Notifications Management in Distributed Development Environments: a Case Study

Hani Bani-Salameh
Department of Software Engineering
The Hashemite University
Zarqa, Jordan
hani@hu.edu.jo

Clinton Jeffery
Computer Science Department
University of Idaho
Moscow, Idaho, USA
jeffery@hu.edu.jo

Abstract—**Interruptions in distributed development environments, caused by notifications, are increasingly necessary during collaborations between developers. This paper introduces the interruptions management problem and presents a solution integrated into the design and management of notifications in a social software development environment called SCI. The paper provides a classification of notification types, and reports results from a study testing SCI's effectiveness and usability. The study explored the effect of managing interruptions with the proposed features on the teams and their progress.**

Keywords- notifications; interruptions management; development environments; SCI

I. INTRODUCTION

A notification is an essential feature in a collaborative system that determines the system's capability to support different collaborative work activities. Notifications distinguish collaborative systems from other general multi-user systems where users are normally not notified of the actions performed by others, such as database management systems [1]. Notifications are visual alerts in the collaborative environment that communicate and deliver valuable information to developers and team members.

Interruptions disrupt developers' ongoing activities whether they are co-located or globally distributed. In co-located software development teams, social cues help developers to determine whether a partner can be interrupted or not. In a distributed collaborative development environment, the social cues are missing, and developers can use presence and awareness tools to create social features and cues that inform others about their availability, progress, and project activity.

Interruptions lead to behaviors that 1) affect the teams' progress and the systems' performance, and 2) cause human errors that may result in: performance time increase, workload increase, and generating errors [2]. In practice, collaborative software development revolves around interactions between developers where they spend most of their time working in a shared environment. Development in such an environment is adaptive to frequent changes happening to the team, and notifications are required to apprise the team of these changes.

Interactions between developers include file and project sharing invitations, and editing and debugging session invitations. Social features add interactions such as friend requests, group invitations, project discussions, emails, instant messages, and feed posts. In such an environment, users get a lot of invitation traffic, which tends to interrupt the users suddenly without regard for their current tasks [3], [4], affecting the development process and productivity. This creates a need for a framework that can transparently manage all kinds of invitations. "Tools for mediating incoming interruptions are necessary in order to balance the concentration required for software development with the need to collaborate and absorb information" [5].

Managing notifications is required for three main reasons. First, notifications can lead to interruption among the development community as mentioned in related research [6]. Second, the recipient of the invitation may be unavailable. In this case, invitations can be held until the invited user gets a chance to see them. The third reason is that managing invitations is required from a usability and scalability point of view. For example, an invitation to a collaboration session might intuitively be delivered via a popup dialog, but such intrusions do not scale well for busy users. Managing several opened windows and switching between them can be cumbersome especially if there are a large number of incoming invitations.

Ironically, in the absence of adequate notification management, many developers constantly interrupt themselves by checking for new mail. They may sacrifice valuable screen space by leaving their mail client open on part of the screen all the time, so as to see new messages immediately, or they may switch repeatedly between different clients to check notifications and application alerts. Either way, this slows developers' progress by increasing cognitive friction. This paper studies notifications within the context of SCI [7], a software development environment (SDE) that allows developers to work together. Developers can use SCI's collaborative tools to perform standard activities such as: programming, testing, and debugging. SCI supports social presence and messaging within teams and communities, roles and activities, as well as finding relevant information and people quickly. Also, this paper describes how the design and management of notifications in SCI is implemented and its effect on team collaboration and activities.

The rest of this paper is organized as follows. Section 2 presents research related to this work. Section 3 gives an overview of the SCI system. Section 4 provides a taxonomy of notification types in SCI. Section 5 introduces the design of the SCI system. Section 6 discusses the management process model. Section 7 introduces the proposed work use case model. Section 8 introduces the supported pending scenarios. Section 9 reports results from the case study. Finally, this paper sums up current state of this work and discusses future directions in Section 10.

II. Related Work

Interruption in distributed software development has been a significant area of focus for HCI researchers over several years. Research in this area includes studying the effects of interruption, solutions to manage interruption, and evaluating the available approaches [8], [9]. This section introduces some related work systems. Liebowitz [10] classifies the existing tools into six categories: triage techniques, leverage and recovery methods, intelligent systems, visual cues, helpful tips, and specialized software programs. Table 1 summarizes these categories.

GateKeeper [5], [11] is a plug-in based framework for collaborative software development. This framework handles and supports different types of interruptions from multiple sources. It is integrated within the Jazz collaboration tool that extends the Eclipse environment.

McGrenere et al [6] created a scenario-based notification architecture for the Jazz CDE. Their goal was to generate a design requirement for the notification management framework. Sen et al [12] introduced an alert filtering system called FeedMe that uses machine learning to infer alert preferences based on user feedback. Their focus on automated, spam-filter-like functionality complements SCI's focus on task- and user/source classification. Wang, Gräther, and Prinz [13] present a dynamic awareness system built to augment the collaboration tool BSCW. It uses a rule-based inference mechanism to follow the user's focus, and filters notifications based on relevance to the current task.

The research presented in this paper focuses on developing a solution for an effective management of notifications that often interrupt users with valuable information. The goal is to allow users to continue receiving all notifications that they desire, while trying to avoid being disrupted by their arrival. This work differs in several respects: First, the interruption management is integrated within a software development environment and has no need to connect to other systems to manage the notifications. Second, no configuration or settings needs to be incorporated by the developer/user.

TABLE I. Notification Tool Categories

Type	Definition	Example(s)
Internal	Notifications generated by the system about others' activities and status	Users' status, comments on walls, new projects, groups and others.
External	Notifications occur through components of the environment.	Emails, instant messages, artifact changes, collaboration requests (editing, debugging)

III. SCI

SCI [7] is a real-time collaborative programming tool with integrated social networking features. The ideas embodied in SCI are important for distributed software developers, e-learning and technical communities. SCI advances the state of the art for collaboration, coordination, and project management in software development.

SCI as a Social Development Environment (SDE) provides a wide range of facilities for synchronous and asynchronous collaboration and information sharing between team members. SCI is a single application that facilitates a harmony between many different subsystems providing semi-independent development functions. SCI includes a spectrum of collaborative tools that can benefit development communities, each of these tools adds its own value to the integrated environment, including Web logs (blogs), Mailing lists, Walls, Chat rooms, Whiteboards, and Wikis. SCI allows developers to work together; developers can use SCI's collaborative tools to perform standard activities such as: programming, testing, and debugging. SCI supports social presence and messaging within teams and communities, roles and activities, as well as finding relevant information and people quickly.

A. Use Scenarios

SCI can facilitate the education of novice programmers. It can help students taking introductory computer programming to collaborate and solve their programming problems, and to improve the student-instructor and student-tutor interaction. SCI can be used in the following scenarios for software engineering educational purposes:

- In the classroom the instructor can start a collaborative session and teach programming and the students can watch the instructor's actions in the IDE, and ask questions. This is valuable especially when a projection of the instructor's desktop is not available or not visible from all student seats.
- A student can use SCI as a stand-alone IDE.
- Students and their peers or teaching assistants can collaborate, discuss, or jointly debug their assignment; groups can use it on longer-duration group projects.

B. Illustrative Example

This section explains how SCI manages the delivered notifications through an illustrative example. In this example, the student is performing a programming task. The instructor usually offers assistance through either email and/or chat. The students' work primarily straddles the collaborative IDE for coding, email, chat, and newsfeed.

SCI can schedule notifications from the system's users that are generated by the features that the user is interacting with, as well as notifications generated automatically by the system. An email about an upcoming programming assignment or a help request arrives to the user's inbox, triggering the system to manage the request. SCI adds the request to a queue and prioritizes the received notifications according to their importance.

Five minutes later, the user finishes editing their code and switches to the chat widget to discuss a code change, and request assistance solving their coding problem. Later, the user resumes focuses on the programming task. After a while, a project's join notification request from the user's partners, is delivered and saved in the queue.

This example shows how the user can maintain an acceptable level of awareness while staying informed, and avoid significant interruption costs while focusing on their task. Through this process, users were allowed to finish their ongoing tasks and then react to the notification, rather than being interrupted while in the middle of coding.

IV. Notifications Classification

"Interruption is an inevitable part of multitasking users' work routine" [8]. In this paper we use the term "notifications" to mean the same thing as a system generated "interruption." The notifications can be broadly classified into two categories: external and internal. Table 2 shows a summary of these categories.

McFarlane [19] creates a taxonomy for interruptions using several different axes. This taxonomy uses several factors, including: interruption source, receiver's nature and characteristics, meaning of interruptions, effect of interruption, user's reaction to the interruption, and others. Like other related research studies [3], [6], [19], SCI classifies the notification types supported and delivered by the system. This classification is presented using a two dimensional matrix. The first dimension reflects the way the notification is delivered and/or initiated (how). The second dimension reflects who initiates the notification (who). This classification refines the notification taxonomy introduced by McGrenere et al. [6]. Table 3 shows a 2x2 classification matrix of SCI's notifications.

TABLE II. Notification Categories

Category	Definition
Triage Techniques	Methods that use triaging, filtering, or prioritization to minimize the interruptions effect [10], such as Gatekeeper [5], [11].
Leverage and Recovery Methods	"Approaches that involve calibrating interruption availability, handling and leveraging interruptions, and recovering." The mechanisms include: *immediate method*, *negotiated method*, *mediated method*, and *scheduled method* [14].
Intelligent Systems	Intelligent systems that can sense when a person is away or busy based on his or her typing and mouse activity, or that decide whether to interrupt the user based on the importance of the incoming message [15].
Visual Cues	The use of visual cues when dealing with the effects of interruptions [16].
Helpful Tips	Tips that deal with the information overload [17].
Specialized Software Programs	Researchers have been working on a research project named Achieve that "aims to develop software that helps users return to a task by reopening programs and websites last used when working on a project [18]."

TABLE III. Classification of SCI's Notifications

	Individual	Group
Direct	Occur as a direct result of the user's intentional actions.	Notifications related to the development process among teams.
Indirect	Notifications occur as a result of external or unintentional actions.	Notifications related to the development process among teams.

- Individual-Direct: notifications happen as a direct result of the user's actions. In general, any notification generated as a result of the users task can be treated as a direct notification. Other examples are:
    - An individual sends an instant message, an email, or starts a VoIP call to an individual or team.
    - An individual sends an announcement to a team.
- Individual-Indirect: notifications occur as a by-product of user actions. Examples of actions that generate notifications in the recipient(s) include:
    - A developer creates a project, and all members are informed that a new project has been created.
    - A developer commits changes to any of the project artifacts, and team members are notified about the code commit event.
    - A developer adds a new comment or changes the project wall of a bug report, and other members of the project are notified about the changes.
    - A developer leaves a collaborative session without warning his/her partner.
- Group-Direct and Group-Indirect: notifications related to the development process among team members.

V. Design

The primary emphasis in the design of the system is to minimize the transition effort from individual work to collaborative sessions and interaction with other users, and then back to individual work, so that transition can easily occur dozens of times during the course of a work period. A Facebook-style visible indicator of pending notifications and email are implemented, adding the ability to review the queue at one's convenience to see who has been waiting for what, and for how long, and to provide the developers with passive awareness notifications. Fig. 1 shows the awareness notification icons. Once the users get an invitation, they can access the notifications and emails by clicking on the icons, and a list of all the available notifications appears. Clicking on any of the list items causes a window to appear showing several actions.

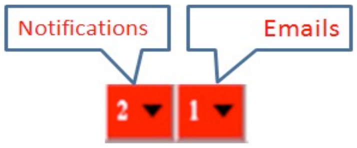


Figure 1. Notification and Email Icons

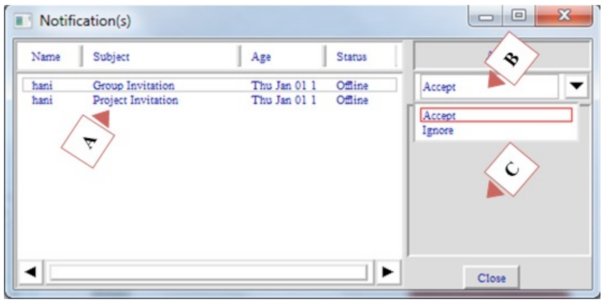
Users can accept, reject, discuss the invitation via a chat session, or email a reply. In collaborative IDE editing and debugging sessions, users may forward the invitation to another user or expert. For example, user **A** invites user **B** for a collaborative IDE session, and user **B** is busy assisting another team member. User **B** may forward the request to user **C**, who can accept or reject it. All invitations and requests are placed in the pending list until an action is taken, except the initiation and joining of the collaborative IDE sessions. The invitation is removed from the list when either the sender or the receiver of the invitation signs out of the system for any reason. The system provides a prioritization for invitations. Invitations from friends and project team members get higher priority than other invitations and requests, meaning that some kinds of invitations are more important (see the code below).

```
#
# A code snapshot that shows how the server
# prioritizes the notifications.
#
  if isFriend (uname, sname) then
     pinv.ppriority := 5
  else if isPartner (uname, sname, "sig") OR
          isPartner (uname, sname, "project")
       then pinv.ppriority := 4
  else if isFriend_of_friend (uname, sname)
  then pinv.ppriority := 3
```

The framework handles duplicate invitations smoothly. Users can browse their notifications by clicking on the notifications icon in Fig. 1 and choosing Browse, and a window appears with all the notifications (see Fig. 2).

Fig. 2 shows the pending invitations management window. Pending invitations are shown in a main window (**A**). The invitations (**A**) are sorted by their priorities as explained in the pseudo-code given above. Users can select the action to reply to the pending item from (**B**), while information about the selected pending item appears in (**C**). Also, users can browse the old emails by clicking on the emails icon in Fig. 1 and choosing Inbox, and a new window will appear showing the emails they received.

The framework requires a shared workspace that consists of participants and pending actions (entities). For every invitation or request that occurs, the participant needs to invoke an operation on an object such as a menu or button (by means of sending it a message) in the proposed framework. For example, if a participant initiates an editing session, the framework needs to reflect this activity by sending a message *add_pending()* to the system. Later, when the invited participant accepts/rejects the invitation, the framework sends a message *del_pending()* to remove the pending entity from the queued operations or entities.



Name	Subject	Age	Status
hani	Group Invitation	Thu Jan 01 1	Offline
hani	Project Invitation	Thu Jan 01 1	Offline

Figure 2. Pending Invitations/Requests Management

In this model, an event is created each time a pending invitation or request is sent. The system stores information about each activity in a log file, such as sender, receiver, time-stamp, type of activity. This information helps to keep track of the activities and ease their management.

VI. USE CASE MODEL OF NOTIFICATIONS MANAGEMENT

The proposed framework initiates the actions (Events) and delivers them to remote users. Thus it consists of both initiation and distribution functionalities. Events occur as a result of the users' actions from the input of the framework and are called *PendingEvents*. *PendingNotification* refers to the information (pending entities) delivered to the users from the output of the system (see Fig. 3).

Inputs from the users are not simply forwarded to the other users; pending notifications require assessment before they will be sent. For each *PendingEvent*, the system assesses the event nature and the target. Only if the *PendingEvent* appears to be new, not a duplicate, the framework will send the event. Once the user accepts the invitation/request, the framework will send the *PendingNotification*.

The pending framework can be described in terms of two key use cases: the initiation functionality and distribution functionality. Each pending activity requires two main actor types: the sender and the receiver.

VII. INTERRUPTIONS MANAGEMENT PROCESS

While performing a task, and on an ongoing activity, a developer receives an interruption signal that might be missed, or identified and interpreted as important or not. Based on the developer's assessment (interpretation) the notifications can be accepted, postponed, or rejected. The interruption management processes passes a set of phases including: *Task, Interruption Delivery, Interruption Identification, Decision, and Response (Outcome)*.

When a user is presented with an interruption activity several outcomes might occur, such as missing the activity and postponing the task. In a collaborative environment, where interaction is important for the success of a team, the situation can become more complicated since team members get multiple simultaneous interruptions that require scheduling and

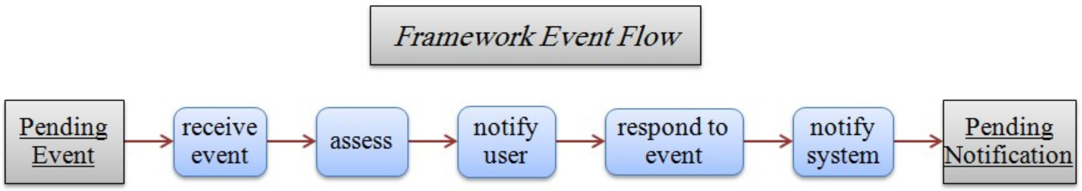


Figure 3. Events Managing Process

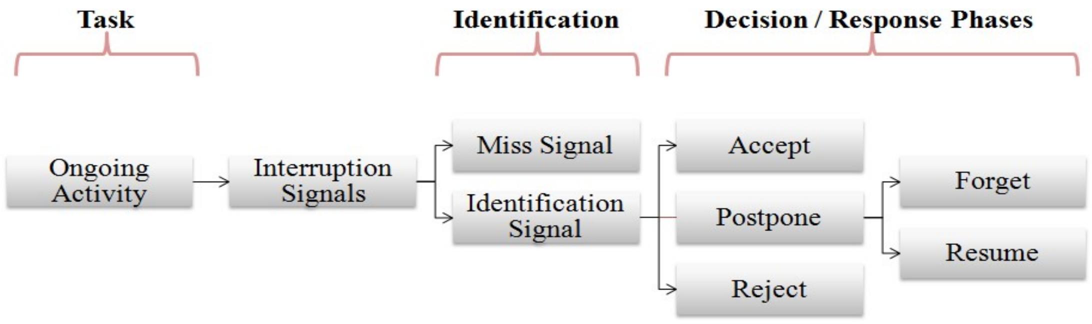


Figure 4. Possible User Response to Interruption Signals.

coordination [20]. Fig. 4 shows the possible user responses and outcomes in such situations.

VIII. Pending Scenarios

This section introduces all the supported pending invitation scenarios, and illustrates the activity using UML activity diagrams. As mentioned earlier, interactions among developers include collaborative session (editing, compiling, and debugging) invitations, friend requests, group invitations, and project discussions.

SCI supports general collaborative software development tasks, but it was built for computer science and software engineering education, particularly distance education, as well as niche programming communities and their developers. There are several requirements scenarios in computer science teaching environments (see Section 3.A). The following is a list of the common scenarios:

- *Invitation to start collaboration session:* the first event in a collaboration session an asynchronous request to start a synchronous interaction.
- *Collaboration session request to take turn:* During a collaborative session, developers take turns controlling the editor or debugger, while other developers watch and discuss the editing and debugging commands and messages. In this take turn action, a collaborator who is in a watch mode during the collaborative editing session requests permission to switch to the edit mode.
- *Friendship request:* another common type of interaction that takes place between developers is adding new friends. In order to build a social community circle, developers request other peers to become their friends. The invited developer has to reply to the friendship request sent by the others, and if the reply is positive, the developers become friends.
- *Project/Group membership request:* two important interactions that occur in a distributed environment are project and group join invitations. Those interactions are common practices between developers who work together finish specific tasks. Developers send invitations to join a project or group, or equest to be members of a specific project or group.

IX. Case Study

Software Engineering and related subjects teach students to do collaborative work and enhance their programming expertise. The following study was inspired by these mentioned goals and is implemented during the “CS120: Computer Science I” class at the University of Idaho.

The study evaluates the effect (net gain or loss in productivity) of integrating new features within the SCI development environment including the awareness notification on collaboration practices such as editing, and debugging. The case study aims to answer two main research questions about notification management:

- *How usable and useful is the notification management feature?*
- *How easy it is to notice the coming notifications and respond to them?*

A. Participants

The study targeted beginner-level C++ programmers who were willing to use the SCI development environment to finish the assigned tasks. Since the study does not depend on the nature of the groups, participants were randomly allocated to groups of two by the class instructor. The population consisted of 8 students, enrolled in the University of Idaho CS120 course, Computer Science I, in the summer of 2011.

B. Results

After completing each task, students were asked to fill out a survey to answer questions about their experience while using the SCI system. The participants were asked questions about their experience with the notifications and notification management framework. Following are two of the survey statements:

- ***(Q1)*** *It was easy to notice the notifications, emails, requests, and invitation that I received from other participants.*
- ***(Q2)*** *I find the notification icons important features in the environment.*

From the responses, 29% of the participants found it difficult to notice the supported notifications of the emails, requests, and invitations. 86% of the participants found the notifications icons important features in the environment. Table 4 shows some of the student's general comments.

The SCI system log files store information about the participants' activities. The log file *notification_response.log* provides an indication on how quickly the users responded to the awareness notifications. This helped determine to what extent the awareness notifications were helpful. Table 5 shows the format for this file.

Table 6 shows kinds of notifications generated by the participants during the case study. The information includes: type, times repeated, total time to respond, and the average response time.

From the data obtained in this log file it appears that the average response time for the notification was 9.37 minutes, and the median is 2.15 minutes. 2.5 minutes for a response time is above the expected response time. These values give an indication that the presented Facebook-style notifications need to be larger and more noticeable, and that SCI requires certain training/familiarization.

TABLE IV. STUDY PARTICIPANTS' COMMENTS

Comments	
C1	"Notifications for starting a collaborative session and request for drivers should be more visible."
C2	"User notification of emails and friend requests has to be actively searched out. If this flashed or changed colors this would be handy."

TABLE V. THE NOTIFICATION_RESPONSE LOG FILE FORMAT

Summary	
Type	What kind of notification (e.g. Project/Join)
From	User who sent the notification.
To	User to whom the notification is sent.
Sent	Time the notification is sent (seconds).
Responded	The time difference between sending the notification and responding to it (seconds).

C. Limitations

There are several limitations in this study. Firstly, the sample population consisted of college students. Corporate and government organizational development environments are different from those in universities; therefore the results should be cautiously generalized to the real development environments.

Secondly, the small sample size of the study presented in this paper limits the generalizability of the results. With a larger sample size, the results would have been different and more generalized. Further study with a large sample in different populations (e.g. students who are taking advanced computer and software engineering courses) will strengthen the results of this paper.

X. SUMMARY AND FUTURE WORK

Collaborative software development revolves around interactions between developers. This paper presents a general notifications management framework that controls and manages the emails and any other kind of notifications a distributed developer using SCI would receive.

This paper also classifies the notification types supported and delivered by the system to reflect both (1) the way the notification is delivered and/or initiated, and (2) who initiates the notification. This framework is designed to enhance the interaction between the team developers. The framework was applied to the design of a notification component for the SCI system, and tested during the evaluation process of the system. Also it was evaluated to be a usable and useful feature.

TABLE VI. NOTIFICATIONS GENERATED BY THE PARTICIPANTS

Notifications	**Times Repeated**	**Total (minutes)**	**Avg. Response Time**
Session/Edit	18	115.5	6.4
Group/Join	2	39.3	19.7
Friend/Add	9	116.9	13.0
Avg.			9.37
Median			2.5

The current notifications management implementation worked well enough for the system users and case study participants, and they responded to the notifications in a short time. A future improvement is by including an option so users specify if they can be interrupted by notification and collaboration invitation, or by allowing the system to predict/detect when a user is interruptible.

The improvement will include: 1) considering the relevance of the interruption content; 2) allowing the users to specify, filter, or limit the types of notification they get. A future work direction might focus on comparing the proposed framework with other notification management methods. Another improvement will focus on implementing the slow-growth approach [6] to help minimizing the response time to the notifications.

References

[1] H. Shen and C. Sun, "Flexible notification for collaborative systems," in Proceedings of the ACM Conference on Computer-Supported Cooperative Work, pages 77-86, November 2002.

[2] J. A. Rukab, K.A. Johnson-Throop, J. Malin, and J. Zhang, "A framework of interruptions in distributed team environments," in MEDINFO 2004: Proceedings of the 11th World Congress on Medical Informatics, Amsterdam: IOS Press, 12821286.

[3] T. Wilson and R. Miller, "Reducing the cost of interruption using gradual awareness notifications," unpublished. Available from MIT at groups.csail.mit.edu/uid/projects/slowgrowth/gradual-awareness.pdf.

[4] B. Bailey, J. Konstan, and J. Carlis, "Measuring the effects of interruptions on task performance in the user interface," in Proceedings of the IEEE Conf. on Systems, Man, and Cybernetics 2000, 757-762.

[5] Y. Dekel and S. Ross, "Eclipse as a platform for research on interruption management in software development," in Eclipse Technology Exchange Workshop at OOPSLA 04, Vancouver, BC, Canada, 2004.

[6] J. McGrenere, J. Li, J. Lo and E. Litani, "Designing effective notifications for collaborative development environments," in The Smart Internet 2010: pp. 65-87.

[7] H. Bani-Salameh, C. Jeffery and J. Al-Gharaibeh, "A social collaborative virtual environment for software development," in 2010 International Conference on Collaborative Technologies and Systems (CTS), DOI 10.1109/CTS.2010.5478525.

[8] S.T. Iqbal, "A framework for intelligent notification management in multitasking domains". PhD Dissertation, University of Illinois at Urbana-Champaign, Department of Computer Science, 2008.

[9] S.T. Iqbal, "Effects of intelligent notification management on users and their tasks," in Proceedings of CHI 2008, Florence, Italy, pages 93-102, April 5-10, 2008.

[10] J. Liebowitz, "Interruption management: a review and implications for IT professionals," in IT Professional vol. 13 no. 2, March 2011.

[11] Y. Dekel, "A plug-in based framework for research on interruption management in distributed software development," in Proceedings of the 2004 OOPSLA 04, Vancouver, BC, Canada, 2004.

[12] S. Sen, W. Geyer, M. Muller, M. Moore, B. Brownholtz, E. Wilcox and D. Millen, "FeedMe: a collaborative alert filtering system," in Proceedings of the 2006 20th Anniversary Conference on Computer Supported Cooperative Work, ACM New York NY, pp. 89-98.

[13] Wang, Y., Gräther, W., and Prinz, W. Suitable Notification Intensity: the Dynamic Awareness System. In Proceedings of the 2007 International ACM Conference on Supporting Group Work. ACM, New York, NY, pp. 99-106.

[14] S. Minassian, M. Muller and D. Gruen, "Diverse strategies for interruption management in complex office activities". IBM Research, 2004.

[15] M. Jackson, "Quelling distraction," in HR Magazine, vol. 53, no. 8, 2008; available at www.shrm.org/Publications/hrmagazine/EditorialContent/Pages/0808jackson.aspx.

[16] C. Speier, J. Valacich and I. Vessey, "The influence of task interruption on individual decision making: an information overload perspective," in Decision Sciences, vol. 30, no. 2, 1999, pp. 337-360.

[17] N. O'Connell, "Interruption overload," in Strategic Direction, vol. 24, no. 10, 2008, pp. 3 - 5. Emerald Group Publishing Limited.

[18] A. Ricadela, "High-tech time management tools," in Bloomberg Businessweek, March 8, 2009.

[19] D.C. McFarlane and K.A. Latorella, "The scope and importance of human interruption in human-computer interaction design," in Human Computer Interaction, 17(1), pages 1-62, 2002.

[20] S. Jayaraman, "Supporting monitoring and interruption management in complex domains through graded multimodal notifications," Ph.D. Dissertation, University of Michigan, 2011.