Decisions and Rationale during the Evolution of a Coordination Infrastructure

Roger M. Ripley and André van der Hoek
University of California, Irvine
Department of Informatics
Irvine, CA 92697-3425 USA
{rripley,andre}@ics.uci.edu

ABSTRACT

For workspace-to-workspace awareness systems, the design of the underlying infrastructure stands out as difficult. We discuss our experience building infrastructures and visualizations that aim to help people better coordinate their work, showing how our infrastructures have evolved through four of our prototype systems: Palantír, Lighthouse, Workspace Activity Viewer, and World View. Through our experiences and the lessons we have learned, we offer requirements for an ideal infrastructure.

1. INTRODUCTION

Coordination of development activities is one of the core activities in a software development project. Awareness tools, and their associated visualizations, aim to support developers by arming them with awareness of parallel activities, allowing them to self-coordinate their actions by placing their work in the context of others' actions.

As part of our research in workspace-to-workspace awareness tools, we have built infrastructures and visualizations that aim to help both developers and managers better coordinate their work by informing them of relevant parallel work, design evolution, loci of activity, and the context of those activities. A common difficulty through all our systems has been the underlying infrastructure.

The remainder of this paper is organized as follows. We briefly review the awareness tools we have developed in Section 2. Continuing with Section 3 we describe our experiences developing, evolving, and using our infrastructures. In Section 4 we discuss requirements for an ideal infrastructure, and conclude with Section 5.

2. SYSTEMS

We will discuss two systems that capture, distribute, and visualize workspace actions (Palantír and Lighthouse), as well as two stand-alone visualizations for already collected data (Workspace Activity Viewer and World View).

2.1 Palantír

Palantír [10, 9] is a workspace awareness tool based on the hypothesis that conflicts in parallel development can be considerably reduced, both in magnitude and number, by providing developers with insight into ongoing project activities. Palantír collects, distributes, organizes, and presents workspace information.

Palantír has different visualizations which present parallel activities in varying degrees of detail and obtrusiveness; however, they are all geared toward the individual developer and reveal the rest of the project from the perspective of that developer. The Palantír-Eclipse integration (Figure 1(a), left) "decorates" the package explorer view with red triangles to quickly alert the developer that parallel activity is taking place. The fully graphical visualization (Figure 1(a), right) complements the Eclipse integration by allowing for exploration of a hierarchical, stackable view of an artifact and its constituents. Other visualizations are similarly complementary; descriptions can be found in [11].

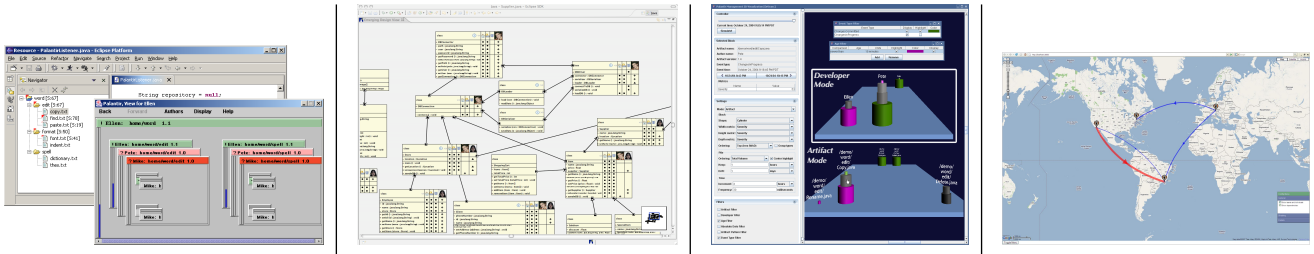
2.2 Lighthouse

Whereas Palantír provides awareness focused around files and blocks of code, Lighthouse [4] looks at the bigger picture by using the system design metaphor. As shown in Figure 1(b), Lighthouse builds a real-time UML-like class diagram to convey development activities. This visualization shows how the current system design is directly emerging from the collaboratively built source code and evolving as the code changes and advances. Design decay, conflicting changes in shared artifacts, and duplicate work can be spotted as they surface. The class diagram is annotated with authorship information, identifying the responsible developers for each step of the system design evolution.

2.3 Workspace Activity Viewer

Both Palantír and Lighthouse are targeted primarily at developers. The Workspace Activity Viewer [8], on the other hand, gives managers an overview of both past and ongoing activities in a project, using information extracted directly from developers' workspaces. As shown in Figure 1(c), it visualizes the developers and artifacts in a project using a 3D metaphor with stacks of cylinders representing artifacts and developers. Parallel work, developer activity, dormancy, and magnitude of change are represented by size and position along the different axes.

The Workspace Activity Viewer reveals how the project is evolving, both socially and technically, via a movie-like playback of the state of the project, showing what developers are



(a) Palantír. (b) Lighthouse. (c) Activity Viewer. (d) World View.

Figure 1: Screenshots of our awareness tools.

active when, and what types of artifacts they contribute to. It also shows social dependencies by revealing when developers are simultaneously working on the same artifact.

2.4 World View

World View [12] provides a location-oriented view of a software development project (Figure 1(d)). It allows project managers and developers to see the inter-team relationships in a global project, by providing the context in which development activities are taking place (teams, location, etc.) on a world map. It has the flexibility of not only visualizing conflicts, but any kind of location-relevant dependency, such as bugs and change requests submitted by one team but not resolved by another. Location can be especially relevant for those dependencies as differing time zones and national holidays can have large effects on the timeliness of resolution.

3. INFRASTRUCTURE EXPERIENCE

Underneath these systems, there is an evolving infrastructure which we change to meet new requirements from our tools and to address shortcomings discovered when evaluating and deploying them.

3.1 Events

In order to make Palantír independent of both the underlying SCM system and the development environment, Palantír silently intercepts the interactions of the developer with the repository (e.g., check-in, check-out, synchronize), monitors the activities of the developer in the workspace (e.g., edit, local save, local delete), and then transmits that information to other "remote" workspaces via events. Artifacts that are modified typically trigger the following sequence of events: Populated, ChangesInProgress, ChangesCommitted, and UnPopulated. When a developer saves their work frequently, but holds off on committing, an artifact will emit multiple ChangesInProgress (one for each local save) followed by a ChangesCommitted (for the eventual check-in). If a developer instead commits immediately after each change, pairs of ChangesInProgress and ChangesCommitted will result. Artifacts that are used by developers for ancillary purposes trigger a simple series of events: Populated followed by UnPopulated. Palantír's entire set of events is described in [10].

Palantír initially used a peer-to-peer architecture, as shown in Figure 2(a), using *Siena* [2] as the event service to broadcast Palantír events. When a new developer joined the Palantír workgroup, each Palantír instance would publish bootstrap events for the artifacts in its workspace, giving the new client a snapshot of every developer's state.

3.2 Centralization

Palantír's peer-to-peer architecture specifies that when a new developer connects, each existing Palantír instance would bootstrap the new one with their current state. This was deficient, however, because when a developer leaves the Palantír workgroup, the knowledge of what they are working on is lost to joining developers, and when a developer rejoins the Palantír workgroup, their visualizations have lost their own history and now only reflect the current status of other developers. Furthermore, since Palantír did not persistently store events, there was no way to analyze the collected data for patterns or to generate historical visualizations.

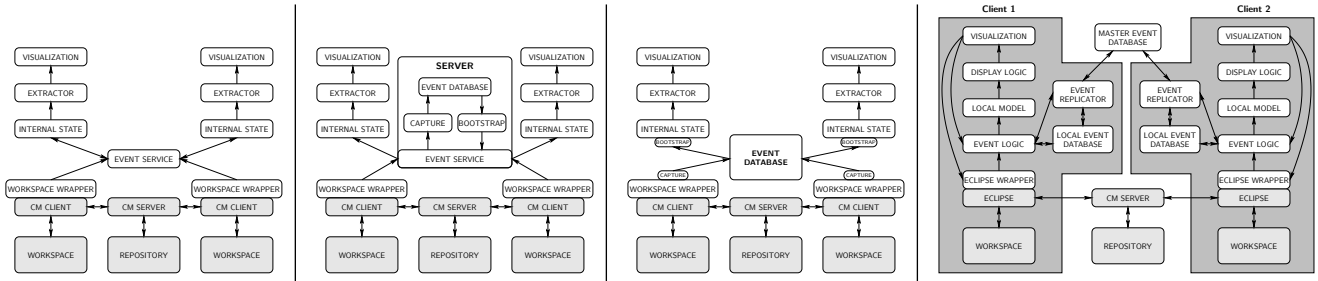
To address these issues, we added a centralized server, as shown in Figure 2(b), which encapsulates the event service—acting as a *Siena* dispatcher, captures all events to a database, and handles publishing bootstrap information from those stored events. The data definition for the events table occurs automatically, based on the attributes in the notifications received by *Siena*.

3.3 Direct database connection

Using *Siena* as our event service had several drawbacks. The first was that ordering of events was not guaranteed, and occasionally dependent events would arrive out of order. The second was that *Siena*'s transports all required a routable unfirewalled connection between client and server and vice versa, preventing clients from being behind network address translation. We wrote a tunneling transport to address this, but it was an awkward solution. We realized that we could push the bootstrap and event capture logic from the server into the client, and have the client connect directly to the database server replacing the event service with the event table. We utilized the `LISTEN` and `NOTIFY` functions in PostgreSQL [6] to wake up the client to retrieve new events without having to poll the server. This made event communication reliable while simultaneously simplifying the system architecture, as shown in Figure 2(c).

3.4 Client event cache

After Palantír was connecting directly to the database, there remained one major infrastructural issue: clients required a full time network connection to the database server. Any drop of the database connection and events being generated by the client would be lost. To remedy this, we modified the event capture module to store generated events locally when there was not an active connection to the database. Palantír visualizations would continue to show the last connected state until a connection was reestablished. If the client started disconnected, it would immediately cache



(a) Palantír: peer-to-peer. (b) Palantír: central server. (c) Palantír: direct database. (d) Lighthouse: database replication.

Figure 2: Infrastructure evolution.

events, but none of the visualizations would be enabled until a connection was able to be established.

3.5 Local database replication

One drawback to the event caching mechanism of Palantír was that if a client started disconnected, they would not be able to use any of the visualizations or analyses, even while their own events were being saved locally.

When designing the event distribution method for Lighthouse, we decided to store and retrieve all events locally using an embedded SQL database, Derby [1]. Unlike Palantír's events, which were sets of name-value pairs, Lighthouse's events were defined as a rich object-oriented model. Because of this, we used Hibernate [7] to map the events objects onto database tables. Periodically the local stored events are pushed to a master event database, while pulling new remote events left by other Lighthouse clients into the local event database, as shown in Figure 2(d). This allows for disconnected operation without losing functionality, and it also allows developers to write visualizations that are irrespective of connection status. The master event database keeps a history of all events, supporting bootstrapping new developers when they join a project, analyses, and visualizations.

3.6 Project history playback

The Workspace Activity Viewer uses Palantír's database of events. In order to accomplish a movie-like playback of workspace state across an entire project—with the ability to rewind, fast-forward, and jump to any position—we have to derive the project state at any particular moment. It is too time-consuming to derive the project state from the event database live during playback, so we preprocess the events to create in-memory indicies.

Rather than building that array of timestamps and pointers into the event database for each developer and each artifact, the infrastructure should facilitate the queries needed by the visualization, and other analysis queries, instead of acting as solely as a repository of historical events.

3.7 Approximating unavailable data

In order to evaluate the Workspace Activity Viewer, we needed real data for the visualization to process. We mined the CVS repositories of several open-source projects hosted on SourceForge [13]. Since the repository only contains records of SCM interactions—not workspace activity data—we simulated workspace activities between check-outs and check-ins. The event model gives us the flexibility of approximating any unavailable data, with the caveat of being judicious when making conclusions based upon that data.

To generate the simulation data, we used the CVS metadata for each commit (who, when, and how much changed) to establish points of known state for each artifact. Before each of these check-ins, we generate events indicating that the artifact is being changed by the developer who checked it in, with the frequency and severity (magnitude of change) increasing as the commit time approaches. As a result, our simulation is accurate at commits, and plausible in between.

3.8 Metadata

The context in which development activities takes place is critical: the teams that developers belong to, the locations they work in, etc. To visualize conflicts between teams, the event information stored by Palantír was insufficient in that it does not store what team individuals belong to, nor where specific teams are located. The database schema had to be expanded to not only include the events table, but also to be able to maintain the metadata required by World View. When updating the metadata, it will not suffice to replace the old value with the new—it must be versioned so that visualizations and analyses can see their evolution as well. As teams change members, members move between different geographic locations, etc., the visualizations need to remain accurate historically to see how geography boundaries affected the software project over time.

4. ANALYSIS / DISCUSSION

Based on our experience, we have determined several dimensions of requirements that should be considered in future coordination infrastructures.

4.1 Events and states

There are two primary and complementary ways to track actions: event streams and captured states. Event streams track *what happened*, but at the cost of having to iterate through the stream in order to know the state of the system at a particular moment in time. Captured states track *what is*, but neglect the cause of the transitions between states. Therefore, we propose tracking both the events and the resultant states, in the same way that a check register records the transaction (event) and the current balance (state).

4.2 Relationships

Events tend to refer to an action performed by a developer upon an artifact in the software project. This can be generalized as two entities entering into a relationship, mutating an already existing relationship, or terminating a relation-

ship. This generalization opens up a realm of possibilities, such as, relationships between authors and teams (authors $\mathcal{X}$, $\mathcal{Y}$, $\mathcal{Z}$ belong to team $\mathcal{W}$, author $\mathcal{S}$ is managed by author $\mathcal{T}$), relationships between artifacts (class $\mathcal{A}$ depends on class $\mathcal{B}$, class $\mathcal{C}$ implements requirement $\mathcal{M}$). The infrastructure should be able to store and version these relationships as they begin, change, and end. The use of relationships can also subsume much of what metadata was used for prior.

4.3 Hierarchical types and actions

In order for there to be more flexibility with regards to mapping actions and artifacts onto visualizations, the infrastructure should support a hierarchy of types and actions. If desired, a visualization that tracks when an class is modified should be able to recognize when a method in that class is modified without requiring a storm of events for the method, class, project, etc., for example. A different visualization could show changes at the method level if it was designed before events were captured with that granularity. Furthermore, actions can have a hierarchy, e.g. renaming a method is a specific type of changing of a method signature. This will allow for visualizations and analyses to be more independent of the type and granularity of data collected.

4.4 Queries and analyses

Beyond captured events, states, and relationships, some visualizations and tools require additional computationally expensive analyses to be performed. Ariadne (an analysis and visualization tool not discussed in this paper) analyzes source code authorship metadata combined with source code call analysis to derive socio-technical dependencies [5]. These types of analyses should be automatically performed and kept up-to-date on the server, so that they are available immediately when needed, similar to nightly builds, or database views. By having these analyses integrated into the infrastructure, the information they provide can be shared with and used to supplement other visualizations, such as World View or Workspace Activity Viewer.

4.5 Client configuration

Using the infrastructure should require a minimum amount of configuration by the end user developer. Configuration should be automatically downloaded from the infrastructure into the client when possible, or even read out of a small configuration file in the SCM repository, requiring only that the developer be able to check out their project from the repository to be able to activate the awareness client.

4.6 Client disconnection

The infrastructure also needs to accommodate temporarily disconnected clients. When a client is disconnected, they need to be unimpeded in their work. Notifications about the disconnected state should be unobtrusive. Users should also be able to maintain the maximum possible benefit from the system while disconnected, e.g. with working visualizations based on the last known state of the project.

5. CONCLUSIONS

In this paper we presented the infrastructures and visualizations that we have built to help both developers and managers better coordinate their work by informing them of relevant parallel work, design evolution, loci of activity, and the context of those activities. We showed how our infrastructure has evolved through Palantír, Lighthouse, Workspace Activity Viewer, and World View, and the difficulties we encountered which spurred this evolution. We finally gave some requirements for an ideal awareness infrastructure. Jazz [3] is a major step in right direction with regards to infrastructure support, but there remains more to be done with regards to supporting *workspace level* awareness.

6. ACKNOWLEDGMENTS

Effort partially funded by a 2004 Eclipse Innovation Grant, a 2005 IBM Eclipse Technology Exchange Grant, a 2006 IBM Technology Fellowship, and by the National Science Foundation under grant numbers IIS-0205724 and IIS-0534775.

7. REFERENCES

[1] Apache Software Foundation. Apache Derby. `http://db.apache.org/derby/`, 2008.

[2] A. Carzaniga, D. S. Rosenblum, and A. L. Wolf. Design and evaluation of a wide-area event notification service. *ACM Transactions on Computer Systems*, 19(3), Aug. 2001.

[3] L.-T. Cheng, S. Hupfer, S. Ross, and J. Patterson. Jazzing up Eclipse with collaborative tools. In *Proceedings of the 2003 eclipse technology eXchange*, 2003.

[4] I. A. da Silva, P. H. Chen, C. V. der Westhuizen, R. M. Ripley, and A. van der Hoek. Lighthouse: coordination through emerging design. In *Proceedings of the 2006 eclipse technology eXchange*, 2006.

[5] C. R. de Souza, S. Quirk, E. Trainer, and D. F. Redmiles. Supporting collaborative software development through the visualization of socio-technical dependencies. In *Proceedings of GROUP '07*, 2007.

[6] PostgreSQL Global Development Group. PostgreSQL Manual: NOTIFY. `http://www.postgresql.org/docs/current/interactive/sql-notify.html`, 2008.

[7] Red Hat Middleware, LLC. Hibernate. `http://www.hibernate.org/`, 2008.

[8] R. M. Ripley, A. Sarma, and A. van der Hoek. A visualization for software project awareness and evolution. In *Proceedings of VISSOFT 2007*, 2007.

[9] A. Sarma, G. Bortis, and A. van der Hoek. Towards supporting awareness of indirect conflicts across software configuration management workspaces. In *Proceedings of ASE '07*, 2007.

[10] A. Sarma, Z. Noroozi, and A. van der Hoek. Palantír: Raising awareness among configuration management workspaces. In *25th International Conference on Software Engineering*, pages 444–454, 2003.

[11] A. Sarma and A. van der Hoek. Visualizing parallel workspace activities. In *IASTED International Conference on Software Engineering and Applications (SEA)*, 2003.

[12] A. Sarma and A. van der Hoek. Towards awareness in the large. In *Proceedings of ICGSE '06*, 2006.

[13] SourceForge, Inc. Welcome to SourceForge.net. `http://sourceforge.net/`, 2008.