2011

A Degree of Conflict Model for Workspace Awareness

Bakhtiar Khan Kasi
University of Nebraska-Lincoln, bkasi@cse.unl.edu

Anita Sarma
University of Nebraska-Lincoln, asarma@cse.unl.edu

A Degree of Conflict Model for Workspace Awareness

Bakhtiar Khan Kasi and Anita Sarma
Computer Science and Engineering Department
University of Nebraska-Lincoln
Lincoln, NE – 68588-0115
{bkasi, asarma}@cse.unl.edu

Tech Report: TR-UNL-CSE-2011-0002

ABSTRACT
Workspace awareness solutions provide ongoing change information at the level of files. This makes the user responsible for identifying how current changes affect their tasks and provides no guidance for planning their (future) tasks. Here, we present our approach to task-based awareness that calculates a degree-of-conflict for tasks and recommends an optimum set of tasks that minimizes the risk of conflicts. Specifically, we present three novel research ideas: (1) transition from the current file-based awareness systems to task-based awareness, (2) transition from reactive conflict detection to proactive conflict prediction, and (3) a degree-of-conflict model that models conflicts per task per workspace, which can be used to recommend an optimum task list for a developer.

Categories and Subject Descriptors
D.2.6 [**Software Engineering**]: Programming Environments – *Programmer workbench.* D.2.7 [**Software Engineering**]: Distribution, Maintenance, and Enhancement – *version control.*

General Terms
Management, Measurement, Human Factors.

Keywords
Workspace awareness, conflicts, degree of conflict model.

1. INTRODUCTION
Software conflicts can arise when developers work on parallel changes in their distributed workspaces. Case studies have shown a high incidence of such conflicts in the software industry [2, 7]. These conflicts are often a result of coordination breakdowns and a lack of understanding of how one's work fits with other parallel changes in the project. An example of such a breakdown is when two developers inadvertently edit the same file in parallel or when an API that was declared to be stable is changed without appropriate notifications to developers using it. In fact, coordination activities constitute a significant portion of a developer's day-to-day activities (sometimes taking up to 78% of their time) [9].

In large distributed projects, developers typically have difficulty in identifying their impact network – individuals on whom they are dependent and individuals who are dependent on them [2]. Current workspace awareness tools attempt to alleviate this problem by (continuously) providing change information of which artifact is being changed by whom and by how much. Some also warn developers of potential conflicts that might occur when the changes are put together [1, 3]. The intention is to enable a developer to realize the significance of the conflict and prod them into self-coordinating [8].

There are several drawbacks to this approach. First, it places the responsibility of understanding the impact of a change and the best possible course of action on the user. Now, in addition to the challenging cognitive task of writing code, a developer also has to "keep an eye out" for relevant changes and determine the best coordination strategy to mitigate the effects of emerging conflicts. Further, this strategy of individuals determining the best course of action for themselves might not be what is the best for the team.

Second, current tools provide awareness at the file level. This is the status quo since files are the basic unit of operation for most development editors and configuration management systems. However, awareness provision at the file level means developers are responsible for reconstructing the change information to identify the effects of ongoing changes on their tasks.

Finally, workspace awareness tools are reactive, that is, they only help identify conflicts once changes are already in progress. While this is helpful, it is insufficient in helping a developer to plan their tasks. Our experiment results of a large scale usability study on workspace awareness shows that users frequently contacted their team members to determine which files others were intending to edit and for which tasks, so as to better plan own their tasks [8].

In this paper, we propose a task-based awareness system that is geared towards overcoming these drawbacks. Our approach tracks resources that are associated with a task and based on the current information of which tasks are being performed in which workspace, recommends the optimum task for a developer such that she faces a minimum chance of a conflict. Through our approach we provide awareness at the task level, which is better aligned with a developer's cognitive unit of work. Further, we push the state-of-art in workspace awareness from being reactive to proactive, by determining which resources will be changed based on their association with tasks that are currently being performed.

Designing a task-based awareness system raises many questions such as: how to provide automated support to associate resources with tasks; how to create an optimum task order that takes into consideration the chances of conflicts and their impact, developer and team priorities; how to provide information without overloading the developer; scalability; and the general effectiveness of the approach in helping a team navigate through their development

Conference'10, Month 1–2, 2010, City, State, Country.

tasks. This paper, serves as an initial investigation of the feasibility of designing a task-based awareness system.

The rest of the paper is organized as follows. In Section 2 we discuss background information on workspace awareness and Mylyn [6], a task centric Eclipse extension. Section 3 presents an illustrative example that is used through the paper. We present our approach in Section 4 and conclude with a discussion on the challenges in creating task-based awareness in Section 5.

2. Background

Our work builds on two main bodies of work. First, workspace awareness, which provides awareness of emerging conflicts as changes take place in private workspaces. Second, Mylyn – a task centered Eclipse extension that uses a degree-of-interest model to identify artifacts that are relevant for a particular developer task.

2.1 Workspace Awareness

Working in a distributed team requires an understanding of how one's own work fits in the context of other ongoing parallel changes and the overall project direction. Obtaining such an understanding of ongoing changes and their impact is difficult [2], and becomes much more difficult in large distributed teams [4].

Many different kinds of conflicts can arise in parallel work when it is performed in distributed workspaces. The impact of each type of conflict and the effort to resolve them vary. We identify three major categories of conflicts (see Section 3 for an example of each kind). First, *merge conflicts* arising because of parallel changes to the same artifact. This type of conflict is typically identified when a developer attempts to check-in their changes while a newer version already exists in the repository. Second, *build failures* arising because of parallel changes to two different artifacts that cause syntactic mismatches and ensuing compilation errors. Such conflicts are typically identified during a system-wide build. Finally, *test failures* arising because of parallel changes to two different artifacts that cause mismatches in program behavioral. Such conflicts are only detected during testing (integration) or may remain as defects in the field.

Workspace awareness tools (e.g., CollabVS [3], FastDash [1], Palantír [8]) attempt to identify such conflicts early while developers are still making changes. The premise of these tools is that conflicts do not appear instantaneously, but occur slowly at the pace of human development. The goal of workspace awareness tools is to prod a developer into taking coordination actions while changes are still work-in-progress and the conflicts still small and relatively easy to resolve. Most tools use visualization cues to notidy developers of impending conflicts.

2.2 Mylyn

Mylyn [6] provides a task-centric interface for the Eclipse IDE, which allows developers to view their tasks (either local tasks or from a remote repository such as Bugzilla), select a task on which they would like to work, and identify the resources associated with that task. Each task has an associated context that includes resources (files or methods) that are: (1) explicitly selected, (2) undergoing edits, or (3) being referenced. Mylyn monitors developer activities to identify relevant resources for the task context.

More specifically, Mylyn monitors direct and indirect interactions of a developer. A direct interaction occurs when a developer explicitly selects a particular file or edits it. An indirect interaction refers to a class of event where program elements and relationships are selected for the task context because Mylyn anticipates them to be of interest. For example, a propagation event occurs when a developer navigates to a different file by using the "open declaration" shortcut in Eclipse. Similarly, Mylyn generates a prediction event to include the parent class of a file that is currently being edited.

Table 1. Task list of Bob and Alice.

Alice's Workspace			Bob's Workspace		
T_{A1}	R	Shape.java (C) Rectangle.java (C)	T_{B1}	M	Rectangle.java (E) Square.java (S) Shape.java (P)
T_{A2}	F	Draw.java (S)	T_{B2}	F	Triangle.java (S)
T_{A3}	F	Plane.java (S)	T_{B3}	F	Plot.java(S)

Additionally, Mylyn uses a degree-of-interest (DOI) model to determine the degree of relevance of a resource in the task context [5]. The model associates an interest value with each resource in the task context. As a user interacts with a program element it's DOI value increases (or gains interest). Similarly DOI values decay when a user does not explicitly select or edit a resource. A resource is removed from the task context if its DOI value falls below a set threshold. Mylyn uses text cues to highlight resources that are of higher interest in the task context.

From the developers' perspective, Mylyn recommends relevant program elements for their current task and provides an uncluttered package explorer interface, which displays only relevant resources. Mylyn helps improve productivity by reducing the time that a developer spends on searching, scrolling, and navigating. Additionally, Mylyn allows developers to easily switch their active tasks by maintaining a record of the task context of each task.

3. Example

Consider a very simplified scenario where Alice and Bob are working on a hypothetical project involving polygons, where classes Square.java, Rectangle.java, and Triangle.java inherit from the abstract class Shape.java. Table 1 summarizes their tasks, the order in which they will be implemented, and the task type (R - refactor, M - modification, F - Feature).

To plan for future additions of new shapes in the code base Alice in method T_{A1} refactors Shape.java to combine the implementations of methods *area(float l, float w),* which calculates area for a rectangle and *area (float s),* which calculates area for a square into a single method. This new method uses an additional parameter for the type of shape (*shape_type*), which is used to calculate the appropriate area. She also modifies Rectangle.java to update its call to *shape.area()* method and commits all her changes.

Meanwhile, Bob in T_{B1} adds new functionality to Rectangle.java (adds *perimeter()* method) and Square.java(adds *area()* method, which in turn calls *shape.area()*). He is unaware of the parallel changes by Alice. On completing his changes he faces a *merge conflict* and realizes that his copy of Rectangle is out of date and needs to be reconciled with changes in the repository. He also faces a *build failure* for Square.java since he used the earlier version of *shape.area(),* which lacked the *shape_type* parameter.

In $T_{B2,}$ Bob creates a new class Triangle.java. Bob ensures that he is calling the new *shape.area()* with the *shape_type* parameter set

as 'T'. However, Alice did not create functionalities for the area of a triangle in Shape.java, which defaults the shape to a rectangle. Bob's changes would therefore lead to a *test failure*.

For simplicity, we assume that Alice's and Bob's other tasks (T_{A2}, T_{A3}, T_{B3}) do not have any dependencies and are independent.

4. Approach

Our work builds on two key insights. First, workspace awareness will be more meaningful when it is aligned with a developer's cognitive unit of work – a development task (i.e., a bug fix, an issue, or a modification request). Second, one can generate an optimal task list, which minimizes the number of conflicts that a developer may face by analyzing ongoing and intended changes.

Here, we present our approach that realizes these two insights through a degree-of-conflict (DOC) model. Our degree-of-conflict model creates a single conflict metric per task. The DOC metric characterizes the number, type, severity, and status (planned edits, workspace edits, commit) of potential conflicts per program element (e.g., files, methods, or variables). From here on, we use the generic term "resource" to refer to a program element. The resource level DOCs are then aggregated for each task across all workspaces that contain the resource. This per task DOC can be used to create a task list that minimizes the possibility of conflicts.

A key step in our approach is to determine early those resources that are likely to be changed for a task, so the system can recommend an optimum task list. This can be done through several ways. First, we can initialize a task context with resources that we can identify by analyzing the description of the bug or issue in the bug repository. Second, we can use the description of a bug to identify similar archived bugs. Then use the resource list of the archived bugs to seed the originals bug's task context. Finally, we can recommend a social process wherein developers start their day by selecting and filtering the task context for their daily task list. We will explore a combination of automated and social processes to determine the best approach.

Figure 1 presents our proposed architecture underlying the DOC model. The *Workspace Wrapper* intercepts Mylyn-generated or user-generated events and sends them to the central server. The *Event Handler* component maintains the current state of changes per workspace in the *Event database* and archives older events in the *History* database. The *Event Handler* is also responsible for transmitting event notifications to relevant workspaces, where events regarding a file are considered relevant for a workspace if that workspace includes that file in any active task context. Our client extensions per workspace include the *Internal State* component that keeps a local cache of all events; the DOC component that calculates the DOCs for different entities (tasks, files, workspaces) per workspace; the *Task Recommender* that component determines an optimum list of tasks with minimum conflicts, which is then presented to the user via the *Visualization* widgets.

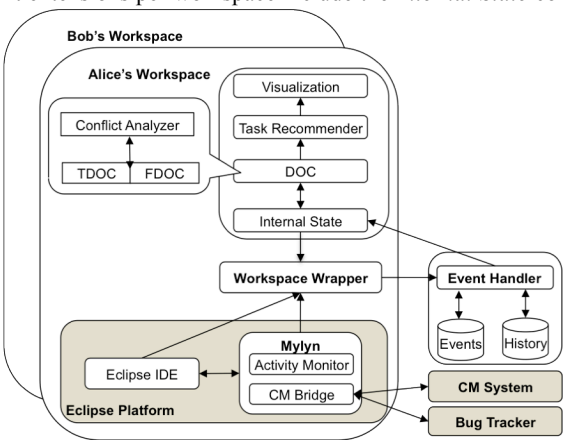

Figure 1. System Architecture.

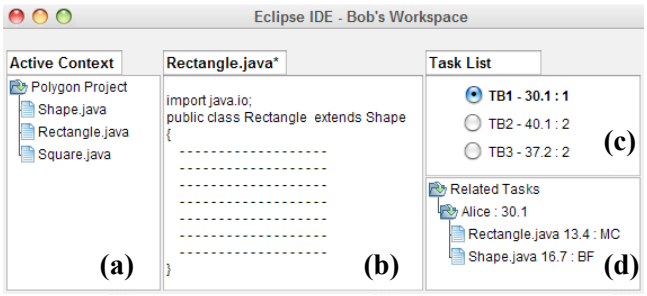

Figure 2. UI Mockup. (a) active task context, (b) editor, (c) user's task list with DOCs along with number of conflicting workspaces, and (d) related conflicting tasks for TB1

We are in the process of implementing our approach and investigating its feasibility. More specifically, we have designed the theoretical framework of the proposed system and some initial user interfaces through paper prototyping (see Fig. 2 for one such example). We plan on interviewing developers to obtain feedback on the UI and determining what information would prove helpful to them. We have also implemented the workspace wrapper, which intercepts user interactions from Mylyn and the CM system and stores them in a central event server. We have modeled the set of events that are necessary for our DOC model, which we are currently implementing. We plan on fine-tuning the DOC model based on our experiences and future use. In the rest of the section we present our degree-of-conflict model.

4.1 Degree of Conflict Model

The DOC model consists of three steps: (1) identify files that are being edited or will be edited for each task, (2) identify the kinds of conflicts and their impact, (3) create a model characterizing different user actions and conflicts into a single DOC metric.

For the first step, we determine program elements that are associated with a task by building on Mylyn's active task context functionality. More specifically, we will initially seed the task contexts with information from the bug repository. Then we depend on a user's interaction with Mylyn's task context for further refinement. The user interactions that we track through include "select", "edit", and "propagation" actions. For example, let us assume Alice in T_{A1} *selects* Shape.java and Rectangle.java. She then begins to *edit* Shape.java. In the meantime, Bob selects Square.java and Rectangle.java for T_{B1}. He realizes that both these files inherit from Shape.java so he investigates the methods provided by Shape.java, which is then added to the active task context as a result of a *propagation* event (see Table 2).

We categorize events concerning a resource into five classes: (1) propagation (P), (2) selection (S), (3) workspace edit (E), (4) check-in (C), and (5) removal (R). The first two events in the list are treated as planned changes and the rest as ongoing changes. Note that the initial seeding of a task context will be treated as propagation events. Each change is associated with a "change type" value: [P:0.5, S:1, E:0.7+, C:10, R:0*]. Our model keeps a running total of each change event per resource, per workspace to create a DOC model for the resource. Therefore, when a resource

is added because of a propagation or select event, the DOC number for that resource is 0.5 or 1, respectively. Each time a user saves her changes in the editor the model adds 0.7 to the resource total. A commit leads to a 10-point addition since it represents changes that definitely will need reconciliation. Removal of a resource causes the DOC to drop to 0. In our example, let us assume that Alice selects Rectangle.java, saves her edits twice, and then commits it, the DOC for Rectangle.java as calculated by Bob's workspace will be $f_{DOC_{Rec}} = 1 + 0.7*2 + 10 = 12.4$.

Additionally, our model increases the DOC values for changes that are conflicting (or may conflict). For example, changes that cause a merge conflict (MC) incur a 1 point increase. Build failures (BF) or test failures (TF), being more difficult to resolve incur higher penalties – 5 and 10 points, respectively. Using our example, Alice's changes to Rectangle.java will cause a merge conflict for Bob, so we add 1 more point making $f_{DOC_{Rec}}$ to go from 12.4 to 13.4. Points allocated to a conflict can also be weighted with the magnitude of the change, for example, multiplying a merge conflict or a build failure with the lines of non-comment lines causing the conflict. Currently, we do not consider such a weighting scheme in our model.

The DOC component for each workspace analyzes the change events transmitted by remote workspaces to calculate the file DOCs, f_{DOC}, for every file in the user's task. It then aggregates the f_{DOCs} across all workspaces that contain that file – F_{DOC}. That is, Bob's workspace calculates all changes from remote workspaces (in this case, only Alice) for each file in T_{B1} (Rectangle.java, Square.java, Shape.java).

The F_{DOC} of every file in a task (e.g., T_{B1} for Bob) is aggregated to create a $t_{DOC_{TB1}}$ score for that task. This is repeated for all tasks in the developer's list and is the basis on which the system can recommend tasks with minimum conflict.

For a selected task (or the active task) the model presents a list of the remote tasks and their associated DOCs that are affecting the task. For example, when Bob selects T_{B1}, all other tasks across all workspaces that conflict with T_{B1}, their DOC, and the type of potential conflict is presented. In our example, only Alice's task has a conflict with a DOC of 30.1 and includes a potential merge conflict (Rectangle.java) with DOC = 13.4 and a build failure (Shape.java) with DOC = 16.7 (1+0.7+10+5) and is displayed as: TA1: 30.1: MC|BF.

We have modeled our DOC metric on Mylyn's degree-of-interest (DOI) model [5]. Specifically, we retain the DOI metrics for workspace events, but, include new DOC metrics for check-in events and conflicts. We will fine-tune our model based on our experiences in building and using the prototype.

5. Conclusions

Current workspace awareness solutions are file-based and reactive, which make the user responsible for determining the impact of ongoing changes to one's current tasks and are inadequate for task planning. We propose a task-based awareness solution that can recommend an optimum task list to minimize conflicts for each user. Our DOC model keeps track of resources that are associated per task and uses this information to identify tasks that have the least DOC number, that is, the least potential for conflicts.

We will fine-tune our DOC model based on our own experience in using it. Currently, we are in the initial phase of implementing our prototype. One of the key challenges that we face is the initial seeding of the task context with associated resources, since the quality of these linkages impacts the quality of the task recommendations. We will explore both automated techniques such as analyzing archived bug reports as well as social processes such as developers linking the resources with tasks early on during bug triaging. Most likely a combination of both approaches will be needed to create an effective system.

Currently, our model recommends a task solely based on the DOC metric. In the future, we plan to extend this model to also consider time and other resource constraints, developer priority, and team requirements. We will explore modeling this problem as a constraint satisfaction problem so that the resulting task recommendations are optimum for a developer as well as for the team.

Finally, we will explore incremental techniques that analyze ongoing changes to identify emerging build and test failures, such that it is computationally inexpensive and scalable across large projects.

6. ACKNOWLEDGMENTS

This research is supported by the National Science Foundation under Grant CCF-1016134.

7. REFERENCES

[1] J. Biehl, *et al.*, "FASTDash: A Visual Dashboard for Fostering Awareness in Software Teams," in *Human Factors in Computing Systems* 2007, pp. 1313-1322.

[2] C. R. B. de Souza and D. Redmiles, "An Empirical Study of Software Developers' Management of Dependencies and Changes," in *Thirteeth International Conference on Software Engineering*, 2008, pp. 241-250.

[3] P. Dewan and R. Hegde, "Semi-Synchronous Conflict Detection and Resolution in Asynchronous Software Development," in *Conference on European Computer Supported Cooperative Work*, 2007, pp. 159-178.

[4] J. D. Herbsleb, *et al.*, "Distance, dependencies, and delay in a global collaboration," in *Proceedings of the 2000 ACM conference on Computer supported cooperative work*, 2000, pp. 319-328.

[5] M. Kersten and G. C. Murphy, "Mylar: A Degree-of-interest Model for IDEs," in *International Conference on Aspect-Oriented Software Development*, ed. Chicago, Illinois: ACM, 2005, pp. 159-168.

[6] M. Kersten and G. C. Murphy, "Using Task Context to Improve Programmer Productivity," in *Fourteenth ACM SIGSOFT International Symposium on Foundations of Software Engineering*, 2006, pp. 1-11.

[7] D. E. Perry, *et al.*, "Parallel Changes in Large-Scale Software Development: An Observational Case Study," *ACM Transactions on Software Engineering and Methodology,* vol. 10, 2001, pp. 308-337.

[8] A. Sarma, *et al.*, "Empirical Evidence of the Benefits of Workspace Awareness in Software Configuration Management," in *ACM SIGSOFT International Symposium on Foundations of Software Engineering*, 2008, pp. 113-123.

[9] N. A. Staudenmayer, "Managing Multiple Interdependencies in Large Scale Software Development Projects," Unpublished Ph.D. Dissertation, Sloan School of Management, Massachusetts Institute of Technology, 1997.