Continuous Coordination (CC): A New Collaboration Paradigm

Ban Al-Ani, Anita Sarma, Gerald Bortis, Isabella Almeida da Silva, Erik Trainer, André van der Hoek, David Redmiles

University of California, Irvine
Department of Informatics
444 Computer Science Building
Irvine, CA 92697-3440 USA
Phone: +1(949) 824-2776
{balani, asarma, gbortis, ialmeida, etrainer, andre, redmiles}@ics.uci.edu

ABSTRACT

The increase in software complexity introduced the need for software development teams and consequently the need to coordinate team members' activities and create a shared awareness. We seek to overcome some the pitfalls of earlier attempts to coordinate software development through a new coordination paradigm we term *Continuous Coordination* (CC). Generally speaking, the CC paradigm complements formal synchronization with support for informal activities. In this paper, we define the CC paradigm within three dimensions and demonstrate how we embodied CC through a spectrum of Eclipse plug-ins.

CATEGORIES AND SUBJECT DESCRIPTORS

H.4.3 [**Information Systems Applications**]: Office Automation-Groupware; H5.3 [**Information Interfaces and Presentation**] Group and Organization Interfaces – Computer Supported Cooperative Work.

GENERAL TERMS

Management, Design, Human Factors

KEYWORDS

Software Engineer, collaborative work, visualization, configuration management, programming, design.

1. INTRODUCTION

Creating software is an inherently complex task because of its changeable and intangible nature. It is further complicated by the dependencies that exist among artifacts and the gamut of rich interactions required among developers. Distributed software development only adds to this plethora of complexities and further emphasized the need for development environments that provide comprehensive support for different aspects of software development (e.g. Curtis et al., 1988).

The proposed paradigm, Continuous Coordination (CC), blends the best aspects of the more formal, process-oriented approach with those of the more informal, awareness-based approach. In doing so, continuous coordination blends processes to guide users in their day-to-day high-level activities with extensive information sharing and presentation to inform users of relevant, parallel ongoing activities. Thus it provides the underlying infrastructure for coordination. Some of the key properties, we identified, for tools that follow this paradigm are that the tools share *relevant* information and do so in a *contextualized* and *unobtrusive* manner. We deem information *relevant* when it is provided to a developer who will utilize it in the foreseeable future. Shared information is *contextualized* and *unobtrusive* when it is embedded in the development environment allowing developers to modify their behavior at a time that is convenient to them.

Other general tool properties are also being explored in our endeavor to increase the effectiveness of the tools developed within the dimensions of CC. For example, we are of the opinion that developers can need differing levels of information abstraction at various stages of development while carrying out different developmental tasks. We sought to develop a range of tools that can offer a spectrum of support. The tools can then be incorporated into different phases of development by developers as they see fit. Thereby increasing flexibility and providing support for developers' low level programming activities through to high level support of managerial activities.

In this paper, we present a definition of Continuous Coordination dimensions and an outline of some of the tools we have developed thus far within these dimensions. They are discussed in terms of the kind of information it provides and to whom.

2. CONTINUOUS COORDINATION

The CC project sought to address a wide range of needs that are typically manifested during the software engineering process when conducted by co-located or distributed teams. Shared awareness, through shared information is one such need. It has been recognized as being both important and challenging (de Souza et al, 2004).

The challenge in sharing information in this way is achieving an appropriate level of detail and providing it at a time that is suitable to the developers. *How* much information should we provide the developer? Providing a constant stream of information can lead the developers to feel overwhelmed whereas infrequent sharing of information can mean that a developer lacks sufficient information to successfully complete a task.

The information provided to the developer depends on his/her role within the team. *What* kind of information does the developer need? For example, a manager would typically need to be aware of

Conference'06, Month 1–2, 2006, City, State, Country.

team structure, work products and interactions. A programmer, however, would generally need to be aware of changes to the design or code made by other team members.

Finally, we also found that while identifying the amount and the type of information needed by the developers must be determined, the manner in which it is presented should also be considered. *When* should information be shared? For example, if a developer chooses to ignore the shared information this should not impede completing the task at hand. Furthermore, shared information should not distract a developer from the task at hand. The information should be available, such that, a developer can access it at a time suitable to him/her becoming part of his/her *peripheral awareness* in the meanwhile.

In summary, the type of information needed by the developer, the triggers to share information and the recipients of shared information form the three principal CC dimensions. Our approach to embodying the CC paradigm within these dimensions will be discussed in the following section.

3. PLUG-INS: EMBODYING CONTINUOUS COORDINATION

We sought to embody the CC paradigm through a series of Eclipse plug-ins, for several reasons. First, we sought to enable developers to incorporate the proposed plug-ins in a manner suited to the process they have chosen to adopt. In adopting this approach we sought to increase the paradigm's flexibility. Second, we sought to tailor information to individual developer needs and consequently the role they play within the project. We decomposed information that is generally shared, such that, it is possible to identify who it would benefit (e.g. programmers, designer, managers...etc) and potentially minimize the amount of redundant information shared. The third and final reason we embodied CC through plug-ins is because we endeavored to promote self-coordination. While a developer may not be able to coordinate the whole project within the restrictions of their assigned roles they are able, through the proposed series of plug-ins, to coordinate tasks with their peers and those who share their artifacts. Furthermore, some of the plug-ins provide a high level of abstraction or visualizations that can be useful to all types of developers e.g. programmers, designers, managers…etc. In such instances, in this report, they will be referred to collectively as developers only.

3.1 Lighthouse

Lighthouse is a coordination platform that is rooted in the concept of emerging design, a real-time representation of the design as it is being implemented in the code by each of the programmers. Lighthouse then projects this emerging design view on top of the initially conceived conceptual design (da Silva et al, 2006).

Figure 1 illustrates how programmers are able to maintain peripheral awareness of ongoing changes made to the project by the team members when adopting the proposed dual-monitor setup. In this set-up a main monitor would have their primary coding environment and an auxiliary monitor would be dedicated to Lighthouse.

Lighthouse development efforts are currently focused on further improving the user interface such that the changes made to the program is reflected in the design more effectively. Once this is stage is concluded the tool will be validated empirically.

1. a. Project management view.

1. b. Programmer's side-by-side view of code and emerging design.

Figure 1. The emerging design overlaid on top of the original conceptual design produced by Lighthouse.

3.2 Palantír

The Palantír plug-in is a workspace awareness tool that provides developers with insight into ongoing development activities in remote workspaces (Sarma et al, 2003). Specifically, Palantír provides information that includes identifying who is conducting a change, what is being changed, calculates a measure of the magnitude of those changes, a measure of the impact of those changes, and graphically displays this information in a configurable and non-obtrusive manner to developers involved in programming (Figure 2).

Palantír breaks the isolation of distributed Configuration Management (CM) workspaces by continuously sharing information of ongoing changes, thereby allowing early detection of conflicts while changes are still in progress. In addition to information regarding which artifacts are being changed by which developer, Palantír distinguishes itself by providing information about the severity and impact of changes. These measures allow developers to gauge which changes are important and require their attention.

Finally, Palantír promotes a model of self-coordination recognizing that many possible and flexible resolutions are possible bringing to distributed development a level of awareness that begins to approach that of local settings. Thus, while the developers are notified of changes, the notification does not impede their work or force them to take immediate action.

Currently, we are in the process of evaluating the effectiveness of Palantír in enabling developers detect potential conflicts earlier and in producing better quality software (i.e. fewer unresolved conflicts) through controlled lab experiments and results are being analyzed.

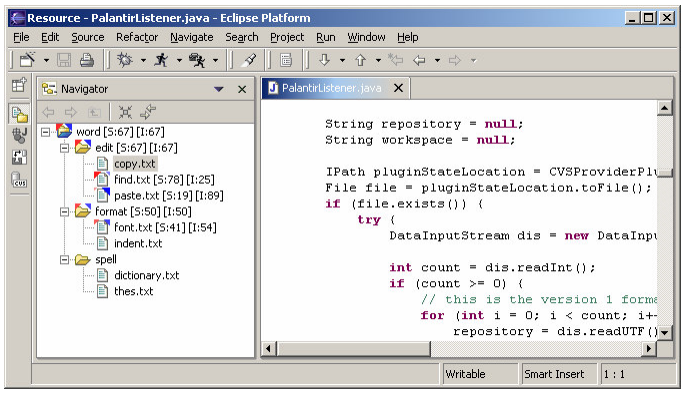
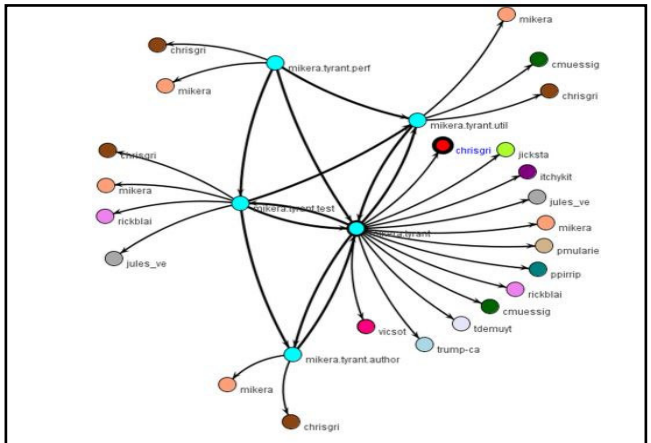


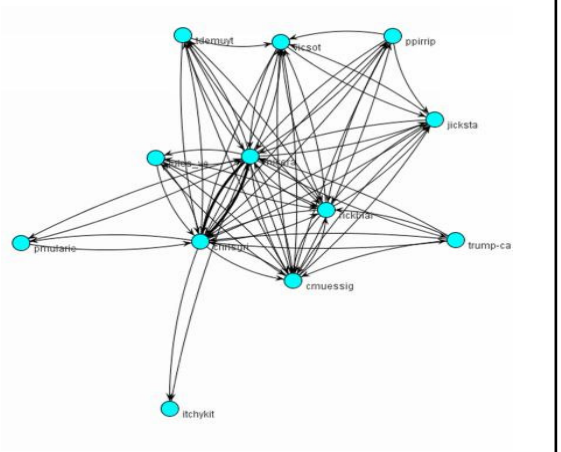
Figure 2. Palantír Visualization.

3.3 Ariadne

Ariadne is a collaborative software engineering tool that aims to enhance developers' awareness of the social dependencies present in their work by seamlessly integrating such information with development activities (Trainer et al, 2005).



3.a. An Ariadne "social call graph" illustrating code dependencies and author dependencies.



3.b An Ariadne "sociogram" illustrating the social network of software developers.

Figure 3. Examples of graphical representations of social dependencies produced by the Ariadne plug-in.

It analyzes software development projects for source-code dependencies and collects authorship information for the source-code from a configuration management repository. The tool then links the source-code dependencies and authorship information to create a social network of software developers (Figure 3.a). We aim to complement this social network graph with current social network analysis techniques, giving programmers and designers an insight into how their work affects other developers and how the work of other developers affects their own. For example, "centrality" is a measure of the power of nodes as a function of their degree of connectedness with other nodes, their closeness to other nodes in the graph, and their positions as intermediaries between other nodes (Figure 3.b). .

We intend to determine the validity of applying information gleaned from social network metrics to enhance programmers' and designers' awareness of their colleagues' development efforts. Centrality offers a measurement of control or ownership of code, and may help developers identify important players in the project team. Equivalence may be used to help developers identify who is using code similarly to prevent duplication of work.

Finally, Ariadne is currently being trialed to visualize the social interaction within the Ariadne project itself. It succeeded in representing these interaction and the relationships between developers involved. However, initial trials also revealed issues relating to the display of textual information (node labels) and readability. These issues and others will be addressed before conducting extensive empirical studies.

3.4 Dashboard

This plug-in is currently in the design phase of development. A "Wizard of Oz" prototype is being utilized to determine what information would support the social interactions (Figure 4). Ultimately we seek to provide a means to simulate informal "watercooler" conversation by providing a central location where project developers can (1) be informed of their work and how it relates to the overall project, and (2) spontaneously engage in exploration of particular issues raised on the board. Insights gained in this phase of development will assist in developing the first working prototype and determining which visualizations are incorporated in the final product.

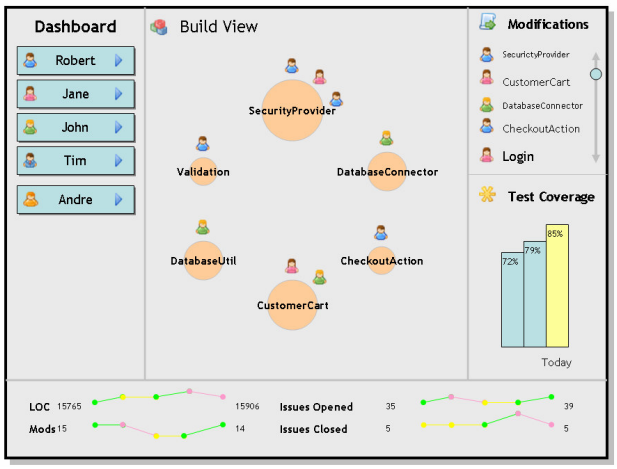


Figure 4. Dashboard provides an abstract view demonstrating the relationship between the developer and the modules under development.

In Figure 4, the programmer's attention is naturally drawn to the spheres, which represent modules that have caused the build to fail. Larger spheres emphasize module importance based on severity metrics. Circling the modules in a clockwise fashion are the names

of the programmers who have most recently modified the module. This visual connection between the programmer and the module is further explored in the modifications view (made accessible through the right panel), which displays in real-time the most recent transactions to the configuration management system. Again, more severe modifications are emphasized by a larger module name and sphere.

3.5 World View

World View plug-in provides a comprehensive view of the team dynamics of a project, regarding the geographical location of teams, the time zones of their operations, and the interdependencies among teams (Figure 5). This view is intended to help developers involved in global software identify global and local team members, interactions between sub-groups and other vital information like how to contact global member and when (Sarma and van der Hoek, 2006).

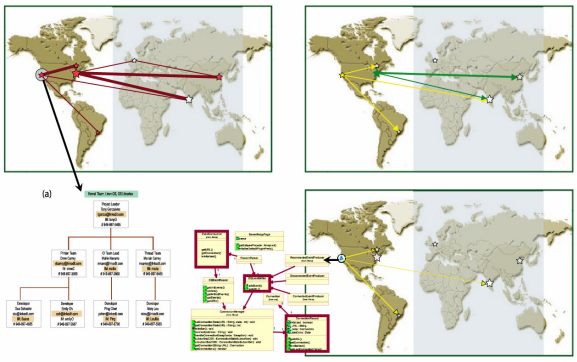


Figure 5. World View screen prototypes. Shaded areas of the map represent countries where it is dark. Active teams in the shaded areas are shown as "white stars".

In Figure 5 teams are represented as "stars" on a world map and interdependencies among teams are shown as "lines" connecting them. The size of the star denotes the size of the team; larger teams are represented as larger stars. Interdependencies among teams are determined based on the number of shared artifacts, which are identified through program analysis of the code base. The thickness of the lines represents the extent of sharing: the thicker the lines, the larger the number of shared artifacts. Through this view, developers can discern at-a-glance which teams are tightly-coupled and through which artifacts (mouse-hovers display the list of shared artifacts).

The directed lines (arrows) in Figure 5 represent the direction of conflicts (changes performed by which team affects which team) and the thickness of the lines denotes the extent of the conflict: the thicker the line, the larger the significance of the conflict. Here, significance is calculated as the number of artifacts that are affected by the change. Teams and their respective "arrows" are color coded to differentiate conflicts arising from different teams. This view can also be configured for the individual developer to show which changes by a specific developer affects other teams. The artifacts responsible for the conflicts are highlighted in red.

Currently, the World View tool is in the exploratory phase with the first prototype to be made available soon.

4. CONCLUDING REMARKS

A new software engineering paradigm was presented in this report, namely: continuous coordination. The project is implemented through a collection of plug-ins. A brief description of each plug in and an outline of the user interface were presented for each. The descriptions sought to demonstrate how each tool fell within the boundaries of the CC dimensions, namely:

1. *Amount of shared information:* each tool provides layers of information such that the developer can adjust the volume and level of detail based on individual need.
2. *Nature of shared information:* the varying forms of information provided by each tool make it possible to focus on information relevant to the task at hand.
3. *Peripheral Awareness:* the information provided by each tool is readily available for the developer to access but does not prevent the developer from continuing with his/her task. The information provided by each tool thus remains within the peripheral awareness of the developers and does not impede their work.

Collectively, these three dimensions seek to define the essence of CC by enabling the developers to *share* an *awareness* of activities carried out during a collaborative development process; such that, developers are neither constrained by the lack of information nor is their work blurred by a high volume of information.

The degree of success achieved by each tool in conforming to these dimensions is yet to be determined through empirical evaluations. However, feedback received from walkthroughs of early prototypes and mock-ups has been positive overall.

5. ACKNOWLEDGEMENTS

This research was supported by the U.S. National Science Foundation under grants 0534775, 0326105, 0093489, and 0205724, by the Intel Corporation, by two IBM Eclipse Technology Exchange grants, and an IBM Technology Fellowship.

6. REFERENCES

[1] Canfora, G.; Cerulo, L., Jimpa: An Eclipse Plug-in for Impact Analysis, *Conference on Software Maintenance and Reengineering (CSMR'06)*, (March 22 - 24, 2006), 341-342.

[2] Curtis, B., H. Krasner, Iscoe, N. (1988). "A field study of the software design process for large systems." Communications of the ACM, 31(11): 1268-1287.

[3] de Souza, C. R., Redmiles, D., Cheng, L., Millen, D., and Patterson, J. 2004. Sometimes you need to see through walls: a field study of application programming interfaces. In *Proceedings of the 2004 ACM Conference on Computer Supported Cooperative Work* (Chicago, Illinois, USA, November 06 - 10, 2004).

[4] da Silva, I.A., Chen, P., Van der Westhuizen, C., Ripley, R. and van der Hoek, A. Lighthouse: Coordination through Emerging Design, *OOPSLA Eclipse Technology Exchange Workshop*, October 2006 (to appear).

[5] Sarma, A., Noroozi, Z., and van der Hoek , A., "Palantír: Raising Awareness among Configuration Management Workspaces", *In Proceedings of Twenty-Fifth International Conference on Software Engineering*, p. 444-454, Portland, Oregon (May 2003).

[6] Sarma, A. and A. van der Hoek, "Towards Awareness in the Large", *First International Conference on Global Software Engineering*, Brazil, to appear (October 2006).

[7] Trainer, E., Quirk, S., de Souza, C. R. B., Redmiles, David F. Bridging the Gap between Technical and Social Dependencies with Ariadne. In: Proceedings of the Eclipse Technology eXchange (ETX) Workshop, San Diego, CA, 2005.