Enabling Project Awareness and Intersubjectivity via Hypermedia-Enabled Event Trails; CU-CS-911-00

Kenneth Anderson
University of Colorado Boulder

Niels Olof Bouvia
Aarhus University

Follow this and additional works at: http://scholar.colorado.edu/csci_techreports

Recommended Citation
http://scholar.colorado.edu/csci_techreports/856

This Technical Report is brought to you for free and open access by Computer Science at CU Scholar. It has been accepted for inclusion in Computer Science Technical Reports by an authorized administrator of CU Scholar. For more information, please contact cuscholaradmin@colorado.edu.
Enabling Project Awareness and Intersubjectivity via Hypermedia-Enabled Event Trails

Kenneth M. Anderson
Department of Computer Science, University of Colorado, Boulder
ECOT 717, Campus Box 430, Boulder CO 80309-0430, USA
E-mail: kena@cs.colorado.edu

Niels Olof Bouvin
Department of Computer Science, Aarhus University
Aabogade 34, DK8200 Århus N, Denmark
E-mail: n.o.bouvin@daimi.aau.dk

ABSTRACT
Supporting project awareness in the context of large-scale software development is difficult. One key problem is identifying appropriate abstractions and techniques for the insertion of project awareness mechanisms into a software development environment with minimal impact. An additional problem is scaling project awareness mechanisms to handle the demands of large-scale software development projects. We present a framework to support awareness and intersubjectivity among software team members through the use of automatically collected, hypermedia-enabled event trails. Event notification and open hypermedia concepts, techniques, and tools are used to support the framework in addressing the two problems identified above. A distinction of the framework is the presence of mechanisms that explicitly support intersubjectivity among team members, enabling a high degree of quality to the project awareness in the team.

Keywords
intersubjectivity, event trails, project awareness, hypermedia, large-scale software engineering

1 Introduction
Software engineers in modern software development projects are challenged by overwhelming information management tasks. These tasks include, but are not limited to: requirements traceability, consistency management in the face of change, and maintaining project awareness between team members who may be distributed across both time and space. This paper reports on work designed to address the third task—project awareness—especially within the context of large-scale software development. Project awareness pertains keeping individual team members informed of the overall project progress, and aware of teammates’ actions. As Dourish and Bellotti state “... awareness is an understanding of the activities of others, which provides a context for your own activity [13].” Awareness thus touches on issues of sharing information between team members such that they can effectively coordinate the activities of their group.

To gain insight into the scope of information management tasks faced by large software development projects, consider recent work in supporting requirements traceability tasks at a major aerospace corporation [2]. In this instance, open hypermedia techniques [27] were applied to just two subsystems of an avionics software package. These two subsystems consisted of approximately 34,000 pages of documentation used to document various aspects of their hundreds of thousands of lines of code. For the requirements traceability tasks, the aerospace engineers were interested in just six different types of relationships, but on a system of this size, over 500,000 instances of these relationships needed to be created and managed by the open hypermedia system!

More relevant to the domain of this paper, consider the number of personnel that can be assigned to large-scale software development projects. In The Mythical Man-Month, Fred Brooks states that, at one point, over one thousand employees were assigned to his project to develop the OS/360 operating system [9]. Modern projects addressing more complex problem domains will, of course, have similar or greater staffing levels but with different characteristics in terms of distribution. All one thousand members of Brooks’ team were co-located at the same facility, whereas modern projects are much more likely to be distributed across several physical sites of the same organization or across multiple organizations (as is typical in the aerospace domain).

We therefore are concerned with two issues with respect to project management. First, how can project awareness mechanisms be inserted into the development environment of large software organizations? What infrastructure is required and what techniques (and their associated tools) are enabled by the infrastructure? Second, how can project awareness mechanisms be scaled such that they can provide utility in large-scale software development contexts?

With respect to the first issue, we develop a framework that makes use of techniques from the fields of open hypermedia and event-based messaging systems to achieve intersubjectivity between team members. We call this framework iScent (intersubjective collaborative event environment). Intersubjectivity can best be defined by the phrase “I know that you
know that I know.” It is difficult to communicate without intersubjectivity, since humans typically need to know that they are being understood before proceeding with a conversation. In our framework, we view the actions performed by software engineers as elements of a conversation. We provide concepts and mechanisms to enable intersubjectivity to be achieved in that conversation, thus supporting awareness. With respect to the second issue, we employ a variety of techniques and strategies to construct a scalable implementation of our proposed framework.

The rest of this paper is organized as follows. In the next section, we discuss related work. We then describe our unique approach to the problem of project awareness in Section 3. Section 4 describes our initial attempt to implement our framework. Section 5 describes our future research plans. We then conclude the paper with a summary of our contributions in Section 6.

2 Related Work
In this section, we cover related work from several disciplines including open hypermedia, event notification systems, process-centered environments, awareness support, and document management systems.

Open Hypermedia
In open hypermedia systems, issues of collaboration have been addressed along a variety of dimensions including the collaborative creation of hypermedia structures [18], concurrency control in collaborative hypermedia systems [34], hypermedia services in shared workspaces [35], and the support for collaboration in Web augmentation systems [7, 8]. Typically, support for awareness is derived from event notification facilities contained in collaborative hypermedia systems. These facilities notify all clients of the actions of the hypermedia system’s users. Thus, two users in a collaborative session will be notified whenever either user creates a link, deletes an anchor, etc. The Arakne environment, a system for Web augmentation, [7] provides support over and above simple event notification for its users by adding the notion of coupling modes [12] within a collaborative session. While these services in collaborative hypermedia systems help to provide awareness with respect to the hypermedia-related actions of members of a software team, they provide little insight into other aspects of the team’s work. In contrast, the iScent framework is designed to support the capture of multiple types of events covering a wide range of work activities. We are currently in the process of integrating our hypermedia systems, Arakne [7] and Chimera [3] into the iScent framework such that we can capture the hypermedia-related activities of software teams within iScent trails.

Event Notification Systems
The iScent framework builds on top of the services provided by event notification systems (See Section 3 for details). In fact, as will be explained in Section 3, the iScent framework is independent of any particular event notification system, such that an implementation can make use of any service that meets the requirements specified in Section 3. Event notification systems were first employed to support tool integration in software development environments. One of the first systems to employ this approach in a local-area network setting was Field [29]. Tool integration via events was also a part of Hewlett Packard’s SoftBench environment [10]. In recent years, event notification systems have been extended to explore issues related to events across wide-area networks [11] and enabling project awareness [15, 16, 28].

With respect to the latter, each of these systems adopt a similar approach to iScent in which an event notification service serves as infrastructure to a higher level set of services. For instance, NESSIE [28] and Elvin [15] each use events generated by applications (and sensors in the case of NESSIE) to provide project awareness through mechanisms such as a ticker tape application in which event notifications scroll across the bottom of a user’s screen. The iScent framework differs slightly from these approaches in that it provides a unifying abstraction around which project awareness is conveyed (event trails) and it provides additional mechanisms (as explained in Section 3) to enable intersubjectivity. For instance, with a ticker tape application, a user knows that his events are being broadcast to other users in his or her workgroup. However, they do not know if the users have actually seen these events scroll by, perhaps because they were away from their desk at the time or the ticker tape application’s display was covered by some other application window. In iScent, several mechanisms combine to achieve the “I know that you know that I know” quality of intersubjectivity: namely that a user’s event has been delivered, its recipient has seen it, and the recipient knows that the sender knows that he or she has seen it. These explicit mechanisms insure that intersubjectivity has been achieved and thus raises the quality and fidelity of the project awareness among iScent users.

Process-Centered Software Environments
Process-centered software development environments [1] provide techniques and tools for making the steps of a software life cycle more explicit and visible to the developers participating in a software development project. Recently, we performed an analysis of these environments, using activity theory [25], to examine their support for computer-supported cooperative work [5]. Our analysis, in part, inspected the interaction paradigms employed by process-centered environments. This aspect is relevant to project awareness, since it is through these paradigms that the state of a software life cycle is conveyed to developers. There are three interaction paradigms used in these environments:

- Task-oriented: This interaction style involves the use of agendas. Agendas manage lists of relevant tasks for each user, as e.g. in SPADE-1 [4].
- Document-oriented: Interaction is achieved in these
systems via documents and document services. In Merlin [21], for instance, a work context graphically displays the relevant documents available to and associated with each user role.

- Goal-oriented: Interaction in this paradigm is centered around a list of goals to be accomplished. In Marvel, for instance, these goals represent currently active rules that can be applied to the state of the process [22].

Each paradigm enables a limited form of project awareness but without the full range of support for intersubjectivity that the iScent framework provides. For instance, few of these systems allow users to specify events such as “notify me when Jane has viewed document A.” In addition, the process formalism being applied constrains the type of awareness that can be achieved among team members. At best, they are aware of their process. In contrast, the iScent framework’s use of trails of events allow the awareness of multiple aspects of a work environment to be captured, whether or not an event is associated with a particular software life cycle.

**Awareness Support**

Support for awareness has a rich history of research. For instance, Dourish and Bellotti examined how shared feedback mechanisms can contribute to project awareness among groups organized around a shared workspace [13]. Shared feedback involves the automated collection and distribution of information that is then presented as background information to the participants of a shared workspace. The iScent framework is an example of a shared feedback mechanism in which events are automatically collected and distributed in the background as team members perform work. These events are then presented to users as hypermedia-enabled trails of events. The trails of events can be organized along many intersecting dimensions. For instance, the trail “all events opening document A” can lead to the trail “what did Jane do after she opened document A.” Each event in a trail is “hypermedia-enabled,” which means that there is enough information associated with an event to allow an open hypermedia system to link the event with its associated application and/or artifacts. In this way, the shared workspace is defined by the event trails themselves; any application or artifact that can be reached from an event is part of the workspace. This allows workspaces to fluidly expand or contract based on the activities of the software team. And, because trails are persistent, it is possible to “travel back in time” and see the structure of a workspace at any point.

One large class of project awareness research involves the use of video to enable project awareness among the members of an organization. Example work in this area includes Portholes [14] and Montage [32]. iScent represents an orthogonal approach to project awareness from these video-based systems; however an interesting intersection between the two domains would be to integrate a video-based awareness system into an event-based framework like iScent such that the video interactions of team members became a part of the project awareness captured by iScent.

A second class of project awareness research involves developing frameworks for adding awareness mechanisms to collaborative applications. Representative samples of this work include [17, 19, 24, 31]. As discussed below in more detail, our approach to supporting awareness does not involve the integration of awareness mechanisms into tools directly. Instead, the iScent framework makes use of event notification systems to transparently capture and distribute event information generated by applications. The iScent infrastructure can then assemble these events into trails. As such, integrating applications into an event notification system remains an issue. However, this task involves significantly less effort than integrating awareness mechanisms directly into an application, since it avoids issues of modifying an application’s user-interface. It can even be achieved when an application’s source code is not available by the use of wrappers, as long as the application provides some form of external interface.

A third class of project awareness research involves creating high-level frameworks for supporting awareness or, more generally, collaborative functionality in applications. The goal here is to specify a conceptual framework that applications can adhere to, backed by an implementation of the framework that will automatically enable collaboration through the application’s use of the framework. Example work in this area includes [6, 20, 26, 30]. Again, iScent places no restrictions on its participating applications other than the requirement that they be integrated with an event notification system. However, conceptually, iScent is able to address issues present in other conceptual frameworks. For instance, Hayashi et al., present a framework designed to support the sharing of knowledge between people, projects, and places [20]. They represent an activity as a chronological thread of snapshots of the information in a workspace. iScent event trails are not restricted to capturing activities along a temporal dimension only. Events are stored persistently and can be assembled into trails along a variety of dimensions. For instance, a user can request to see Jane’s trail of events for the work she did last Wednesday, however they can also request to see those same events organized by the projects she was working on, or by the applications that she used that day, or any other axis that can be represented by the value of an event’s fields. In addition, the concepts of people, projects, and places can all be inferred from standard event fields such as the user who generated an event (e.g. “John”), their work location at the time (e.g. “John’s laptop”), and the project they were working on (e.g. “iScent documentation”).

**Document Management**

LaMarca et al., take an innovative approach for supporting document-centered collaboration in the Placeless Documents project [23]. Here, coordination and collaborative functionality is associated with documents rather than applications. This is enabled by associating active code with all
operations that can occur on documents, including reading, writing, deleting, etc. When an operation occurs, the active code can maintain coordination and collaborative constraints by posting event notifications, performing additional operations, and even denying the original operation in the first place for example. LaMarca et al., use this framework to describe an application, Shamus, that supports software engineers with implementation activities such as checking code in and out of a configuration management system and automatically compiling code and documentation whenever code is changed. They argue that the placeless documents architecture enables a finer granularity of awareness to be achieved than is normal in software development. For instance, Shamus is able to inform developers working on the same piece of code where exactly each developer is working within a file. Previously, developers could only know, at best, that more than one person is working on the same file at once.

In a similar fashion, the iScent framework is an attempt to increase the granularity of awareness that can be achieved via the use of hypermedia-enabled trails of events. As will be discussed in detail in Section 3, an event can be at any level of granularity ranging from key press events to document events to process level events, depending on the amount of detail provided by its source application. While iScent has made a tradeoff because it depends on application integration—a restriction that is not encountered in the Placeless Documents approach—it then has the ability to capture a wide range of events, including those that are not associated with documents. In addition, by making use of open hypermedia, we allow users to quickly traverse from (a visualization of) an event to its related application and/or documents. This latter feature allows users to access documents in a uniform way as part of a collaborative process enabled and managed by the event trails themselves. Thus, the iScent framework can be seen as another point in the design space first mapped by the Placeless Documents project, because it also does not attempt to place coordination and collaboration functionality into applications. It instead associates this functionality with event trails as a new abstraction for project awareness.

3 Approach
We now present the iScent framework. Since the transport layer of iScent is an event notification system, we begin our presentation with a brief review of the capabilities and characteristics of these systems. We then present the architecture of the iScent framework and describe in detail its constituent parts. The utility of the iScent framework is illustrated via a scenario which includes a step by step explanation of how iScent enables intersubjectivity. We then conclude this section by briefly addressing privacy concerns raised by the iScent approach.

Event Notification Systems
iScent uses an event notification system as a transport layer. These systems are, in general, based on the concepts of events, producers/consumers of events, and event subscriptions/notifications. Typically, an event is a set of key/value pairs. Producers publish events to an event server which routes these events to consumers based on their subscriptions. One benefit of this arrangement is that producers are completely unaware of the location of interested consumers (and are thus not dependent on these consumers in any way). Likewise consumers are unaware (and not dependent on) producers. This arrangement can lead to significant benefits. For instance, the C2 architectural style makes use of these characteristics to provide substrate independence in software architectures [33]. Other advantages include:

- Producers and consumers focus only on events meaningful to them. They have no need to understand the entire event space being managed by the event system. They are, thus, straightforward to create and configure.
- Event systems make efficient use of a network. When an event is produced, only those consumers who subscribed to the event are notified.
- If several event servers are used (which is, for instance, possible with the Siena system [11]), the routing of events can be further optimized (e.g. an event is only sent to an event server if it has clients that are interested in that event). This facilitates the use of an event notification system across a wide-area network.
- The publish/subscribe model enables dynamic service discovery. For instance, a consumer can publish an event requesting a specific service. If there is a producer that provides the service, it will notify the consumer, and the consumer can subscribe to it.

By making use of an event notification system, the iScent framework provides these same advantages and characteristics to its users.

The Architecture of the iScent Framework
The architecture of the iScent framework is presented in figure 1. The components of the framework are:

- iScent Client Applications iScent applications produce iScent events.
- Sink A sink consumes all iScent events. It is responsible for making events persistent and provides an interface that allows other components to issue queries over the stored events.
- Trail Viewer A trail viewer specializes in the visualization and structuring of collections (“trails”) of iScent events. A trail viewer provides hypermedia capabilities over its trails such that software engineers can traverse from an
event to the event’s associated information. These hypermedia capabilities are provided via integration with an open hypermedia system [27].

**Watchdog** A watchdog is a persistent trigger than can generate iScent events when a specified condition is met. Watchdogs are used to support awareness since they play a key role in achieving intersubjectivity (as described below).

**Kennel** A kennel is a collection of watchdogs. A kennel will often be associated with a sink, however they can also exist independently.

Communication between iScent components is primarily handled by iScent’s integrated event notification system. Communication via events is symbolized in figure 1 by solid arrows, i.e. no point to point network connections exist between components connected by solid arrows. Dashed arrows signify point-to-point socket connections, which are used when large packets of information must be transferred between components. Event systems typically limit the size of events to ensure adequate performance. Therefore, queries with large result sets must be transferred outside the event system.

A typical iScent configuration consists of one or more iScent applications deployed on a set of user machines, and a set of sinks and kennels distributed across a set of server machines. Note, however, that a user can install a sink on his or her local machine. The main benefit of this configuration is the ability to use iScent “unplugged,” i.e. without contact with other sinks or event servers. Once a machine is plugged into the network, the local sink can offer its stored events to other sinks for replication and distribution to other team members.

The configuration of iScent components involves linking each component to a local event server and, in the case of sinks, specifying the types of iScent events to store. An event server will likely be coupled with other event servers which are in turn connected to other sinks and iScent applications. It is thus possible to create highly-distributed large-scale networks of iScent components.

**An iScent Scenario**

The utility of the iScent framework is now demonstrated via a scenario. While the iScent framework is general enough to be applied in many work settings, our first target setting is the task of software development.

Jane, a system developer, has just returned from vacation. She begins her first day of work by starting a trail viewer. While she has been away, many events matching her criteria have been generated. The trail viewer retrieves these events, and presents them in juxtaposed trails sorted according to topic and time of occurrence. There is too much information to digest, so Jane engages a set of filters that she has previously defined, which simplifies the view considerably by folding matching patterns of events into meta events. Jane sees in the CVS trail that several new files have been checked into a CVS repository. Checking her e-mail, she notices a message from John telling her about one of the new files and asking her to take a look at its associated design rationale in the documentation. She clicks on the event representing the check-in of the file to see further details, and notices a link to the file’s documentation. She follows the link which loads the documentation into a Web browser. While reading, she notices that John has put a watchdog on the documentation, and five minutes later, John calls to hear her opinion of the design rationale.

This scenario illustrates key aspects of iScent’s functionality. Since iScent events are persistent, it is easy for Jane to come up to speed with what has happened while she was on vacation. Jane need not worry about specifying where events should be retrieved; this is handled automatically by iScent (while she was away, the entire structure of sinks could have been reorganized without her noticing or caring). Rather than presenting a single, large list of events, the presentation of events is structured, and can be further manipulated. To provide further structuring mechanisms, it is possible to create links to and from the events stored in the sinks. Because Jane’s Web browser is iScent aware, her viewing of the documentation creates an iScent event, which can trigger a watchdog. This allows John to wait until Jane has actually discovered and viewed his changes, before contacting her.

**Enabling Intersubjectivity**

We now parse the scenario into single iScent actions to illustrate how the iScent framework enables intersubjectivity among software developers.

When John checked in a new file, an iScent event representing his action was created and stored in a project-related

---

Figure 1: The Architecture of the iScent Framework
Thus, Jane was aware of the context for John’s call because he knew that Jane had expertise on the topic of the new file, he sought her opinion of the design through e-mail. Wanting to be told when she had read the documentation, John used his trail viewer to create a watchdog, set to trigger when Jane retrieved the documentation with her Web browser. This watchdog was stored at a project-related kennel, and the kennel issued a subscription matching the criteria of the watchdog. By posting the watchdog, John automatically created a subscription matching the event that the watchdog would send if triggered. To use the iScent vocabulary, John wanted to be alerted if his and Jane’s trails of actions intersected at the point of the documentation. Having done this, John was free to work on other things, knowing that he would be notified when to contact Jane.

When Jane started her trail viewer, the trail viewer issued a series of query events on the topics she had configured it to follow (essentially the subscriptions she had created earlier). In this case, Jane was interested in iScent events of the types “CVS” and “Documentation.” These query events were consumed by a project-related sink which then confirmed that it could answer her query. The confirmation events held information (a network address and a port number) for the trail viewer to retrieve the trails generated by the sink based on the queries. The trail viewer proceeded to contact the sink and retrieve the trails.

Once retrieved, the trail viewer displayed the trails, and Jane could begin to manipulate them to gain an overview of the changes made while she was on vacation. In this situation, the trails were arranged by type (“CVS” and “Documentation”) along a time axis. When Jane read the e-mail from John, she found the matching event, and followed a link from it to the documentation. Following the link caused the documentation to be loaded into Jane’s Web browser. Since the Web browser was iScent-aware, it generated an iScent event that recorded the action of loading the documentation. At the project kennel, John’s watchdog was triggered by this event. The watchdog, in turn, generated an event containing its ID and the information about the person, who had triggered it. Because a trail viewer by default subscribes to events that contains a “triggered by” field matching its user, the event was received by Jane’s trail viewer, alerting her to the existence of a watchdog and to the identity of its creator. Simultaneously, the same event was received by John, and he now knew that he could contact Jane, in the near future, to discuss the design rationale.

By reading the email, Jane knew that John wanted her to read the documentation. When she actually read the documentation, John was alerted, so that “he now knew that she knew”. Conversely, Jane knew by being alerted by a watchdog, that “John knew that she now knew” about the design rationale. Thus, Jane was aware of the context for John’s call because intersubjectivity had been achieved.

**Trails**

Trails are a cornerstone of the iScent framework. Through the use of iScent applications, a user generates a trail of iScent events (a “scent trail”). These are stored by sinks, and can later be retrieved through queries. A trail is defined as a set of iScent events. Thus, the actions of a user can be captured by a trail of iScent events matching the user over time. Another trail could be the events matching a certain tool over time, or following the history of a document. These trails intersect, for instance, when a user makes use of a certain tool, or modifies a certain document. Since trails are modeled as sets of events, trails support the normal set operations: join, intersection, and difference. These operations match on selected common fields among the constituent events, and can be performed by a trail viewer or by expressing a query to reflect the desired operations.

The user manipulates trails in a trail viewer. Trails can be plotted along axes defined by the user. Returning to the scenario, Jane plotted two trails consisting of CVS events and documentation events respectively along a time axis in order to see the correlation between code produced and documentation written. When plotting a trail, one field common to all the constituent events is used to order the events along an axis. As a time stamp is common among all iScent events (see Table 1 for the default iScent fields), a common ordering is time based, though other orderings can be used.

As a set of events, a trail can be arbitrarily large. One of the design goals of iScent is to produce highly detailed events. However in some circumstances too much detail can be a liability, since a user can easily lose his or her sense of the “big picture” behind the events. To address this, without abandoning high fidelity events, trails can be filtered. A filter is essentially pattern matching on events, such that matching events are replaced in a trail viewer with a meta event (or a “folded” event). One could for instance define a pattern to replace a sequence of CVS events relating to a single file with a single event representing all actions pertaining to the file. The user can of course unfold a folded event to inspect the constituent events. Once defined, filters are stored in a trail viewer and can be applied as the user desires. Folded events are iScent events and can, as such, also be subjected to filtering.

Trails of events can be transient, e.g. existing only on a user’s screen, or they can be stored — either as a specification of the queries and filters that produce the trail, or as a sequence of event ids (all iScent events have unique ids, as described in Section 4). Finally, it is possible to export (or import) trails as XML files for external use. The export format is identical to the format used by sinks to send trails to trail viewers.

**Privacy Concerns**

Not all actions taken by an iScent user is necessarily relevant for other users. Sometimes it makes sense to track all ac-
tions, such as a user’s CVS operations, and sometimes it does not, such as what Web pages a user visited. User can and should therefore be able to configure filters that designate the types of events they want to publish. In the context of a Web browser, one could, for instance, publish all events regarding technical documentation. This not only addresses privacy concerns, but also improves the overall signal-to-noise ratio in the trails stored by sinks.

4 Implementation
We have constructed a prototype implementation of the iScent framework, including initial implementations of the following components: sinks, watchdogs, and kennels. These components are sufficient to test the creation and storage of iScent events and to validate the iScent type system and query facility. Our work on a trail viewer is preliminary and has involved the design of its user interface and the construction of a tool that can query and retrieve event trails from sinks. Our initial prototypes have been stress tested through the use of batch tools that generate tens of thousands of events and then issue queries that validate the operation of the sinks, watchdogs, and kennels. These initial efforts represent a proof-of-concept of the iScent framework; our future work will involve fleshing out the trail viewer and integrating client applications. Below, we discuss the various issues that our initial implementation has raised.

Requirements of the Event Transport Layer
The iScent architecture relies on an event notification system to provide the transport layer for iScent events. The requirements on the transport layer are:

- Events must consist of key/value pairs
- Values can be arbitrary strings
- Subscriptions to events must be supported
- Subscriptions to events should be specific key/value pairs (e.g., “name equals John Smith” or “age greater than 24”)

These requirements are met by most event notification systems. Therefore, the iScent framework is independent of any particular event system. The current version of iScent utilizes the Siena event notification system [11], which easily satisfies the above requirements.

Scalability of iScent
iScent’s scalability is dependent on the scalability of its associated event notification system. Our implementation of iScent therefore depends on the scalability of Siena, which has been characterized by Carzaniga et al. in [11]. One of the main benefits of Siena is that Siena servers can be coupled together in a network. Given a situation where many iScent components and Siena servers are in use, Siena automatically provides optimal routing of events from producer to consumer to maximize performance and minimize latency.

Multiple Sinks
iScent can be configured with any number of sinks. This has several advantages over a single store:

- Sinks can be local to a workgroup, and run on a local machine. This provides improved performance as the sink only stores events produced by its workgroup.
- While maintaining local sinks, the addition of larger sinks collecting all events generated by, e.g., a department, provides additional benefits. Data is automatically replicated between sinks for greater data security, and if one sink is down or slow due to high load, other sinks can provide access to the desired events.
- These configurations are achieved by installing more or less specialized subscriptions into sinks that determine the set of events they store and share. A configuration is flexibly manipulated since a sink can be added or removed at run-time simply by changing its subscriptions.

Querying for Events
Sinks provide for the persistence of events and handle all queries. Queries are sent to sinks via Siena. This allows queries to be handled in an efficient and scalable manner.

A query is an iScent event that specifies the type of desired events and (ranges of) values in these events. A sink subscribes to the query events matching what it stores, and queries are thus automatically routed (by the underlying transport layer) to the sinks that can provide answers. When an iScent component issues a query event, it also subscribes to confirmation events matching the query. If there are sinks that are able to fulfill the query, they will issue confirmation events which are then received by the querying party. While most event notification systems can efficiently route events, they are in general not optimized for very large events (Siena for instance has an upper limit of 64 K). Query results can be arbitrarily large, and rather than returning the query result itself in an event, the confirmation event contains contact information for the answering sink. Upon retrieval of the confirmation events, the querying party establishes direct socket connections to the sinks and retrieves the events (it can also choose to select only one of the sinks for retrieval). This lessens the load on the event transport layer considerably. To lessen the network load further, the queries are compressed and decompressed automatically. The retrieval of query results is the only time where an iScent component makes a direct point to point connection to a sink — at all other times it generates iScent events that are automatically propagated by the event transport layer.

The query and confirmation events are themselves iScent events, and are, as such, stored by the sinks. Apart from the awareness possibilities in these events, it also allows a system administrator to monitor how the sinks are used, and
exploit this information to move sink data and subscriptions to improve network performance.

A sink uses an SQL database for storage, and automatically generates a new table, when an iScent event of a new type (see below for information on types) is received. This facilitates rapid query resolution. The current query interface has been designed to make the most of the SQL database. Queries can be combined using logical operators, and is able to create very precise queries.

**iScent Events**

An iScent event is the atom of the iScent framework. An event is composed of key/value pairs (or “fields”). All events have a default set of key/value pairs, shown in Table 1.

<table>
<thead>
<tr>
<th>Key</th>
<th>Example Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>User</td>
<td>John Smith <a href="mailto:john@comp.com">john@comp.com</a> org.iscent.app.iScentApp</td>
</tr>
<tr>
<td>Producer</td>
<td>iscent.iScentApp</td>
</tr>
<tr>
<td>Host</td>
<td>138.128.34.10</td>
</tr>
<tr>
<td>Timestamp</td>
<td>956809312389</td>
</tr>
<tr>
<td>id</td>
<td>9KjxG3iBMCvHALTrOe=6xW</td>
</tr>
<tr>
<td>Types</td>
<td>type1 type4</td>
</tr>
</tbody>
</table>

Table 1: The default iScent key/value pairs

Thus, an event can provide information about its user, how it was created, where it was created, and when. The id is a 128 bit integer (here presented in base64 form), created as a MD5 hash signature of the rest of the event. The use of MD5 ensures both a unique id and data integrity (since the MD5 signature is recomputed at arrival and compared with the existing one). In addition, most events use the Types field to declare their type(s), i.e. specifying which other fields the event contains.

**The iScent Type System**

iScent events are typed, and are checked upon creation and arrival. The type system is component based and extensible. Types are defined through iScent type declaration events and are stored by sinks. A type declaration consists of a (globally unique) type name, one or more fields with key names unique within the type, and a definition of the types of these fields.

All values in an event (with the exception of the default values listed in Table 1) are encoded in XML. The field definition is a DTD specifying the format of legal values for that field. The DTDs are stored by sinks and are used by them to validate the events they consume.

The basis of the iScent type system is the declaration of simple types, but iScent events can have more than one type. When an event declares several types, it guarantees that it contains valid values for all the fields defined in all of the types. The names of keys are a combination of their original iScent type name and their own field name, which eliminates name collisions between types. Combining types can be very useful; a class of iScent applications can, for instance, share a common “header” type, plus their own specialized types. As the number of iScent applications grows, so do the types defined. These types then provide building blocks that can be used by other iScent developers when constructing new iScent applications.

**Watchdogs and Kennels**

A crucial part of supporting intersubjectivity is the watchdog. The watchdog is used to detect the occurrence of certain iScent events (for instance, opening a document), and alerts the creator of the watchdog when the condition has been met. Currently a watchdog consists of two parts: criteria that must be met, and an event to send when triggered. Watchdogs can be set to trigger only once, and then disappear, or to continue to trigger when conditions are met. If a watchdog reported only to its creator, it would increase the creator’s project awareness but it would not provide intersubjectivity, since the people triggering the watchdog would be unaware of its existence. To enable intersubjectivity the watchdog also reports to the person who has triggered it. Trail viewers automatically subscribe to watchdogs events triggered by their user, and will alert its user when such an event is received. Likewise, by creating a watchdog, a user automatically subscribes to responses from it. These responses will be received and displayed by a user’s trail viewer.

To ensure the persistence and vigilance of watchdogs, they reside in kennels. A kennel is a server that registers the conditions of its watchdogs and creates matching subscriptions. When an iScent event is received, it is checked against the conditions and, if a match is found, triggers the creation of an event specified by the appropriate watchdog.

The triggering mechanism is currently a simple if-then mechanism. Future work will involve the creation of more sophisticated watchdogs. The most likely approach for improvement will be the creation of watchdog Java classes, that can be uploaded into a kennel. These Java watchdogs can maintain state between triggering events, so they can, for instance, be used to detect patterns of events and generate an event only after a pattern has been detected. Another topic for future work is the migration of watchdogs. Currently watchdogs are sent to the nearest kennel (or rather the kennel that first accepts the watchdog event). To optimize performance and minimize the network load, it would be better if watchdogs could migrate to a kennel close to the producers of events that match the watchdog’s criteria.

**5 Future Work**

There are many avenues for future work in further developing the iScent framework and its associated implementation. Chief among them is the need for evaluating the iScent framework’s ability to address the project awareness needs of modern software development projects. We plan to perform this type of evaluation with two very different methods. The first method is to conduct laboratory-based usability studies
on the user interface of the trail viewer and the kennel. We need to determine if the conceptual model presented by the iScent framework provides utility to software engineers and if the user interface is effective in delivering the functionality of the iScent framework into the hands of its end users. The second method is to conduct field studies of the iScent framework in use at an industrial site. The key problem here is identifying industrial partners and obtaining a commitment to participate in industrial collaboration. We have successful track records in university/industry interchange [2] and are beginning the process of obtaining industrial collaborators to support this line of research. The goal of the field studies will be to increase our understanding of the work practices currently used in industry and how the iScent framework either hinders or enhances these procedures.

Secondary plans for future work on the iScent framework involve exploring techniques to increase the fidelity of event trails from integrated iScent applications. Application events of high fidelity leads to better “conversations” between engineers and increases the intersubjectivity that can be achieved by the iScent framework. However, it must not be difficult to integrate applications into the iScent framework and thus techniques are needed to reduce the effort required to integrate an application into the framework. Again, such efforts will enhance the project awareness that can be achieved by the iScent framework by giving it more sources of events and thus increasing the percentage of project work it captures. In addition, we will be exploring new trail visualizations, new query mechanisms over trails, more flexible support for watchdog behaviors, and better tool support for managing iScent networks.

6 Conclusions
We have presented a framework designed to support project awareness in large-scale software development contexts that takes a unique approach to the problem. Leveraging already existing open hypermedia and event messaging infrastructures, the iScent framework combines the use of hypermedia trails with event publish/subscribe techniques to enable intersubjectivity between software engineers and thus promote wide-spread project awareness throughout a software development team. We have described the conceptual layout of the framework and provided insight into an experimental implementation of it. The implementation employed aggressive reuse of fielded infrastructure, database technology, and distributed system techniques in order to scale the implementation to a level where it becomes feasible to conduct field studies of the new technology in industrial settings. Having met these initial scalability concerns, our attention now turns to evaluating the utility of our formalisms and the effectiveness of our prototypes.

REFERENCES


