A CONTEXT-AWARE NOTIFICATION FRAMEWORK FOR DEVELOPERS OF COMPUTER SUPPORTED COLLABORATIVE ENVIRONMENTS

A Dissertation
presented to
the Faculty of the Graduate School
University of Missouri-Columbia

In Partial Fulfillment
of the Requirements for the Degree

Doctor of Philosophy

---

by
CHRISTOPHER J. AMELUNG

Dr. James Laffey, Dissertation Supervisor

MAY 2005

The undersigned, appointed by the Dean of the Graduate School, have examined the dissertation entitled.

A CONTEXT-AWARE NOTIFICATION FRAMEWORK FOR DEVELOPERS OF COMPUTER SUPPORTED COLLABORATIVE ENVIRONMENTS

Presented by Christopher J. Amelung

A candidate for the degree of Doctor of Philosophy

And hereby certify that in their opinion it is worthy of acceptance.

Dr. James Laffey

Dr. Dale Musser

Dr. Joi Moore

Dr. Feng-Kwei Wang

Dr. Jeffrey Uhlmann

DEDICATION

I dedicate this work to my loving wife Chastidy Dillon-Amelung and all of my family back home.

ACKNOWLEDGEMENTS

My greatest debt of gratitude goes to my mentors: Drs. Dale Musser and James Laffey. Without their guidance and support, this dissertation would not have been possible.

Dr. Musser introduced me to this Ph.D. program and gave me the confidence to fight conformity and think about research differently. Throughout my education, he never failed to inspire me with his creativity and his desire to always innovate. If it were not for Dale, I would have never started this work for which I am so proud.

Just as Dr. Musser started me on this journey, Dr. Laffey guided me to the end. I have never known someone so dedicated and willing to help others to grow in their own profession. Dr. Laffey inspired this research topic, encouraged me when my focus wavered, and never failed to offer his expertise whenever it was needed. I look forward to continued collaboration with these two individuals for they are not just my mentors, but my friends as well.

I would also like to thank everyone involved with the Sakai project, Dr. Fitzpatrick for her inspiring work on Locales, and Dr. Carzaniga for his work on Siena. I could not imagine this research without their prior contributions.

Finally, I would like to thank my wife for her patience and understanding. It has been a long road, but we made it.

To all of these people, I am forever in debt.

A CONTEXT-AWARE NOTIFICATION FRAMEWORK FOR DEVELOPERS OF COMPUTER SUPPORTED COLLABORATIVE ENVIRONMENTS

Christopher J. Amelung

Dr. James Laffey, Dissertation Supervisor

ABSTRACT

Researchers have identified user presence, awareness, and a sense of community as important components of Computer Supported Collaborative Environments (CSCE) (Dourish & Bellotti, 1992; Erickson & Kellogg, 2003; Moody, 2000; Prinz, 1999). To support user actions and interactions sufficient to create and sustain a sense of community, recent CSCE have been developed with notification systems to provide activity notifications to users. However, these notification systems typically transmit generic notifications as actions occur and do not provide mechanisms for analyzing and providing notifications based on user preference or social context. A challenge facing developers of CSCE is to create a notification system for delivering awareness information based on the ever-changing preferences, interests, and social contexts of users.

To address this challenge, this study articulated and advanced a theoretical framework for developers to use when integrating activity notifications into existing CSCE. The proposed framework is based on the importance of user preferences and social context and is derived from the Locales Framework (Fitzpatrick, 1998). The principles of this new development framework are Social Context, Awareness in Context, Activity Discovery, Trends in Activity, Meaning of Activity, and Notification Customization.

To evaluate the concepts of this framework, this study developed a context-aware activity notification system for an existing CSCE based on the framework's proposed principles. During the development process, it was determined that not only could the Framework for Notification be used to provide notifications based on user preference and social context, but the use of the proposed Framework afforded a richer understanding of the collaborative needs of users for both the theorists discussing the implications of activity notifications and the developers working to provide those notifications.

LIST OF TABLES

LIST OF ILLUSTRATIONS

TABLE OF CONTENTS

A Context-aware Notification Framework for Developers of Computer Supported Collaborative Environments

CHAPTER I

INTRODUCTION

Researchers have identified user presence, awareness, and a sense of community as important components of Computer Supported Collaborative Environments (CSCE) (Dourish & Bellotti, 1992; Erickson & Kellogg, 2003; Moody, 2000; Prinz, 1999). To support user actions and interactions sufficient to create and sustain a sense of community, recent CSCE have been developed with notification systems to provide activity notifications to users. However, these notification systems typically transmit generic notifications as actions occur and do not provide mechanisms for analyzing and providing notifications based on user preference or social context. A challenge facing developers of CSCE is to create a notification system for delivering awareness information based on the ever-changing preferences, interests, and social contexts of users.

Designing notification systems for CSCE is a challenging task that requires the researcher to consider both the technical requirements and the social implications of activity notifications. In a perfect world, a notification system would monitor every possible action occurring in an environment and compare that action with every possible condition to provide the most relevant notification

to a user. In reality, developing that system for online collaborative environments is impractical with the technology currently available. Instead, developers of notification systems must focus on the aspects of activity notification that are most relevant to the social requirements of users. To maintain this focus, developers need a conceptual framework to guide the development of notification systems for CSCE.

To facilitate the development of new, innovative notification systems, this study articulated and advanced a theoretical framework for development based on social context and user preference that could be used to integrate activity notifications into existing CSCE. That framework is called the Framework for Notification.

The Framework for Notification

The Framework for Notification presented in this study was designed for developers to use when creating notification systems for existing CSCE. The Framework was based on the importance of user preference and social context and requires the developer to consider why notifications are presented rather than just delivering a notification when an action occurs.

The principles of this Framework, presented in Table 1.1, were derived from an analysis and interpretation of the Locales Framework, from a study of the concept of embodied interaction, and from a review of existing network-based notification systems.

Table 1.1 Principles of the Framework for Notification

SOCIAL CONTEXT	The place where user actions and interactions occur. Social context partially determines the salience of awareness information. Social context is defined by the current membership, the collective goals of individuals, and the recent activity in the context.
AWARENESS IN CONTEXT	Deliver notifications to users when the notification is relevant to the user's social context.
ACTIVITY DISCOVERY	Allow the discovery of activity outside the user's current context to promote the formation of new social contexts.
TRENDS IN ACTIVITY	Maintain activity and notification histories to determine the impact notifications have on user actions and interactions. Trajectory of activity partially determines the salience of awareness information.
MEANING OF ACTIVITY	Provide mechanisms for users to interpret and construct meaning from the activity occurring in a context.
NOTIFICATION CUSTOMIZATION	Provide notification customization so the user has the final decision on the notifications received.

The Locales Framework (Locales) was based on Strauss' Theory of Action, Vygotsky's Activity Theory and the results of a systems engineer's study on the wOrlds system (Fitzpatrick, 1998). It was chosen to be the theoretical foundation for the Framework for Notification because (1) it was designed "to enhance support for social world interactions", (2) it was designed to be general and adaptable to meet the various requirements of computer supported collaboration, (3) it provides a common language for researchers to discuss interactions in a social world, and (4) because despite its general and adaptable design, it is difficult to interpret and to put into practice. The results of this study showed that the Framework for Notification is more accessible to developers

than Locales and consequently, affords the development of notification systems based on the principles of Locales.

In addition to Locales, the concept of embodied interaction was used to design the principles of this proposed Framework. Dourish, in his book “Where the Action Is”, defines embodied interaction as “the creation, manipulation, and sharing of meaning through engaged interaction with artifacts” (2001, p. 126). This study explains how activity notifications can be used to engage users with artifacts in a CSCE and how configurable notification preferences may impact the creation of meaning for those users.

During this study, the Framework for Notification was used to develop a context-aware activity notification system for the existing collaborative environment known as Sakai (2005). Throughout the development process, the Framework for Notification was analyzed based on the real-world problems encountered during development and molded into its current form. As a result of developing the notification system, it was determined that not only could the Framework for Notification be used to provide notifications based on user preference and social context, but the use of the proposed Framework afforded a richer understanding of the collaborative needs of users for both the theorists discussing the implications of activity notifications and the developers working to provide those notifications.

Statement of Problems

During an analysis of the Locales Framework and a review of existing notification-enabled systems, problems with the existing systems were found. The researcher was interested in solving the following problems:

- A lack of notification systems based on the actions and social context of users
- An absence of activity notifications for personal embodiment
- An inability to include all aspects of the Locales Framework into a single CSCE

Purpose of Study

The purpose of this study was to advance a theoretical framework for development that could be used to integrate activity notifications into existing CSCE to solve the problems listed above.

The general hypothesis tested was that developers following the principles in the proposed Framework for Notification could develop a notification system for a CSCE that supported the five aspects of the Locales Framework (see Chapter two). To test this hypothesis, the researcher carried out a software research study to develop and integrate a Context-aware Activity Notification System (CANS) into an existing CSCE using the proposed Framework for Notification.

Software Research

This study is software (i.e., development) research; consequently the concept of “software research” must be defined. According to Walker and

Bresler, "*development research* is disciplined inquiry conducted in the context of the development of a product or program for the purpose of improving either the thing being developed or developer" (Walker & Bresler, 1993). Software research is inquiry *by development* for the purpose of solving a problem. The purpose of *this* software research was not to determine the effectiveness or validity of social theory, but to design and test a framework for development to advance the notification capabilities of existing CSCE. For the purpose of this study, the definition of software research is defined as *the systematic investigation into a non-trivial activity or problem, within the context of development, that has not been solved or was poorly implemented and can now be carried out in a new, innovative way*.

Research Goals

> "In a [software] research project, the goal of a project evolves in the work. That is, the programmer starts with a broad idea, but cannot specify the detailed behavior of the program until it is written. He or she starts with a partial understanding, attempts to program some better understood corner of the project, then interacts with the resulting program to see in what direction to proceed." (Harvey, 1991)

The following three goals reveal the intended outcome of the study:

Goal 1: Articulate a development framework for integrating activity notification into existing CSCE.

Goal 2: Design and implement a notification system into a CSCE to advance the concepts of the proposed framework.

Goal 3: Design, develop, and test use-case scenarios to demonstrate the effectiveness of the proposed framework.

Definition of Terms

Activity Notification – (a.k.a. Event Notification) an indicator of activity in a CSCE.

Awareness – "an understanding of the activities of others, which provides a context for your own activity" (Dourish & Bellotti, 1992).

Development Framework – a theoretical framework, or guide, for researchers to use when developing software.

Development Research – (a.k.a. Software Research) the systematic investigation into a non-trivial activity or problem, within the context of development, that has not been solved or was poorly implemented and can now be carried out in a new, innovative way.

Computer Supported Collaborative Environment – a network-based collaborative system where users work, generate activity, and receive notifications.

Embodied Interaction – “the creation, manipulation, and sharing of meaning through engaged interaction with artifacts” (Dourish, 2001).

Embodiment – “the property of our engagement with the world that allows us to make it meaningful” (Dourish, 2001).

Event – “any significant change in the state of an observed object” (Fitzpatrick, Mansfield, et al.).

Event Notification System – (a.k.a. Event Notification Service) an application independent service that supports notification-based environments through the efficient routing of events between producers and consumers.

Framework for Development – a theory-based model used to guide developers in the construction of software.

Social Context – the socially constructed place for user actions and interactions defined by current membership, the collective goals of individuals, recent activity, and the communicative affordances of the technology.

Social Worlds – "the prime structuring mechanism for interaction" (Fitzpatrick, 1998).

Software Research – the systematic investigation into a non-trivial activity or problem, within the context of development, that has not been solved or was poorly implemented and can now be carried out in a new, innovative way.

Summary

In this study, a Context-aware Activity Notification System (CANS) was integrated into an existing CSCE using the principles of the proposed Framework for Notification for the purpose of examining, testing and revising the Framework. The outcome of this study is a Framework for Notification that advances the social interaction and activity notification capabilities of CSCE.

The following chapters describe the results of this study. Chapter two presents the Locales Framework to the reader and discusses the interpretation of Locales used in this study. Chapter three discusses the review of existing CSCE that was used to determine how previous systems approached the problem of providing activity notifications. Chapter four presents the proposed Framework for Notification and describes how the Locales Framework influenced the design of its principles. Chapter five discusses the development of CANS, presents the results of using CANS in a user-scenario evaluation, and discusses the results of a comparison of the notification capabilities of Sakai before and after CANS. Finally, Chapter six presents the implications of this study and provides suggestions for future research.

CHAPTER II

INTERPRETING THE LOCALES FRAMEWORK

The purpose of this study was to advance a theoretical framework for development that could be used to integrate activity notifications into existing CSCE. To achieve that purpose, this study adapted an existing theoretical framework that provided a common language for understanding user interactions in online social worlds. The existing framework is known as the "Locales Framework" (Fitzpatrick, 1998).

This chapter begins with a review of the Locales Framework (Locales), followed by consideration for embodied interaction, which was not articulated in Locales. Embodied interaction is defined by Dourish as "the creation, manipulation, and sharing of meaning through engaged interaction with artifacts" (Dourish, 2001, p. 126). Then, based on the recommendations of previous research, the Locales Framework is interpreted and localized to the task of integrating activity notifications into CSCE.

Locales Framework

The theoretical framework on which this study was based is the Locales Framework, by Geraldine Fitzpatrick. Locales was chosen for this study because (1) it was designed "to enhance support for social world interactions", (2) it was designed to be general and adaptable to meet the various requirements of computer supported collaboration, (3) it provides a common language for

researchers to discuss interactions in a social world, and (4) because despite its general and adaptable nature, it is difficult to interpret and to put into practice.

The Locales Framework was based on Strauss' Theory of Action, Vygotsky's Activity Theory and the results of a systems engineer's study on the wOrlds system (Fitzpatrick, 1998). Three primary assumptions guided the design of this framework. Those assumptions were: (1) user's actions constantly evolve based on the actions of others, (2) interaction occurs within *social worlds*, and (3) the *interactional* needs of users are important (Fitzpatrick, 1998). Based on these assumptions, Fitzpatrick designed the Locales framework around five fundamental principles, referred to as "aspects" by Fitzpatrick. Table 2.1 lists these aspects with descriptions taken from Mansfield, Kaplan, Fitzpatrick, et al (1997).

Table 2.1 Five Aspects of the Locales Framework

Locale Foundations	"about (1) providing adequate media and mechanisms in available domains to support sharing of objects, tools and resources, (2) supporting a group's notion of membership and related processes, and (3) facilitating appropriate privacy and access mechanisms"
Civic Structure	"concerns the facilitation of interaction with the wider community beyond a person's immediate workgroups and locales"
Individual Views	"looks at (1) the different individual views that can be held of the same locale, and (2) the individual's view over multiple locales"
Interaction Trajectory	"concerns all the temporal aspects of the group's locale and its associated people and entities"
Mutuality	"concerns the degree to which presence and awareness must be supported in collaborative work for the purpose of maintaining a sense of shared place"

While the general and adaptable nature of this framework is one of its strengths, according to Fitzpatrick, it is also one of its primary weaknesses. Fitzpatrick indicates that a "considerable creative interpretative effort" is required to use the framework and the processes for working with the framework need to be made more accessible to accommodate widespread use (1998, p. 225). In addition to this noted weakness, applying the concept of embodied interaction to Locales could enhance the interactional capabilities of a CSCE. The importance of embodied interaction is discussed further in the next section.

Interpreting Locales

Applying the concepts of the Locales Framework to the development of notification systems requires clarification and interpretation of its five aspects. This study's interpretation of these aspects along with a description of embodied interaction is presented in this section.

Locale Foundations

The first aspect of the Locales Framework is Locale Foundations. The Locale Foundations aspect characterizes "the social world or worlds of interest" and "the site and means that become the locale they [users] use to achieve their shared work" (Fitzpatrick, 1998, p. 98). To interpret that sentence into a framework for development, one must understand the meaning of social world, site and means, and locale.

A social world is the “prime structuring mechanism for interaction” and the site and means that enables interactions (Fitzpatrick, 1998). A social world is defined by its:

- Collective goal or shared purpose
- Membership
- Duration and life-cycle
- Internal structure, roles and culture of members
- Focus of work and tasks

Site and means involves the technologies that afford interaction in a domain of activity. A locale is comprised of the collaborative needs of the social world and the site and means used to meet those collaborative needs.

Because this study is about integrating activity notifications into *existing* CSCE, one of the first tasks a developer must perform is the identification and analysis of the social worlds and the existing site and means of the CSCE. It is important to remember that social worlds can be both composed of sub-worlds and simultaneously members of larger communities so an accurate identification is important. Guidelines for identification are presented with the Framework for Notification (Chapter four).

Civic Structures

Civic Structures includes the structure of locales, the relationship between locales including the discovery and navigation between locales, and the creation and deletion of locales. Civic Structures is the aspect of the Locales Framework

that places the membership of social worlds in a larger context and recognizes the importance of interactions between locales within social worlds. To determine the structure and relationship of locales, Fitzpatrick suggested that researchers create a conditional matrix to analyze the existing civic structures (1998). In addition, it is important to analyze the CSCE's existing capabilities to interact between locales. Without an understanding of the civic structures in the CSCE, it is difficult to provide meaningful activity notifications between locales.

Individual Views

The Individual Views aspect focuses on the needs of individuals in social worlds. It is important to realize that a social world is "comprised *[sic]* of individuals who each bring their own perspective, history, needs and agenda to the group" (Fitzpatrick, 1998, p. 115). Even when individuals share the goal of a social world, they still use the artifacts of that world in their own unique ways. During their involvement in the social world their perspectives and goals often change and the environment must be adaptable to these changes.

The Individual Views aspect also emphasizes that individuals are rarely involved in only one locale. Instead individuals are members of multiple locales and social worlds independent of their current locale. Consequently, a CSCE must accommodate activity notifications from multiple locales simultaneously.

Interaction Trajectory

Interaction Trajectory is "about the social world in action in its locale(s), and the co-evolution of action, locale, and social world as the trajectory unfolds"

(Fitzpatrick, 1998, p. 123). According to the Interaction Trajectory aspect, actions only make sense when actions and interactions are viewed as a whole over time. This aspect is not just about providing a history of activity, but describes the importance of past, present, and future interactions.

User's actions in a CSCE are part of the process through which users achieve their goals and actions are interrelated with other actions occurring in the locale. Recognizing that activity is part of a process and realizing that the knowledge of activity impacts the goals and outcomes of processes is important to the development of a CSCE notification system. During the development of an activity notification system, the Interaction Trajectory aspect plays a significant role in supporting asynchronous activity and in providing notifications based on recent, and possibly, upcoming actions of users.

Mutuality

The principle of Mutuality states, "for interaction to happen in a locale, there are basic requirements for presence and awareness" (Fitzpatrick, 1998, p. 134). Fundamentally, this aspect identifies the need for participants to be represented (present) in an environment and a way of knowing about other's activity (aware) in that environment. Cooperative work in social worlds requires mutuality. Likewise, an activity notification system must address the concepts of presence and awareness as described in the Mutuality aspect.

Prior to development, a researcher should determine what mechanisms currently exist and what mechanisms are needed to provide presence and

awareness. It is important to remember that artifacts are a primary representation of individuals in a virtual environment and can be used to define user presence. Providing information about actions occurring on artifacts can be used to provide awareness to users. It is also important to realize that as Fitzpatrick indicates "there are no intrinsic supports for *mutuality* in the virtual medium" because "geometric space and corporeality have no meaning" in a virtual environment (Fitzpatrick, 1998, p. 140). While it is difficult to represent the natural world in a virtual environment, a natural representation is not a necessity for presence and awareness. There are several strategies for supporting mutuality that are revealed in this study's review of prior work.

Embodied Interaction

Dourish, in his book "Where the Action Is", defines embodiment as "the property of our engagement with the world that allows us to make it meaningful" (2001, p. 126). Only through an engagement with the CSCE can users create meaning from the activity occurring within it. Embodied interaction is defined by Dourish as "the creation, manipulation, and sharing of meaning through engaged interaction with artifacts" (2001, p. 126). In other words, users create a sense of meaning by interacting with artifacts in a CSCE.

The CSCE cannot generate meaning for users; it can only provide information that can be used by users to generate meaning. It is up to each individual user of a CSCE to construct their meaning from activity. It is up to a notification system to provide that information through the delivery of activity

notifications. This aspect also reveals that knowing when and how others interact with one's own artifacts may generate a greater sense of presence and worth in the environment.

Summary

This chapter presented an interpretation of the Locales Framework and provided various suggestions on how a developer should apply each aspect of this Framework to the development of a notification system. Those suggestions were:

1. Identify and analyze the existing social worlds and site and means of the CSCE.
2. Remember that social worlds may be composed of sub-worlds and simultaneously members of larger communities.
3. Create a conditional matrix to determine the existing civic structures and their influence on each other in the CSCE.
4. A CSCE must accommodate notifications from multiple locales.
5. Individuals must be able to configure their own notification profiles.
6. Activity within a CSCE should be recorded so that trends can be identified.
7. User artifacts are a primary representation of individuals in CSCE.
8. Knowing how others in a CSCE interact with one's own artifacts is important.

The next chapter presents a review of existing CSCE and compares those environments to the Locales Framework to determine the non-trivial problems that were not solved before or were poorly implemented.

CHAPTER III

REVIEW OF RELEVANT WORK

This review of relevant work began with an analysis of existing computer supported collaborative environments (CSCE) that provided awareness information through activity notifications (see Appendix A). From this analysis, three systems were chosen for their applied notification strategies and reviewed in detail. The systems were chosen because each one represented unique strategies for implementing activity notifications that were relevant to this research. During this review, these systems were used to develop a better understanding of the Locales Framework by identifying how some Locales aspects have been met and why some have not. This chapter ends with the conclusions reached during this review of relevant work and a comparison of how these existing systems addressed the principles of the Locales Framework. But first, a brief history of collaborative environments is presented.

A Brief History of Collaborative Environments

The first occurrence of social interaction via networked computers transpired in 1962 at MIT (Leiner, Cerf, et. al., 2000). Since that time, an increasing number of people have used networks and network-based systems to communicate on work, education, and leisurely activities. Today, the World Wide Web, and its related technologies, supports hundreds of millions of users.

Because of this immense popularity of networks, especially the Internet, many network-based systems have emerged.

Originally, these networked systems focused on information-management activities and were not created for the promotion of community-centered activities. However, an emerging shift from information-management to community-centered environments has been found. Ishida emphasizes the importance of social interaction and community-centered activities in networked environments with the statement: “The term *community* is being used as a metaphor for the next stage of computing technologies, including the methodologies, mechanisms and tools for creating, maintaining and evolving social interaction in human societies” (Ishida, 1999).

Through the technological advancements of the Internet and the realization that social interaction in networked environments is important, several network-based systems have been designed or modified to focus on social collaboration and community-based activities. The next section of this Chapter describes three of those environments.

Existing Computer Supported Collaborative Environments

From the systems listed in Appendix A, three systems were chosen for a detailed review based on two important factors: (1) applied framework for awareness and (2) technological implementation. These systems were chosen because the type of awareness information they provided and the methods of implementation they used were relevant to this research. Several strategies

employed by these systems were adapted and advanced for the notification system developed during this study.

This review began with an analysis of the features and technical achievements of each environment followed by a comparison of these systems to the Locales Framework. The fundamental goals of these systems were evaluated against the Locales Framework to attain a better understanding of how these systems adhered to the principles of activity notification according to Locales. It should be noted that failure to meet aspects of Locales is not an indication of poor software, but instead reveals the need for this research by identifying areas of improvement in CSCE.

iScent

iScent is an acronym for InterSubjective Collaborative Event Environment. iScent is a web-based environment that strived to meet the awareness needs of large software organizations through “the use of automatically collected, hypermedia-enabled event trails” (Anderson & Bouvin, 2000). The iScent architecture is framed around the importance of “I know that you know that I know”, or intersubjectivity (Anderson & Bouvin, 2000). Another way to think of intersubjectivity is the notification system not only notifies users of activity, but notifies users when they are being watched and by whom.

Intersubjectivity occurs within iScent because when someone is notified about an action, the person who performed the action is notified about the notification. Traditionally, notification systems only send out a notification to

indicate activity has occurred. However, iScent sends both the activity notification and the notification of notification. Refer to Table 3.1 for a step-by-step walk-through of intersubjectivity in iScent.

Table 3.1 Sequence of Notifications to Achieve Intersubjectivity

Step 1: User X performs action Y
Step 2: User Z is notified about Step 1
Step 3: User X is notified about Step 2

How It Works

The iScent framework supports awareness by identifying activity as an element of conversation. Awareness of activity-based conversations is supported through iScent's capacity for intersubjectivity. iScent achieves awareness and intersubjectivity by linking applications, event histories, event monitoring, and event notification in a dynamic and user-configurable way. Table 3.2 lists the major components of the iScent configuration.

Table 3.2 iScent Components

Component	Description
iScent Aware Application	An application on a user's computer that generates iScent events.
Sink	Provides an interface for propagating events to interested users. Sinks are responsible for event persistence and are interconnected through an event notification system called Siena.
Trail Viewer	Responsible for providing hypermedia-enabled event trails to users.
Watchdog	Monitors events based on user-configurable settings.
Kennel	A collection of watchdogs.
Siena	Event notification system employed by iScent.

An iScent environment is composed of a collection of iScent applications on users' personal computers, a collection of sinks and kennels on remote servers, and an event notification system to route events between sinks. Figure 3.1 models a typical iScent configuration. Figure 3.1 is reproduced from Anderson and Bouvin's article "Supporting Project Awareness on the WWW with the iScent Framework" (2000).

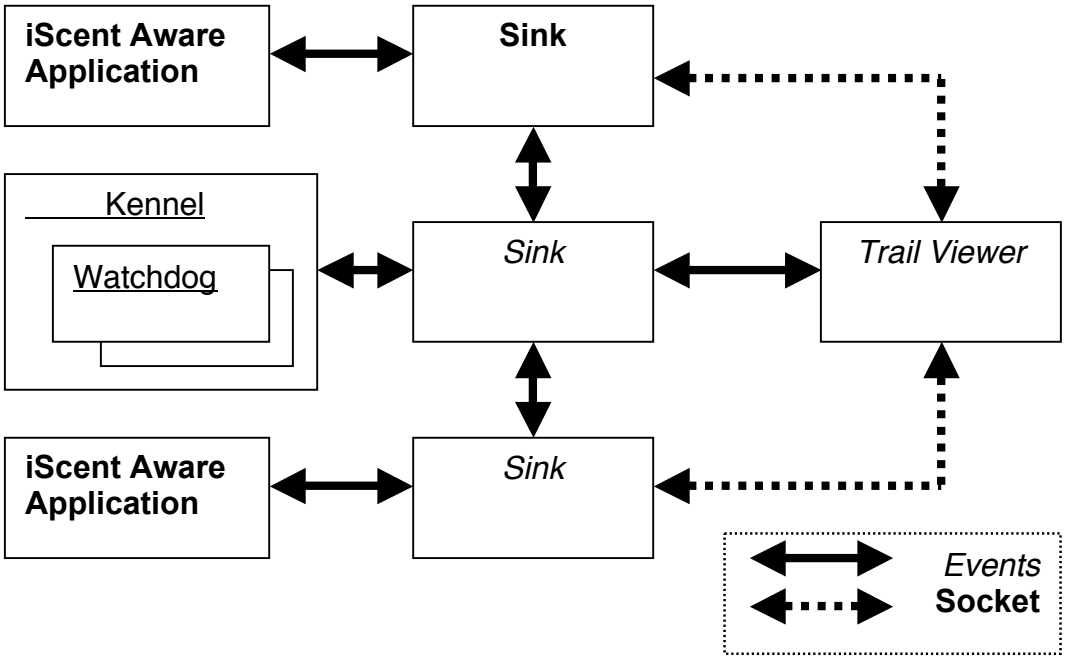


Figure 3.1 iScent Architecture

The first component of iScent is the iScent aware application. As of 2000, the only iScent application available was a web browser found in the Arakne Environment (Anderson & Bouvin, 2000). While, theoretically, any application can generate an iScent event, the iScent framework does not allow notifications to be embedded in those applications. As a result, event notifications must be presented in a separate window.

An iScent sink collects and stores iScent events. Sinks, and the event notification system, are the backbone of the iScent framework. Multiple sinks are joined through the event notification system and employ that system "to propagate events to the interested parties" (Anderson & Bouvin, 2000). One interesting feature of sinks is how they can be added to the iScent environment as demand increases. Once a sink is connected to the environment, it presents its stored events for distribution among other sinks. Through research and testing, Anderson and Bouvin found "that associating a sink and an event server with each Web server in a particular environment is a good rule of thumb for configuring iScent frameworks" for optimal performance (2000). Another interesting feature of iScent sinks is a sink can be installed locally on a user's computer, which allows users to work offline. Upon connection to the network, and the iScent environment, recent activities and notifications are shared among connected sinks.

The next component of the iScent configuration is the trail viewer. In the iScent framework, a collection of events stored in sinks are called event trails. The trail viewer is a Java servlet that provides a hypermedia-based interface to view, edit, and query those event trails. In other words, the iScent trail viewer is the interface for viewing and interacting with event histories. Additionally, the trail viewer provides users with the ability to set watchdogs on certain event trails.

Watchdogs, another component of iScent, are created to monitor the occurrence of certain activities in a person's event trail. When the specified

activity appears in the monitored event trail, the watchdog dispatches a notification to the person who created the watchdog. Consequently, a user is notified about the activity of other users. At the same time, a notification is also sent to the owner of the event trail. Through this dual notification, intersubjectivity is achieved.

Another iScent component is the kennel. An iScent kennel is a collection of watchdogs. The kennel allows users to monitor multiple activities simultaneously by allowing the user to have multiple active watchdogs. A single watchdog monitors the occurrence of a single event. Through the use of kennels, iScent allows users to monitor multiple events.

The final component of the iScent configuration is the event notification system called Siena. Siena (Scalable Internet Event Notification Architectures) is a content-based routing event notification system (Carzaniga, 1998). iScent utilizes the Siena system for all routing of events between iScent components. Additional information related to Siena is provided in Chapter five.

Relevant Features

iScent incorporated several notification features in its framework that were relevant to this research. This section examines those features in detail. The features of iScent relevant to this research are presented in Table 3.3.

Table 3.3 iScent Features

Feature	Description
Event Trails	Hypermedia-enabled event histories that can be monitored for notification and stored for later use.
InterSubjectivity	“I know that you know that I know”.
Large Group Sizes	Supports large numbers of users by being based on standard web-based technologies.
Scalable Architecture	Sinks are plug-and-play. Additional sinks can be dynamically added to support demand.

The first important feature of iScent related to this study was event trails. Event trails are the recorded history of activity that occurs within iScent applications. As discussed, sinks store the activity histories and trail viewers provide access. Access to the histories provides users with a mechanism to review the past actions of users. In addition to this feature, event trails can be monitored to provide real-time activity notification and intersubjectivity.

“I know that you know that I know”. Anderson and Bouvin proposed this phrase as the definition of intersubjectivity. In iScent, the researchers focused on the importance of *being understood* in conversation by incorporating the ability to be notified when someone is notified of his or her activity. Whether or not knowing “that you know that I know” actually facilitates understanding was not concluded in Anderson and Bouvin’s research, but the concept of intersubectivity was relevant to this study.

The third relevant feature of iScent was its ability to support a large number of users. iScent achieves this important feature by basing its architecture on standard web-based technologies. As previously stated, the web and its

related technologies currently support hundreds of millions of users and are growing in user capacity every year. Currently, a network-based collaborative work environment would not propose to support such a vast number of users, but developing a collaborative system on these existing technologies affords growth and expandability of the CSCE.

The final feature that was identified as relevant to this research is closely related to the previous feature: dynamic scalability. This feature is presented separately because of its implementation in the iScent environment. The dynamic scalability of iScent results from the plug-and-play nature of iScent's sinks. As previously stated, in an ideal iScent configuration, a sink resides on its own server and multiple sinks can be added to iScent as user activity increases. To be integrated into the system, the sink only needs to be aware of one other sink. From that single connection, the capabilities of a sink and the server on which it resides, are made available to the entire awareness system.

Groove Workspace®

Groove Workspace® (Groove) is desktop software that facilitates small group interaction through the use of peer-to-peer networking (Groove Networks, 2003). Groove is tightly integrated with Microsoft Office, and utilizes email as a basic collaboration tool. Unlike many CSCE, Groove is built on a decentralized architecture, which simply means that Groove does not rely on a central server for collaboration between clients. Groove works like the infamous Napster by

sharing files and activity notifications between networked computers without requiring a central data store.

How It Works

According to Groove Networks, "[a]ll application logic and data is stored locally on the desktop of each member of a shared space" (Groove Networks, 2003). As a result, the complete set of data in the system is duplicated on every user's computer. When activity occurs, data and notifications are sent as XML messages using the peer-to-peer network. Only the changes in data are sent with each notification, thereby preventing the need to send all data on every notification delivery. Table 3.4 list the primary components of Groove and provides a brief description of each.

Table 3.4 Groove Components

Component	Description
Microsoft Office/ Lotus Notes	Collaborative desktop applications used by Groove
Microsoft SharePoint Portal	Server used to "manage documents, lists, view, and configuration information"
Microsoft SharePoint Team Services	Provides collaboration through the use of team and shared websites.

The software used by Groove is primarily Microsoft Office and Lotus Notes, but an application programmer interface is available for third-party software development. The applications of Groove employ the capabilities of Microsoft's SharePoint Team Services for collaboration. According to the Microsoft Corporation, "SharePoint Team Services used a hybrid model of Web

server, file system, Windows registry, and SQL Server-based storage to manage documents, lists, views, and configuration information" (2003). The SharePoint Portal component enables the peer-to-peer communication capabilities of Groove. In other words, Groove Networks used the capabilities of Microsoft's SharePoint services to develop a collaborative work environment for small business groups. The technological features relevant to this study are presented in the next section.

Relevant Features

Groove utilizes several interesting activity notification features useful to a CSCE. This section examines the features of Groove relevant to this research. An overview of these features is presented in Table 3.5.

Table 3.5 Groove Workspace® Features

Bandwidth optimization	Only the changes to documents are sent over network; Allows users to choose desired notifications and updates.
Decentralized architecture	Places activity primarily on users' computers leaving the server to act as a router.
Live co-editing	Two or more users can edit a Word document in real-time.
Live co-viewing	Groups can view documents or PowerPoint presentations in real-time.
Offline editing	Ability to work offline with automatic synchronization on next network connection.
Online awareness	Shows online users and activity of those users.
Roles & permissions	Allows managers to set access permissions within shared spaces.
Security	Encrypts data transferred over the network.

The first feature of Groove relevant to this study was its ability to support real-time, multi-user editing of documents. Users of this software can work on the

same Microsoft Office document simultaneously and see the changes made by others in real-time.

Another relevant feature of Groove was bandwidth optimization. Bandwidth optimization is achieved by sending only the recent changes made in a co-edited document instead of transmitting the entire document over a network every time an action occurs. Groove only sends the new information and an updated document is created by combining the new changes with the previously unchanged document, thus minimizing network traffic and enhancing network performance.

A third feature of Groove that was relevant to this study is the ability to allow users to work offline. Users can work offline on their personal computer. When the user reconnects to the Internet, Groove automatically synchronizes the environment by uploading any changes made by the user and downloading any changes made by others.

Sticker

Sticker, a desktop Java applet, allows individuals "to send and receive short text messages via the Elvin event notification service" (Phillips, 2002, p. 2). Sticker is one of many descendants of Tickertape and offers advanced features over its predecessors (Distributed Systems Technology Centre, 2003). Sticker is built on the latest version of Elvin (version 4) and supports online presence, secure messaging, and advanced message management.

How It Works

Sticker is a Java applet that resides on a user's desktop and utilizes the event notification service called Elvin for the distribution of event notifications. Sticker detects events by monitoring user-specified Sticker groups and/or by searching for keywords, or expressions, occurring within the content of notifications that occur throughout the system. Sticker groups are available in two forms: chat groups and news groups (Phillips, 2002).

In a chat group, the Sticker client acts as both an event producer and consumer. Individual users produce events by sending them to a specific Sticker group, whereby other users in that group receive, i.e., consume, those events. Used in this manner, Sticker offers the interactive capabilities of a chat or Instant Messaging application. Figure 3.2 shows the two-way communications available in a typical chat group.

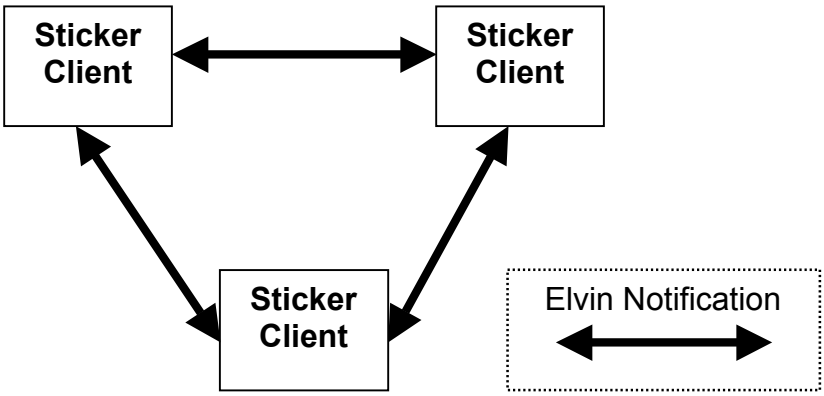


Figure 3.2 A Typical Bi-Directional Chat Group Configuration

In a news group, a Sticker client receives event notifications that are automatically generated from various external sources, typically websites and

UseNet groups. In this mode, Sticker functions as a general event notification application by consuming notifications without any direct action required by the user. Figure 3.3 models the one-way communication scheme available in a news group.

In addition to listening to specific Sticker groups, users can generate expressions that can be monitored throughout all Sticker groups. This feature affords users with the ability to customize the notifications they receive.

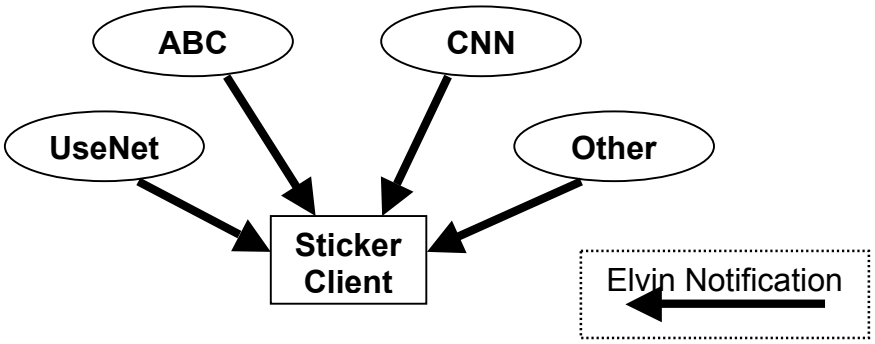


Figure 3.3 A Typical Unidirectional News Group Configuration

Whenever a notification is sent to a Sticker client, that notification is displayed as a scrolling Sticker message in the Sticker window. The notification contains an identifying icon, the name of the group it belongs to, the name of the entity that created the notification, and the notification's message. A notification will continue to scroll until it is deleted or it expires. If the notification is allowed to expire, it will change color over time to indicate age. This *time-sensitive* feature is a common attribute of many Elvin-based applications (Distributed Systems Technology Centre, 2003).

While events are automatically produced by applications in news groups, a user specific event requires the user to physically compose the event notification. When a user produces an event, he or she can choose the group to send it to, the amount of time before it expires, and the actual notification message. In addition, the user can encrypt the message and attach external files to the notification.

Relevant Features

Sticker incorporates many features relevant to this study. This section examines those features in detail. The features of Sticker relevant to this research are presented in Table 3.6.

Table 3.6 Sticker Features

Producer/Consumer	Applications can act as producers and/or consumers of event notifications.
Notification filtering	Notifications can be monitored for expressions thereby providing more control over notifications received.
Online presence	Shows list of online users with ability to add users to presence groups
Secure messaging	Provides data encryption through the use of Elvin's security features
Supports customized applications	API available for the development of customized Sticker applications
Time sensitive notifications	Notifications can be set to expire on a certain date and, over time, change color to indicate age.
Minimal space requirement	Requires a small amount of screen space and can be placed in the periphery.

The producer/consumer model of Sticker was a feature relevant to this study. A producer is an application capable of producing event notifications.

When an event is produced, the application transmits a notification to the event notification system for distribution to event consumers. A consumer is an application capable of consuming event notifications.

Another relevant feature of Sticker was its ability to allow consumers to subscribe to specific events, thereby reducing the number of unwanted notifications sent to consumers. Sticker refers to this feature as *notification filtering*.

The third relevant feature of Sticker was its ability to provide time sensitive notifications. Notifications can remain active until a user-specified deletion date. As the deletion date approaches, the notification changes color to signify age. In addition to these features, Sticker also allows for secure messaging and the development of customized applications.

To conclude this review of existing collaborative environments, Table 3.7 shows the features in iScent, Groove, and Sticker that were found relevant to this study and were used to influence the design of this researcher's proposed Framework for Notification and Context-aware Activity Notification System. The next section of this Chapter discusses how these existing environments compared to the Locales Framework.

Table 3.7 Relevant Features of Existing CSCE

Feature	Description	CSCE
Bandwidth optimization	Only the changes to documents are sent over network; Allow users to choose desired notifications and updates instead of receiving every notification.	Groove
Decentralized architecture	Places activity primarily on users' computers leaving the server to act as a router.	Groove

Event trails	Hypermedia-enabled event histories that can be monitored for notification and stored for later use.	iScent
InterSubjectivity	“I know that you know that I know”.	iScent
Large Group Sizes	Supports large numbers of users by being based on standard web-based technologies.	iScent
Live co-editing	Two or more users can edit a Word document in real-time.	Groove
Live co-viewing	Groups can view documents or PowerPoint presentations in real-time.	Groove
Minimal space requirement	Requires a small amount of screen space and can be placed in the periphery.	Sticker
Notification filtering	Notifications can be monitored for expressions thereby providing more control over notifications received.	Sticker
Offline editing	Ability to work off-line with automatic synchronization on next network connection.	Groove/ Sticker
Online awareness/ presence*	Shows online users and activity of those users.	Groove/ iScent/ Sticker
Producer/Consumer	Applications can act as producers and/or consumers of event notifications.	Sticker
Roles & permissions	Allows managers to set access permissions within shared spaces.	Groove
Scalable architecture	Sinks (refer to section iScent for more information) are plug-and-play. Additional sinks can be dynamically added to support increased demand.	iScent
Secure messaging	Encrypts data transferred over the network.	Sticker/ Groove
Support customized applications	Supports continued development through a set of APIs.	Sticker/ Groove
* Feature supported in limited fashion		

A Comparison to Locales

In this section, iScent and Sticker are compared to the Locales Framework to determine which aspects of Locales were met in previous work and in what ways notification systems can more fully meet the requirements of Locales. Groove was not included in this evaluation because it was studied for its unique technical approaches to activity notifications and does not map well to the Locales Framework.

Locales Foundations

The Locales Foundations aspect is “about (1) providing adequate media and mechanisms in available domains to support sharing of objects, tools and resources, (2) supporting a group’s notion of membership and related processes, and (3) facilitating appropriate privacy and access mechanisms” (Mansfield, Kaplan, Fitzpatrick, et al, 1997, p. 2). For a system to achieve the qualifications of this aspect, it needs a group-like structure that supports sharing and cooperation and restricts access to members of the structure.

iScent does not base its architecture on a group structure and therefore, does not meet the primary requirement of Locales. iScent’s primary goals are related to framing actions as elements of a conversation, meeting the performance requirements of large organizations, and achieving intersubjectivity (Anderson & Bouvin, 2000). Consequently, the system was designed to support a single community of users where an interest in actions determines notifications instead of founding initial interest on membership in a group, or social context. While this design deviates from the Locales Foundations aspect, iScent’s focus on communication and intersubjectivity establishes a framework capable of achieving some of the remaining aspects of Locales.

Sticker supports artifact sharing by allowing users to send and receive notifications and by allowing users to share objects and resources through the use of attachments. However, Sticker does not provide a community of work in the sense that users work together with a set of shared set of tools in a virtual

environment. Artifacts are not worked on collaboratively in the environment; instead artifacts are simply transferred through the environment. Each user works on his or her desktop computer and then communicates through Sticker. Sticker does an adequate job of meeting the communicative requirements of Locales Foundations, but it does not fulfill the community of work component of this aspect.

Civic Structures

The Civic Structures aspect "concerns the facilitation of interaction with the wider community beyond a person's immediate workgroups and locales" (Mansfield, Kaplan, Fitzpatrick, et al, 1997, p. 3). This aspect does not apply to iScent because, as previously stated, iScent does not base its architecture on a group structure. Multiple locales do not exist within iScent and consequently, there is not a requirement for "the facilitation of interaction with the wider community" built into the design of iScent because there is no narrower community. There is only a single community in iScent.

Sticker meets the basic requirement of this aspect by allowing cross group monitoring. Users can define parameters to monitor throughout all groups and users will receive notifications when the search parameters are met. However, Sticker requires a mechanism to facilitate the formation or evolution of locales once notifications are received. Sticker allows users to receive notifications from other locales, but does not provide a good mechanism to support interaction after the notification is received.

Individual Views

The Individual Views aspect "looks at (1) the different individual views that can be held of the same locale, and (2) the individual's view over multiple locales" (Mansfield, Kaplan, Fitzpatrick, et al, 1997, p. 3). iScent fails to meet the second component of this aspect because it does not support multiple locales. Since both systems allow users to customize the notifications received, it was concluded in this study that both systems support the first component of this aspect. Of course, more robust and intelligent configuration agents should always be designed and developed because, as discussed later in this study, allowing users to configure notifications empowers users and may facilitate the user's ability to assign meaning to activity notifications.

Interaction Trajectory

The Interaction Trajectory aspect "concerns all the temporal aspects of the group's locale and its associated people and entities" (Mansfield, Kaplan, Fitzpatrick, et al, 1997, p. 3). iScent, through the use of event trails, provides the best example for this aspect. Event trails are hypermedia-enabled event histories that can be monitored for notification and stored for later use. By monitoring these event histories, iScent supports the delivery of notifications from the present and recent past. However, iScent does not support the notifications of *upcoming activity* through an analysis of recent activity. While this is a technically imposing challenge that could be found unreasonable, notification of future activity is a potential area of improvement over existing CSCE.

Sticker does not fulfill the requirements of this Interaction Trajectory aspect because, like iScent, it does not concern all "temporal aspects" of a locale. However, Sticker does employ a method of temporal notification through the use of time sensitive notifications. Providing age-identified notifications is a strategy that should be considered in any notification system.

Mutuality

The Mutuality aspect "concerns the degree to which presence and awareness must be supported in collaborative work for the purpose of maintaining a sense of shared place" (Mansfield, Kaplan, Fitzpatrick, et al, 1997, p. 3). There are two important components of this aspect: (1) "the degree to which presence and awareness must be supported in collaborative work" and (2) "the purpose of maintaining a sense of shared place".

Each system provides some form of awareness otherwise it would not have been reviewed. However, the systems only transmit basic notifications and do not attempt any type of notification analysis based on social context or user preference. For example, instead of simply notifying a user when a *buddy* logs in, the system could remind the user that s/he and the buddy are working on a shared document that is due in two weeks. In addition, the extent to which these systems support collaborative work and a sense of shared place are the important criterion in this evaluation. iScent and Sticker support collaborative work through the dispersal of event notifications, but they fail to generate a sense of shared place because neither system includes the actual work contexts and

shared tools for users to work cooperatively. These systems support Locales through various communicative mechanisms, but do not combine that capability with a shared context for work.

By articulating a framework that integrates notifications into an existing CSCE, the communicative benefits of the iScent and Sticker can be combined with the work-centered structure of the existing CSCE to provide a more authentic social environment.

Embodied Interaction

Only through an engagement with the CSCE can users create meaning from the activity occurring within it. Embodied interaction is defined by Dourish as "the creation, manipulation, and sharing of meaning through engaged interaction with artifacts" (2001, p. 126). Neither system supported an interaction with artifacts because both systems were simply notification systems and not integrated with a work environment.

Conclusions

Based on the review of previous work and a comparison of existing CSCE to the Locales Framework, the following non-trivial problems that were not previously solved or poorly implemented were revealed. These problems were (1) a lack of notifications based on social context and a trajectory of actions, (2) an inability to support interaction after notification is received, (3) a failure to generate a sense of shared work place, (4) an absence of activity notifications for

personal embodiment, and (5) an inability to include all aspects of the Locales Framework into a single CSCE.

Sticker, through the use of Elvin, began to address the importance of context with the use of Sticker groups, but did not *analyze* and present notifications based on user preferences or current social context. The environments reviewed simply passed notifications from an event producer to an event consumer. A notification system that generates notifications based on social context and recent activity is needed so users can draw meaning from activity occurring in a CSCE. This problem is the most difficult one to solve both conceptually and technically. Any attempt to instill *intelligence* into a computer-based system will face multiple technical challenges and need to be designed robustly enough to overcome the currently unforeseen needs of a notification system.

The second problem is an inability to support interaction after notification. The systems reviewed supported event notifications but did not afford interaction between contexts and users after the notification was received. Technically, this problem should not be difficult to overcome – record the originator of the notification and develop an interface to allow communication back to this person or program. Socially, this problem poses several obstacles including security, privacy, and overall management on the amount of notifications received by a user.

A failure to generate a sense of shared work place was a shortcoming found in both iScent and Sticker. Each system focused on notifications and did not integrate notifications into the tools used by users. These systems did not provide a context for work, but rather provided a mechanism for notifications. Because this research is about the integration of notification in an *existing* CSCE, the challenges of this problem cannot be discussed until the CSCE is chosen.

The fourth problem with existing collaborative environments is the absence of activity notifications to support embodiment and embodied interaction. Dourish, in his book *Where the Action Is…*, reveals that embodiment is "a foundational property, out of which meaning, theory, and action arise" (Dourish, 2001, p. 125). Current collaborative environments focus on the general notification of activities. An environment that notifies users about activity based on their current social context and personal preferences could enhance the creation of meaning from activity notifications. If users know why they are receiving notifications because they defined their own notification preferences, then less cognitive processing may be required to translate that notification into meaningful information that can be acted upon.

The final issue with existing computer supported collaborative environments is that no single system meets all of the requirements of the Locales Foundations. Some of the systems achieved a few aspects, but no one system symbolized a complete implementation of Locales.

Summary

During this study, only problems (1) a lack of notifications based on context and a trajectory of actions, (4) an absence of activity notifications for personal embodiment, and (5) an inability to include all aspects of the Locales Framework into a single CSCE were studied. Problem 2, the inability to support interaction after notification is received, was deferred because it is a user interface issue that was beyond the scope of this study. Problem 3, the failure to generate a sense of shared work place, was another problem that, while interesting and should be addressed in future research, was beyond the scope of this study. The next chapter presents the proposed Framework for Notification that was based on the previous interpretation of the Locales Framework and this review of relevant work.

CHAPTER IV

FRAMEWORK FOR NOTIFICATION

This chapter presents the principles of the proposed Framework for Notification. The following chapter describes the methods used to examine, test and revise this Framework. The final chapter in this study discusses the implications of using this Framework, the lessons learned while using it, and the directions for future research related to this Framework.

The Framework for Notification

The principles of the Framework for Notification are: Social Context, Awareness in Context, Activity Discovery, Trends in Activity, Meaning of Activity, and Notification Customization (Table 4.1). These principles are derived from an analysis and interpretation of the Locales Framework (Chapter 2) and a review of prior work (Chapter 3). This Framework was validated during an inquiry by development (Chapter 5). Like the Locales Framework, the boundaries between these principles are fluid and should not be studied in isolation of each other, but considered related viewpoints for understanding the notification and awareness requirements of a CSCE.

Table 4.1 Principles of the Framework for Notification

SOCIAL CONTEXT	The place where user actions and interactions occur. Social context partially determines the salience of awareness information. Social context is defined by the current membership, the collective goals of individuals, and the recent activity in the context.

AWARENESS IN CONTEXT	Deliver notifications to users when the notification is relevant to the user's social context.
ACTIVITY DISCOVERY	Allow the discovery of activity outside the user's current context to promote the formation of new social contexts.
TRENDS IN ACTIVITY	Maintain activity and notification histories to determine the impact notifications have on user actions and interactions. Trajectory of activity partially determines the salience of awareness information.
MEANING OF ACTIVITY	Provide mechanisms for users to interpret and construct meaning from the activity occurring in a context.
NOTIFICATION CUSTOMIZATION	Provide notification customization so the user has the final decision on the notifications received.

Social Context

The foundation of the Framework for Notification is the social context. Often referred to as a group, workgroup, or class in existing CSCE, the social context is more than is implied by a pre-defined CSCE structure. The social context is the socially constructed place for user actions and interactions defined by current membership, the collective goals of individuals, recent activity, and the communicative affordances of the technology. The membership of a context is the current collection of individuals belonging to the context. The collective goals of individuals are the shared interests of everyone in the context. Recent activity is a record of the actions that have occurred in the context and the communicative affordance of the technology is the structure and collaborative tools provided in the context.

In a CSCE, a social context is the entire environment, a group within the environment, a collection of groups, a single application in a group, or even an individual user or artifact. For example, a discussion board application that is used by the members of a group is a social context according to this Framework. While all of these constructs are social contexts, only one or a few are relevant to a user's interests and actions at any given time. Consequently, a notification system should be designed to accommodate these different contexts and changing interests of users.

To understand how a social context impacts the actions of users, the developer should analyze contexts from the viewpoint of a single user and realize how contexts reside within other contexts. For example, from the perspective of a user, a discussion board is contained within a CSCE group, the group is contained within the CSCE, the CSCE is part of the Internet, and all of these contexts reside within the physical world. The user may at any time, be interrupted by a phone call, someone knocking on the door, or an email external to the CSCE; each of which affects the actions of that user within the CSCE. Without access to the activity in each of these contexts, it is difficult for a user to derive an accurate meaning from another user's actions in the CSCE. For example, the knowledge that someone just received a phone call is meaningful to a user trying to initiate a CSCE-based chat with the person on the phone. Figure 4.1 illustrates the hierarchical relationship of social contexts in a CSCE called Sakai. Sakai is the CSCE used for the implementation test in this study.

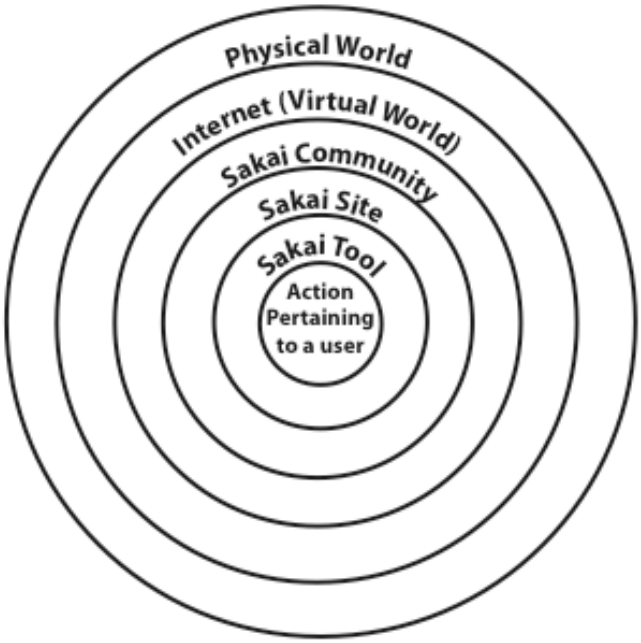


Figure 4.1 Conditional Matrix for Sakai

Figure 4.1 is a conditional matrix of Sakai, the CSCE used in this study. The focus, or center, of the matrix is the action pertaining to a user. Each circle represents a social context that influences the action of that user. Social contexts and the relationships between contexts are dynamic entities that change and evolve rapidly as each aspect of the context changes. For example a user may switch tools or sites rapidly within Sakai. These switches may impact the salience of awareness information as changing contexts relates to changes in goals. A notification system must be equally dynamic to remain effective in the delivery of activity notifications. More information about this conditional matrix is presented in Chapter five.

The following principles articulate how aspects of social action and information about that action moderate the Social Context principle. Each of the

principles provides different viewpoints and strategies for developing a social context aware notification system.

Awareness in Context

According to Dourish and Bellotti, awareness is “an understanding of the activity of others” (Dourish & Bellotti, 1992, p. 1). To understand the activities of others, the most basic requirement is to be notified about the occurrence of those activities. When someone opens a document a user is currently working on, those users should be notified with relevant information. The users need to be made *aware* of the actions of others as it relates to their preferences and social context.

This principle articulates the mechanisms implied in the previous principle because notifications must be presented when the user is in the appropriate social context. For example, activity in a context should have a higher level of importance when the user is within the scope of that social context. If the same action occurs when the user is outside the social context, the notification should have a lower priority and be more peripheral to the user’s attention.

For the developer to achieve this principle, a mechanism to monitor actions of users and to retain information about context must be integrated into the CSCE. Unlike systems like Sticker, where notifications are delivered when an action occurs, a notification system that achieves this principle must function contextually rather than only synchronously. A user should not be notified simply

because an activity just occurred but because the activity is relevant to that user's current social context.

Activity Discovery

The Activity Discovery principle focuses on the potential benefit for the user who knows about activity occurring outside their current social context. To afford the growth and development of new social contexts, a non-intrusive method of discovering activity outside a user's social contexts must be made available.

This principle has two parts. Part one is users need to know about the activity occurring in contexts where their attention is not currently focused. In other words, users are members of multiple social contexts and they should receive notifications about activity occurring in their other social contexts. This activity is less relevant to the user because it is not occurring in their current context and consequently, should be treated as a lower priority notification. To achieve this part of the Activity Discovery principle, a notification system should categorize notifications based on different contexts and priority levels. A notification with a higher priority should be made more obvious to a user than a low priority notification.

The second part of this principle is users should be made aware of activity occurring in unknown social contexts to promote the growth and development of new contexts in the CSCE. An unknown social context is one where the user is

currently not a member and consequently, does not have access to the actions and interactions of its members.

To achieve the latter part of this Activity Discovery principle, the notification system should provide the functionality to analyze a user's notification preferences against activity occurring in all available contexts. Then, the results of this analysis should be presented to each user anonymously to let them decide if they want to act upon this information. Following the iScent principle of Intersubjectivity, the owner of the social context and the interested user are made aware of the potential interaction (Anderson & Bouvin, 2000). To illustrate, consider User X who, based on his notification preferences, is interested in grant writing. The members of social context Y, which is owned by User Z, have been discussing the challenges of writing NSF grant proposals for the past month. Based on this information, the notification system should notify User X and User Z of a new interaction opportunity. Each user must confirm an interest in this opportunity before either is notified about the actual activity to ensure privacy.

Trends in Activity

The Trends in Activity principle is about user actions and interactions in a social context over time. By developing a notification system capable of recording activity information, it is possible to support asynchronous collaboration, provide users with a history of activity, and provide data for researchers to study the effects of awareness and user interactions in CSCE.

To achieve this principle, a notification system must record both the activity and notification histories of social contexts. By comparing the data stored in both histories, the actions of users and interactions between users caused by activity notifications can be analyzed. By analyzing these histories, a notification system could be designed to deliver notifications to users based on current, and potentially, future activity. For example, a system capable of identifying that one team member is located in another country, could notify a user about the conflicts in schedules when he or she schedules a synchronous chat room discussion.

This principle is important because activity is part of a process and notifications about activity impact subsequent actions in that process. Knowing that an action has recently taken place alerts users to the needs and requirements of future actions and interactions. For instance, knowing that my dissertation chair just reviewed a draft of my dissertation alerts me to the need to review his feedback and make the needed revisions. Without the knowledge of the available feedback, I would not know about or incorporate the feedback into the process of writing my dissertation.

Meaning of Activity

Dourish, in his book “Where the Action Is”, defines embodied interaction as “the creation, manipulation, and sharing of meaning through engaged interaction with artifacts” (2001, p. 126). When developing a notification system for a CSCE, it is important to remember that activity notifications are only as good as the sense the user can make of the notification. The notification system

should support the creation of *meaning* from activity notifications. The notification system cannot impose meaning through notifications. It is up to each individual user to construct meaning from the activity.

Current CSCE focus on the general notification of activities. This principle proposes that a CSCE embodies users by notifying them of actions on their personal artifacts. To achieve this principle, a notification system should provide each user with the option to select artifacts to monitor, select the activity on those artifacts to monitor, and determine how and when the resulting notifications should be received. In this way, the notification information is not simply disembodied information about activity in a system; but rather, notification information fits into the user's mental map of activity and context.

Artifacts and actions define user presence in CSCE. By revealing the activity of others resulting from or related to the person's actions and artifacts, the user will connect the new set of activity to the current practices and may experience an enhanced sense of value and worth. For example, by notifying a user about the amount and type of activity on an artifact he or she created, the user derives a sense that the artifact has a reputation with other members of the social context. As a result, the user may put more effort into improving the artifact or contributing to additional artifacts.

Notification Customization

To meet the requirements of the previous principles and to incorporate the Individual Views aspect of Locales, a well-designed notification configuration tool

is required in a notification system. Even when a user shares the interests and goals of a social context, each individual user has individual interests that require individual notification settings.

To achieve this principle, the notification system should be designed so users can select activity based on social context, select activity for specific artifacts, and choose different priorities for different notifications. Users should also be able to block notifications or change how notifications are received. This notification configuration component must be integrated closely with the CSCE to support the level of configuration required by this Framework. Additional information on creating a robust notification configuration system is presented in the next Chapter.

Relationship to Locales

As mentioned earlier, the proposed Framework for Notification was initially derived from Fitzpatrick's Locales Framework. This section describes how Locales influenced the proposed Framework.

This Social Context principle is derived from the Locales Foundations aspect and Strauss' concept of "social worlds". Like the Locales Foundations aspect, this principle is the foundation for the Framework for Notification. Each subsequent principle in this Framework is an articulation of different ways to understand and support this foundational principle.

In Locales, a social world is the "prime structuring mechanism for interaction" and the site and means that enables interactions (Fitzpatrick, 1998,

p. 90). “Locales are mechanisms to relate the resources needed to support social world interactions” (Fitzpatrick, Mansfield, & Kaplan, 1996, p. 4). While these concepts are needed and important when discussing interactions in collaborative environments, the distinctions between locale and social worlds were found to be unnecessarily abstract for the developer of a CSCE notification system. The proposed Framework uses the single concept of “social context” to represent the structures where interactions between users occur.

Paul Dourish, in his book “Where the Action Is”, refers to the term “context” as a social construct where interactions take place (2001). He also emphasizes the importance of deriving meaning from the actions occurring within those contexts. As a result, the term “social context” is used in the proposed Framework to represent any element of a CSCE around which collaboration and interaction may occur, i.e., a user’s artifact, a CSCE tool, a CSCE community, etc. A finding of this study was that by translating and simplifying the Locales Foundation aspect into the Social Context principle of this Framework, the processes for working with Locales becomes more accessible to accommodate widespread use, which was a problem Fitzpatrick identified with using Locales as a framework for development (Fitzpatrick, 1998).

The Awareness in Context principle is conceived from the concepts of awareness and presence in the Mutuality aspect and the Locales Foundations aspect. To support interaction and collaboration in an online environment, users must be notified about the activities of others. Without awareness of activity,

interaction in a CSCE becomes more difficult because users must actively seek out activity information for themselves. The Locales Foundation aspect influenced this principle by enforcing the concept of having a “focal point” for collaboration. Notifying users of activity related to their current social context, or focal point, serves as a filter to reduce the number and increase the quality of CSCE notifications.

The Activity Discovery principle is directly related to the Civic Structures aspect of Locales. Like the terms “social world” and “locale”, “civic structure” is useful to researchers discussing awareness and user interactions but difficult to translate into system functionality when designing notification systems. Consequently, this principle is named “Activity Discovery” to make the Framework more accessible to developers and those not trained to understand the language of Locales.

As the Civic Structure aspect of Locales implies, interaction beyond a user’s current context is important for the growth of existing and new social contexts. Membership is a defining quality of a social context and providing mechanisms to support the infusion of new members into a context is important to sustain the growth and evolution of the collaborative environment.

The Trends in Activity principle is the result of an analysis and interpretation of the Interaction Trajectory aspect in Locales. A core notion of this aspect is “that actions can only make sense in relation to interaction and the broader interactional context” (Fitzpatrick, 1998, p. 123). Likewise, activity

histories should be made available to the users to support the creation of meaning related to past activity. Activity and notification histories should be kept to study user interactions resulting from activity notifications. Knowing the history of actions in a context impacts the future actions of users in that context. Analyzing the resulting interactions provides researchers with data to study the effects of awareness in CSCE.

The Meaning in Activity principle is drawn partially from the Mutuality aspect in Locales, but is more fully articulated by Dourish's concept of "embodied interaction" (2001). The Mutuality aspect focuses on the importance of presence and awareness in a shared place and embodied interaction is a construct for explaining how notifications support meaningful activity. This principle is used to determine why notifications are received and how the user interprets them.

The Notification Customization principle is derived from an analysis of the Individual Views aspect of Locales and the process of developing a notification system. The Individual Views aspect states that individuals have different views of social contexts. Even though the members of a context share a common goal for that context, each individual values activity in that context differently. Consequently, users should be given the ability to define the notifications they value.

Summary

The purpose of this chapter was to present the outcome of work toward the first goal of this study: articulate a development framework for integrating

activity notification into existing CSCE. This chapter described that Framework for Notification. The next Chapter presents the methodology used to evaluate the Framework. Finally, Chapter six discusses the results of this study and suggests directions for future research related to this Framework.

CHAPTER V

METHODS

The purpose of this study was to advance a theoretical framework for development that could be used to integrate activity notifications into existing CSCE. To achieve this goal, the study used a software research design that included collecting data during software development, during product evaluation, and during a system comparison between the modified CSCE and the original. Over the course of this research, the Framework for Notification was envisioned, articulated, and modified into the form presented in Chapter four.

Figure 5.1 and Table 5.1 provide a quick overview and introduction to the system and terminology used to describe the product. The product developed in this study is called a Context-aware Activity Notification System (CANS). Figure 5.1 illustrates the layers and relationships between the layers of CANS. CANS is composed of three layers: an Event Notification System (ENS), a CANS Link, and a modified CSCE (Figure 5.1). Table 5.1 defines the terms of the system.

The following convention is used when discussing various components of software developed in this study. A Java class or object is referred to simply by the class name and is italicized to simplify reading for those unfamiliar with object-oriented programming. No distinction between class and object is made. For example, the term *Consumer* refers to the functionality provided in the Consumer.java file. Whenever the word “application” is used, as in “Consumer

application", the researcher is referring to the custom-built software developed within the CSCE.

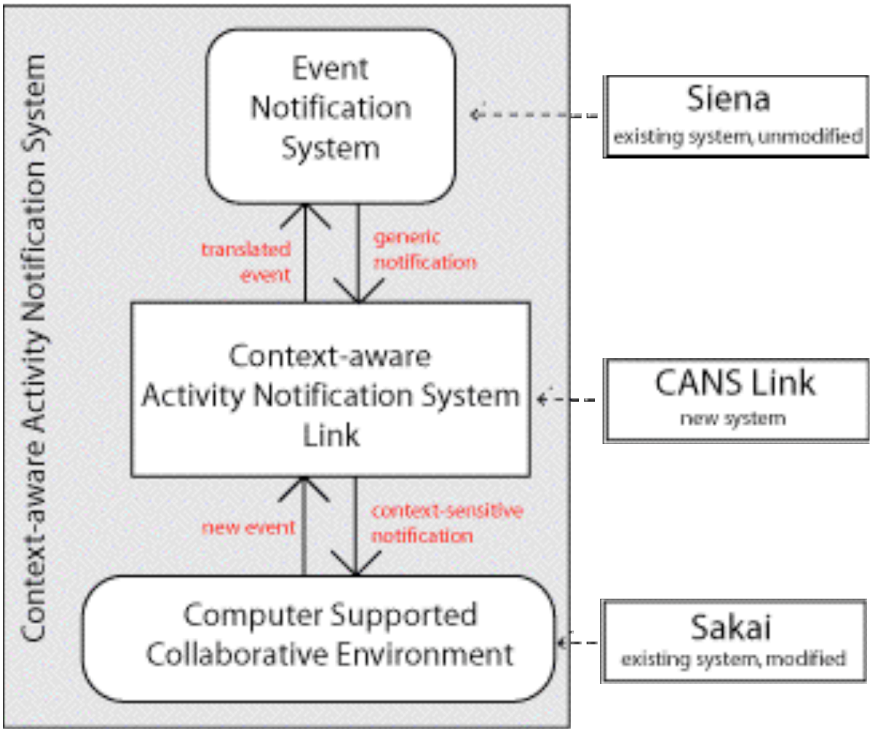


Figure 5.1 Basic Context-aware Activity Notification System

Table 5.1 Relevant Terms

Context-aware Activity Notification System (CANS)	The complete notification system created in this study. CANS is comprised of a modified CSCE, a newly created CANS Link, and an unmodified ENS.
CANS Link	A middle-ware application that serves as a communication and information-processing channel between a CSCE and an ENS.
CANS Link: Server, Consumer, Producer, Administrator	The four components of the CANS Link layer. The "Consumer", "Producer", and "Administrator" reside within the CSCE and are responsible for initiating and receiving communication with the "CANS Link: Server". The "Server" communicates directly with the ENS.
Event Notification System (ENS)	An application independent service that supports notification-based environments through the efficient routing of events between producers and consumers. Sometimes referred to as a "Service" rather than a

	"System" in the literature, only the term "System" is used for consistency.
Sakai	The CSCE used in this study.
Siena	An acronym for "Scalable Internet Event Notification Architectures". Siena is the ENS used in this study.
Event Producers	An application that generates user events.
Event Consumers	An application that receives notifications about user events.
Internal Notifications	Notifications that are received and displayed within the CSCE.
External Notifications	Notifications that are received and displayed outside the CSCE in 3rd party software.

Context-aware Activity Notification System

CANS represents not only a new product but also a new approach for providing activity notifications in CSCE. In the past, a CSCE either employed an internal notification scheme or incorporated the direct services of an external Event Notification System (ENS). Either way, the notifications were transmitted to the user as they occurred and failed to address the asynchronous nature of CSCE or failed to address all aspects of the Locales Framework and thus limiting its potential to support user interactions in the collaborative environment. The unique contribution of this development effort is the creation of the CANS Link - the communication bridge between the CSCE and the ENS.

The top layer of CANS is the ENS. An ENS is an application independent service that supports notification-based environments through the efficient routing of events between producers and consumers. Event producers send events to the ENS and event consumers receive notifications. For this study, the researcher used the existing ENS developed by Antonio Carzanigia called Siena

(1998). The benefit gained from using an ENS is found in its ability to efficiently transfer events from a producer to a consumer. Its limitation is that it only provides generic notifications to consumers in real-time and fails to support asynchronous activity or notification analysis based on social context and user preference.

The bottom layer of CANS, i.e., the user interaction layer, is the CSCE. The CSCE is the collaborative environment where users work, generate activity, and receive notifications. The CSCE is both a producer of events and a consumer of notifications. In CANS, the CSCE produces events to and consumes notifications from the CANS Link, not the ENS directly. Sakai, a new collaborative environment, which was still under development by the University of Michigan, Indiana, MIT, and Stanford during this study, was selected as the CSCE (Sakai, 2005). Sakai provided limited activity notification capabilities through the use of an internal notification system prior to this study. Refer to the section of this Chapter titled “System Comparison” for additional information related to the capabilities of the original CSCE versus the modified CSCE.

The CANS Link, the middle layer of CANS, is the new and innovative component of this system that was developed to address the research problems identified in Chapter three. The CANS Link is a middle-ware application that serves as a communication and information-processing channel between a CSCE and an ENS. It is innovative and non-trivial because it not only mediates the communications between the CSCE and the ENS to deliver notifications

based on user preference and social context, but also because it provides a level of notification customization not available in previous CSCE and because it separates the required notification processing and analysis from the CSCE. The full benefits of this distributed architecture are presented later in this Chapter but to summarize, this separation distributes notification processing and analysis across servers to reduce the negative impacts of analyzing and processing activity notifications.

Software Development Product

As previously stated, CANS is made up of three layers: the ENS, the CANS Link, and the CSCE. This section describes each layer in CANS and explains how each layer works together as shown in Figure 5.2.

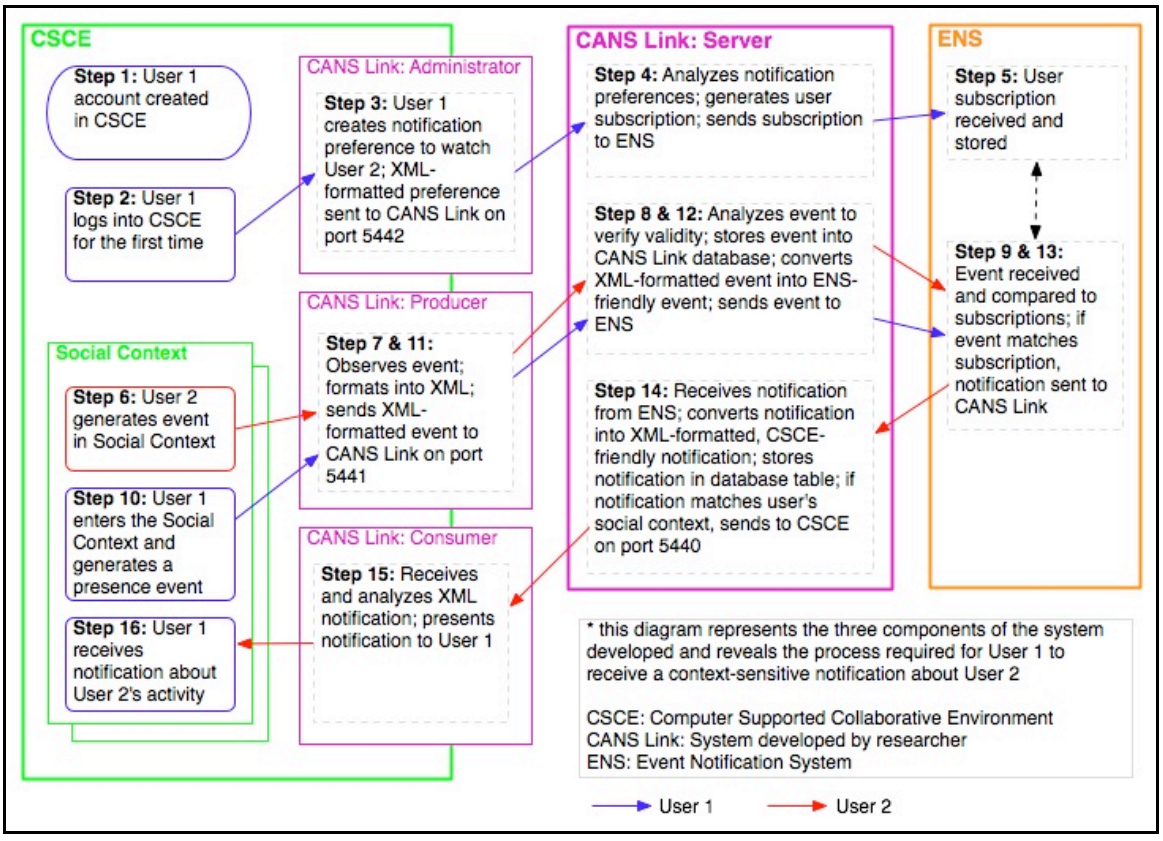


Figure 5.2 CANS Flow Diagram

The ENS Layer

The ENS selected for this development effort was the Scalable Internet Event Notification Architectures (Siena) (Carzanigia, 1998). Siena, developed by Antonio Carzanigia, is “a generic *scalable publish/subscribe event-notification service*” that was chosen because (1) it provided efficient selection and delivery of events, (2) it facilitated an expanded CANS architecture through its *scalable* design, and finally because (3) it supports future use and adaptation through it being licensed as an open source product. The ENS was not modified in this study.

The ENS works with the CANS Link by maintaining user notification preferences and using those preferences to subscribe to “notifications of interest” in the ENS. When an event occurs, the CANS Link sends the event to the ENS. The ENS evaluates the event against each user’s subscription and delivers a notification back to the CANS Link. The CANS Link then stores that notification in a database table and delivers it to the appropriate user based on that user’s preferences and current social context. The ENS compliments the CANS Link by distributing the information processing required in a notification system across multiple servers. This distributed architecture improves system performance when compared to an architecture where event detection and notification analysis is executed on the same server because each component of the distributed system can function without being negatively impacted by the performance of the other.

The CSCE Layer

Before activity notifications were integrated into an existing CSCE, the CSCE had to be selected and analyzed. For this study, the new CSCE known as Sakai (http://sakaiproject.org) was chosen for its flexible system architecture, rudimentary notification capabilities, and current interest shown by the higher-education community (Figure 5.3).

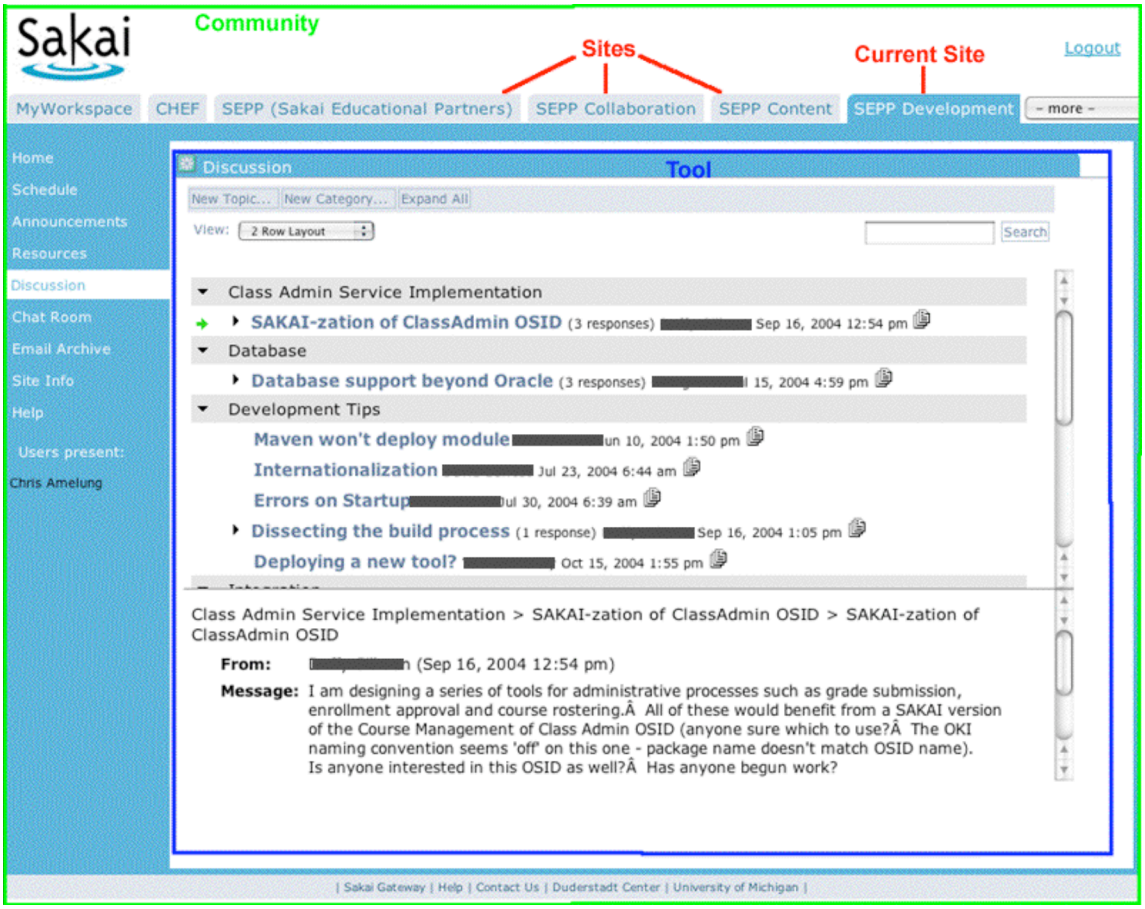


Figure 5.3 The Social Contexts of Sakai

Sakai is an innovative open source project founded by the University of Michigan, Indiana University, MIT, Stanford, the uPortal Consortium, and the Open Knowledge Initiative, and supported by a worldwide partnership of other

Universities. Sakai was built to support the integration of a set of educational software tools into a single online collaborative community.

As recommended in Fitzpatrick's thesis on designing for cooperative work, a conditional matrix was created during this study to analyze the existing social worlds and civic structures, i.e. social contexts, of Sakai. Developing a conditional matrix is a procedure used to identify and analyze the social conditions affecting a specific phenomenon (Strauss & Corbin, 1994). Data for the conditional matrix were collected through direct observation of Sakai (see Figure 5.3) and is presented in Tables 5.2 and 5.3.

Table 5.2 Context Structure in Sakai

Sakai Community									
Site						Site			
Tool			Tool		Tool	Tool		Tool	
User	User	User	User	User	User	User	User	User	User

All activity in Sakai occurs within Sakai Tools and consequently the function of a tool is typically the most influential condition on the action, as shown later in Figure 5.4. The tools enable, or influence, not only the action that occurs but also the actions that *can* occur. Table 5.3 presents the activity that can occur within tools in version of Sakai used in this study.

Sakai Tools reside within, and are accessed through, a Sakai Site. Referring back to Figure 5.3 and Table 5.2 shows how multiple Sites exist within a single Sakai Community. The Figure also reveals the Tools available to the user's current site. For example, the current Sakai Site shown in Figure 5.3,

named “SEPP Development”, possesses the following Tools: Schedule, Announcements, Resources, Discussion, Chat, Email, Site Info, and Help.

Table 5.3 Sakai Tool Actions

Tool	Action (event)
Announcement	new, read, revised, deleted
Assignment	new, read, revised, deleted, submitted
Chat	new, read, revised, deleted
Discussion Board	new, read, revised, deleted
Mailbox	new, read, revised, deleted
News	new, read, revised, deleted
Realm	new, deleted, updated
Resource	new, read, revised, deleted
Schedule	new, read, revised, deleted, imported
Site	new, deleted, visited, updated
User	new, deleted, updated, navigation

Finally, Sites reside within a Sakai Community. A Community represents an installation, or instance, of Sakai. At the time of writing, communication and collaboration between Sakai Communities was not supported. However, as this study will show, an interesting and unexpected outcome of this development effort was the ability to provide cross-community communication through the use of a notification system.

From these data, a Sakai Conditional Matrix was created (Figure 5.4). The matrix is a series of expanding concentric circles with the phenomenon residing in the center and conditions of decreasing influence appearing as the circles radiate outward. The phenomenon, or center, of the Sakai Conditional Matrix is the action pertaining to a user in the Sakai environment. This “action” could be the presence of a user or resource, or the action performed by a user on a

resource. In Figure 5.3, the “action” is the activity of reading a discussion board post.

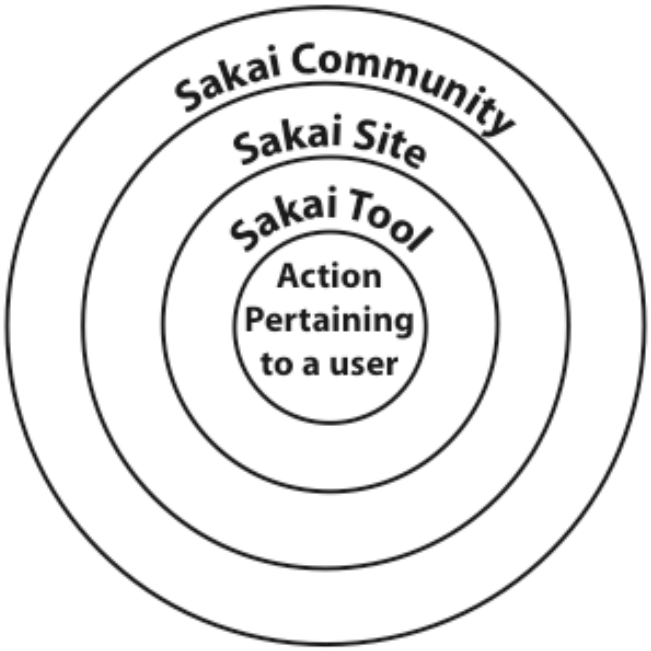


Figure 5.4 Sakai Conditional Matrix

The Sakai Conditional Matrix (Figure 5.4) represented the conditions, i.e., social contexts, influencing the actions of a user in Sakai. However, during the development process of this study, the realization that Sakai resides within additional social contexts was reached. Figure 5.5 shows two additional layers added onto the initial conditional matrix. These two layers - Internet (Virtual World) and Physical World - imply that even when a user is working within a CSCE, that user is influenced by events *external* to the CSCE. It is desirable that users be notified about all activities relevant to their current social context and notification preferences, even events external to the CSCE.

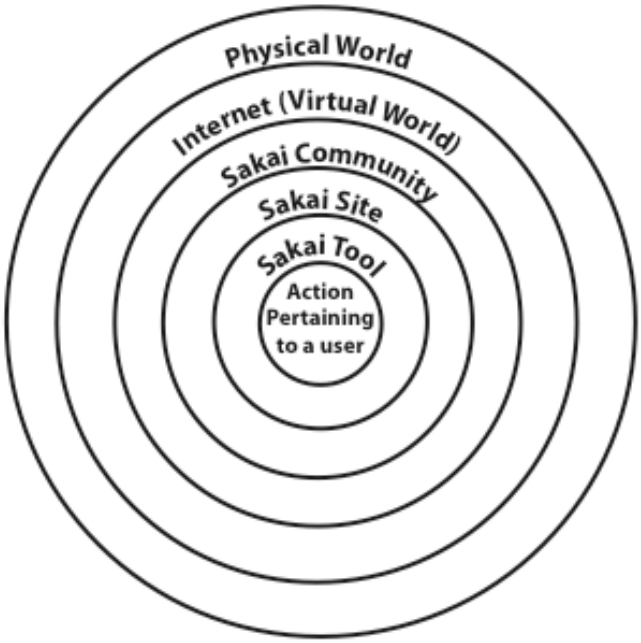


Figure 5.5 Comprehensive Conditional Matrix for Sakai

To conclude this analysis of Sakai's existing social contexts, it should be noted that while the process of designing the conditional matrix afforded a rich understanding of the social contexts related to notifications in the CSCE, the matrix is a limited representation of conditional (social context) influence. The matrix implies a strict hierarchical relationship between conditions. In other words, the matrix implies that the Sakai Tool is always more influential than the Sakai Site because Tool layer is closer to the center of the circle. However, during the development process and through interaction with the resulting product and Framework, it was found that any condition in the matrix would occasionally exert its influence on the user despite the implied influential structure of the conditional matrix. An example of this situation is the need for the CSCE's system administrator to notify users of a mandatory system reboot. This

community-level event impacts every user action in the CSCE without regard for Tool or Site.

The CANS Link Layer

The previous two sections presented the ENS and CSCE layers of CANS. Those two layers existed prior to this study. The CANS layer presented in this Chapter is a new, innovative component to notification systems that was developed by the current research.

The CANS Link is a multi-threaded server written in the Java programming language and a collection of three custom applications embedded inside the CSCE. The server, referred to as “CANS Link: Server”, resides between the ENS and CSCE and is responsible for storing the event history of the CSCE and the users’ notification preferences, for providing a communication channel between the CSCE and ENS, and for analyzing and distributing activity notifications to users. It uses XML transmitted over dedicated sockets to communicate with the CSCE and ENS and can provide RSS feeds to external asynchronous event monitors. User preferences, event histories, subscriptions, and notifications are stored in a MySQL database to support context-aware activity notifications and to support data analysis in future research.

The CANS Link applications operate within the CSCE and communicate directly with the CANS Link: Server via sockets. These applications are called the CANS Link: Producer, CANS Link: Consumer, and the CANS Link: Administrator (Figure 5.6).

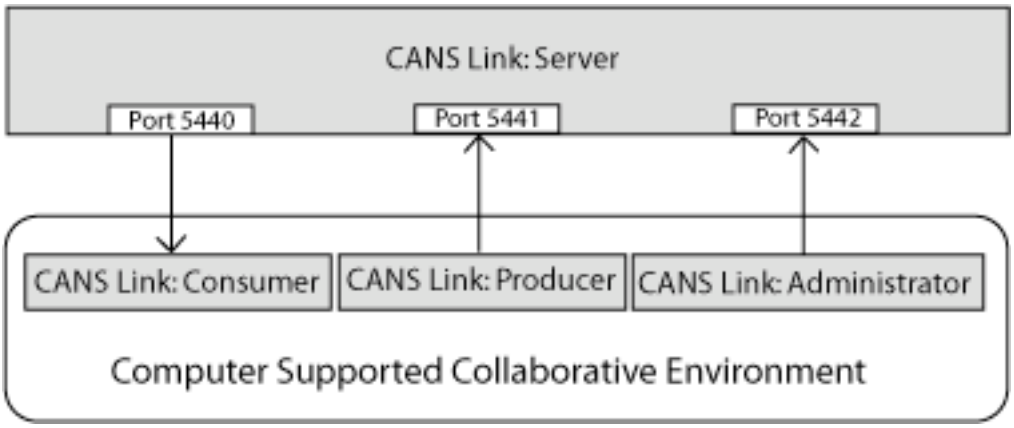


Figure 5.6 CANS Link Integration into CSCE

A detailed description of each application is presented next, followed by a review of the database tables used by the CANS Link.

The CANS Link: Server is a multi-threaded Java server. This server is made from a collection of Java class files. Figure 5.7 shows the class diagram for this server – additional CANS Link documentation is located in Appendix B. The top-level class used to initiate the CANS Link: Server is found in the file named StartServer.java. This class is responsible for:

- Establishing a connection with the ENS
- Loading each User object into memory and initializing User subscriptions with the ENS
- Starting the socket server

Once *StartServer* completes user initialization and communication with the ENS has been established, server control is transferred to *ConnMgr*. The *ConnMgr* controls each port's active socket and dispatches new connections to *Client* on a new thread. The *Client* creates the appropriate: *Producer*, *Consumer*, or *Administrator*, based on the communication port used. The CANS Link: Server

only accepts *Consumer* communication on port 5440, *Producer* communication on port 5441, and *Administrator* communication on port 5442. A user requesting a service on the incorrect port is assumed invalid and ignored.

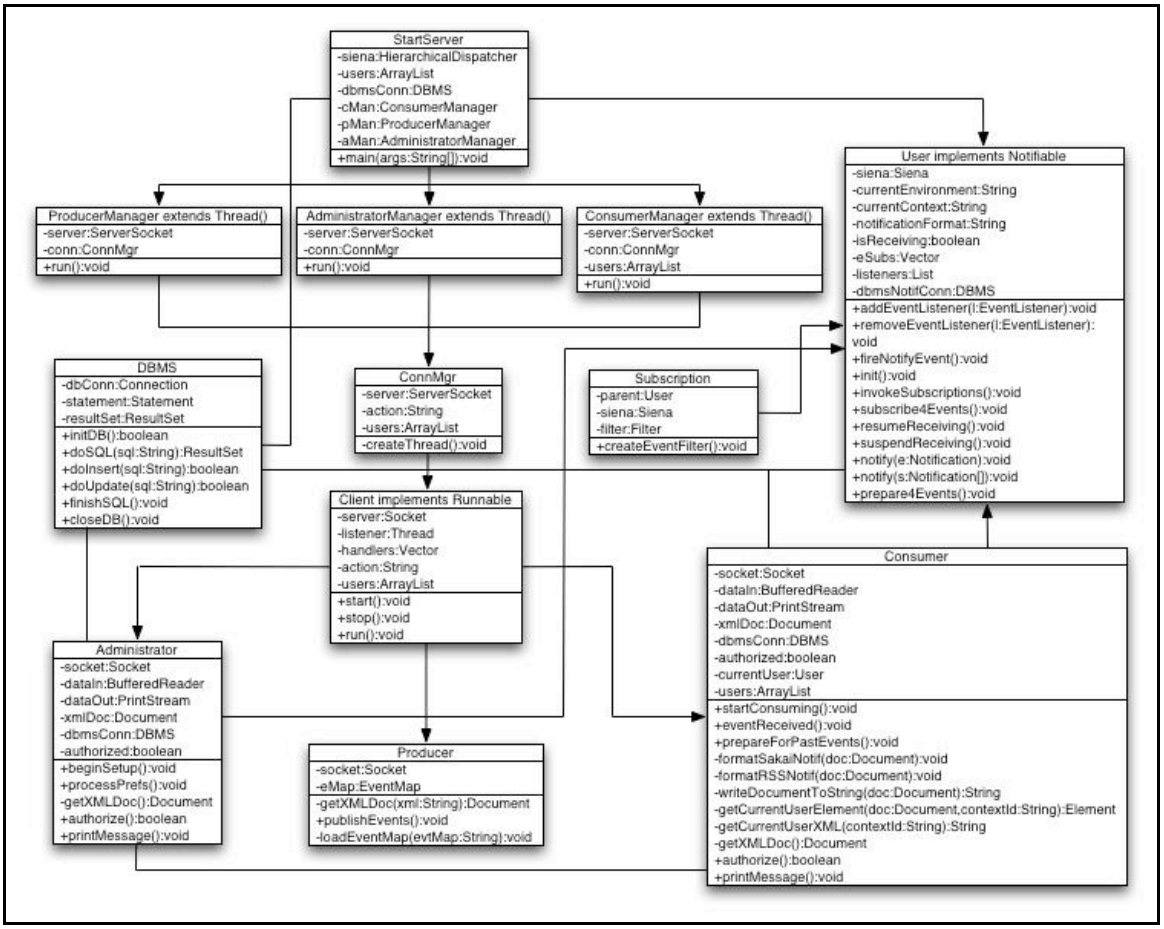


Figure 5.7 CANS Link: Server Class Diagram

Event Production

After a user connects to the CANS Link: Server, system operation depends on the user's request and the communication port used. If the user connects as an event producer, the *Producer* is instantiated. The purpose of the *Producer* is to convert the XML-formatted event into a Siena-formatted event and

then deliver that event to Siena. The following XML is an example of the required format used by the *Producer*:

```
<environment id="sakai">
    <context id="dev" name=”Development” type="group">
        <event action="content.new" event_object="Dissertation.rtf"
            event_tool="1111104282053-13" author_id="chris"
            author_name="Chris Amelung" date="2005-03-23 08:08:08" />
    </context>
</environment>
```

The environment tag is the root of the XML document. An environment tag may contain multiple context tags and a context tag may contain multiple event tags if the client application is designed to aggregate user events before sending to the *Producer*. Additional information about the required XML structure is provided in Appendix C.

XML is translated into a Siena-formatted structure using a Siena extension called SXML (Siena, 2005, SXML section). SXML provides methods for importing a set of rules, i.e., mapping, to describe an XML structure. These mappings are used to create a Siena event. Additional specifications and information on using SXML is located at http://serl.cs.colorado.edu/~serl//siena/sxml/index.html. The SXML mappings used in the Producer are:

```
string environment_id="/environment/@id";
string context_id="/environment/context/@id";
string context_name=”/environment/context/@name”;
string context_type="/environment/context/@type";
string event_action="/environment/context/event/@action";
string event_object="/environment/context/event/@object";
string event_tool="/environment/context/event/@tool";
string author_id="/environment/context/event/@author_id";
string author_name="/environment/context/event/@author_name";
string event="/environment" xml;
```

The CANS Link: Producer application within Sakai communicates with the *Producer* in the CANS Link: Server. The Producer application detects events occurring within Sakai, formats the event to adhere to the XML structure, and then delivers the event to the CANS Link: Server on port 5441. In this study, data collected during the Sakai analysis were used to design and develop this Producer application. A detailed analysis of this application is not relevant to this report because the programming strategies used to create this application are unique to Sakai and would be necessarily unique for other CSCE.

Notification Consumption

The second client type of CANS Link is a consumer. The *Consumer* in conjunction with the *User* accepts notifications from the ENS; then stores, translates, and delivers XML-formatted notifications to event consumers. The *Consumer* communicates with the CSCE and the *User* communicates with the ENS (Figure 5.8). The *Consumer* monitors each user connection and delivers notifications based on user preference and social context. The *User* is the persistent representation of a user in CANS that continuously accepts notifications from the ENS.

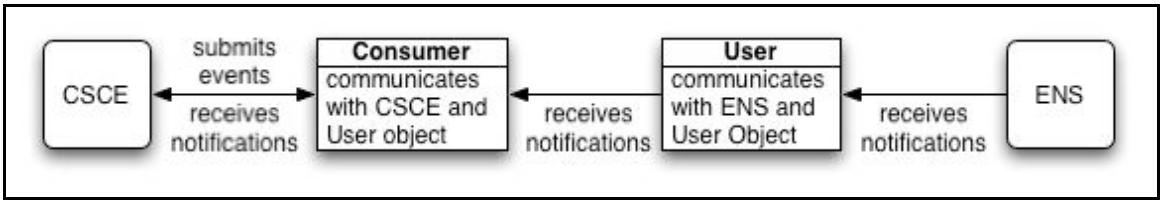


Figure 5.8 Communication Structure of Consumer and User Classes

When a user connects to the CANS Link: Server on the consumer port, it must authenticate itself before it can receive notifications. The *Consumer* waits

for an authentication XML message from the user and verifies the account information against the user information in the MySQL database. A valid authentication XML message is formatted as follows:

```
<environment id="sakai" date="2005-03-22 19:11:43">
        <user id="chris" pw="letmein"/>
</environment>
```

Once the user is authorized to receive notifications, the *Consumer* notifies the user's application of valid authentication and queries the event history table in the database for recent events. If events have occurred that match the user's notification preferences the *Consumer* records the notification in the database. The client application does not accept notifications until it has received the validation on user authentication from the CANS Link: Server. User authentication in the *Consumer* is critical to ensure notifications are delivered to the correct consumer.

The *Consumer* receives new notifications from the *User* by implementing the Java EventListener class. When the *User* receives a new notification from the ENS, it notifies the *Consumer*. The new notification is stored in the "notifications" database table and analyzed to determine when and how to deliver it to the user. If the notification matches the user's preferences and current social context, an XML-formatted notification is created and sent to the user. The following XML is an example of the required format for notifications sent to Sakai:

```
<?xml version="1.0" encoding="UTF-8"?>
<notification date="2005-03-22 07:11:43">
   <environment id="sakai" userid="2" username="Chris Amelung"
      usercount=”2”>
      <context id="System " type="system" priority=”1”>
         <present since="1 DAY">
            <user author_id="chris"/>
            <user author_id=”jim” />
            <user author_id=”chastidy” />
         </present>
         <event action="user.login" author_id="chris"
            author_name="Chris Amelung" date="2005-03-22 19:11:43"
            object="" tool="">
            <description>Chris Amelung just logged in.</description>
        </event>
      </context>
   </environment>
</notification>
```

This XML notification was generated when a user logged into Sakai. In addition to the information provided in the event tag, this notification also indicates how many users are currently active in the CSCE and how many users have been in the “System” context within the last day. Appendix D provides additional specifications on this XML format.

The CANS Link: Consumer application receives notifications from the *Consumer* and displays notifications to users within Sakai. Unlike the CANS Link: Producer, this application was not written specifically for Sakai because it doesn’t interact directly with the Sakai environment. The only requirements of this application are that it can communicate across a socket and that it adheres to the XML format of *Consumer* notifications. For this study, the CANS Link: Consumer

application was written in Flash MX 2004 using ActionScript 2.0. Figure 5.9 shows an example of the Consumer application's interface.

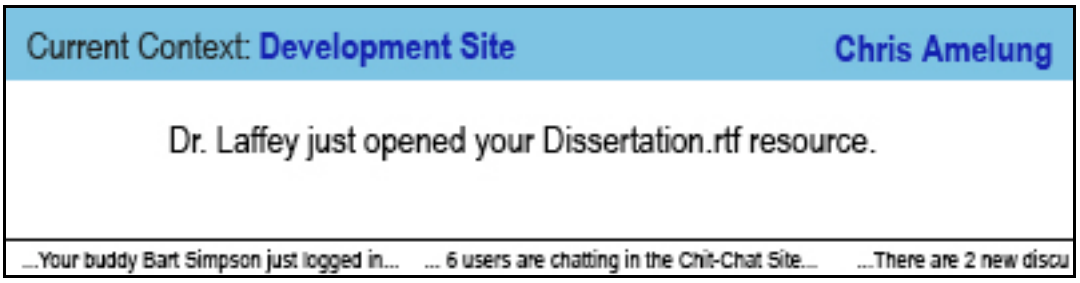


Figure 5.9 CANS Link: Consumer Application Example

Notification Customization

The final client type in CANS Link is an administrator. A user connects to the CANS Link: Server as an administrator to define or modify notification preferences. In the version of CANS developed for this study, two different user types were supported: administrator and user. An administrator can set system preferences and default user preferences for the entire CSCE. A user can adjust those default notification schemes based on personal preference.

When CANS is first setup for Sakai, an administrator must turn on and define the default notification preferences. Initially, all notifications are turned off. Each notification has two states: on and off. Notifications that have a state set to "off" are not available to users of the CSCE. When a new user is created, that person adopts the default notification preferences defined by the administrator and may define personal preferences if desired. Figure 5.10 is a screenshot of the instructions presented to a user when setting general notification preferences. This Figure shows that users can receive notifications with different

priorities and as indicated by the phrase "… you will be given the opportunity to select specific users and resources within Sakai to monitor…"

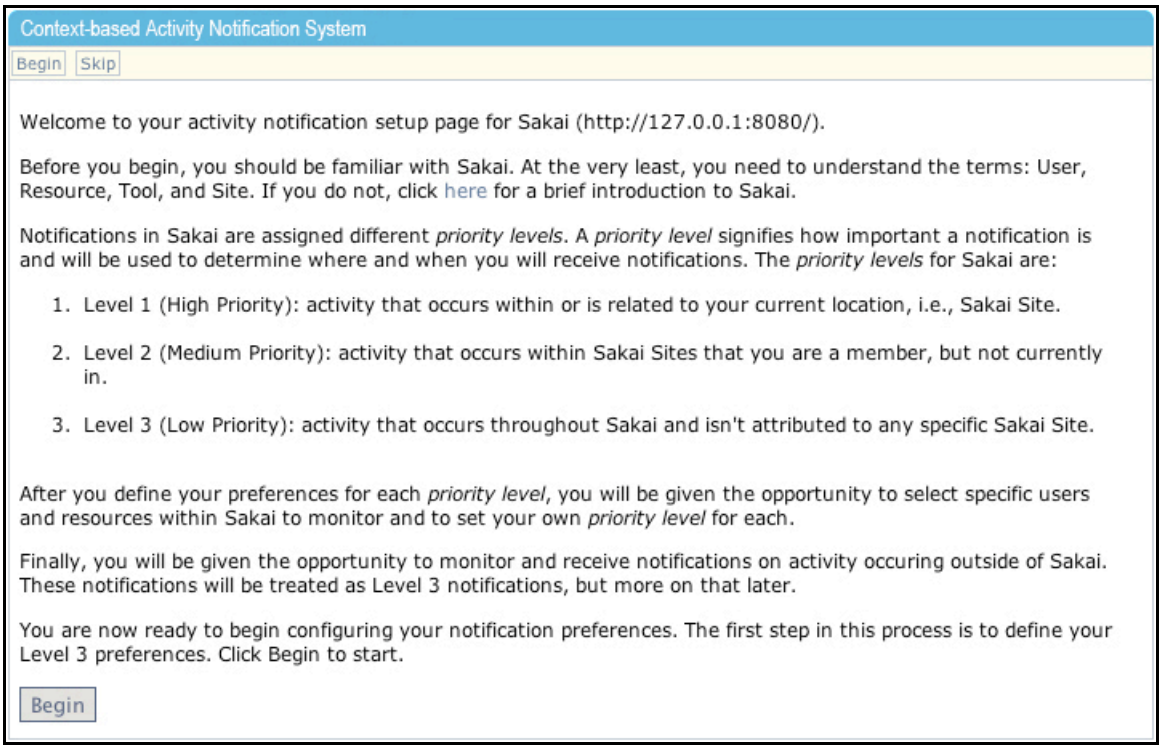


Figure 5.10 CANS Link: Administrator Welcome Screen

The *Administrator* in the CANS Link: Server is responsible for accepting and setting preference changes in the CANS "subscription" database table. Like the *Consumer*, the *Administrator* requires authentication before accepting any notification change requests. The *Administrator* follows the same protocol used by the *Consumer*. Refer to the previous section titled "Notification Consumption" for the authentication specifications.

The CANS Link: Administrator application communicates with the *Administrator* to provide notification customization to Sakai users. Figure 5.11 is a screenshot of the user interface and preference options developed for setting

level 3 notifications in Sakai. The complete sequence of screenshots for setting notification preferences is provided in Appendix F.

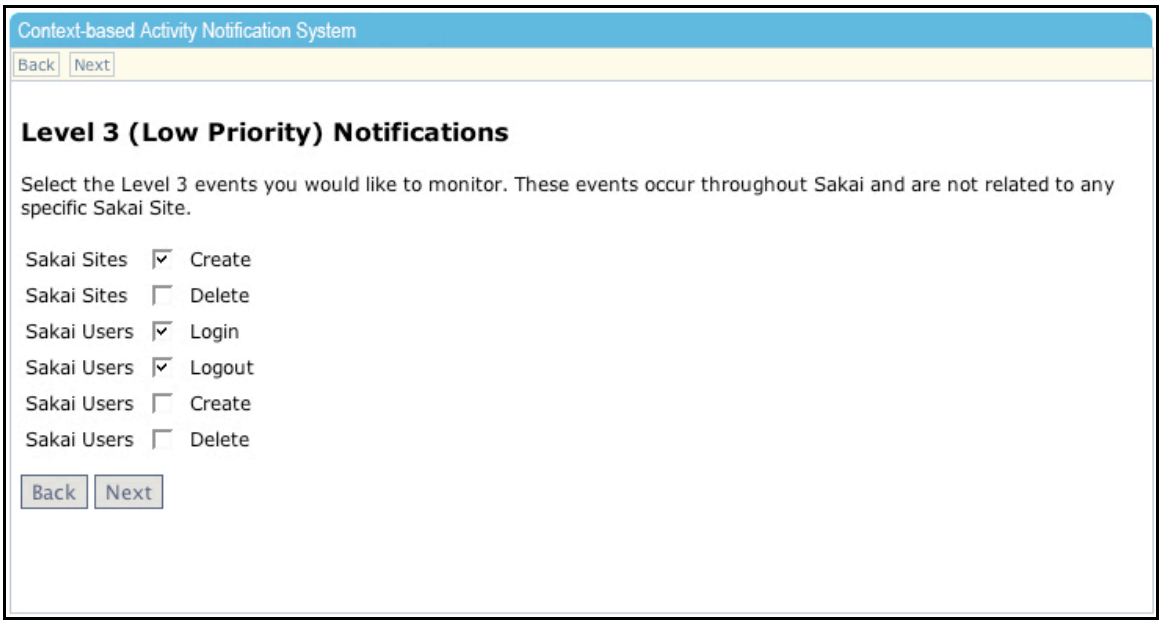


Figure 5.11 CANS Link: Administrator Low Priority Notifications

On each click of the “Next” button (Figure 5.11), an XML-formatted notification change for the user is sent to the *Administrator*. The following XML is an example of the required format enforced by the *Administrator*:

```
<environment id="sakai" date="2005-03-22 19:11:43">
    <user id="chris" request="update">
        <event id="site.add" context_id="System" priority="3" status="on"/>
        <event id="site.del" context_id="System" priority="3" status="off"  />
        <event id="user.login" context_id="System" priority="3" status="on"/>
        <event id="user.logout" context_id="System" priority="3" status="on"/>
        <event id="user.add" context_id="System" priority="3" status="off"/>
        <event id="user.del" context_id="System" priority="3" status="off" />
    </user>
</environment>
```

The above XML example corresponds with Figure 5.11. Additional specifications for this XML format are provided in Appendix E.

In addition to the customization of general notifications, the CANS Link supports custom notification preferences for individual groups, users and objects within Sakai. This feature allows users to identify items of interest in the CSCE and set personal notification preferences on those items. In the version of CANS developed in this study, individuals can monitor the items presented in Table 5.4.

Table 5.4 Events Available for Specific Group, User, and Object Monitoring

Context	Events
Group	New Discussion Board Posts New Chat Messages New Resources
User	Login Logout Navigation
Object	Revised Viewed

Naturally, when individuals monitor and receive notifications concerning specific groups, users, or objects, possible privacy violations arise. CANS addresses this privacy issue by adopting Anderson and Bouvin's principle of Intersubjectivity - "I know that you know that I know" (2000). While not fully implemented at the time of this writing, CANS requires users to grant permission for others to monitor and receive notifications about their activity. Then, when another user chooses to monitor the available activity, CANS sends a notification to the user being monitored to notify him or her that someone is watching his or her activity. Figure 5.12 is an example of the message that is presented to users prior to setting preferences on specific groups, users, and objects in the CSCE.

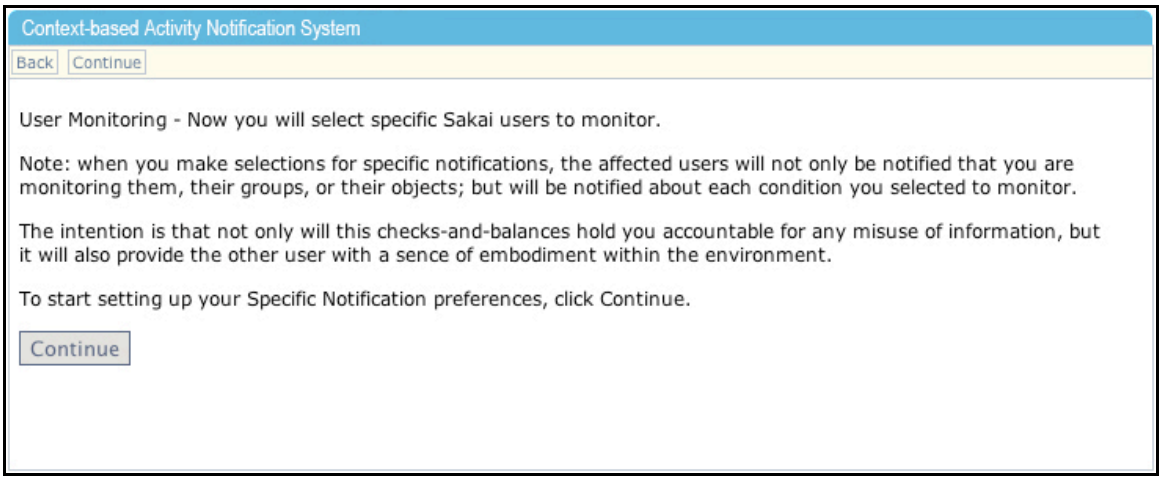


Figure 5.12 CANS Link: User Monitoring Instructions

The CANS Link Database

The CANS Link is a database driven notification server. Various tables in a MySQL database hold data and support the configurability and expandability of this server. Table 5.5 shows the database tables used in this version of the CANS Link. The remainder of this section is a discussion of the purpose and function of each database table.

Table 5.5 CANS Link Database Tables

Table Name	Description
environments	Environments supported by CANS
contexts	Social contexts within the CSCE
events	Events for each social context
subscriptions	Users subscriptions
event_history	History of every event submitted to CANS
notifications	History of every notification generated and received
users	Users in CANS
accounts	Account information for each user

The “environments” table stores information regarding each environment that CANS uses; including multiple CSCE, RSS readers, and various 3rd party

applications that can produce events or consume notifications. For this study, Sakai is the only environment supported; but any network-based application capable of generating events or displaying notifications could be integrated with CANS.

Because Sakai is the only environment represented in this study, and the name of the database table is “environments”, the term “environment” is used in place of the acronym CSCE for the remainder of this Section.

The “contexts” table contains information related to the various social contexts in CANS. When an environment is added to the CANS Link, the environment submits its contexts to the CANS Link to initialize this table. When a new group, user, or site is created in the environment, a corresponding context is also added to the “contexts” table.

The structure of a context is composed of the following attributes: name, type, state, environment id, and priority level. These attributes are columns in the “contexts” table. The environment id attribute associates the context with the appropriate environment and the name attribute stores the human-readable title for the context. The state attribute is used to grant or deny access to users in the environment. For example, a context with a state of “off” does not allow activity monitoring and notifications, but a state of “on” permits activity notifications.

The type and priority level attributes of a social context are important to the CANS Link. The type attribute is used to associate each context with a distinct context type. This association facilitates rapid event identification and

supports the customization of notification preferences in the CANS Link: Administrator application. As identified in the previous section, individuals select specific groups, users, and objects to monitor. This attribute is used to support that feature by allowing each type to be represented.

The priority level attribute is used to define the level of priority for notifications in each context. When a context is created in the CANS Link, it is given a default notification priority. This notification priority is used by the CANS Link: Consumer application when displaying notifications to users. Notifications with a priority of "1" are considered more relevant to the user than a priority of "2", which is more important than a priority of "3". Consequently, the Consumer application displays the priority "1" notification more prominently. For example, the notification "Dr. Laffey just opened your Dissertation.rtf resource" in Figure 5.13 represents a priority "1" notification; whereas, the notification "6 users are chatting in the Chit-chat site" in the same Figure represents a priority "2" notification.

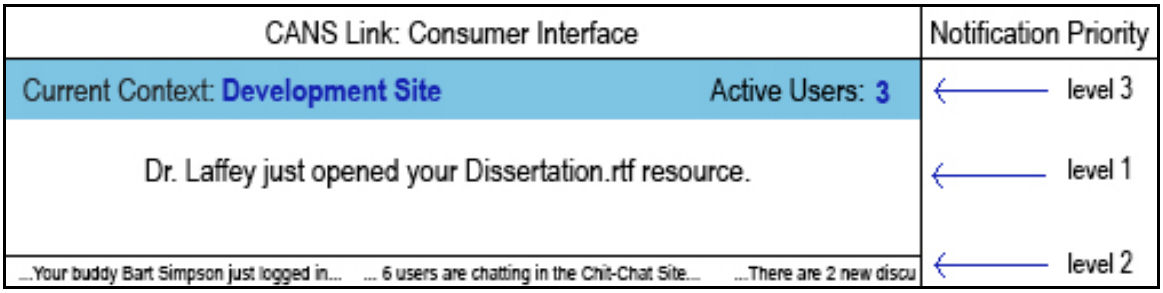


Figure 5.13 CANS Link: Consumer Application with Priority Levels

The "events" table stores information about each event that can occur in the environment. Each event is assigned to a single context. An event is defined

by the attributes: name, type, state, and context id. The purposes of the attributes name, state, and context id are the same as the corresponding attributes in the "contexts" table. The "type" attribute is an environment-defined identifier of an event. Refer back to Table 5.2 for a list of Sakai events.

Events are detected by first verifying that the event occurred in the appropriate environment. Then the social context is checked to determine contextual relevance. Finally, if the event type matches the user's notification preference, an activity notification is generated. In other words, a user's notification preference must first match the environment, then the context, and finally the event type before a notification is generated for a user. Notification preferences are stored in the "subscriptions" database table.

The "subscriptions" table stores each user's notification preferences. This table records the context id, event id, and the id of the user who created the subscription. The context and event ids are used to point to the data stored in "contexts" and "events" table. The "events_history" table stores every event that is sent to the CANS Link. Not only does this table provide a record of the activity that passed through CANS, but it also supports the delivery of asynchronous notifications. When a user logs into CANS, this table is checked to determine if any relevant events have occurred since the user's last notification. If events have occurred, appropriate notifications are generated and stored in the "notifications" table.

The “notifications” table is a repository for activity notifications. When a notification is generated, it is stored in this table. Because CANS supports asynchronous notifications, this table has a “notified” column to indicate if a notification was delivered to its recipient. If a notification is stored in this table, when the user is not available to receive it; the notification’s notified attribute is set to “false”. Only after the user receives the notification does this attribute get changed to “true”. The final two database tables are “users” and “accounts”. These tables store information related to each user.

Product Test

The previous sections in this Chapter provided information about the notification system developed during this study. Each level of CANS was described in detail and the unique contribution of the CANS Link layer was presented. This section describes a notification scenario that was used to (1) determine if CANS could deliver notifications based on user preference and social context and (2) collect data for the system comparison presented in the next section.

Notification Scenario: Description

This scenario was based on an object in a group within Sakai. The Sakai terms for object and group are resource and site respectively, but in this discussion the generic terms of object and group are used. The object was a text document in Sakai titled “Dissertation Feedback.txt”. The group was titled “Dissertation Defense”. Four different Sakai users were involved in this scenario.

The users were Chris, Dale, Jim, and Joi. Chris was the group owner and was the user who created the object. The other three users were members of the group. The permissions in the group were set to allow group members to view and revise the object. The actions performed by each of these users were designed and executed by this researcher to demonstrate the collaborative capabilities of CANS. While these names correspond with real people, those individuals were not a part of this activity.

Notification Scenario: User Preferences

The goal of this activity was to determine the notifications each of these users would receive while interacting with the object and in the group. Before presenting the actions and notifications that occurred, it is necessary to present each user's notification preferences. Without the knowledge of these preferences, an analysis of the actions and notifications is impossible. Table 5.6 shows the notification preferences relevant to this scenario. These preferences represent the Group, User, and Object preferences available via the CANS Link: Administrator application (Table 5.4).

Table 5.6 Notification Preferences for Scenario Participants

User: Chris		
Event	Group, User, or Object	Outcome
Login	Dale, Jim, Joi	Notification occurs when the users Dale, Jim, or Joi login to Sakai.
Viewed	Dissertation Feedback.txt	Notification occurs when any group member opens the object.
Revised	Dissertation Feedback.txt	Notification occurs when any group member revises the object.
New Chat	*current group*	Notification occurs when any group member posts a chat message in the

		same group where Chris is located.
New DB Post	Dissertation Defense	Notification occurs when any group member posts a new discussion board message.
New Object	Dissertation Defense	Notification occurs when any group member creates a new object.
User: Dale		
Login	Chris, Jim, Joi	Notification occurs when the users Chris, Jim, or Joi login to Sakai.
Logout	Chris, Jim, Joi	Notification occurs when the users Chris, Jim, or Joi logout of Sakai.
Viewed	Dissertation Feedback.txt	Notification occurs when any group member opens the object.
Revised	Dissertation Feedback.txt	Notification occurs when any group member revises the object.
New Chat	*current group **	Notification occurs when any group member posts a chat message in the same group where Dale is located.
New DB Post	Dissertation Defense	Notification occurs when any group member posts a new discussion board message.
New Object	Dissertation Defense	Notification occurs when any group member creates a new object.
User: Jim		
Login	Chris, Dale, Joi	Notification occurs when the users Chris, Dale, or Joi login to Sakai.
New Chat	*current group **	Notification occurs when any group member posts a chat message in the same group where Jim is located.
New DB Post	Dissertation Defense	Notification occurs when any group member posts a new discussion board message.
New Object	Dissertation Defense	Notification occurs when any group member creates a new object.
User: Joi		
--none--	--none--	--none--
This user can generate events, but does not have any active notification preferences to receive notifications.		

* represents the group the user is currently in and does not represent any specific group.

To summarize the data in Table 5.6, Chris and Dale's preferences were designed for activity occurring in the group, on the object, and with the group members. Jim's notification preferences were set to detect the creation of new items in the "Dissertation Defense" group and member logins, but he did not have preferences set for the "Dissertation Feedback.txt" object. Joi was a member of the Dissertation Defense group, but did not have any notification preferences defined.

Notification Scenario: Activity and Notifications

This section presents the activities that occurred and the resulting notifications each user received. Each numbered item represents an activity and the notifications resulting from the activity are represented by the lettered sub-items. The sequence of activities and notifications were:

1. Joi logs into Sakai.
2. Joi navigates to the Dissertation Defense group.
3. Dale logs into Sakai.
    a. Dale receives a priority 2 notification about Joi's recent login.
    b. Dale receives a priority 2 notification about Joi's recent navigation to the Dissertation Defense group.
4. Chris logs into Sakai.
    a. Chris receives a priority 2 notification about Joi's recent login.
    b. Chris receives a priority 2 notification about Joi's recent navigation to the "Dissertation Defense" group.

    c. Chris receives a priority 2 notification about Dale's recent login.

5. Joi posts a new chat message in the "Dissertation Defense" group.
    a. Dale receives a priority 2 notification about Joi's new chat message.
    b. Chris receives a priority 2 notification about Joi's new chat message.
6. Chris navigates to the "Dissertation Defense" group.
7. Dale navigates to the "Dissertation Defense" group.
8. Joi posts another chat message in the "Dissertation Defense" group.
    a. Dale receives a priority 1 notification about Joi's new chat message.
    b. Chris receives a priority 1 notification about Joi's new chat message.
9. Jim logs into Sakai.
    a. Jim receives priority 2 notifications about Joi's recent login, navigation to the "Dissertation Defense" group, and new chat messages.
    b. Jim receives priority 2 notifications about Chris' recent login and navigation to the "Dissertation Defense" group.
    c. Jim receives priority 2 notifications about Dale's recent login and navigation to the "Dissertation Defense" group.
    d. Dale receives a priority 2 notification about Jim's recent login.

e. Chris receives a priority 2 notification about Jim's recent login.

10. Jim logs out of Sakai.
    a. Dale receives a priority 1 notification about Jim's recent logout.
11. Dale opens the "Dissertation Feedback.txt" object.
    a. Chris receives a priority 1 notification about Dale opening the object.
12. Chris opens the "Dissertation Feedback.txt" object.
    a. Dale receives a priority 1 notification about Chris opening the object.
13. Dale closes the "Dissertation Feedback.txt" object.
14. Chris revises the "Dissertation Feedback.txt" object.
    a. Dale receives a priority 1 notification about Chris revising the object.
15. Dale navigates out of the "Dissertation Defense" group.
16. Chris revises the "Dissertation Feedback.txt" object.
    a. Dale receives a priority 2 notification about Chris revising the object.
17. Joi logs out of Sakai.
    a. Dale receives a priority 1 notification about Joi's recent logout.
18. Dale logs out of Sakai.
19. Chris logs out of Sakai.

Notification Scenario: Results

This notification scenario was performed to determine if CANS could deliver notifications based on user preference and social context. The data presented in Table 5.6 showed how user preferences differed. The most obvious difference was between Joi and the other users. As shown in the sequence of activities, Joi did not receive any notifications because she did not have any preferences defined.

Dale and Chris' preferences were very similar. However, Dale had a preference for user logouts that Chris did not. In the sequence of activities, Dale was the only user to receive notifications about a user logging out of Sakai. These two users also had notification preferences set for the "Dissertation Feedback.txt" object. When either user viewed or revised that object, the other was notified of the action. While not shown in this scenario, if the users Jim or Joi had viewed or revised that object, Chris and Dale would have received notifications about their actions on that object as well.

Based on this information, the ability of CANS to allow unique user notification preferences and to deliver notifications based on those preferences has been demonstrated. The ability of CANS to deliver notifications based on the user's current social context was also demonstrated in steps 5 through 8. Chris and Dale received priority 2 notifications about Joi chatting before they entered the "Dissertation Defense" group, i.e., a social context. However, these users

both received priority 1 notifications about the same activity after they navigated into the group.

Another feature of CANS revealed in this scenario was the ability to provide asynchronous and synchronous notifications to users. When Jim logged into Sakai at step 9, he received all relevant notifications that occurred before he was in the environment.

As this section has shown, CANS is capable of delivering notifications based on user preference and current social context. The next section shows how CANS extends the original notification capabilities of Sakai.

System Comparison

Prior to this study, Sakai offered few activity notification options to users. Figure 5.14 is a screenshot of the notification preferences that were available to users before CANS. As shown in the Figure, Sakai supported High- and Low-priority notifications and allowed users to monitor the occurrence of new announcements, emails, and resources.

Activity notifications in the original Sakai were delivered to users via email. As shown in the “Low Priority Notifications” section of Figure 5.14, users could receive individual emails when new announcements, emails, and resources became available or in a daily digest. Users could also choose to block notifications. In addition to notifications represented in this Figure, Sakai supported a “Users Present” feature by listing the names of each user active in a

Sakai Site. This feature was the only notification users received within the Sakai environment.

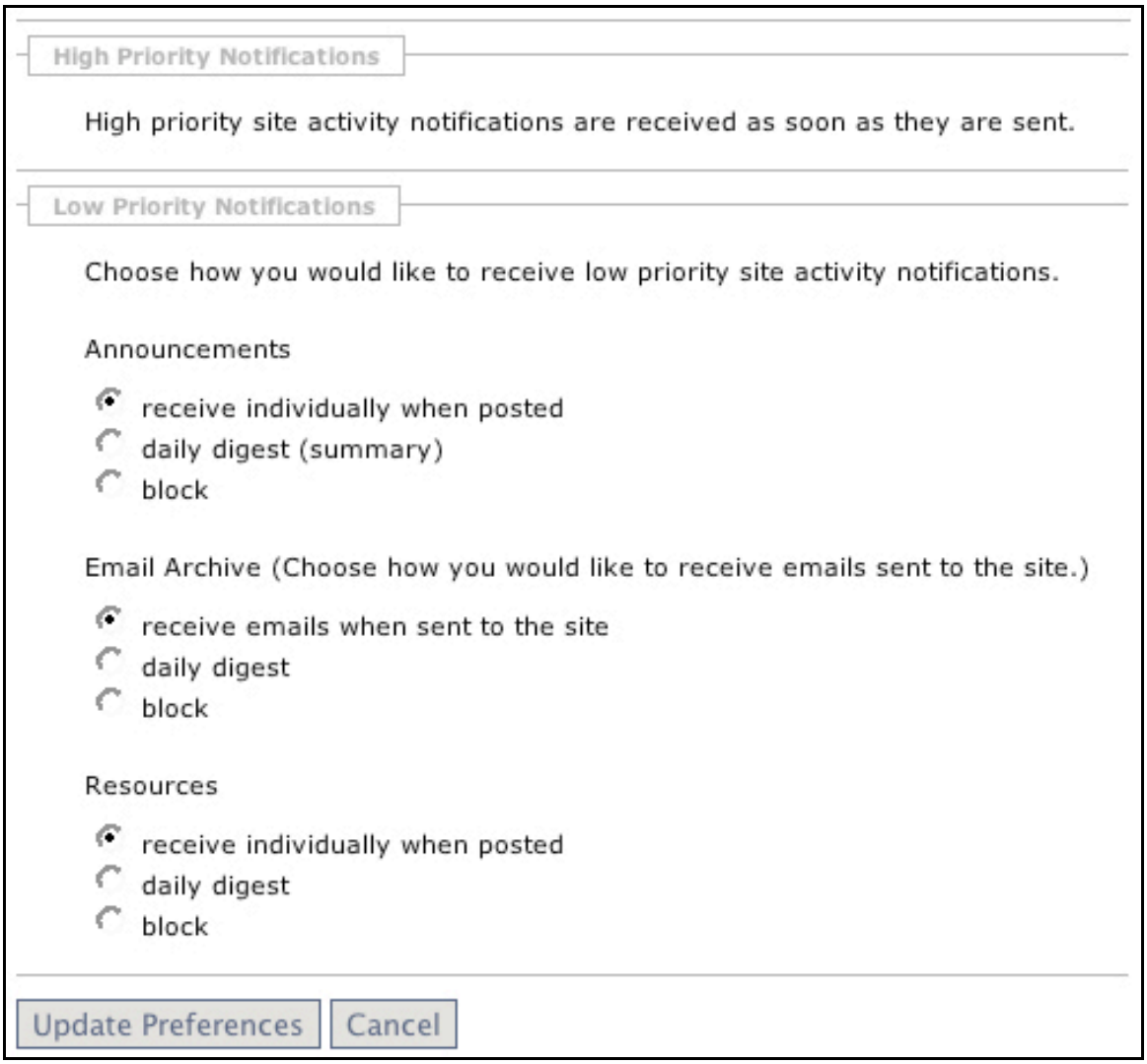


Figure 5.14 Initial Sakai Notification Administration Features

In contrast to Sakai's initial notification system, CANS is a robust and configurable system that enhances user awareness in Sakai. CANS supports the delivery of notifications within Sakai and, through its distributed and configurable nature, to external monitors.

By taking advantage of the efficiency and scalability of Siena; the configurability of the CANS Link; and the overall distributed nature of CANS, the basic model of CANS presented in Figure 5.1 can be extended to incorporate multiple environments. Figure 5.15 shows an extended model of CANS. The arrows in this Figure indicate the direction of communication between each component. As shown in this Figure, external event producers and consumers connect to the CANS Link. Through this connection, events occurring outside of the CSCE can be made available to users working within the collaborative environment. Due to the scalability of Siena, multiple CANS (Figure 5.1) can be linked together through the ENS. This extended system provided by CANS is not supported in the original version of Sakai.

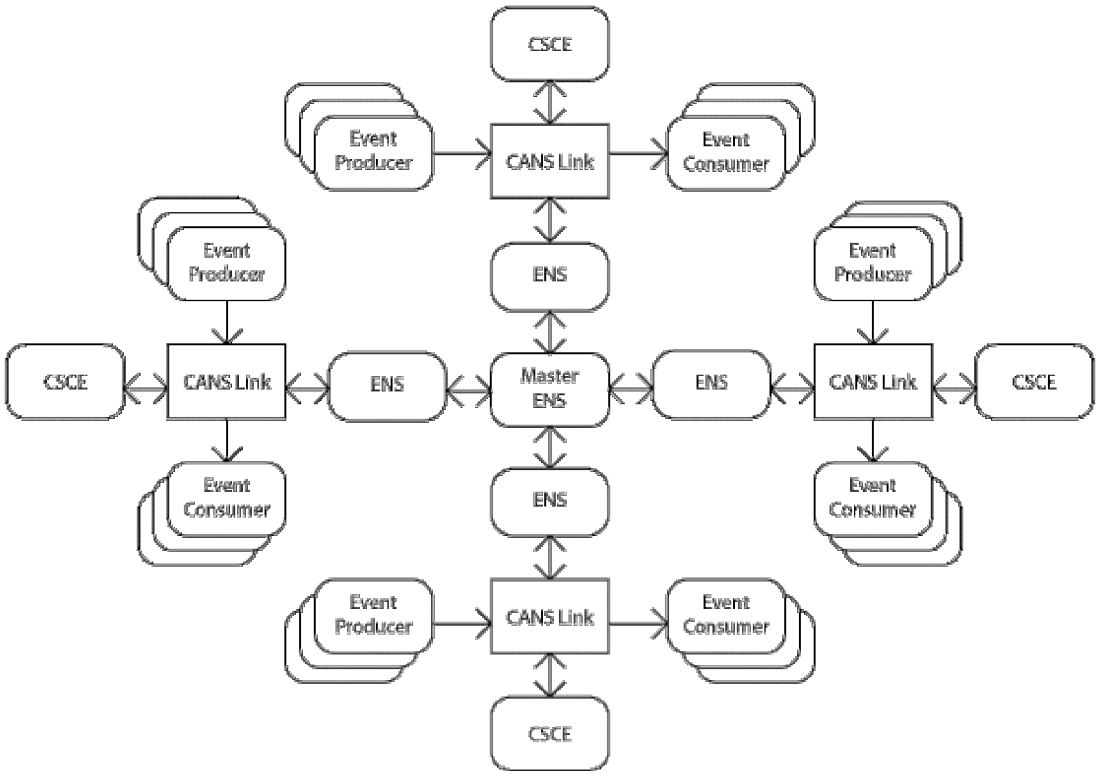


Figure 5.15 Expanded Context-aware Activity Notification System

CANS also monitors a wider range of events occurring within the original Sakai (Table 5.3) and CANS provides users extended notification configuration opportunities through the CANS Link: Administrator application. Table 5.7 shows the enhancements CANS provides over the original Sakai.

Table 5.7 System Comparison

Sakai		
Feature	Pre- CANS	Post- CANS
Asynchronous Notifications	Limited	Yes
Synchronous Notifications	Yes	Yes
Context-aware Notifications	No	Yes
Internally-Displayed Notifications	Limited	Yes
Externally-Displayed Notifications	Limited	Yes
External Events	No	Yes
Event History	Limited	Yes
Notification History	No	Yes
Support Interaction after Notifications	No	No
Monitor Specific Group Events	No	Yes
Monitor Specific Object Events	No	Yes
Monitor Specific User Events	No	Yes
Notification Preference Customization	Limited	Yes

The “Limited” features of pre-CANS Sakai indicate that the feature was supported in a rudimentary fashion. For example, asynchronous notifications are only provided by sending a daily digest of all activity in a single email once a day. In addition, “Users Present” was the only notification received within Sakai and the only method of receiving notifications outside Sakai was through email. Finally, as shown in Figure 5.15, the pre-CANS Sakai supported very few customization features.

The only feature not supported by CANS in Table 5.7 was “Support Interaction after Notifications”. The purpose of this feature was to facilitate user

interaction after the receipt of activity notifications. To support interactions between users, the notification system would either need to provide its own communication mechanism or control the CSCE software. An example of the latter is that the notification system might launch the CSCE's chat software and automatically add the appropriate users to the chat room. Neither option was found appropriate for a context-aware activity notification system during this study. It is recommended that to fully meet these interactional requirements the CSCE itself should be modified to accommodate the Framework for Notification.

Summary

This Chapter presented the research methods used to determine the extent to which the principles of the Framework for Notification could be used to develop and integrate a notification system into an existing CSCE. This chapter showed the outcome of using that Framework: CANS, and showed how CANS advanced the notification capabilities of Sakai by providing notifications based on user preference and current social context. CANS does not represent a finished product, but instead reveals the potential for future research made available through following the principles of the Framework for Notification. The next Chapter discusses how the Framework influenced the CANS design, the implications this development effort had on the Framework, and opportunities for further research.

CHAPTER VI

DISCUSSION AND IMPLICATIONS

To facilitate the development of new, innovative notification systems, this study articulated and advanced a theoretical framework for development based on social context and user preference that could be used to integrate activity notifications into existing CSCE. That framework is called the Framework for Notification. To understand the requirements of notification in CSCE, build new knowledge for how to meet those requirements and test the implications of using this framework, a notification system was designed and developed based on the framework's principles. During development, the initial framework was examined, tested and revised into the principles presented in Chapter four. In the end, a notification system and a framework for development was created to provide activity notifications based on user preference and social context.

This Chapter discusses the implications of the new Framework for Notification by explaining how the proposed Framework influenced the design of the notification system developed during this study. Then the results of using this Framework to create a Context-aware Activity Notification System (CANS) and consideration for the initial research problems presented in this study are discussed. Finally, theoretical and practical implications and suggestions for future research are identified.

The Framework for Notification

The Framework for Notification presented in this study was designed for developers to use when creating notification systems for existing CSCE. The Framework is based on recognition of the importance of user preference and social context and challenges the developer to consider why notifications are presented to users as opposed to just delivering notifications when an action occurs. As described in Chapter four, the Framework uses the principles of Social Context, Awareness in Context, Activity Discovery, Trends in Activity, Meaning of Activity, and Notification Customization to articulate how a developer can create a notification system based on users' social contexts and personal preferences in a CSCE.

The principles of this Framework were derived from an analysis and interpretation of the Locales Framework, a study of Dourish's concept of *embodied interaction*, and a review of existing network-based notification systems. The Locales Framework influenced the design of the Framework for Notification by indicating how a user's actions evolve based on the actions of others, how interaction occurs within social worlds, and how the interactional needs of users are important for user awareness and collaboration (Fitzpatrick, 1998). The concept of embodied interaction influenced the Framework by describing how users need to derive meaning from interactions with artifacts (Dourish, 2001). During the review of prior work, an absence of affordances to

support these principles was found in existing notification systems, which reinforced the need for this research (see Chapter three and Appendix A).

Advancing the Framework

The development of CANS was used to interact with and advance the principles of a proposed framework for developers. The initial framework, originally called the Framework for Notification Integration, focused more heavily on the social context of work and less on the user's need and purpose for receiving awareness information than the Framework for Notification. This section describes how the development of CANS advanced the Framework for Notification.

During this study, this researcher used the evolving Framework to create eleven different versions of CANS. After each version, the proposed framework was analyzed and modified to be more understandable for developers, more useful for developing context-aware notification systems, and more attuned to the users acting and interacting in CSCE.

The initial framework was composed of six principles: Social Context, Activity Discovery, Trends in Activity, Awareness through Notification, Presence through Embodiment, and Notification Customization. Figure 6.1 reveals the principles of the initial framework and how it related to the original notification system design.

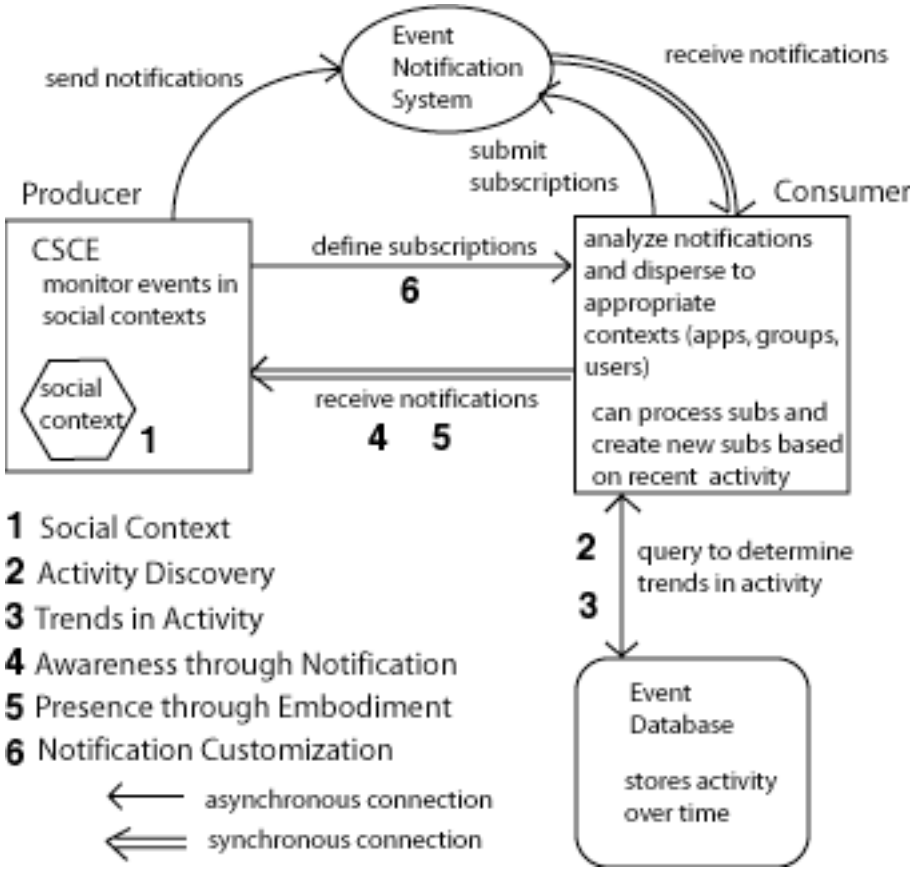


Figure 6.1 Diagram of Original Notification System

Throughout this study, the Social Context principle itself did not change significantly, but the other five principles, which are related to Social Context, did change. During an analysis of Sakai, the realization that a user's social context extends beyond the boundaries of the collaborative environment caused the researcher to make the Activity Discovery principle more attuned to the actions occurring in the Internet and physical world.

The Trends in Activity principle was altered during development when it was realized that the notification system could store both the activity and notification histories of the CSCE. This understanding led to an effort to create a flexible database design and data structure to support data analysis.

Principles four and five (see Figure 6.1) of the original framework were found to be difficult to interpret and to put into practice during the development of CANS. Consequently, the Awareness through Notification principle was transformed into Awareness in Context and the Presence through Embodiment principle was changed to Meaning of Activity. The most significant change of these two principles was in the Meaning of Activity principle. While working with the CANS Link and the CANS Link: Administrator, the importance of supporting the user's ability to construct meaning from the actions arising in social contexts was reached and subsequently articulated in the Meaning of Activity principle in the proposed Framework.

While the final principle of Notification Customization didn't change in name, the importance of this principle was realized during software development. After each version of CANS, the researcher interacted with the system and uncovered new features that were not previously considered. During this process, it was realized that no matter how valid the concept of Social Context might be in activity notification systems, every user is unique and thus requires unique awareness information. Only by creating a configurable notification system can a developer provide a mechanism to enhance social collaboration in online communities. Figure 6.2 reveals the final principles of the Framework for Notification and shows how those principles fit into the design of CANS.

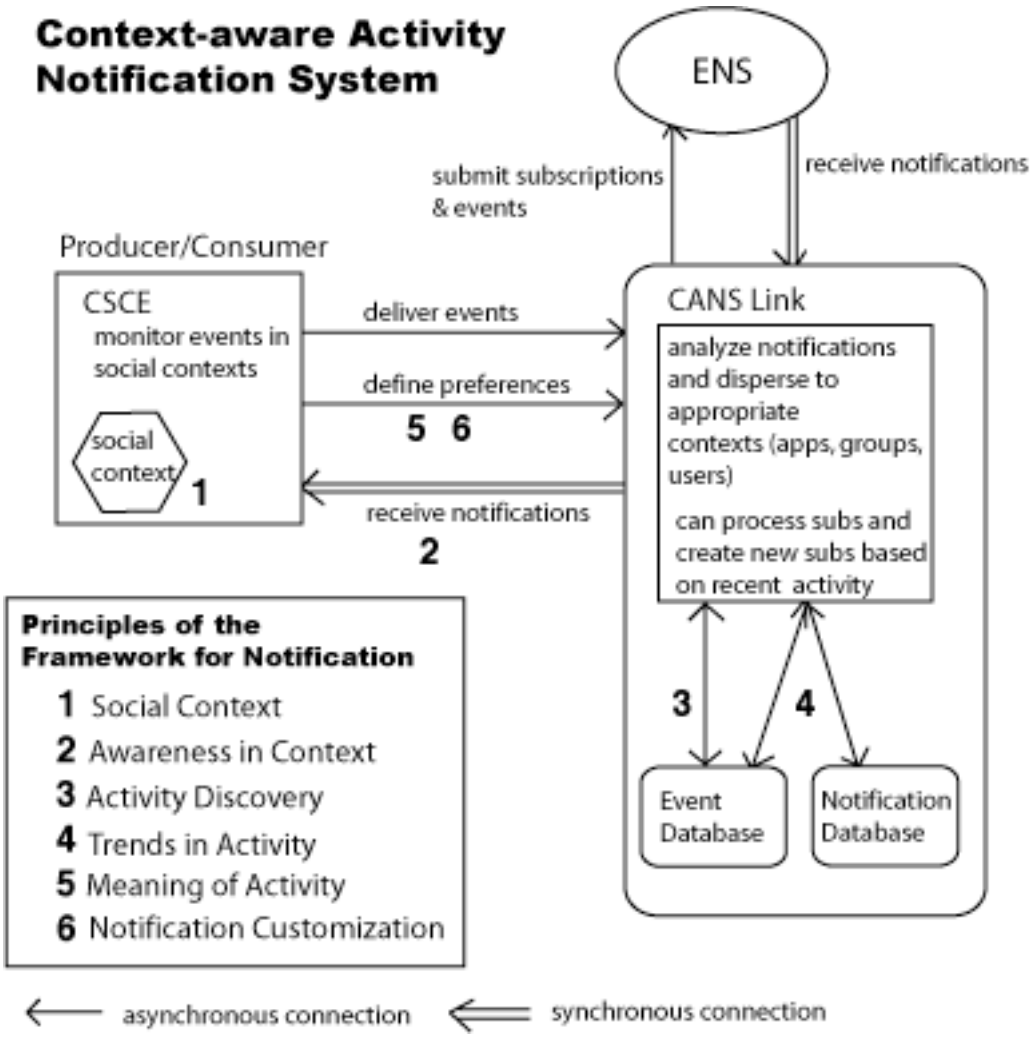


Figure 6.2 Final Framework and CANS

This section described how the development process influenced the Framework for Notification. The next section discusses how the proposed Framework also influenced the development of CANS.

The Context-aware Activity Notification System

As described in Chapter five, CANS was composed of three separate layers each of which may reside on separate servers to improve system performance. Those layers were: an Event Notification System, the CANS Link, and a Computer Supported Collaborative Environment. The structure and

relationship of these layers was influenced by the results of the review of prior work presented in Chapter three and the principles of the Framework for Development. In the review of prior work, the existing notification systems of iScent, Groove, and Sticker were analyzed for their unique notification strategies. The relevant features of each of those systems were identified and used in the development of CANS. Refer back to Figure 3.7 for a list of those relevant features.

The primary contribution of this study is the Framework for Notification and the demonstration through the architecture of CANS for how the Framework can be used to influence the design and development of new notification systems for existing CSCE. The Framework for Notification articulates that a notification system must deliver notifications to support user preference and current social context; otherwise, it will not meet the full collaborative needs of the users in the CSCE.

Framework for Notification's Influence on CANS

During the development of CANS, the Framework for Notification influenced design and development in many ways. First and foremost, the Framework emphasized the need to base the delivery of notifications on the user's current social context. A social context is defined as the socially constructed place for user actions and interactions defined by current membership, the collective goals of individuals, recent activity, and the communicative affordances of the technology. From this definition, a social

context in Sakai, the CSCE used in this study, was identified as an object, a tool, a group, and/or the entire community.

The Social Context principle required the researcher to analyze CSCE activity from the point of view of a user to determine how notifications could be delivered based on social context. An action in a CSCE was conceptualized into a hierarchical structure of identifying attributes as identified in the SXML mapping presented in Chapter five. In that mapping the following attributes were identified and used to define an action in Sakai: environment id, context id, context name, context type, event action, event object, event tool, author id, and author name. From those attributes, all context-aware notifications in CANS were generated and delivered to users in the CSCE.

Second, the Framework influenced the design of CANS by revealing the users' need to know about activity occurring outside their current context. The Framework showed how a user is primarily interested in the activity in their current social context, but still has a need to know about relevant activity occurring elsewhere. The Activity Discovery principle indicates that this external activity should be presented to users, but not as pronounced as notifications related to their current context. This principle led to the development of different *priority levels* for notifications in CANS. These priority levels are used to determine how users receive notifications.

The Activity Discovery principle also revealed the need for a notification system to monitor activity external to the CSCE and to present those notifications

internally within the CSCE. As shown in the Sakai Conditional Matrix in Chapter five, users actions within a CSCE are also influenced by activity occurring elsewhere on the Internet and in the physical world. Knowing about those external actions within the CSCE may influence the actions and interactions of users.

Third, the Framework for Notification influenced the design of CANS by identifying how activity is part of a process and how the knowledge of activity through notifications impacts the goals and outcomes of those processes. Through the principle of Trends in Activity, the Framework caused the researcher to incorporate into CANS not only an activity history, but a notification history as well. The process of storing both histories was not found in prior work, but was found to be very important when determining the trajectories of user activity. Through an analysis of these two histories, it may be possible to measure the effect notifications have on user actions and interactions. While measuring this effect was beyond the scope of this study, analyzing this data is a suggestion for future research presented later in this Chapter.

The next influence of the Framework for Notification came from the Meaning in Activity principle. This principle states that a notification system should provide a mechanism for users to interpret and construct meaning from the activity occurring in a context. During development and interaction with CANS, the researcher struggled with this concept but then came to the conclusion that a user may be able to construct meaning from activity

notifications by assigning meaning to the notification *before* the activity occurs. A notification system cannot impart meaning onto a notification; it is up to the individual to determine the meaning. If a user configures the notification system so that he or she knows why, when, how, and where a notification is received, then that user may be more likely to understand the reason for and thus the meaning of the notification when it is received. The extent to which this mechanism supports this principle has not been determined and is another area of future research to be discussed. Nevertheless, the importance of this principle remains. A notification system should provide a mechanism for users to interpret and construct meaning from activity notifications. The principle of Meaning of Activity contends that notifications provide the most benefit when they have meaning to the user.

Finally, the principle of Notification Customization influenced the design of CANS by identifying the need for users to make the final decisions on the notifications they receive. In all the existing notification systems reviewed, the systems either did not provide mechanisms to configure notification preferences or did not provide users with a set of notification preferences as configurable as developed in CANS.

As described in this study, a user in a CSCE works and interacts within various social contexts and those contexts are used to determine the notifications a user will receive. However, while working with the Framework for Notification, the realization was made that the interests of users and social contexts are not

static, but are dynamic and sometimes unpredictable. To compensate for this unpredictability, a notification system should allow users to define their own notification preferences that transcend the bounds of whatever collaborative construct the notification system initially imposes on users, even the construct of a social context.

Limitations

Before considering the implications of this study, the limitations of this research are described. The first and most significant limitation of this study is: this researcher designed both the Framework for Notification and the Context-aware Activity Notification System. This limitation affects the generalizablitiy of the framework for fully supporting the development of awareness systems. Because the framework and system were both developed by the same person, this study has not tested the ability of the framework to support the development of a system by another developer. It is difficult to measure how fully this Framework was articulated and in what ways further articulation may be needed. This researcher may have made assumptions about the Framework that were not fully explained in Chapter four. Once other developers use the Framework, it may be found that one or more of the principles of the Framework for Notification must be reanalyzed and explained more fully to support more widespread use.

The other limitation of this study is that the entire CANS was developed on a single Apple G4 laptop because access to multiple servers and a more authentic production environment was not possible. Knowing how the CANS

system performed on a distributed network as described in Chapter five would have allowed the researcher to assess the performance capabilities of the notification system. This researcher acknowledges the importance of performance testing, but also recognizes the unreliability of the performance data from the current implementation. Continuing the development of CANS and testing it on multiple servers is a suggestion for future research presented later in this Chapter.

Implications

The results of this study suggest that developers can use the Framework for Notification to create a context-aware notification system for existing CSCE based on the aspects of the Locales Framework. One of the limitations with previous notification systems, as identified in Chapter three, was an inability to include all aspects of the Locales Framework into a single collaborative environment. Because the Framework for Notification was derived from an interpretation of the Locale's aspects and the Framework was used to successfully create CANS; it was concluded that, in this study, CANS addressed all aspects of Locales.

Another limitation of existing systems has been a lack of notifications based on the actions and social contexts of users. This limitation was overcome in this study because the Framework for Notification articulates the importance of social context and was used to develop a notification system based on the

actions and social contexts of users in the CSCE, as shown in the previous Chapter.

The final limitation that was identified in Chapter three, that this study would address, was the absence of activity notifications for personal embodiment in previous systems. The implications of this limitation and how it was addressed with CANS were not fully realized through this development research and will require further study. The implications related to these results and others from this study are discussed below.

Theoretical Implications

The purpose of this study was to advance a theoretical framework for development that could be used to integrate activity notifications into existing CSCE. To achieve that purpose, this study adapted an existing theoretical framework (Locales) to create the new Framework for Notification. The principles of this new Framework offer theoretical guidance for researchers exploring activity notifications in CSCE.

By using the Locales-influenced Framework for Notification to create CANS, this study showed how the proposed Framework could be used to incorporate all aspects of Locales into a CSCE. The implication of this outcome is that by translating Locales into a more succinct language that focuses on the concept of user preferences and social context, instead of the more abstract concepts of social world, site and means, locales, civic structures, etc.; this new Framework is more accessible to the developers responsible for creating

notification systems in CSCE. This study does not imply that developers are incapable of understanding Locales; it simply addresses how developers can use a transformational framework to achieve the theoretical aspects of Locales because the need for some form of transformative support was recognized by Fitzpatrick when she articulated the difficulty of interpreting and using Locales in everyday use.

The results of this study indicate that Locales can be applied to existing systems. In the past, Locales was only used to study its influence on the development of new collaborative environments, i.e., wOrlds and Orbit (Fitzpatrick, 1998 & Dourish, 2001). However, because this study developed a notification system for an *existing* CSCE, the implication is that through the use of the Framework for Notification, the Locales Framework can be applied to existing collaborative environments as well – a concept not previously addressed by Fitzpatrick.

The next theoretical implication is that this new Framework serves as a communication bridge between the theorists interested in the purpose and meaning of activity and the information scientists responsible for understanding the technology needed to support activity notifications in online environments. The Locales Framework was based on sociological understandings and, as claimed by Fitzpatrick, provides a common language for researchers to discuss interactions in collaborative environments (1998). Likewise, the Framework for Notification provides systems researchers and developers with a language to

discuss the creation of notification systems to support context-aware activity notifications. The implication is that because the system-centered Framework for Notification is a result of an interpretation of sociologically based Locales and each framework serves as a common language for the respective disciplines, researchers from both disciplines may be able to communicate more easily through the use of these two frameworks.

The next implication of this study is that even when a user is working within a CSCE, that user's actions and interactions are not limited to that virtual environment. As physical beings, people are part of the natural world and consequently are influenced by activity in that world even when working in a CSCE. As the conditional matrix in Chapter five showed (Figure 5.5), the Internet and the physical world are additional social contexts relevant to a users actions when they are in a CSCE. As a result, consideration for social contexts that extend beyond the bounds of a CSCE should be included in discussions related to activity notifications and awareness in online collaborative environments. If a person is meeting with someone physically in his or her office, that information may be relevant to the user in a CSCE.

An outcome of this study was that actions occurring within a CSCE could be defined through a set of attributes. As already discussed earlier in this Chapter, an action in Sakai was defined by an environment id, context id, context name, context type, event action, event object, event tool, author id, and author name. From those attributes, every context-aware notification in CANS was

generated and delivered to users in the CSCE. The implication of this outcome is that researchers and developers should consider how an action is constructed in a CSCE and determine what attributes can be identified to provide context-aware notifications, which also implies that consideration for what the appropriate attributes are should be studied in future research.

The final theoretical implication of this study was that customizable notification preferences might support a user's ability to create meaning from activity notifications. The contention is that users who define the reasons for notifications prior to receiving the notification require less cognitive processing to determine the meaning of the notification. As previously stated, the validity of this concept was not measured during this study and requires further research.

Practical Implications

A result of this study is that a context-aware notification system can be integrated into an existing CSCE using the Framework for Notification. The practical implication is that more consideration can now be given to activity notifications in new and existing collaborative environments. Even though a trend from information- to community-centered environments was illustrated in Chapter three, an analysis of the new Sakai environment showed that user awareness and activity notifications was not a priority in Sakai development. If it were, Sakai would offer more notification features than identified in the "System Comparison" section of Chapter five. A shift in developer's thinking toward the belief that tools to support work processes should include tools that make activity visible and

support user interactions is needed. To improve efficiency in the monitoring of activity and to support the embedding of notifications into the contexts of a CSCE, new environments should be designed with the importance of activity notifications and the principles of the Framework for Notification in mind.

Another practical implication of this study is that a notification system can be developed to provide notifications to a CSCE about activity occurring elsewhere on the Internet or even in the physical world. Through the use of RSS feeds or physical devices like cameras and Bluetooth enabled cell-phones or PDAs, event producers could be created to send activity information to a notification system like CANS. For example, a Bluetooth enabled cell-phone could automatically notify a network-connected sensor when a person of interest arrives at his or her office. The sensor could deliver a properly formatted XML event to CANS, which would then format that event into a notification for users in the CSCE. Likewise, through the use of CANS, events occurring within Sakai could be delivered to external event monitors that reside on a user's cell-phone, computer desktop, or various other devices. These implications from the articulation of the Framework for Notification and the development of CANS are interesting and lead to some of the suggestions for future research presented next.

Suggestions for Future Research

Further research is encouraged to confirm the usability of the proposed Framework for Notification and to advance the development and use of the

Context-aware Activity Notification System. As stated earlier, the development of CANS is incomplete. Further work is required to fully implement all of the features available in CANS and to transfer the system to a collection of servers for extensive stress testing and debugging. At that point, CANS should be integrated into a production environment and tested with real users in Sakai to determine its effect on user actions and interactions.

Additional development research should be performed to determine the extensibility of this system as shown in Figure 5.13. CANS was designed to incorporate external event producers and consumers with the activity in Sakai. This functionality was built into CANS to support the finding that user actions within a CSCE are not only influenced by the social contexts within the collaborative environment, but also the contexts external to the CSCE. Only through further development and testing can researchers determine the practical and theoretical implications as well as the collaborative benefits of this extensible system.

Another suggestion for future research is related to methods of displaying activity notifications to users. CANS provides different priority levels for notifications, but does not enforce any user interface requirements or techniques regarding the use of these priority levels. Research has been done related to the development of social proxies and other techniques for displaying notifications, i.e., the Babble system by Erickson, but the architecture of CANS could extend that research by supporting a rapid prototyping approach for exploring different

notification visualization strategies. Because the notification-processing component of CANS is separated from the notification delivery layer, customization of the notification interface is relatively straightforward.

Another area of future research the Framework for Notification and CANS affords is the study of activity trajectories in a CSCE. During the development of CANS, a mechanism to store both the activity and notification histories of the CSCE was developed. The activity history is important because it is used to support the delivery of notifications asynchronously. The notification history is needed to determine the notifications a user has received and, more interestingly, to analyze how notifications influence activity. Both histories store all the attributes of action as presented earlier with SXML mapping and also a time-stamp of when the activity or notification occurred. Through an analysis of the action's attributes and when an action occurred, a trajectory of activity might be established. Initially, this analysis will require someone to process the data and generate the trajectory, but additional development within CANS may support the automatic analysis of this information, which could then be used to dynamically suggest new notification preferences to users.

Finally, the effect notification customization has on the user's ability to construct meaning from activity should be researched. Users are embodied in the CSCE when their engagement with artifacts allows them to create meaning. Whether the current approach of empowering users with notification customization affords the creation of meaning in ways that are important to

activity has not yet been determined, but the opportunity for further study and refinements for the principle of Meaning of Activity is made possible by implementing CANS.

Conclusion

At the beginning of this study, three research goals were identified to show the work of this study. Those goals were (1) articulate a development framework for integrating activity notification into existing CSCE, (2) design and implement a notification system based on the proposed framework for an existing CSCE, and (3) design, develop, and test use-case scenarios to demonstrate the effectiveness of the proposed framework.

The first goal of articulating a development framework was successfully achieved through the design of the Framework for Notification as presented in Chapter four. This Framework shows the importance of a user's personal preferences and current social context when providing activity notifications in a CSCE. It alerts developers to the idea that activity is part of an ongoing process in collaborative environments and activity notifications can reveal the trajectory of actions and influence the goals and outcomes of that ongoing process. This Framework also proposes that notification customization is more important than previously identified in past research. Allowing users to customize how, when, why, and where they receive notifications allows the user to understand notifications more fully, which may facilitate the creation of meaning from activity notifications.

The second goal of developing a notification system based on the principles of the proposed Framework was achieved through the creation of CANS – the Context-aware Activity Notification System. During development, the principles of the Framework and its focus on the importance of social context was used to create a notification system that delivered notifications not simply because of resent occurrence in the CSCE, but because the user's current social context made the notification relevant to the user's current interest and activity. Additionally, the Framework's principle of Notification Customization influenced the development of a notification system that allowed users to configure almost every aspect of an activity notification thereby empowering the user with their personal notification preferences.

The final goal of this study to test the notification system through the use of use-case scenarios was demonstrated in the latter half of Chapter five. Four different user accounts, each with their own notification preferences, were defined and used to identify the notifications CANS would provide when those four users acted and interacted in a group and around a specific object. As was shown in that Chapter, CANS successfully presented each user with notifications based on user preference and current social context.

In conclusion, this study has advanced a theoretical framework for development that integrates activity notifications into an existing CSCE. Additionally, this work provides future researchers with a configurable research tool that can be used in future studies related to social computing and awareness

in online collaborative environments. That tool is known as the Context-aware Activity Notification System (CANS).

REFERENCE LIST

Anderson, K., & Bouvin, N. (2000). Supporting Project Awareness on the WWW with the iScent Framework. *ACM SIGGROUP Bulletin*, 21(3):16–20, 2000.

Budzik, J., Fu, X., & Hammond, K. (2000). Facilitating Opportunistic Communication by Tracking the Documents People Use. *Proceedings of the CSCW 2000 International Workshop on Awareness and the WWW*.

Cadiz, J., Fussell, S., Kraut, R., Lerch, F., & Scherlis, W. (1999). *The Awareness Monitor: A Coordination Tool for Asynchronous, Distributed Work Teams.* Unpublished Manuscript. Human-Computer Interaction Institute, Carnegie Mellon University. Pittsburgh, PA.

Cadiz, J., Venolia, G., Jancke, G., & Gupta, A. (2001). *Sideshow: Providing Peripheral Awareness of Important Information*. Technical Report, MSR-TR-2001-83. Microsoft Research, Microsoft Corporation. Redmond, WA.

Cadiz, J., Venolia, G., Jancke, G., & Gupta, A. (2002). Designing and Deploying an Information Awareness Interface. *Proceedings of the ACM conference on Computer Supported Cooperative Work,* pp.314-323, New Orleans, LA.

Carzanigia, A. (1998). Architectures for an Event Notification Service Scalable to Wide-area Networks. Plitecnico Di Milano.

Dourish, P. (2001). *Where The Action Is: The foundations of embodied interaction*. MA: MIT Press.

Dourish, P., & Bellotti, V. (1992). Awareness and Coordination in Shared Work Spaces. *Proceedings of the ACM conference on Computer-Supported Cooperative Work*. Toronto, Canada.

Erickson, T., & Kellogg, W. A. (2003). Knowledge Communities: Online Environments for Supporting Knowledge Management and its Social Context. *Beyond Knowledge Management: Sharing Expertise.* Cambridge, MA. MIT Press, pp. 299-326.

Espinosa, A., Cadiz, J., Rico-Gutierrez, L., Kraut, R., Scherlis, W., & Lautenbacher, G. (2000). Coming to the Wrong Decision Quickly: Why Awareness Tools Must be Matched with Appropriate Tasks. *Proceedings of the CHI 2000 conference on Human factors in computing systems*, pp. 392-399, NY: ACM.

Fitzpatrick, G. (1998). The Locales Framework: Understanding and Designing for Cooperative Work. PhD Thesis, Department of Computer Science and Electrical Engineering. The University of Queensland.

Fitzpatrick, G., Mansfield, T, & Kaplan, S. (1996) Locales framework: Exploring foundations for collaboration support. In J. Grundy and M. Apperley, editors, *Proceedings Sixth Australian Conference on Computer-Human Interaction*, pages 34–41. IEEE Computer Society Press.

Fitzpatrick, G., Mansfield T., et al. (1999). Instrumenting and Augmenting the Workaday World with a Generic Notification Service Called Elvin. *Proceedings of ECSCW*. Copenhagen, Denmark.

Fitzpatrick, G., Parsowith, S., Segall, B., & Kaplan, S. (1998). Tickertape: Awareness in a Single Line. short paper in *CHI'98 Summary*, ACM Press, Los Angeles, pp. 281-282.

Groove Networks. (2002). *Groove Networks*. Retrieved November, 2003, from http://www.groove.net.

Groove Networks. (2003). *Groove Desktop Collaboration Software: Product Backgrounder*. Retrieved November, 2003, from http://www.groove.net/pdf/backgrounder-product.pdf.

Handel, M., & Wills, G. (2000). TeamPortal: Providing Team Awareness On the Web. *Proceedings of the International Workshop on Awareness and the WWW. ACM CSCW'2000,* 21(3), pp. 3-12. Philadelphia, PE.

Harvey, B. (1991). Symbolic programming vs. the A.P. curriculum. The Computing Teacher, 18(5), 27-59, 56.

Intraspect. (2003). *Intraspect - Products*. Retrieved on October, 2003, from http://www.intraspect.com/products/.

Ishida, T. (Ed.). (1999). *Community Computing and Support Systems: Social interaction in networked communities*. New York: Springer.

Jang, C., Steinfield, C., & Pfaff, B. (2000). Supporting Awareness among Virtual Teams in a Web-Based Collaborative System: The TeamSCOPE System. *ACM SIGGROUP Bulletin*, 21 (3):28-34, 2000.

Leiner, B., Cerf, V., Clark, D., Kahn, R., Klienrock, L., Lynch, D., Postel, J., et. al. (2000). *A Brief History of the Internet*, version 3.31. Retrieved November, 2003, from http://www.isoc.org/internet/history/brief.shtml.

Mansfield, T., Kaplan, S., & Fitzpatrick, G. (1997). Evolving Orbit: a progress report on building locales. *Proceedings for Group'97*, ACM Press, Phoenix, AZ.

Microsoft Corporation. (2003). *SharePoint Products and Technologies 2003 Software Development Kit (SDK)*. Retrieved November, 2003 from http://www.microsoft.com/downloads/details.aspx?FamilyId=AA3E7FE5-DAEE-4D10-980F-789B827967B0&displaylang=en.

Moody, P. B. (2000). The Role of Awareness in Social, Collaborative and Shared Activities. *Proceedings of the International Workshop on Awareness and the WWW*. ACM CSCW'2000 Conference. Philadelphia, Pennsylvania. ACM Press, New York.

Orbit-Gold. (2003). *Orbit Gold*. Retrieved October, 2003, from http://www.dstc.edu.au/Research/Projects/Worlds/Prototypes/OrbitGold.html.

Pfaff, B. (2001). *TeamSCOPE Tutorial and Reference*. Retrieved November, 2003, from http://cscw.msu.edu/scope/doc/scope.html.

Phillips, M. (2002). *Sticker User's Guide*. Retrieved September, 2003, from http://elvin.dstc.com/projects/sticker/sticker_user_guide.html.

Prinz, W. (1999). NESSIE: An Awareness Environment for Cooperative Settings. *Proceedings of The Sixth European Conference on Computer Supported Cooperative Work* (ECSCW'99). Copenhagen, Denmark. pp. 391-410.

Distributed Systems Technology Centre. (2003). *Project: Sticker*. Retrieved November, 2003, from http://elvin.dstc.com/projects/sticker/.

Sakai. (2005).*Sakai*. Retrieved February, 2005, from http://www.sakaiproject.org/cms/index.php?option=com_content&task=blogcategory&id=141&Itemid=258.

Siena. (2004). *SXML.* Retrieved December, 2004, from http://serl.cs.colorado.edu/~serl//siena/sxml/index.html.

Steinfield, C., Jang, C., & Pfaff, B. (1999). Supporting Virtual Team Collaboration: The TeamSCOPE System. *Proceedings of GROUP'99: International ACM SIGGROUP Conference on Supporting Group Work*. pp. 81-90, ACM Press. Phoenix, AZ.

Strauss, A., & Corbin, J. (1994). *Grounded theory methodology*. In N. Denzin & Y. Lincoln (Eds.) The Handbook of Qualitative Research. (pp. 273-285). Thousand Oaks: Sage Publications.

Walker, D., & Bresler, L. (1993). Development research: Definitions, methods, and criteria. *Paper presented at AERA*. Atlanta, GA.

APPENDIX

Appendix A

Collaborative Network-based Systems

System	Audience	Features	Information	Presentation	Implementation
Aware-ness Monitor (Cadiz 1999)	Students participating in a realistic business simulation called "Management Game"	- Help teams monitor changes to important resources while imposing minimally on their attention. - Provides passive awareness of others' activities. - Provides an extensive range of external information along with limited within-team information.	- With-in Team Information: work progress & load; flow of communication; who knows what information - External Information: environmental changes outside the team context	- monitor: presents a piece of data and the set of rules to specify how the data should be watched. - Awareness Information is presented to the user through a dedicated desktop application - Synchronous	3-Tier System - Tier 3: Microsoft SQL Server Database - holds events of interest and settings for each users' monitor. - Tier 2: Programs communicate with Database via stored procedures (Transaction SQL statements). - Tier 1: 2 programs work together to deliver awareness information to the user. -- Awareness Server: examines user settings & information for all monitors. -- Awareness Client: presents awareness information to user & allows user to configure
BSCW	Distributed Groups	- Provides notification on object-related events - Synchronous, Java-based monitor applet - Catch-up events: tells system users are aware of events	- New, Read, Changed, Moved, & Touched object events - Online users - Availability of users - Activities of online users	- Monitor applet: presents activity information synch - Activity icons in shared workspaces - Email	- Runs on web server using Python interpreter - Java-based Monitor applet.

CoMMITT (Espinosa 2000)	Distributed, Asynchronous Groups	- To help distributed, asynchronous groups solve problems by providing task awareness. - Used to explore documents with information about medical patients and to discuss possible diagnoses with teammates.	- Team member's activity, notes, and comments on documents. - Documents are displayed in groups. - Asynchronous	- Uses two displays to present information about documents: -- Top display: contains list of all documents with group, name and icon w/ bars to represent activity. -- Bottom display: contains a chronological listing of all the documents explored by one's team members. - Activity is represented on a 5-point scale where each attempt to view a document increases the amount of bars shown. Activity is adjusted for time on a logarithmic scale - recent activity will generate more bars.	- Implemented using Microsoft's Visual Basic and Active Server Pages. - Data were accessed from a relational database.
Groove (2003)	Person-to-person, small group collaboration	- Peer Connection - content shared online/offline. - Security - end user authentication and data encryption. - Storage - activity stored locally in the Groove XML object store. - Synchronizatio	- Online status - Document creation and modification. - Comm. - email, chat, discussion boards, instant messaging. - Co-browsing	- Presentation is integrated into the applications. - Groove Web - allows presentation of information on handhelds and non-Windows platforms.	- Integrated with Microsoft Office. - Uses Microsoft Sharepoint Team Services & Portal for publication and notification of activity. - Peer-to-peer communication handled via XML message passing.

		n - shares local copies of data through peer-to-peer connections. - Device Presence - indicates when a computer is available to receive data. - Groove Web – allows integration with handhelds and non-Window platforms.			
I2I (Budzik 2000)	Distributed Users	- Automatically clusters documents based on content. - Groups related documents into a single conceptual space. - Attempts to build communities of interest. - Automatically builds lexical representations of users' current activity - Allows users to set privacy level	- System activity - Who's online - Related documents - Active chat information - "Calling Cards" – note indicating desire to communicate	- Embeds information directly into applications, or - Information can be displayed in a separate window - Asynchronous/ Synchronous	- Integrates Microsoft Word, Internet Explorer, and Netscape using COM - Application Adapter sends information to a Central Broker - Broker responsible for persistent information, uses vector space modeling to determine similarity between documents
intraspect (2003)	Business	- Provides the architecture and infrastructure on which to build, deploy, and manage collaborative business solutions.	- Email, web pages, and other information objects	- Personal private workspace – readable only by user. - Solution-specific shared spaces – provide business context for users to work together	- Built on an open, Java based platform. - Uses HTTP, SMTP, WebDAV, XML, LDAP, J2EE. - APIs to support the addition to enterprise services

iScent (Anderson 2000)	Large Scale Software Development Teams	- Presents framework to support awareness and intersubjectivity among team members through the use of automatically collected event trails.	"I know that you know that I know." - User X can register to watch an event performed by User Y. When User Y performs an instance of that event, User X is notified of the event and User Y is notified of User X's notification - Intersubjectivity.	- iScent application presents awareness information. - Trail Viewers display event histories. - Synchronous	- One or more iScent applications are deployed on a set of user machines. - Once a machine is plugged into the network through iScent Sinks, the Sink can offer its stored events to other Sinks for replication & duplication to other team members. - The Event Notification System (Siena) handles all routing of events. Sinks are connected through Siena.
Loops	Medium-sized corporate groups	- Supports online conversations. - Social proxy: visual tool indicating online user status.	- Online users - Activity level of users. - Activity in conversation areas.	- Social Proxy: a graphical representation of conversations and activity.	- Server: TCP/IP-based; uses XML for data transfer. - Client: Macromedia Flash
Orbit-Gold (2003)	Distributed groups	- Goals: to test the Locales Framework; to be usable by the project members as a collaboration tool; and to be robust enough from eventual external release	- Objects, tools, & resources. - Creation & deletion of locales. - History of activity.	- Browser – user logs into the system through web browser. - Workspace – displays objects - Navigator – displays information about other users and locals. - Orbit client – unifies notifications with the workspace. - Asynchronous/ Synchronous	- Core System: Locale Service which manages user sessions/ details, & stores definitions of locales - Behind the Locale Service is the repositories (BSCW, Envy, CVS) - User Interface: Orbit client (Java) - Medium for awareness notification is the Elvin Notification Service

Shadow netWorkspace	Learning Communities	- To bring the benefits of advancing internet-based technology and network services to bear on the work of improving teaching, learning and schooling. - To build and deliver a common platform of integrated network applications that is readily available and usable by every teacher and student in the world.	- Last time logged in. - Discussion board posts since last login. - New group members since last login.	- Awareness monitor widget which resides on Personal and group desktops. - Information presented contextually based on groups. - Email notifications of Discussion Board posts. - Asynchronous	- Applications register as having awareness information. - New information is retrieved from registered applications when the desktop is refreshed.
Side Show (Cadiz 2001; Cadiz 2002)	Work Teams	- An awarness interface with the goal of helping people stay aware of large amounts of dynamic information without overloading or distracting them. - Supports peripheral awareness, polling of information, & alerts. - Design Principles: always present, minimal motion, personal, extensible, supports quick-drill-down and escape, & scalable.	- Information from Outlook Calendars, Email, RAID bugs, websites, other. - Team members availability. - Information from the Internet.	- Awareness information is presented in an always-present 50 pixel-wide bar comprised of tickets. - A ticket presents small summary of information. - Mouse-over tickets provide more detailed information. - Double-clicking a ticket provides all the information for that ticket. - Tickets can be grouped for more efficient use of space. - Some tickets can alert users through use of motion.	Unknown

Sticker (Phillips 2002)	Distributed Co-Workers	- Goals: to provide a complete set of ticker and presence features; a single tickertape client for a wide range of platforms without sacrificing functionality; a set of software components that can be used to build customized tickertape/ presence applications.	- Can potentially handle any awareness information - Works by listening for messages sent to a ticker groups. -- Chat groups: for conversations between people -- News groups: carry news messages generated from online news services	- Ticker windows: messages scroll across window until deleted/ expires. - Online presence: see who's online, chat with them, & find more detailed info about them - Synchronous	Uses the Elvin Notification Service
Team Portal (Handel 2000)	Geographically Distributed Work Teams	- Monitors presence information. - Presents the information to team members. - Allows team members to rapidly initiate communication. - Implementation guided by the need to provide: Real-time visualization, Compact and easy to use views, & Range of Presence Information.	- Relies primarily on physical presence information, such as location, last login, and keyboard/mouse activity. - Uses the computer's activity level as a proxy for presence. - Focuses on changes in shared documents for mutual awareness. - Public calendar information. - People can report their own status such as present, busy, meeting, other.	3 views - TimesView: displays location of each data source. - CalendarView: displays data "of interest". Right-clicking a day brings up a context-sensitive menu for more information. - PeopleView: spreadsheet-like representation of the people on the team the user is focusing on.	- Intergrates data from different sources using an extension of the relational database model - linking paradigm. - Additional attributes are associated with data sources to determine the user's "degree of interest (DofI) for each element. - Written in Java

			- Synchronous		
TeamSCOPE (Jang 2000; Pfaff 2001; Steinfield 1999)	Internationally distributed engineering design teams	- Focus on facilitation of group members' awareness of group activities, communications and resources. - Provide a shared workspace where group members can store and retrieve shared objects. - Support asynchronous group interaction through the ability to post group messages. - Provide group members with ongoing information about status of group objects, group communication, resources available, and schedules/availability. - Support group use of other external communication resources. - Work with regularly used communication tools outside the collaborative tool. - Have ubiquitous accessibility via	- Activities tracked include: all file-related activities, activity related to message boards, & calendar events. - Tracks only changes made through web and ftp interfaces. - Tracks which activities have been reported to each user. - Activity history - Synchronous: only in notification of team members logged in status.	- Uses website for each team member to display summary of activity on login. - Email summaries of activity. - Searchable activity history.	- Activity awareness comes from a system log of all activities in the team's directory - Gathers activity records in a central location & offers some flexibility for users to structure event summaries. - No real-time component to allow the server to push awareness information to user.

		the Internet from a web browser. - Easily customized for different groups.			
Ticker Tape (Fitzpatrick 1998; Fitzpatrick 1999; Parsowith 1998)	Distributed Co-Workers	- To facilitate interaction between co-workers - To facilitate social interaction and provide a sense of group cohesion - To facilitate leisure activities (reading news/sports, other information)	- Can potentially handle any type, and source of, awareness information	- Ticker window: messages scroll across window until deleted/expired. - Information fades as it ages. - Allows users to initiate bi-directional communications.	Uses the Elvin Notification Service

APPENDIX B

API Documentation for CANS Link

This appendix contains API documentation for CANS Link. The following documentation was created using Sun's javadoc tool (http://java.sun.com/j2se/javadoc/) and then modified by this researcher to fit this document.

Class CansEvent

java.lang.Object
  CansEvent

---

public class **CansEvent**
extends java.lang.Object

This Class is used by the Consumer object to store information related to recent events.

Constructor Summary
CansEvent(java.lang.String notifId, java.lang.String userId, java.lang.String authorId, java.lang.String authorName, java.lang.String environment, java.lang.String contextId, java.lang.String contextName, java.lang.String contextType, java.lang.String eventAction, java.lang.String eventObject, java.lang.String eventTool, java.lang.String eventXml, java.lang.String eventDate)

Constructor for CansEvent Class.

Method Summary
java.lang.String **getAuthorId**()
    Returns event author's id.
java.lang.String **getAuthorName**()
    Returns event author's name.
java.lang.String **getContextId**()

Returns context id for event.
java.lang.String **getContextName**()
Returns context name for event.
java.lang.String **getContextType**()
Returns context type for event.
java.lang.String **getEnvironment**()
Returns environment id for event.
java.lang.String **getEventAction**()
Returns CSCE action for event.
java.lang.String **getEventDate**()
Returns date of event.
java.lang.String **getEventObject**()
Returns CSCE object for event.
java.lang.String **getEventTool**()
Returns CSCE tool for event.
java.lang.String **getEventXml**()
Returns xml for event.
java.lang.String **getNotifId**()
Returns notification id.
java.lang.String **getUserId**()
Returns user's id.

Methods inherited from class java.lang.Object
clone, equals, finalize, getClass, hashCode, notify, notifyAll, toString, wait

---

Constructor Detail

CansEvent

public **CansEvent**(java.lang.String notifId,
java.lang.String userId,
java.lang.String authorId,
java.lang.String authorName,
java.lang.String environment,
java.lang.String contextId,
java.lang.String contextName,
java.lang.String contextType,
java.lang.String eventAction,
java.lang.String eventObject,
java.lang.String eventTool,
java.lang.String eventXml,
java.lang.String eventDate)

Constructor for CansEvent Class.

Parameters:
notifId - notification id
userId - user id
authorId - Author id for event
authorName - Author name for event
environment - Environment id for event
contextId - Context id for event
contextName - Context name for event
contextType - Context type for event
eventAction - Action of the event
eventObject - CSCE object for the event
eventTool - CSCE tool for the event
eventXml - XML representing the event
eventDate - Date the event occurred

Method Detail

getNotifId

public java.lang.String **getNotifId**()

Returns notification id.

---

getUserId

public java.lang.String **getUserId**()

Returns user's id.

---

getAuthorId

public java.lang.String **getAuthorId**()

Returns event author's id.

---

getAuthorName

public java.lang.String **getAuthorName**()

Returns event author's name.

---

getEnvironment

public java.lang.String **getEnvironment**()

Returns environment id for event.

---

getContextId

public java.lang.String **getContextId**()

Returns context id for event.

---

getContextName

public java.lang.String **getContextName**()

Returns context name for event.

---

getContextType

public java.lang.String **getContextType**()

Returns context type for event.

---

getEventAction

public java.lang.String **getEventAction**()

Returns CSCE action for event.

---

getEventObject

public java.lang.String **getEventObject**()

Returns CSCE object for event.

---

getEventTool

public java.lang.String **getEventTool**()

Returns CSCE tool for event.

---

getEventXml

public java.lang.String **getEventXml**()

Returns xml for event.

---

getEventDate

public java.lang.String **getEventDate**()

Returns date of event.

Class Client

java.lang.Object
  Client

All Implemented Interfaces:
java.lang.Runnable

---

public class **Client**
extends java.lang.Object
implements java.lang.Runnable

Manages the Thread and creates either a Producer, Consumer, or Administrator Object.

Field Summary
protected java.lang.String **action**
protected static java.util.Vector **handlers**
protected java.lang.Thread **listener**
protected java.net.Socket **socket**
protected java.util.ArrayList **users**

Constructor Summary
Client(java.net.Socket socket, java.lang.String action)
        Constructor for Client Class.
Client(java.net.Socket socket, java.lang.String action, java.util.ArrayList users)
        Constructor for Client Class.

Method Summary
 void **run**()
        run method for Thread.
 void **start**()
        start method for Thread.
 void **stop**()
        stop method for Thread.

Methods inherited from class java.lang.Object
clone, equals, finalize, getClass, hashCode, notify, notifyAll, toString, wait

Field Detail

socket
protected java.net.Socket **socket**

---

listener
protected java.lang.Thread **listener**

---

handlers
protected static java.util.Vector **handlers**

---

action
protected java.lang.String **action**

---

users
protected java.util.ArrayList **users**

Constructor Detail

Client

public **Client**(java.net.Socket socket,
java.lang.String action)

Constructor for Client Class.

Parameters:
socket - socket object
action - indicator for connections role or action: producer, consumer or administrator

---

Client

public **Client**(java.net.Socket socket,
java.lang.String action,
java.util.ArrayList users)

Constructor for Client Class.

Parameters:
socket - socket object
action - indicator for connections role or action: producer, consumer or administrator
users - array list of users in the system

Method Detail

start

public void **start**()

start method for Thread.

---

stop

public void **stop**()

stop method for Thread.

---

run

public void **run**()

run method for Thread. Creates Producer, Consumer, or Administrator Object depending on the Class variable: action

Specified by:
run in interface java.lang.Runnable

Class ConnMgr

java.lang.Object
  ConnMgr

---

public class **ConnMgr**
extends java.lang.Object

Master connection manager for NADS sockets.

Constructor Summary
ConnMgr(java.net.ServerSocket server, java.lang.String action)
        Constructor for ConnMgr Class.
ConnMgr(java.net.ServerSocket server, java.lang.String action,
java.util.ArrayList users)
        Constructor for ConnMgr Class.

Methods inherited from class java.lang.Object
clone, equals, finalize, getClass, hashCode, notify, notifyAll, toString, wait

---

Constructor Detail

ConnMgr

public **ConnMgr**(java.net.ServerSocket server,
            java.lang.String action)

Constructor for ConnMgr Class.

Parameters:
server - server socket object
action - indicator for connections role or action: producer, consumer or administrator

---

ConnMgr

public **ConnMgr**(java.net.ServerSocket server,
java.lang.String action,
java.util.ArrayList users)

Constructor for ConnMgr Class.

Parameters:
server - server socket object
action - indicator for connections role or action: producer, consumer or administrator
users - array list of users in system

Class Consumer

java.lang.Object
  Consumer

All Implemented Interfaces:
EventListener

---

public class **Consumer**
extends java.lang.Object
implements EventListener

Consumer Class that receives event notifications from Siena

Constructor Summary
Consumer(java.net.Socket socket, java.util.ArrayList users)
        Constructor for Consumer Class.

Method Summary
 boolean **authorize**()
        Authorizes current user and returns boolean result.
 void **eventReceived**()
        Called when a new event notification is received for the user.
 void **prepareForPastEvents**()
        Determines the events that have occurred since the user's last notification.
 void **printMessage**(java.lang.String message)
        Prints message to socket.
 void **startConsuming**()
        Initiates the notification listening process for a user.

Methods inherited from class java.lang.Object
clone, equals, finalize, getClass, hashCode, notify, notifyAll, toString, wait

---

Constructor Detail

Consumer

public **Consumer**(java.net.Socket socket,
java.util.ArrayList users)

Constructor for Consumer Class.

Parameters:
socket - socket object
users - array list of all users in system

Method Detail

startConsuming

public void **startConsuming**()
throws java.io.IOException

Initiates the notification listening process for a user. Listens to Consumer socket and authorizes user on this socket. After user is authorized, user notifications subscriptions are invoked and the user is set to receive new notifications.

Throws:
java.io.IOException

---

eventReceived

public void **eventReceived**()

Called when a new event notification is received for the user.

Specified by:
eventReceived in interface EventListener

---

prepareForPastEvents

public void **prepareForPastEvents**()

Determines the events that have occurred since the user's last notification.

---

authorize

public boolean **authorize**()

Authorizes current user and returns boolean result.

Returns: boolean result for user authorization

---

printMessage

public void **printMessage**(java.lang.String message)

Prints message to socket.

Parameters:
message - text message

Class DBMS

java.lang.Object
  DBMS

---

public class **DBMS**
extends java.lang.Object

Manages connection and interaction with MySQL database.

Constructor Summary
DBMS()
        Constructor for DBMS Object.

Method Summary
 void **closeDB**()
        Closes connection to database.
 boolean **doInsert**(java.lang.String sql)
        Inserts data into database.
 java.sql.ResultSet **doSQL**(java.lang.String sql)
        Executes SQL statement.
 boolean **doUpdate**(java.lang.String sql)
        Executes SQL Update.
 void **finishSQL**()
        Finishes SQL statement.
 boolean **initDB**()
        Initializes connection to database.

Methods inherited from class java.lang.Object
clone, equals, finalize, getClass, hashCode, notify, notifyAll, toString, wait

---

Constructor Detail

DBMS

public **DBMS**()

Constructor for DBMS Object.

Method Detail

initDB

public boolean **initDB**()

Initializes connection to database.

---

doSQL

public java.sql.ResultSet **doSQL**(java.lang.String sql)

Executes SQL statement.

Parameters:
sql - String representing SQL statement to be executed
Returns:
result set from SQL statement

---

doInsert

public boolean **doInsert**(java.lang.String sql)

Inserts data into database.

Parameters:
sql - String representing SQL statement to be executed
Returns:
result from SQL statement: true or false

---
doUpdate

public boolean **doUpdate**(java.lang.String sql)

Executes SQL Update.

Parameters:
sql - String representing SQL statement to be executed
Returns:
result from SQL statement: true or false

---

finishSQL

public void **finishSQL**()

Finishes SQL statement.

---

closeDB

public void **closeDB**()

Closes connection to database.

Class EventHistory

java.lang.Object
  EventHistory

---

public class **EventHistory**
extends java.lang.Object

This Class is used to store events that occurred since the user's last notification.

Constructor Summary
EventHistory(java.lang.String userId, java.lang.String authorId,
java.lang.String authorName, java.lang.String environment,
java.lang.String contextId, java.lang.String contextName,
java.lang.String contextType, java.lang.String eventAction,
java.lang.String eventObject, java.lang.String eventTool,
java.lang.String eventXml, java.lang.String eventDate)

Constructor for EventHistory Class.

Method Summary
java.lang.String **getAuthorId**()
Returns event author's id.
java.lang.String **getAuthorName**()
Returns event author's name.
java.lang.String **getContextId**()
Returns context id for event.
java.lang.String **getContextName**()
Returns context name for event.
java.lang.String **getContextType**()
Returns context type for event.
java.lang.String **getEnvironment**()
Returns environment id for event.
java.lang.String **getEventAction**()
Returns CSCE action for event.
java.lang.String **getEventDate**()
Returns date of event.
java.lang.String **getEventObject**()

Returns CSCE object for event.
java.lang.String **getEventTool**()
Returns CSCE tool for event.
java.lang.String **getEventXml**()
Returns xml for event.
java.lang.String **getUserId**()
Returns user's id.

Methods inherited from class java.lang.Object
clone, equals, finalize, getClass, hashCode, notify, notifyAll, toString, wait

---

Constructor Detail

EventHistory

public **EventHistory**(java.lang.String userId,
java.lang.String authorId,
java.lang.String authorName,
java.lang.String environment,
java.lang.String contextId,
java.lang.String contextName,
java.lang.String contextType,
java.lang.String eventAction,
java.lang.String eventObject,
java.lang.String eventTool,
java.lang.String eventXml,
java.lang.String eventDate)

Constructor for EventHistory Class.

Parameters:
userId - user id
authorId - Author id for event
authorName - Author name for event
environment - Environment id for event
contextId - Context id for event
contextName - Context name for event
contextType - Context type for event
eventAction - Action of the event
eventObject - CSCE object for the event
eventTool - CSCE tool for the event
eventXml - XML representing the event

eventDate - Date the event occurred

Method Detail

getUserId

public java.lang.String **getUserId**()

Returns user's id.

---

getAuthorId

public java.lang.String **getAuthorId**()

Returns event author's id.

---

getAuthorName

public java.lang.String **getAuthorName**()

Returns event author's name.

---

getEnvironment

public java.lang.String **getEnvironment**()

Returns environment id for event.

---

getContextId

public java.lang.String **getContextId**()

Returns context id for event.

---

getContextName

public java.lang.String **getContextName**()

Returns context name for event.

---

getContextType

public java.lang.String **getContextType**()

Returns context type for event.

---

getEventAction

public java.lang.String **getEventAction**()

Returns CSCE action for event.

---

getEventObject

public java.lang.String **getEventObject**()

Returns CSCE object for event.

---

getEventTool

public java.lang.String **getEventTool**()

Returns CSCE tool for event.

---

getEventXml

public java.lang.String **getEventXml**()

Returns xml for event.

---

getEventDate

public java.lang.String **getEventDate**()

Returns date of event.

Class Producer

java.lang.Object
  Producer

---

public class **Producer**
extends java.lang.Object

Producer Object that publishes new events to Siena

Field Summary
protected java.net.Socket **socket**

Constructor Summary
Producer(java.net.Socket socket)
        Constructor for Producer Class.

Method Summary
 java.lang.String **getRootTagName**()
        Returns the root tag name in the XML Document.
 java.lang.String **getSienaConnectInfo**()
        Generates and returns connection information for Siena.
 void **publishEvents**()
        Publishes XML formatted event to Siena.

Methods inherited from class java.lang.Object
clone, equals, finalize, getClass, hashCode, notify, notifyAll, toString, wait

---

Field Detail

socket

protected java.net.Socket **socket**

Constructor Detail

Producer

public **Producer**(java.net.Socket socket)

Constructor for Producer Class.

Parameters:
socket - socket object

Method Detail

publishEvents

public void **publishEvents**()

Publishes XML formatted event to Siena.

---

getRootTagName

public java.lang.String **getRootTagName**()

Returns the root tag name in the XML Document.

---

getSienaConnectInfo

public java.lang.String **getSienaConnectInfo**()

Generates and returns connection information for Siena.

Returns string to connect to Siena in the format of reciever:host:port

Class SienaConn

java.lang.Object
  SienaConn

---

public class **SienaConn**
extends java.lang.Object

This Class is used to provide Siena connection information.

Constructor Summary
SienaConn()
        Constructor for SienaConn Class.

Method Summary
static Siena **getSienaConn**()
        Returns the Siena connection information

Methods inherited from class java.lang.Object
clone, equals, finalize, getClass, hashCode, notify, notifyAll, toString, wait, wait, wait

---

Constructor Detail

SienaConn

public **SienaConn**()

Constructor for SienaConn Class.

Method Detail

getSienaConn

public static Siena **getSienaConn**()

Returns the Siena connection information

Class StartServer

java.lang.Object
  StartServer

---

public class **StartServer**
extends java.lang.Object

This Class starts the CAN Server.

Constructor Summary
StartServer()

Method Summary
static void **main**(java.lang.String[] args)
        Main method for the StartServer Class.

Methods inherited from class java.lang.Object
clone, equals, finalize, getClass, hashCode, notify, notifyAll, toString, wait, wait, wait

---

Constructor Detail

StartServer

public **StartServer**()

Method Detail

main

public static void **main**(java.lang.String[] args)
            throws java.io.IOException

Main method for the StartServer Class. StartServer is run from the command line. This method establishes a connection to Siena and the MySQL database. It then builds an ArrayList of all the users in the system and initializes each user. Finally, this method activates port 5440, 5441, and 5442.

Parameters:
args - command line arguments
Throws:
java.io.IOException
See Also:
siena.HierarchicalDispatcher

Class Subscription

java.lang.Object
  Subscription

---

public class **Subscription**
extends java.lang.Object

This Class is used to create Siena subscriptions for each User.

Constructor Summary
Subscription(User user, Siena siena, java.lang.String env, java.lang.String evt, java.lang.String contId, java.lang.String contLevel, java.lang.String contNotes, java.lang.String contName)

Constructor for Subscription Class.

Method Summary
 void **createEventFilter**()
        Creates a Siena Filter object that represents the user's subscription.
 Filter **getFilter**()
        Returns the Siena Filter object.

Methods inherited from class java.lang.Object
clone, equals, finalize, getClass, hashCode, notify, notifyAll, toString, wait, wait, wait

---

Constructor Detail

Subscription

public **Subscription**(User user,
                Siena siena,
                java.lang.String env,

java.lang.String evt,
java.lang.String contId,
java.lang.String contLevel,
java.lang.String contNotes,
java.lang.String contName)

Constructor for Subscription Class.

Parameters:
user - User object
siena - Siena connection object for user
env - String representing the CSCE, ex. 'sakai'
evt - The event the subscription will monitor
contId - The context id for the subscription
contLevel - The priority level for the subscription
contNotes - The context id for subscriptions on specific users, groups, or objects
contName - The context name for the subscription

Method Detail

createEventFilter

public void **createEventFilter**()

Creates a Siena Filter object that represents the user's subscription.

---

getFilter

public Filter **getFilter**()

Returns the Siena Filter object.

Class User

java.lang.Object
  User

---

public class **User**
extends java.lang.Object

User Class for each user in the notification system.

Constructor Summary
User(java.lang.String id, Siena siena)
        Constructor for User Class.

Method Summary
 void **addEventListener**(EventListener l)
        Adds event listener to the List listeners.
 void **fireNotifyEvent**()
        Performs action when an event occurs.
 java.lang.String **getCurrentContext**()
        Returns user's current context (location).
 java.lang.String **getCurrentEnvironment**()
        Returns user's current environment.
 DBMS **getDBMSConn**()
        Returns database connection object for user.
 java.lang.String **getFirstName**()
        Returns user's first name.
 java.lang.String **getLastName**()
        Returns user's last name.
 java.lang.String **getLoginID**()
        Returns user's login id.
 java.lang.String **getNotificationFormat**()
        Returns user's desired notification format.
 Siena **getSiena**()
        Returns user's Siena Object.
 java.lang.String **getUserID**()
        Returns user id.
 java.lang.String **getUserName**()

Returns user's full name.
void **init**()
Initializes the User Object.
void **invokeSubscriptions**()
Determines preferences and creates subscription for User.
void **notify**(Notification e)
Automatically called when a new event is received.
void **prepare4Events**()
Prepares User object for event notifications by establishing connection to database.
void **removeEventListener**(EventListener l)
Removes event listener from the List listeners.
void **resumeReceiving**()
Resumes notification receiving status for user.
void **setCurrentEnvironment**(java.lang.String e)
Sets current environment for user.
void **setNotificationFormat**(java.lang.String f)
Sets notification format for User: Sakai or RSS.
void **subscribe4Events**()
Subscribes user to receive events based on subscription.
void **suspendReceiving**()
Suspends notification receiving for user.

Methods inherited from class java.lang.Object
clone, equals, finalize, getClass, hashCode, notify, notifyAll, toString, wait

---

Constructor Detail

User

public **User**(java.lang.String id,
Siena siena)

Constructor for User Class.

Parameters:
id - user id
siena - Siena connection object for user

Method Detail

addEventListener

public void **addEventListener**(EventListener l)

Adds event listener to the List listeners.

Parameters:
l - event listener object

---

removeEventListener

public void **removeEventListener**(EventListener l)

Removes event listener from the List listeners.

Parameters:
l - event listener object

---

fireNotifyEvent

public void **fireNotifyEvent**()

Performs action when an event occurs.

---

init

public void **init**()

Initializes the User Object.

---

invokeSubscriptions

public void **invokeSubscriptions**()

Determines preferences and creates subscription for User.

---

subscribe4Events

public void **subscribe4Events**()

Subscribes user to receive events based on subscription.

---

resumeReceiving

public void **resumeReceiving**()

Resumes notification receiving status for user.

---

suspendReceiving

public void **suspendReceiving**()

Suspends notification receiving for user.

---

notify

public void **notify**(Notification e)

Automatically called when a new event is received. Inserts new event in database for later analyses and inserts notification into notification table. Then checks to see if user should receive notification and fires notify event method.

Parameters:
e - event notification object

---

prepare4Events

public void **prepare4Events**()

Prepares User object for event notifications by establishing connection to database.

---

setCurrentEnvironment

public void **setCurrentEnvironment**(java.lang.String e)

Sets current environment for user.

---

setNotificationFormat

public void **setNotificationFormat**(java.lang.String f)

Sets notification format for User: Sakai or RSS.

---

getUserID

public java.lang.String **getUserID**()

---

getLoginID

public java.lang.String **getLoginID**()

Returns user's login id.

---

getFirstName

public java.lang.String **getFirstName**()

Returns user's first name.

---

getLastName

public java.lang.String **getLastName**()

Returns user's last name.

---

getUserName

public java.lang.String **getUserName**()

Returns user's full name.

---

getSiena

public Siena **getSiena**()

Returns user's Siena Object.

---

getDBMSConn

public DBMS **getDBMSConn**()

Returns database connection object for user.

---

getCurrentContext

public java.lang.String **getCurrentContext**()

Returns user's current context (location).

---

getCurrentEnvironment

public java.lang.String **getCurrentEnvironment**()

Returns user's current environment.

---

getNotificationFormat

public java.lang.String **getNotificationFormat**()

Returns user's desired notification format.

APPENDIX C

Producer XML DTD

The Document Type Definition for the XML sent by event producers.

```
<!DOCTYPE environment [

<!ELEMENT environment ( context+ ) >
<!ATTLIST environment id CDATA #REQUIRED >

<!ELEMENT context ( event+ ) >
<!ATTLIST context id CDATA #REQUIRED >
<!ATTLIST context name CDATA #REQUIRED >
<!ATTLIST context type CDATA #REQUIRED >

<!ELEMENT event EMPTY >
<!ATTLIST event action CDATA #REQUIRED >
<!ATTLIST event author_id CDATA #REQUIRED >
<!ATTLIST event author_name CDATA #REQUIRED >
<!ATTLIST event date CDATA #REQUIRED >
<!ATTLIST event event_object CDATA #REQUIRED >
<!ATTLIST event event_tool CDATA #REQUIRED >

]>
```

APPENDIX D

Consumer XML DTD

The Document Type Definition for the XML sent by event consumers.

```
<!DOCTYPE notification [

<!ELEMENT notification ( environment ) >
<!ATTLIST notification date CDATA #REQUIRED >

<!ELEMENT environment ( context+ ) >
<!ATTLIST environment id CDATA #REQUIRED >
<!ATTLIST environment userid CDATA #REQUIRED >
<!ATTLIST environment username CDATA #REQUIRED >
<!ATTLIST environment usercount CDATA #REQUIRED >

<!ELEMENT context ( present, event ) >
<!ATTLIST context id CDATA #REQUIRED >
<!ATTLIST context type CDATA #REQUIRED >
<!ATTLIST context priority CDATA #REQUIRED >

<!ELEMENT event ( description ) >
<!ATTLIST event action CDATA #REQUIRED >
<!ATTLIST event author_id CDATA #REQUIRED >
<!ATTLIST event author_name CDATA #REQUIRED >
<!ATTLIST event date CDATA #REQUIRED >
<!ATTLIST event object CDATA #REQUIRED >
<!ATTLIST event tool CDATA #REQUIRED >

<!ELEMENT present ( user ) >
<!ATTLIST present since CDATA #REQUIRED >

<!ELEMENT user EMPTY >
<!ATTLIST user author_id CDATA #REQUIRED >

<!ELEMENT description ( #PCDATA ) >

]>
```

APPENDIX E

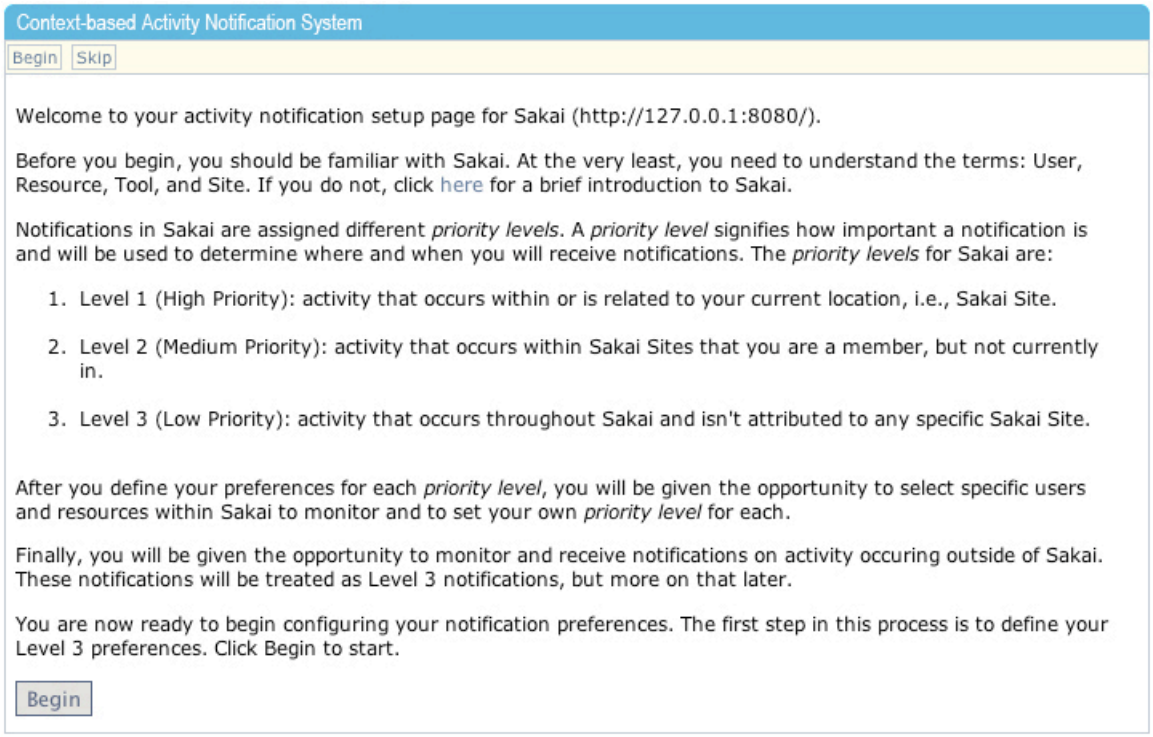
Administrator XML DTD

The Document Type Definition for the XML sent by notification administrator applications.

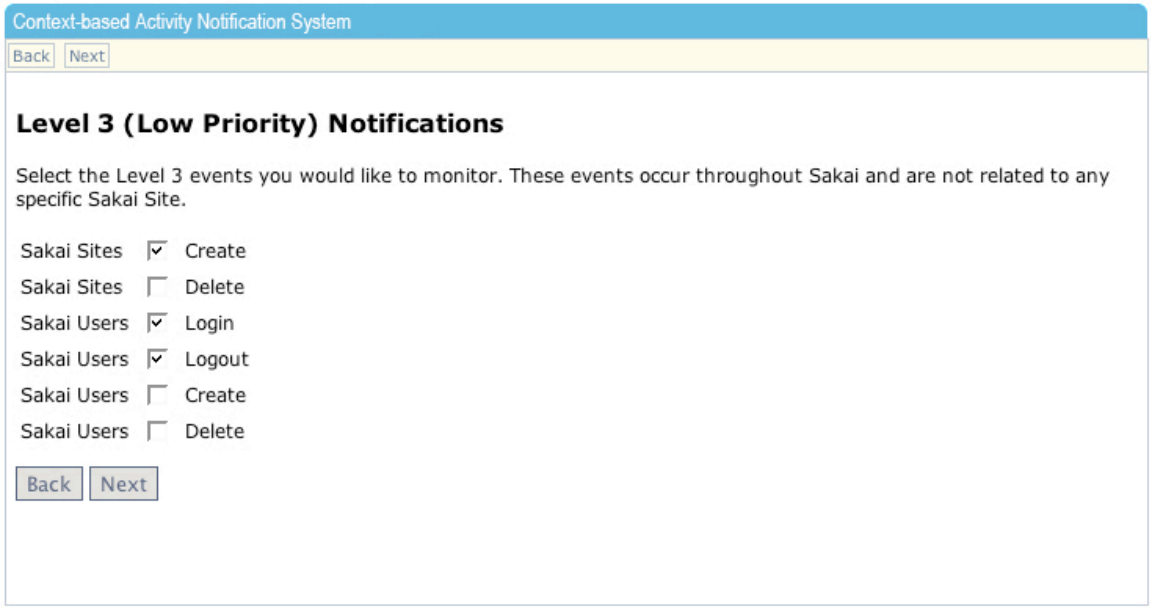
```
<!DOCTYPE environment [

<!ELEMENT environment ( user ) >
<!ATTLIST environment date CDATA #REQUIRED >
<!ATTLIST environment id CDATA #REQUIRED >

<!ELEMENT user ( event+ ) >
<!ATTLIST user id CDATA #REQUIRED >
<!ATTLIST user request CDATA #REQUIRED >

<!ELEMENT event EMPTY >
<!ATTLIST event context_id CDATA #REQUIRED >
<!ATTLIST event id CDATA #REQUIRED >
<!ATTLIST event priority CDATA #REQUIRED >
<!ATTLIST event status CDATA #REQUIRED >

]>
```

APPENDIX F

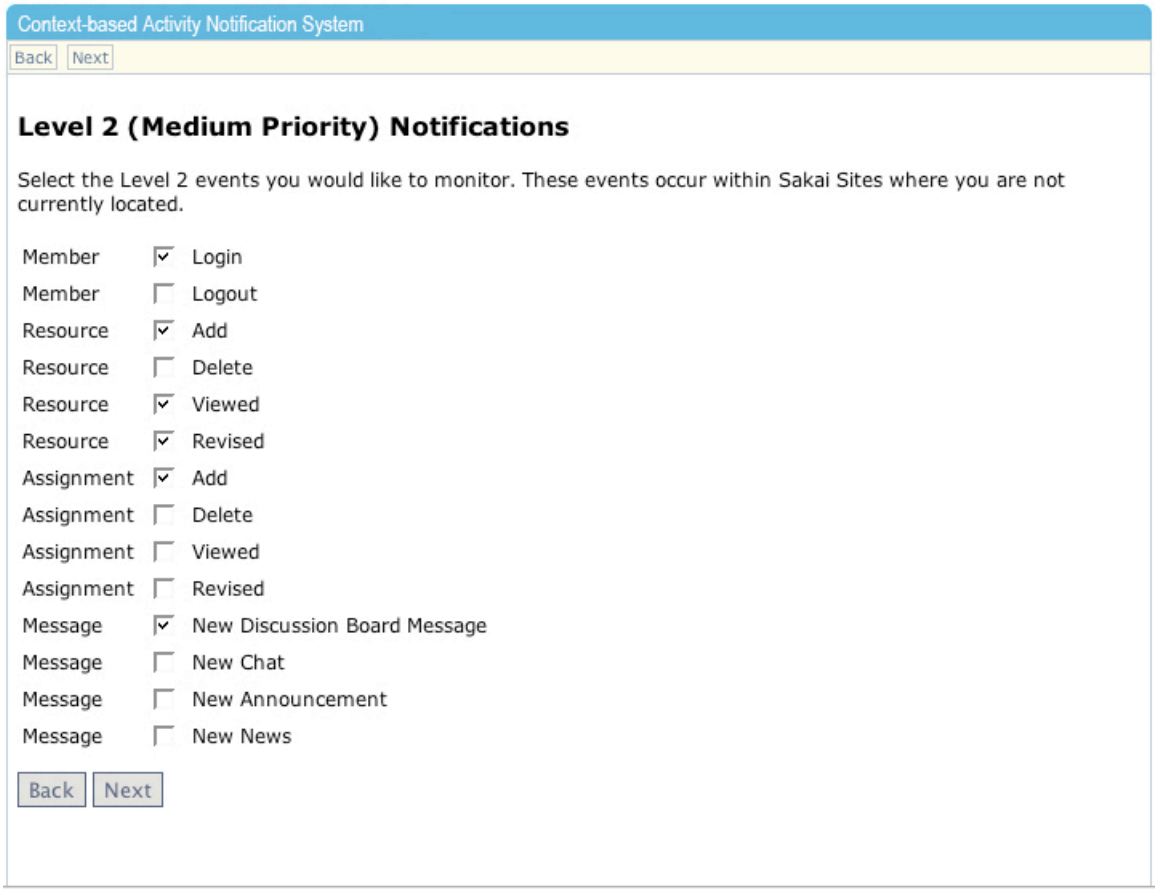
CANS Link: Administrator Screenshots



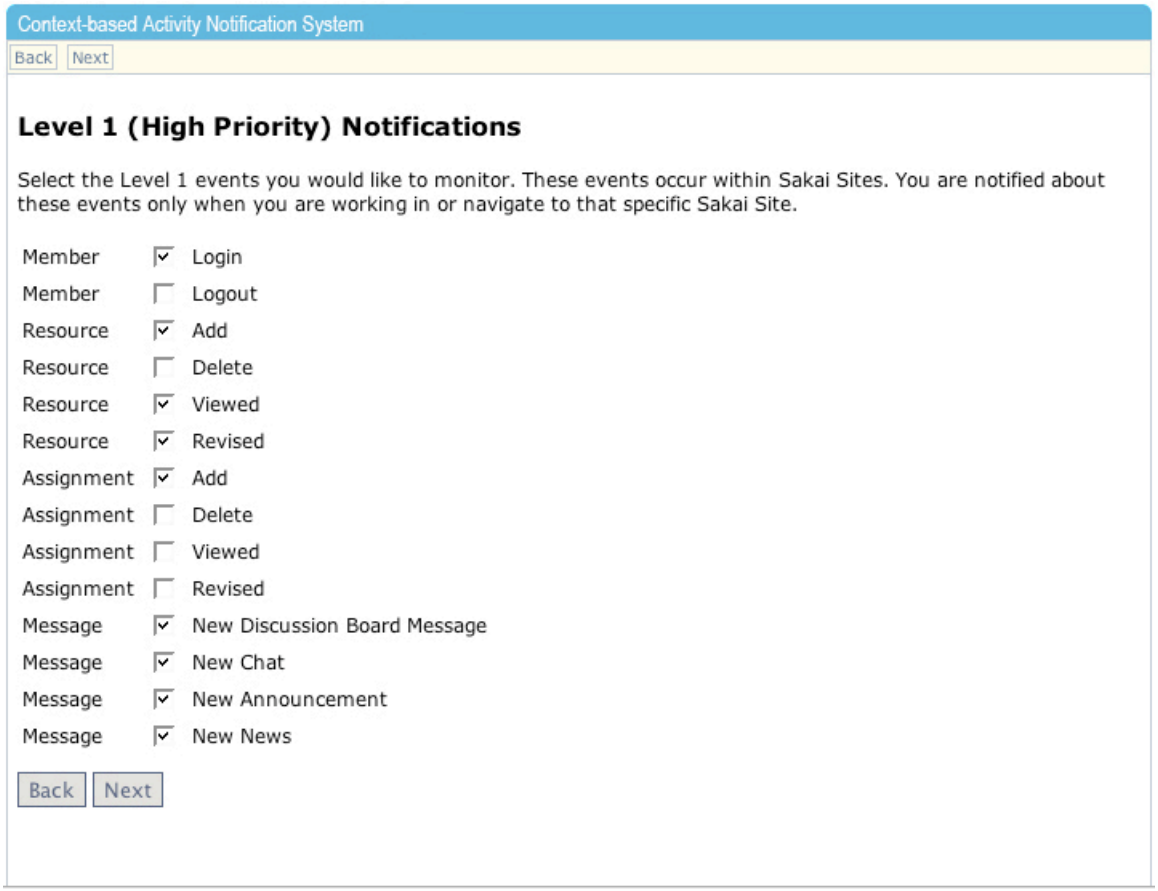
The first screen presented to users setting their notification preferences. This screen instructs users on the different priority levels for notifications and contains information about how to start setting preferences.



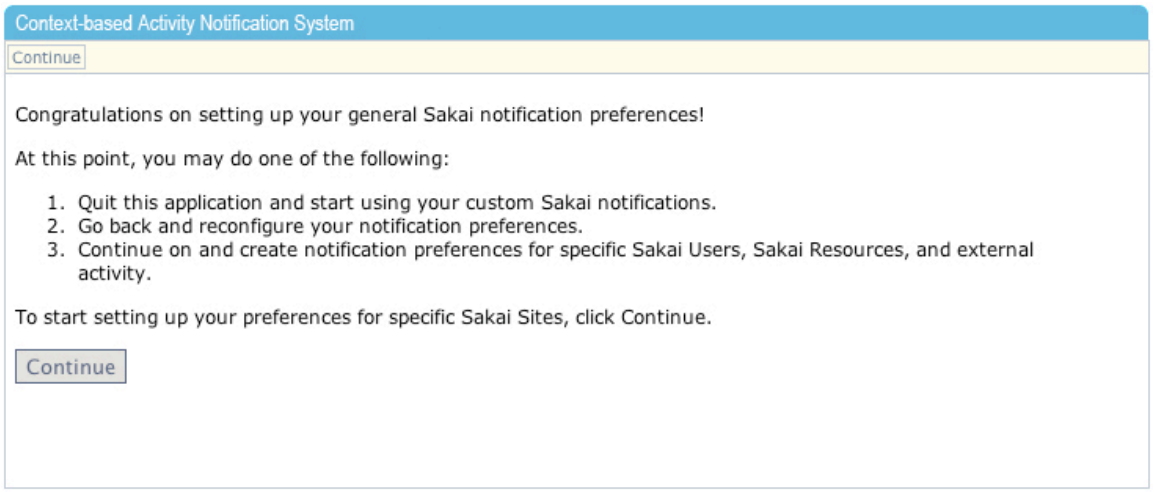
This screen is used to set priority 3 notifications for actions occurring throughout Sakai, i.e., activity not belonging to a Sakai Site.



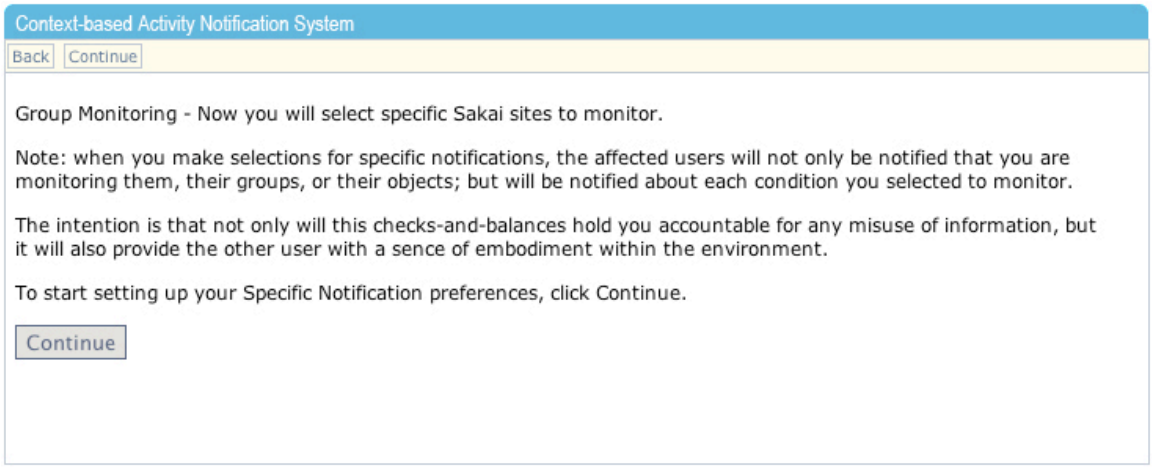
This screen is used to set priority 2 notifications for actions occurring in Sites that the user is a member of, but not currently located.



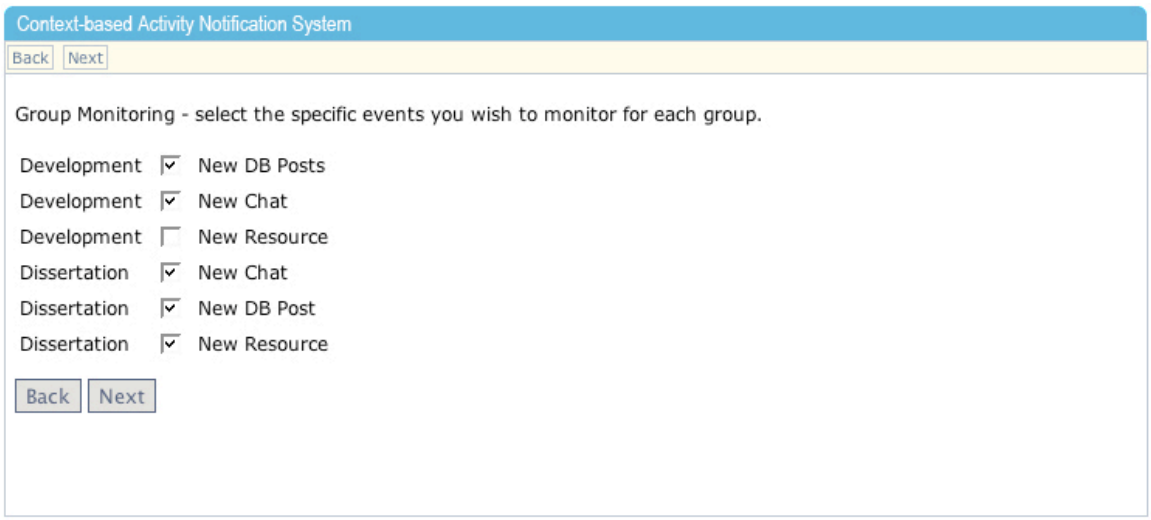
This screen is used to set priority 1 notifications for actions occurring in the same Site where the user is located.



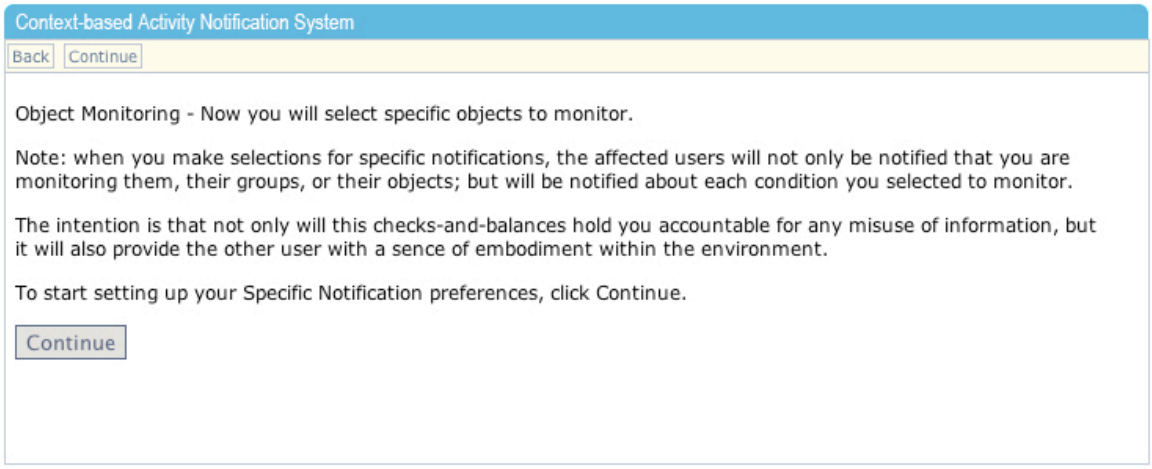
This screen is presented to users after they complete the general notification preferences shown above. At this point the user may quit or continue to set notification preferences on specific groups, objects, and users.



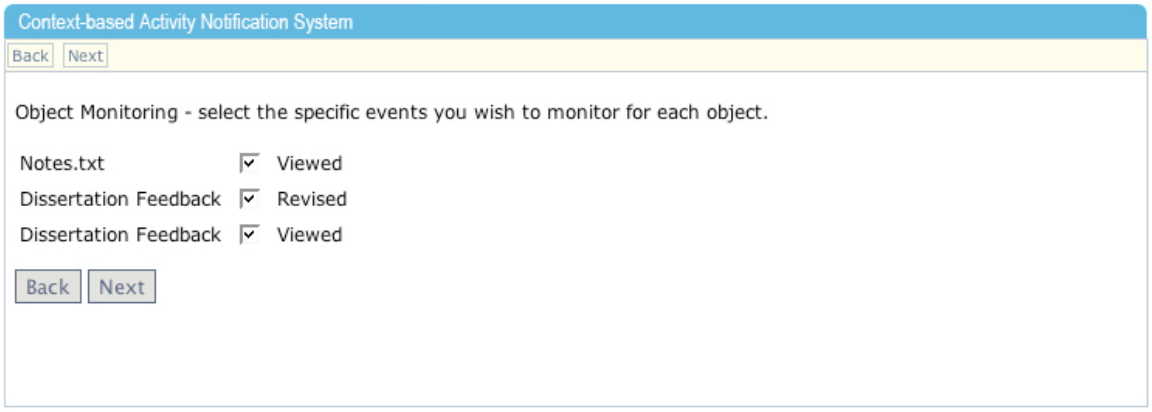
This screen provides information about the purpose of group notification preferences.



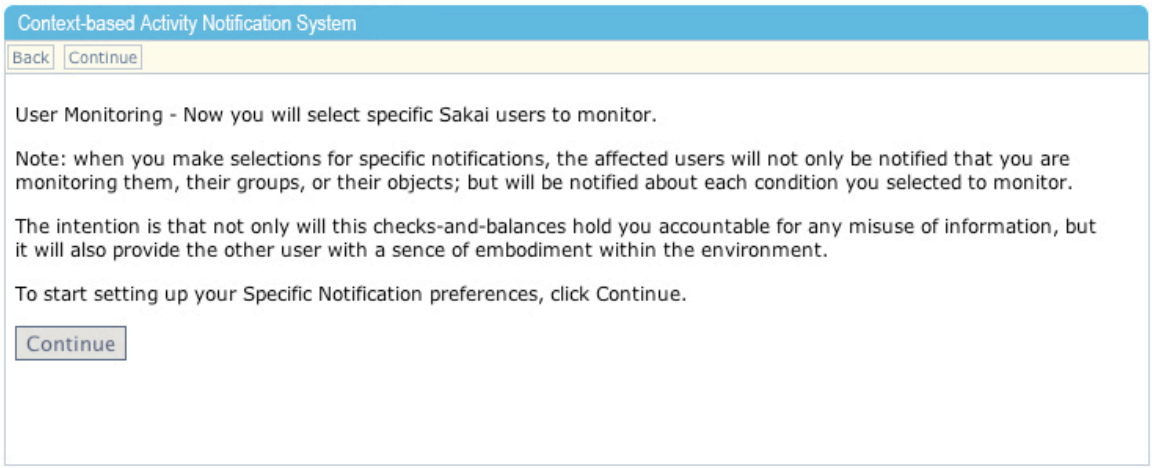
This screen is used to select events for specific Sites, i.e., groups, in Sakai.



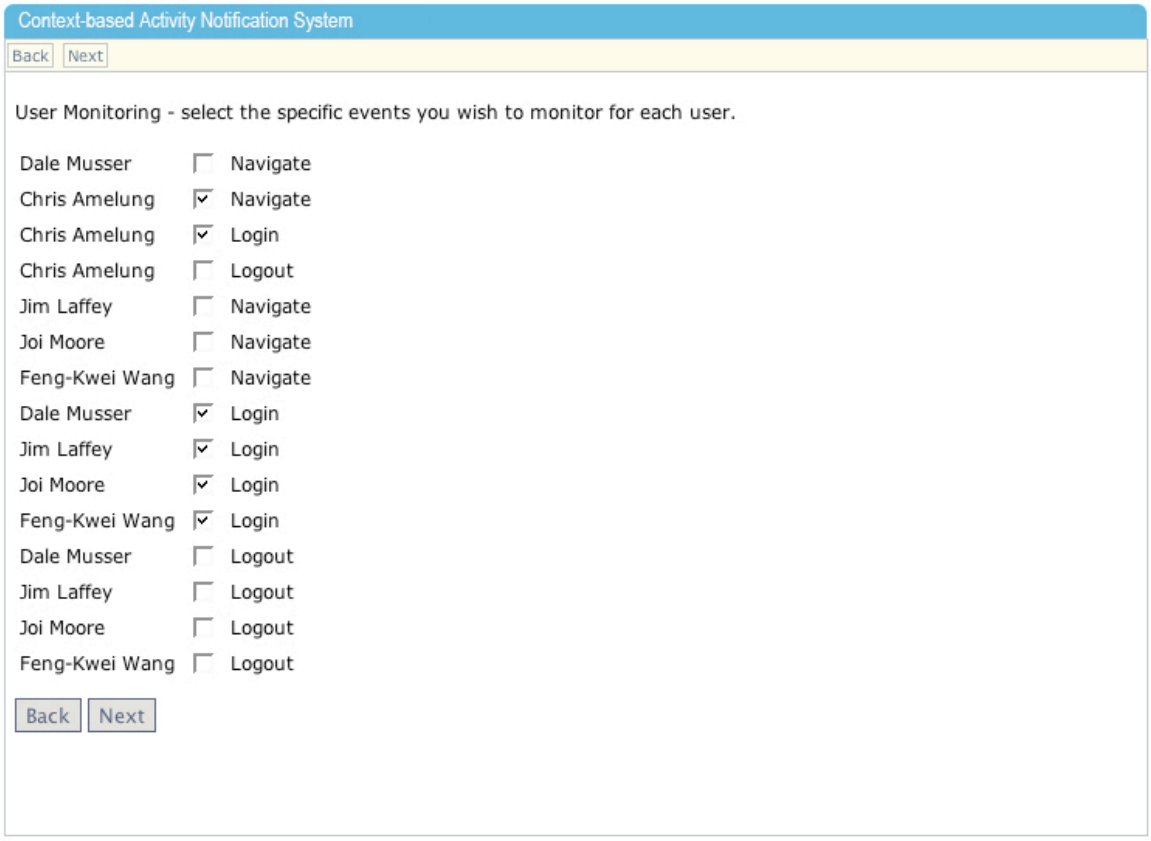
This screen provides information about the purpose of object notification preferences.



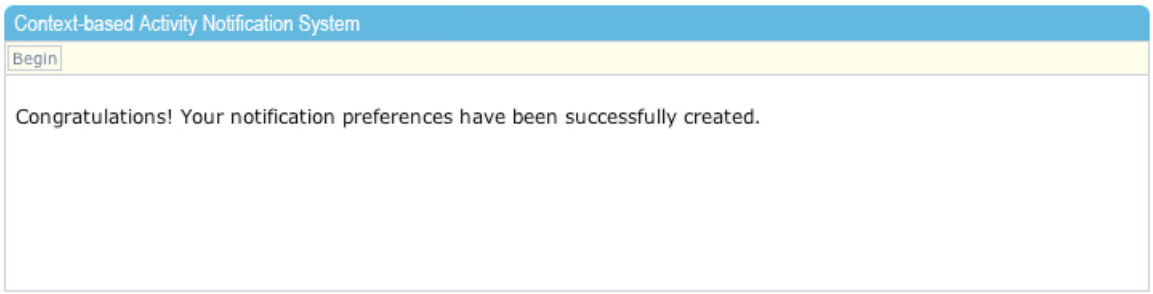
This screen is used to select events for specific Resources, i.e., objects, in Sakai.



This screen provides information about the purpose of user notification preferences.



This screen is used to select events for specific users in Sakai.



The final notification preference screen

VITAE

Christopher J. Amelung is an Assistant Professor at the University of Missouri – Columbia. His prior work experiences include Instructional Manager for an innovative support environment for students enrolled in online digital media and web development courses known as the Digital Media ZONE (http://zone.missouri.edu) and lead developer of an online collaborative work environment called Shadow netWorkspace™. He received both his Bachelor's degree in Interdisciplinary Studies in 1997 and his Masters of Educational Technology degree in 2000 from the University of Missouri – Columbia.