change typically trigger the following sequence of events: (1) POPULATED, (2) CHANGESINPROGRESS, (3) CHANGESCOMMITTED, and (4) UNPOPULATED. Of course, the pair CHANGESINPROGRESS and CHANGESCOMMITTED may be repeated if a developer continues to make changes to the artifact before removing it from the workspace. Artifacts that must be present in a workspace for ancillary purposes trigger a simpler series of events: POPULATED followed by UNPOPULATED.

Exactly *when* these two event sequences are triggered depends on the particular configuration management system, its configuration management policy, and the specific wrapping mechanism that is used. Important, however, is that most, if not all, configuration management policies can be mapped onto the above two sequences of events in one way or another (see Section 5).

Most of the other events capture infrequent, but nonetheless critical workspace activities. The event CHANGESREVERTED captures the fact that a developer has undone some changes and reversed an artifact back to its initial, unchanged state in the workspace. The event SYNCHRONIZED is similar, but instead signifies that an artifact has been updated to reflect its latest state in the repository. Finally, the events ADDED, REMOVED, RENAMED, and MOVED capture the results of adding, removing, renaming and moving artifacts in the project hierarchy—all events that may indicate potential conflicts amongst workspaces.

Table 2. Palantír events.

Event	*Meaning*	*Data*
POPULATED	Artifact has been placed in a workspace	artifactID, parentArtifactID
UNPOPULATED	Artifact has been removed from a workspace	artifactID
SYNCHRONIZED	Artifact has been synchronized with repository	syncArtifactID, artifactID
CHANGESINPROGRESS	Artifact has changed in the workspace	wsArtifactID, artifactID, comment
CHANGESREVERTED	Artifact has been returned to its original state	artifactID, wsArtifactID
CHANGESCOMMITTED	New version of artifact has been stored in repository	newArtifactID, wsArtifactID, comment
ADDED	New artifact has been added	artifactID, parentArtifactID, comment
REMOVED	Artifact has been removed altogether	artifactID
RENAMED	Artifact has been renamed	artifactID, newName
MOVED	Artifact has been moved from one artifact to another	newArtifactID, newParentArtifactID, artifactID
SEVERITYCHANGED	Amount of change to an artifact has changed	artifactID, min, actual, max, explanation

The last event, SEVERITYCHANGED, has no direct relation to any particular configuration management operation. It is used by Palantír as a mechanism for communicating a measure of the amount of change that an artifact has undergone at a given point in time. Simply knowing which artifacts are changing in other workspaces is useful, but having an associated indication of severity conveys more information. Small changes typically will be easy to reconcile. Large changes may signify a regression towards potentially difficult conflicts in an eventual integration.

A trivial severity measure is binary and simply indicates whether or not any kind of change has occurred. A slightly more complicated measure of severity can be calculated by dividing the number of lines that has been added, removed, and changed by the total number of lines in an artifact. A domain-specific configuration management system could even institute a measure that depends on the type of artifact being managed (e.g., by tracking interface changes in a particular programming language).

Of note is that the event SEVERITYCHANGED only provides a way of communicating severity, but does not prescribe a particular severity measure. In fact, different configuration management systems may use different severity measures as implemented in their respective workspace wrapper. In general, it should be noted that one configuration management system should adopt one kind of severity measure throughout in order to provide a uniform view to all of its developers.

Palantír events must distinguish incarnations of the same artifact in different workspaces. Additionally, they must distinguish an artifact that is in a workspace in its original, repository-equivalent state from that same artifact in that same workspace after it has undergone some changes. At the same time, however, Palantír must be able to detect related artifacts in order to be able to identify potential conflicts. For these three reasons, Palantír defines artifact identifiers incrementally and uses different parts of the identifier for different purposes (e.g., organizing artifacts per workspace, sorting versions of artifacts, matching artifacts in different workspace to each other). The resulting structure of artifact identifiers is as follows:

object-id:name:version:[REP|WS]:author-id

The object identifier uniquely identifies an artifact in the configuration management repository and is typically assigned by the configuration management system. (If not, an equivalent can often be obtained by using the path to the version archive of the artifact in question.) Because the object identifier may not be human readable, and because it sometimes identifies an artifact as a whole rather than a specific version, the name and version of an artifact are an integral part of the Palantír artifact identifier. To distinguish an artifact that has changed from one that has not, the qualifiers WS and REP are used (WS indicating that the artifact's contents are unique to the workspace, REP indicating that the contents are still the same as the original version in the repository). Finally, to distinguish different workspaces, a unique author identifier is used as an integral part of the artifact identifier.

As an example, consider the following artifact identifier:

17608:write.c:1.1:WS:Ellen

It identifies version 1.1 of artifact 17608 in the configuration management repository as it has been materialized in the workspace of ELLEN under the name WRITE.C.

Currently, artifact identifiers as described above make two simplifying assumptions. First, they assume that each developer has a single workspace. Second, they assume that each artifact is placed in a workspace only once. These limitations are a result of the prototype nature of Palantír. They can be easily overcome by enhancing the artifact identifier slightly to make it unique per author‘s workspace and per copy of an artifact in a workspace.

4.2 Workspace wrapper

Workspaces and their access mechanisms differ per configuration management system. While most of the architecture of Palantír can be independent of that fact because it is driven by the generic events defined in the previous section, workspace wrappers must be specific to a configuration management system in order to translate its workspace conventions into Palantír events. To do so, a workspace wrapper should execute the following steps for each action performed by a developer: (1) intercept the action, (2) interpret the action, (3) determine whether the action is relevant to Palantír, (4) if relevant, gather appropriate information concerning the action, and (5) construct and emit an event.

Some of the more advanced configuration management systems, such as CM/Synergy [31] and ClearCase [1], provide integrated triggering mechanisms and associated scripting languages that ease the development of a workspace wrapper. Other systems do not have such facilities, and their command line interface must be wrapped with a separate program. Such a program should take the place of the original executable and operate by first invoking the original executable and then emitting events as necessary. In both wrapping techniques a slight delay in user response will be incurred due to the added complexity of constructing and emitting events. The interaction mechanism of the user with the configuration management system, however, remains the same and the workspace wrapper operates in a non-obtrusive way without altering the state of a workspace.

4.3 Internal state

The internal state component maintains an overview of the activities in both the local and remote workspaces that is independent of the particular visualization that a developer may use. This is necessary because a typical visualization can only show a subset of all workspace activities without cognitively overloading a developer. Nonetheless, the overall state of the workspaces must be maintained as a cache such that a developer can change the viewpoint of their visualization without having to wait for all necessary information to be obtained on demand.

In addition to maintaining the details of the various workspace activities, the internal state component serves a second critical role in Palantír. In particular, it is the component that is responsible for subscribing to relevant types of events and processing the individual events that it receives from Siena as a result of those subscriptions. Scalability is of a pertinent concern in the particular set of subscriptions that Palantír maintains. Rather than simply receiving every event concerning every artifact, Palantír leverages the structure of artifact identifiers to only subscribe to events regarding artifacts that are in the current workspace. Some indirect conflicts may be missed as a result of this policy. The benefit of achieving a first level of scalability in terms of the number of events that must be handled, combined with the benefit of not overloading a user with information regarding all artifacts in all workspaces, however, clearly outweighs this issue. Especially since studies have shown that workspaces generally overlap only partially rather than completely [28], we can expect significant savings in the number of events that must be handled by each internal state component.

An additional benefit of this approach is that Siena internally uses the subscriptions to optimize event routing among its distributed event servers [7]. While Siena's servers already can handle a significant number of events per second, this optimization improves their overall performance even further.

A timing problem arises when two developers each populate their workspace with some of the same artifacts. One developer will be first, and their internal state component will subscribe to the relevant events and receive notifications as the other developer populates their workspace. The developer that is second, however, will not be aware of the activities of the first developer since its internal state component creates its subscriptions after the relevant notifications were sent. To address this issue, Palantír has a small set of internal bootstrap events that it uses to synchronize new workspaces with the state of other, previously existing workspaces.

As a consequence of its central role in processing events and sharing bootstrap information with other workspaces, the internal state component should always be executing for every workspace. While a developer may choose not to run any visualizations, they must always start the internal state component such that at least other developers can benefit from Palantír and become aware of potential conflicts as they arise.

4.4 Extractor

While the internal state component of Palantír provides a first cut at automatically selecting the events of interest (namely those performed in other workspaces but pertaining to the artifacts in the local workspace), users often can narrow these down even further. Guided by their experience over time, they may for example decide that the only events of interest are those with a severity measure of fifty perfect or higher. Alternatively, they may wish to only monitor a few select workspaces, since their knowledge of project context may help them in selecting only those developers who are working on closely related tasks that have a high chance of interfering.

Specifically for this purpose, the Palantír architecture contains an extractor component that, based upon a set of developer preferences, selects a subset of all events to be

visualized. Shown in Figure 2, this component allows a developer to select which types of events they want to know about, by which authors, over which time span, and with what minimal severity. In this particular case, a developer is interested in a number of different events, by Ellen only, over the past 5 minutes, with a severity of at least fifty percent.

4.5 Visualization

The last component in the architecture of Palantír is the visualization component. It is responsible for organizing and displaying the activities as they happen in the various workspaces. Thus far, we have built two visualizations, both of which can be used in parallel. The first is a simple ticker tape similar to the ticker tape of Elvin [17]. Shown in Figure 2, it scrolls through the set of events as selected by the extractor component. While limited in displaying one event at a time, it serves an important role as a non-obtrusive alert mechanism. For example, Figure 2 shows a ninety percent severity for a CHANGESCOMMITTED event, which almost certainly necessitates further investigation.

Note that the order in which events appear in the ticker tape can be sorted according to event, author, or severity.

Shown also in Figure 2, the fully graphical visualization complements the ticker tape with a mechanism to maintain an overall view of workspace activities. Instead of focusing on one event at a time, the fully graphical visualization organizes and colors the events such that they highlight potential conflicts among workspaces. For example, the first stack highlights that, in addition to the workspace owner, two other developers are in the process of making changes to the artifact EDIT.

The visualization is hierarchical, and supports browsing of the artifacts much like a web browser. Severity measures are displayed as change bars, and can be used to help localize the source of a potential conflict. For example, if a high-level artifact shows a high severity value, a developer can double click on the artifact to see and examine severity values of its constituents; one or more of those values is likely to be high and causing the conflict.

Several other features of the fully graphical visualization are of note. First, it shows pair-wise conflicts among developers. Second, it sorts the individual artifacts per severity, such that the artifacts with the highest severity appear first in the window. Both these features make it easier to recognize and localize potential conflicts.

As a final note, we observe that the fully graphical visualization helps in detecting potential indirect conflicts. It highlights all relevant workspaces activities with associated severity values, not just those for direct conflicts.

5. Two Example Integrations

To evaluate the effort involved in enhancing an existing configuration management system with Palantír, we implemented two integrations: one with RCS and one with CVS. While old, these two systems represent the core of virtually every configuration management system. Moreover, they are examples of a pessimistic (RCS) and an optimistic (CVS) approach. We discuss each integration below, and conclude the section with some observations regarding other potential integrations.

5.1 RCS

RCS [32] is a simple configuration management system that, while supporting branching, is often used with locking only. Our particular integration is geared towards that (pessimistic) mode of operation.

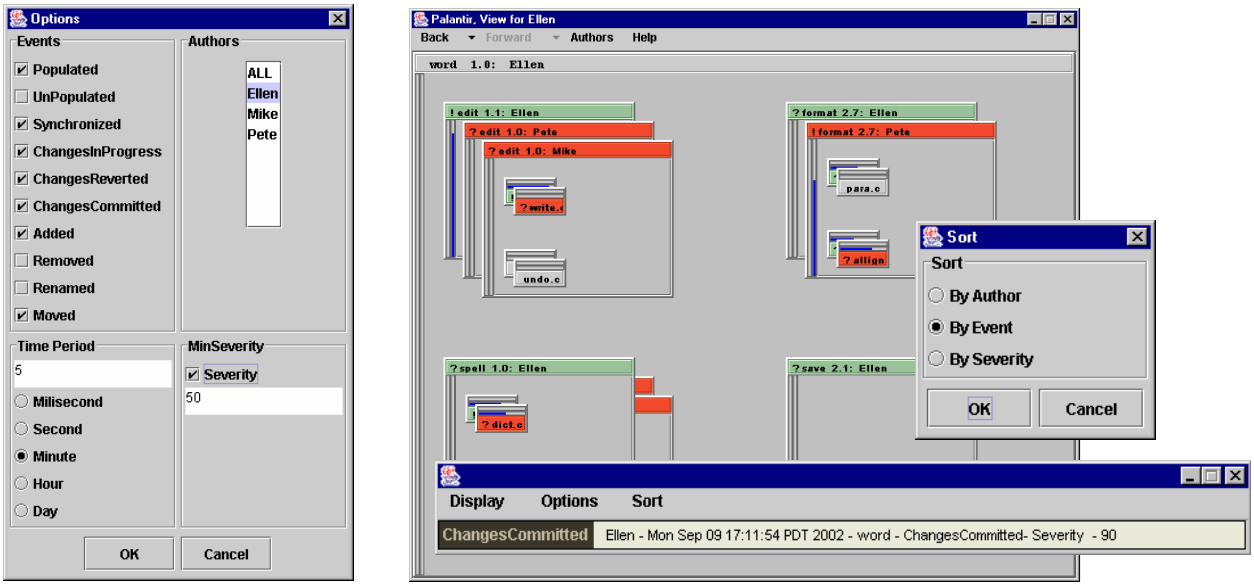

Figure 2. Extractor ("Options" window), ticker tape visualization, and fully graphical visualization.

We were able to develop a workspace wrapper for RCS within the span of a single day. The wrapper constitutes about 500 lines of Java code, and largely concerns wrapping the RCS executable with our own executable that invokes RCS, calculates a severity measure, and emits corresponding events. Individual RCS commands are mapped, per artifact, onto Palantír events as follows.

check out: POPULATED
check-out + lock: [POPULATED +] CHANGESINPROGRESS
uncheck-out: CHANGESREVERTED
check-in: SEVERITYCHANGED + CHANGESCOMMITTED

The end result of this mapping is that developers become aware of which other developers are modifying which artifacts, but until the artifacts are checked in, they do not know the actual severity of the changes. This drawback could easily be overcome by building a simple workspace daemon that periodically calculates and emits the severity of each artifact currently locked.

The severity measure used in our RCS integration is the same as we use for the CVS integration. Using the output of the DIFF program, we calculate the percentage of lines that has been added, removed, and modified.

Compared to just RCS, its integration with Palantír has the advantage that developers are aware of which other artifacts are being modified in parallel, know at all times which new versions of artifacts are available, and have at least the opportunity to detect potential indirect conflicts.

5.2 CVS

CVS [4] is an optimistic configuration management system that allows parallel work. The integration of Palantír with CVS was also performed in a single day, and again constitutes about 500 lines of Java code. The particular mapping of commands to events, however, is slightly different (once again, per artifact).

check out: POPULATED
edit: CHANGESINPROGRESS
update: SYNCHRONIZED
commit: [CHANGESINPROGRESS +] SEVERITYCHANGED + CHANGESCOMMITTED

In addition to these commands, we also wrapped the addition and removal of artifacts. This represents a particularly interesting event since additions and removals are local to a workspace until the parent artifact is checked in. This represents a clear case in which Palantír shares workspace information that otherwise would not be known to others.

A single CVS command recursively operates on all artifacts in a workspace, and thus, can emit multiple events. Of particular interest is CVS EDIT, which announces to CVS that an artifact will be modified (see Section 6.1). It is not mandatory that this command is used before an artifact is changed and committed. In such cases, a commit should first result in the emission of a CHANGESINPROGRESS and awareness in effect is delayed. Again, this could be overcome by building a simple daemon.

Compared to just CVS, its integration with Palantír has the advantage that developers are more rapidly aware of potential conflicts, know of planned artifact additions and removals, and know at all times when new versions of artifacts are available and what their change severity is.

5.3 Other potential integrations

Integrating other configuration management systems may in fact be easier and more powerful than RCS or CVS. Newer configuration management systems provide trigger mechanisms and associated scripting languages in which to implement a workspace wrapper. One particularly interesting feature of some newer configuration management systems is their use of a virtual file system to implement workspaces. The virtual file system intercepts every read and write to an artifact. Palantír could leverage this mechanism to send out a SEVERITYCHANGED event every time an artifact is edited or modified. As compared to the RCS and CVS integrations, this would result in the most up-to-date workspace information possible.

6. Related work

Palantír builds upon work in the areas of configuration management and computer-support collaborative work. We discuss relevant contributions in each area below.

6.1 Configuration management

CVS [4] already provides what is perhaps the oldest awareness mechanism in the field of configuration management. Through the use of watches, developers specify which artifacts they want to monitor. Before changing an artifact, developers must announce their intent of doing so by invoking the CVS EDIT command. This triggers notifications to be send, usually via e-mail, to those developers who are watching the artifact in question. Compared to Palantír, CVS watches require explicit manual action, provide limited awareness information concerning direct conflicts only, and do not scale due to the use of e-mail as the presentation mechanism.

Coven [8] supports awareness through soft locks. Before changing any artifacts, developers place soft locks with associated messages on those artifacts. When another developer attempts to place a soft lock on an artifact that already has a soft lock, that developer is presented with the message attached to the lock. They then are given the option to abort, which has no further effect, or continue, which informs the other developers that parallel work is

now ensuing. Coven suffers drawbacks similar to CVS: it requires extra manual actions and it provides, via e-mail, limited information concerning direct conflicts only.

EPOS [27] and Adele [15] do not directly address awareness, but rather provide sophisticated mechanisms to coordinate artifact sharing among workspaces. EPOS supports four different policies that can be instituted among different pairs of workspaces: (1) all artifacts are shared immediately, (2) artifacts are pushed to other workspaces, (3) artifacts are pulled from other workspaces, and (4) artifacts are implicitly propagated through the central repository [35]. Adele extends this model with programmable process support for specifying sharing policies on a per-artifact (type) basis [14]. In both cases, the primary objective is workspace integration, not awareness, and Palantír complements the functionality provided.

6.2 Computer-supported collaborative work

BSCW [3] is a web-based, shared, centralized workspace with integrated versioning facilities that allow it to be used as a configuration management system. Awareness is provided statically, via web-based icons that enrich the web page for each artifact with information concerning its state, and dynamically, via a Monitor Applet that continuously informs authors of what activities are taking place in the central workspace. Compared to Palantír, BSCW lacks pair-wise comparisons, severity information, and a mechanism to scope the events of interest.

TUKAN [29] and COOP/Orm [22] are distributed, cooperative software development environments that support fine-grained editing and versioning of artifacts. In TUKAN, authors can select different collaboration modes and, upon accessing or modifying an artifact, are informed via "weather" icons whether potential conflicts exist with other authors. In COOP/Orm, active diffs instantly communicate changes to other developers who can see those changes both in the version tree and the actual artifact. Unfortunately, both TUKAN and COOP/Orm provide awareness on a per-artifact basis and only when the artifact is actually accessed. An overall view of activities in other workspaces, such as provided by Palantír, is lacking.

State Treemap [25] is an awareness widget for multi-synchronous groupware that relies on many of the same concepts as Palantír. Its visualization component shows which artifacts are being modified (both locally and remotely) and which artifacts already have been committed to the configuration management repository. Out-of-sync artifacts and potential conflicts, thus, are quickly discovered. Because State Treemap lacks author information, it does not provide pair-wise workspace comparisons. Severity information is also missing, though an enhancement for change impact calculations is underway [26].

Other approaches, such as Elvin [17], iScent [2], and CoVer [21], are subsumed by the systems discussed.

7. Conclusions

Any configuration management system exhibits a fundamental tension between the need for individual developers to work in isolated workspaces and the need for the overall team to maintain control over the integration of the individual changes into the overall system. To address this tension, we have developed Palantír, a novel configuration management workspace awareness tool. Palantír deliberately, but non-intrusively, breaks workspace isolation by continuously informing developers of the activities of others developers. By letting developers know who modifies which artifacts by how much, Palantír complements current automated procedures with the capability of human *intervention* when potential problems are recognized.

Palantír exhibits three key properties: (1) its coordination mechanism is based on workspace rather than repository information, (2) it continuously instead of sporadically informs developers of other ongoing efforts, and (3) it provides an overall view of other workspaces that supports the detection of both direct and indirect conflicts. The resulting system increases awareness among developers, and helps them in coordinating their tasks such that future integration problems can be avoided.

The current incarnation of Palantír represents only the beginnings of our investigations into the use of awareness in configuration management systems. Now that we have demonstrated that it is indeed possible to enhance existing configuration management systems with awareness, we intend to develop additional visualizations, explore the use of virtual file systems as a way of creating a more informative workspace wrapper, and investigate the use of a measure of change impact to complement our measure of change severity. Additionally, we will integrate Palantír with an industrial-strength configuration management system to study whether it truly scales and whether we can empirically determine its impact on coordination among developers in a actual, real-life development setting.

Acknowledgments

We thank the other members of our research group for their valuable suggestions during the design, implementation, and testing of Palantír.

This research is supported by the National Science Foundation under Grant Number CCR-0093489. Effort also sponsored by the Defense Advanced Research Projects Agency, Rome Laboratory, Air Force Materiel Command, USAF under agreement numbers F30602-00-2-0599 and F30602-00-2-0607. The U.S. Government is authorized to reproduce and distribute reprints for governmental purposes notwithstanding any copyright annotation thereon. The views and conclusions contained herein are those of the authors and should not be interpreted as necessarily representing the official policies or endorsements, either expressed or implied, of the Defense Advanced Research Projects Agency, Rome Laboratory or the U.S. Government.

References

[1] L. Allen, et al. *ClearCase MultiSite: Supporting Geographically-Distributed Software Development.* Proceedings of the International Workshop on Software Configuration Management: ICSE SCM-4 and SCM-5 Workshops Selected Papers, 1995: p. 194-214.

[2] K.M. Anderson and N.O. Bouvin. *Supporting Project Awareness on the WWW with the iScent Framework.* Proceedings of the International Workshop on Awareness and the WWW, 2000.

[3] W. Appelt. *WWW Based Collaboration with the BSCW System.* Proceedings of the Conference on Current Trends in Theory and Informatics, 1999: p. 66-78.

[4] B. Berliner. *CVS II: Parallelizing Software Development.* Proceedings of the USENIX Winter 1990 Technical Conference, 1990: p. 341-352.

[5] E.A. Bier and S. Freeman. *MMM: A User Interface Architecture for Shared Editors on a Single Screen.* Proceedings of the ACM Symposium on User Interface Software and Technology, 1991: p. 79-86.

[6] C. Burrows and I. Wesley, *Ovum Evaluates Configuration Management.* Ovum Ltd., Burlington, Massachussetts, 1998.

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

[8] M.C. Chu-Carroll and S. Sprenkle. *Coven: Brewing Better Collaboration through Software Configuration Management.* Proceedings of the Eighth International Symposium on Foundations of Software Engineering, 2000: p. 88-97.

[9] CollabNet, *SourceCast*, http://www.collabnet.org/products/sourcecast/, 2002.

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

[11] S. Dart. *Concepts in Configuration Management Systems.* Proceedings of the Third International Workshop on Software Configuration Management, 1991: p. 1-18.

[12] P. Dewan and R. Choudhary, *A High-Level and Flexible Framework for Implementing Multi-user Interfaces.* ACM Transactions on Information Systems, 1992. 10(4): p. 345-380.

[13] P. Dourish and V. Bellotti. *Awareness and Coordination in Shared Workspaces*. Proceedings of the ACM Conference on Computer-Supported Cooperative Work, 1992: p. 107-114.

[14] J. Estublier. *Defining and Supporting Concurrent Engineering Policies in SCM.* Proceedings of the Tenth International Workshop on Software Configuration Management, 2001.

[15] J. Estublier and R. Casalles, *The Adele Configuration Manager*, in Configuration Management, W.F. Tichy, Editor. 1994: p. 99-134.

[16] G. Fitzpatrick, et al., *Supporting Public Availability and Accessibility with Elvin: Experiences and Reflections.* Computer Supported Cooperative Work, 2002 (to appear).

[17] G. Fitzpatrick, et al. *Augmenting the Workaday World with Elvin*. Proceedings of the Sixth European Conference on Computer Supported Cooperative Work, 1999: p. 431-451.

[18] R.E. Grinter. *Using a Configuration Management Tool to Coordinate Software Development*. Proceedings of the Conference on Organizational Computing Systems, 1995: p. 168-177.

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

[20] C. Gutwin and S. Greenberg. *Workspace Awareness for Groupware*. Proceedings of the CHI'96 Conference Companion on Human Factors in Computing Systems, 1996: p. 208-209.

[21] A. Haake and J.M. Haake. *Take CoVer: Exploiting Version Support in Cooperative Systems.* Proceedings of the INTERCHI'93, 1993: p. 406-413.

[22] B. Magnusson and U. Asklund. *Fine Grained Version Control of Configurations in COOP/Orm*. Proceedings of the Sixth International Workshop on Software Configuration Management, 1996: p. 31-48.

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

[24] A. Mockus and J. Herbsleb. *Expertise Browser: A Quantitative Approach to Identifying Expertise*. Proceedings of the 2002 International Conference on Software Engineering, 2002.

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

[26] P. Molli, H. Skaf-Molli, and G. Oster. *Divergence Awareness for Virtual Team through the Web*. Proceedings of the Integrated Design and Process Technology, 2002.

[27] B.P. Munch. *Versioning in a Software Engineering Database - the Change-Oriented Way*. Ph.D. Thesis, DCST, NTH, 1993.

[28] D.E. Perry, 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.

[29] T. Schümmer and J.M. Haake. *Supporting Distributed Software Development by Modes of Collaboration.* Proceedings of the Seventh European Conference on Computer Supported Cooperative Work, 2001: p. 79-98.

[30] SourceForge.net, *SourceForge*, http://sourceforge.net/, 2002.

[31] Telelogic, *CM/Synergy*, http://www.telelogic.com/-products/synergy/cmsynergy/index.cfm, 2002.

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

[33] A. van der Hoek, et al., *A Testbed for Configuration Management Policy Programming.* IEEE Transactions on Software Engineering, 2002. 28(1): p. 79-99.

[34] C. Walrad and D. Strom, *The Importance of Branching Models in SCM.* IEEE Computer, 2002. 35(9): p. 31-38.

[35] A.I. Wang, et al. *Improving Cooperation Support in the EPOS CM System.* Proceedings of the European Workshop on Software Process Technology, 1998: p. 75-91.

[36] D. Wiborg Weber. *Change Sets versus Change Packages: Comparing Implementations of Change-Based SCM.* Proceedings of the Seventh International Workshop on Software Configuration Management, 1997: p. 25-35.