RC 22089 (99042) 12 June 2001 Computer Science

IBM Research Report

An Intelligent Notification System

Vincent Bazinette, Norman H. Cohen, Maria R. Ebling,
Guerney D. H. Hunt, Hui Lei, Apratim Purakayastha,
Gregory Stewart, Luke Wong, Danny L. Yeh
IBM Research Division
Thomas J. Watson Research Center
P. O. Box 704
Yorktown Heights, NY 10598

Research Division
Almaden - Austin - Beijing - Haifa - India - T. J. Watson - Tokyo - Zurich

An Intelligent Notification System

Vincent Bazinette, Norman H. Cohen[1], Maria R. Ebling, Guerney D. H. Hunt, Hui Lei, Apratim Purakayastha, Gregory Stewart, Luke Wong, and Danny L. Yeh

IBM Thomas J. Watson Research Center
Yorktown Heights, NY 10598
{vbazine,ncohen,ebling,gdhh,hlei,apu,gstewar,lukew,dlyeh}@us.ibm.com

Abstract. Today, pervasive computing often centers around accessing web information from handheld devices. One attractive alternative is for a user to subscribe to relevant information and to be notified when such information becomes available. In this paper, we describe a system that allows users to specify a wide variety of interesting, and possibly complex, events, which may require the system to aggregate data from numerous content sources. Furthermore, because users typically have different devices available to them at different times, the system delivers the message to one or more of the users' devices based upon their general preferences as well as their present context. Our Intelligent Notification System reduces the demand for user attention by delivering the requested information to the user on a convenient device in a timely and appropriate fashion.

1 Introduction

The quantity of information available on the Internet is immense and it increases each day. Anyone needing to detect when certain events occur or when certain conditions are met is faced with the difficult challenge of detecting those events or of monitoring those conditions. Compounding this difficulty is the fact that, at its core, the Web is a pull technology. Consequently, the user must actively search out the needed information on a regular and frequent basis.

Consider Susan, an executive who regularly invests in the stock market. She would like to invest in PQR Corporation, but only when the price falls below $14 per share and her current on-line trading account balance is at least $5000. Because of her interest in this company, she also keeps a close watch on the business news for any articles mentioning PQR. Using today's technology, she must monitor her account balance and, when it is above $5000, she then must monitor the price of the stock. She checks the stock price on-line whenever she gets a chance during the day and, when time permits, she also scans the CNNfn web site. Because her job responsibilities must take priority, she frequently does not even get the opportunity to check the stock price of PQR.

To address these difficulties and alleviate the burden such activities place on users, we have built a system, the Intelligent Notification System, that allows users to specify events and

[1] Contact author

conditions of interest to them. The system monitors information from a variety of sources, such as news wires and stock tickers. When the event occurs or the conditions are met, the system notifies the user via a convenient means, controlled by the user, such as instant messaging, e-mail, cellular phone, or WAP.

Using the Intelligent Notification System, Susan monitors PQR with much less effort. She submits two subscriptions. One subscription notifies her of any news wire articles mentioning PQR. The other notifies her when the stock price falls below $14 per share and her current account balance exceeds $5000. When an article of interest is published or when the stock and account requirements are met, the Intelligent Notification System notifies Susan. As an executive however, Susan is often in meetings with customers. In such situations, instead of the normal chime of her WAP phone, she would like the system to alert her *gently* using either instant messaging (if she happens to be logged on and not making a presentation) or her pager (which she keeps set on vibration mode).

This vision presents a number of interesting research challenges. One such challenge relates to the quantity of information brought to the user's attention. The system must allow users to define criteria that are sufficiently selective to avoid overwhelming them with irrelevant data (and incurring high wireless-communication fees). Yet, the criteria must be sufficiently broad to allow for an adequate and appropriate match rate. Another challenge is that, in the era of pervasive computing, users may have more than one device and they may switch from one device to another many times throughout the day. Further, depending upon users' current activities, not all of their available devices may be appropriate. Our system must be capable of knowing which device is appropriate to use at any given time. Additional challenges include scalability and privacy.

In this paper, we begin by discussing the desired features and challenges presented by this system. We then discuss the architecture of the system, including each of the major components. Next we discuss some outstanding research issues that we are working to address. We conclude with a discussion of related work and the contributions of this research.

2 Desired Features and Challenges

The key features of this system include programmable aggregation, multi-modal notification, and exploitation of user context. Each of these features, including its desired functionality and the challenges it presents, is discussed in the following sections.

2.1 Programmable aggregation

Most web portals today (e.g., my.yahoo.com) offer some form of content aggregation that collects data from various sources and presents it to the user in a personalized manner. The various data sources are often explicitly identified via URLs (e.g., local weather information from weather.com) and the personalization is centered on presentation. The data from various sources is not materially altered or correlated with other data. In contrast, our system allows a user (or an application running on behalf of the user) to control how data from various sources are aggregated and correlated.

When Susan places her request, our system will first determine that Susan wants live financial information and will open a connection to an appropriate stock data feed. It will then monitor the data feed for the appropriate condition (PQR is trading below $14). When the condition occurs, our system will query Susan's on-line trading account (with proper prearranged authorization from Susan) for current balance and projected cash requirement information. If the balance is greater than $5000, the Intelligent Notification System will initiate notification.

There are a number of technical challenges in programmable data aggregation as we envision it. First, a large number of diverse data sources may need to be discovered and supported. Applications will require support for accessing data from the web, from wire-service–type data feeds, from various databases, and from files. Applications will typically not want to specify the actual physical data source but instead describe the data source in some symbolic form, such as "PQR stock" (or XML equivalent). The system will need to figure out the actual data source appropriate for the request. In some cases, a live data feed will be appropriate; in others, some archival data source will be appropriate. The system must also allow applications to specify aggregation logic as well as provide various libraries of common aggregation functions for faster application development. Second, we envision programmable aggregation to be distributed, especially when we transcend from the consumer domain to the enterprise domain. Data from various sensory data sources (other than location) may need to be aggregated at the edges of the network such that only the aggregated data, and not the voluminous raw data, are propagated across the network. Consider a traffic application that suggests shortest paths to motorists from their current locations to their destinations based on data from various traffic sensors. This entails computation of traffic density metrics based on vast amounts of sensor data. Such density metrics are best computed at the edges of the network in server(s) near the sensors and only the computed metrics used in the shortest path calculations.

2.2 Multi-modal Notification

Susan's requirements imply that we support notifying our users by a variety of means. Users may want notification by voice over wired or wireless phones; they may want text-based notification via email, instant messaging, pagers, or SMS; in addition to text, they may want enhanced notifications (e.g., one containing an image) via instant messaging or e-mail; they may further want hard-copy notifications on a fax machine or a networked office printer. To support this diverse set of requirements, our system uses a large number of device gateways (e.g., WAP gateway) and services (e.g., instant messaging), and performs on-the-fly transcoding of the notification content to the format suitable for the destination device. Due to space limitations on some devices, the actual notification may involve presenting a summary of the content (e.g., just a headline) to the user. To offer this functionality, our service maintains the actual content in persistent store and allows the user to access the full content when desired (sometimes very conveniently by simply clicking a URL embedded in a text notification).

Further, our system notifies Susan according to her *preferences*. These preferences may involve static directives, such as sending urgent notifications via cellular phone and FYI notifications via e-mail. They may also involve dynamic directives, such as using low-key notifications while Susan is in a meeting. The preferences may also involve further user directives such as a preference for instant messaging over pager messages when the user happens to be logged on.

UND is only useful if the service it provides is easily useable by the average person. The principal challenges are in the user interface and the presentation of information. How much control does a recipient need? How do we present the preferences options in an easily comprehensible way? What sort of delivery guarantees are needed and how can we provide them?

2.3 Exploitation of User Context

To deliver notifications in the most appropriate manner, our system must gather information about Susan's context (e.g., whether or not Susan is in a meeting). Context includes such information as daily activities (inferred from an electronic calendar), on-line presence, and location. Rather than deriving and managing user context in an ad hoc manner, we have chosen to place the functionality in the infrastructure and have developed a context service. An infrastructural context service brings two advantages. First, it simplifies the development of other pervasive computing applications and encourages them to exploit diverse sources of user context. Second, it amortizes the costs of introducing new context sources across multiple applications.

Developing a context service presents a number of challenges. First, context information, by its nature, is personal and sensitive. Our instincts impressed upon us the need to protect the privacy of context subjects to the greatest extent possible. A privacy protection mechanism for context services must make provisions for the specification and enforcement of individualized policies. Second, the space of context information is tremendous. There are many different aspects of the computing context and the same aspect may be represented at different semantic levels. Services supporting a general notion of context must accommodate heterogeneous context sources and must allow new context sources to be easily added. Third, applications' needs for context information are as diverse as the applications themselves. Therefore, a context service should provide structure to heterogeneous context information and present an integrated view to applications. It should supply the quality of context information upon request and allow applications to assert their quality of information requirements. Further, it should support both synchronous and asynchronous operations.

3 Architecture

Figure 1 depicts the architecture of the Intelligent Notification System, as it would be deployed in a Web server environment. In such an environment, end users access the system via a web browser. Application servlets access the Intelligent Notification System through a Java API and allow users to register events of interest to them (*trigger servlet*) and to access content after a match has occurred (*content-retrieval servlet*). In addition, *content adapters* collect content from various sources, such as news feeds and web sites, and push[2] that content to the Intelligent Notification System. In addition, users may request that a message be sent to a

[2] Although a content adapter may obtain its content through either a push mechanism (e.g., subscribing to a service that automatically sends new content) or a pull mechanism (e.g., actively polling a known content source), it is responsible for *pushing* that content to the Intelligent Notification System.

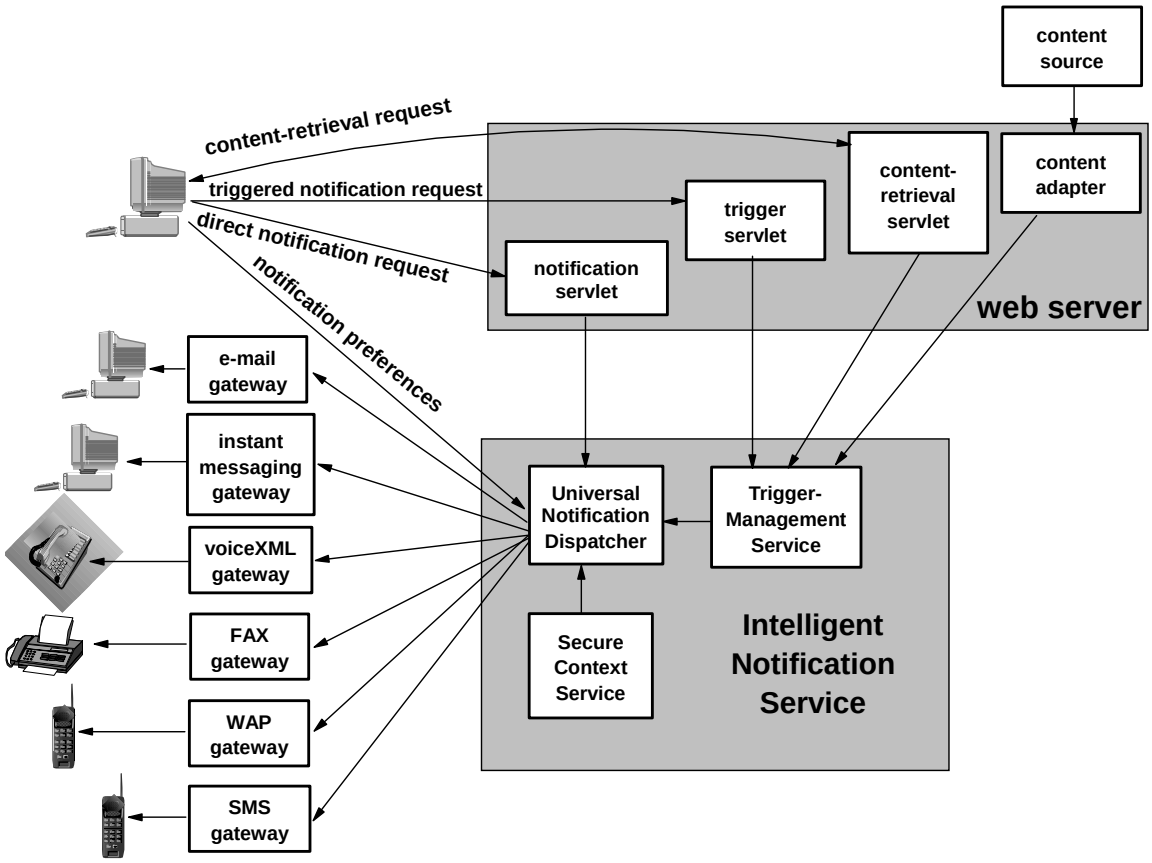

Figure 1. Architecture of the Intelligent Notification System and its environment.

specified user or users directly (*notification servlet*), but this functionality is orthogonal to this paper and will not be discussed further.

The three major components of the Intelligent Notification System are the Trigger Management Service, the Universal Notification Dispatcher, and the Secure Context Service.

- **Trigger Management Service.** The Trigger Management Service maintains a record of user interests and accepts content from various sources. When content arrives that is of interest to one or more users, the Trigger Management Service identifies the match and takes appropriate action, including simply notifying the user, according to each user's wishes.
- **Universal Notification Dispatcher.** The Universal Notification Dispatcher transmits messages to specified end users through gateways for a variety of devices, selected according to the end users' preferences. Some of these preferences are contingent on the end user's current context, which the Universal Notification Dispatcher obtains from the Secure Context Service.

- **Secure Context Service.** The Secure Context Service maintains context information about its users. It allows users to specify who can access the various types of context it maintains. When a context request arrives from the Universal Notification Dispatcher or any other application or service, the Secure Context Service authorizes the requester and releases only that context which the user has approved.

Each of these components is discussed in more detail in the sections that follow.

3.1 Trigger Management Service

The Trigger Management Service manages a user's events of interest and matches the incoming messages from content adapters to users' events of interest; if a match occurs, the service may store content associated with the event into a persistent content store and/or send notification to end users. A user's events of interest are termed *triggers*. A trigger is associated with a *trigger handler*, with one or more *content sources*, and with a *firing condition*. A content source is a named entity to which a content adapter may direct messages. For example, a content adapter obtaining content from the Reuters News Service might direct a message to a content source named "`ReutersNewsService`" each time it obtains a news story. The message generated by a content adapter may include named attributes. For example, a message corresponding to a news story might have attributes named "`headline`" and "`body`". A firing condition is an SQL92 predicate in which the names of these attributes may appear. In our example, the firing condition "`headline LIKE '%PQR%'`" would test whether the word "PQR" appears in the subject line of a news story. In this example, when the "ReutersNewsService" content source receives a news-story message whose `headline` attribute contains the word "PQR", a match occurs, and all triggers that have this firing condition are said to *fire*.

When a trigger fires, the Trigger Management Service invokes the `handleMatch` method of its trigger handler. This method executes one or more actions on behalf of the user and returns a result specifying further actions to be performed by the Trigger Management Service. One possible further action is to store the news story into a persistent storage for later retrieval by the user, and another possible further action is to request the Universal Notification Dispatcher to send a notification to the user. The result returned by the trigger handler may also specify that the trigger should be removed, so that it will not fire again. (This would be appropriate, for example, if a user was interested in being notified the *first* time that the price of a particular stock fell below $14, but not every subsequent time the stock was traded at or below that price.) A trigger-handler object may contain variables that maintain state from one call on its `handleMatch` method to the next.

The Trigger-Management Service uses a *matching engine* to direct messages from content adapters to the applicable triggers. The applicable triggers for a given message are those whose content source is the one to which the content adapter wrote the message, and whose firing conditions are true for the attribute values contained in the message. Our architecture uses Gryphon [Agu99], a content-based publish-subscribe system, for this purpose. When a match occurs, Gryphon invokes a callback function that executes the trigger handler associated with a trigger, examines the result returned by the trigger, and performs the further actions specified by

the result. The actions performed by the trigger handler may be conditional on other sources of data, such as the time of day or the user's current bank balance.

The Trigger-Management Service contains two persistent stores, a *persistent-trigger store* and a *persistent content store*. In our architecture, each store is implemented as a DB2 table, accessed through the Java JDBC API. However, we isolate those parts of the implementation that are DB2-specific, and even those that are specific to a relational database, thus making it easy to plug in different persistent stores.

Each time a new trigger is registered, it is stored in the persistent trigger store. The result returned by a trigger handler's `handleMatch` method specifies whether the state of the trigger-handler object should be updated in the persistent store. A `handleMatch` method that modifies the state of its trigger-handler object should return a result requesting that the persistent store be updated; however, many `handleMatch` methods do not modify the states of their trigger-handler objects, so we allow the `handleMatch` method to return a result indicating that the potentially expensive persistent-storage update is unnecessary. The registry of active triggers is initialized from the persistent trigger store when the Trigger-Management Service starts up. Thus, if the service is stopped and later restarted, all triggers that had been registered beforehand are still registered, and all trigger-handler objects are in the state they were in prior to the restart.

When the result returned by a trigger handler's `handleMatch` method indicates that content is to be stored persistently for later retrieval, that content is stored in the persistent content store. The store generates a numeric key that may be embedded in a notification and used by a servlet to retrieve the content later. Content is added to the persistent content store with an expiration time. A scavenger thread runs periodically and removes expired content from the store.

The architecture of the Trigger-Management Service is summarized in Figure 2.

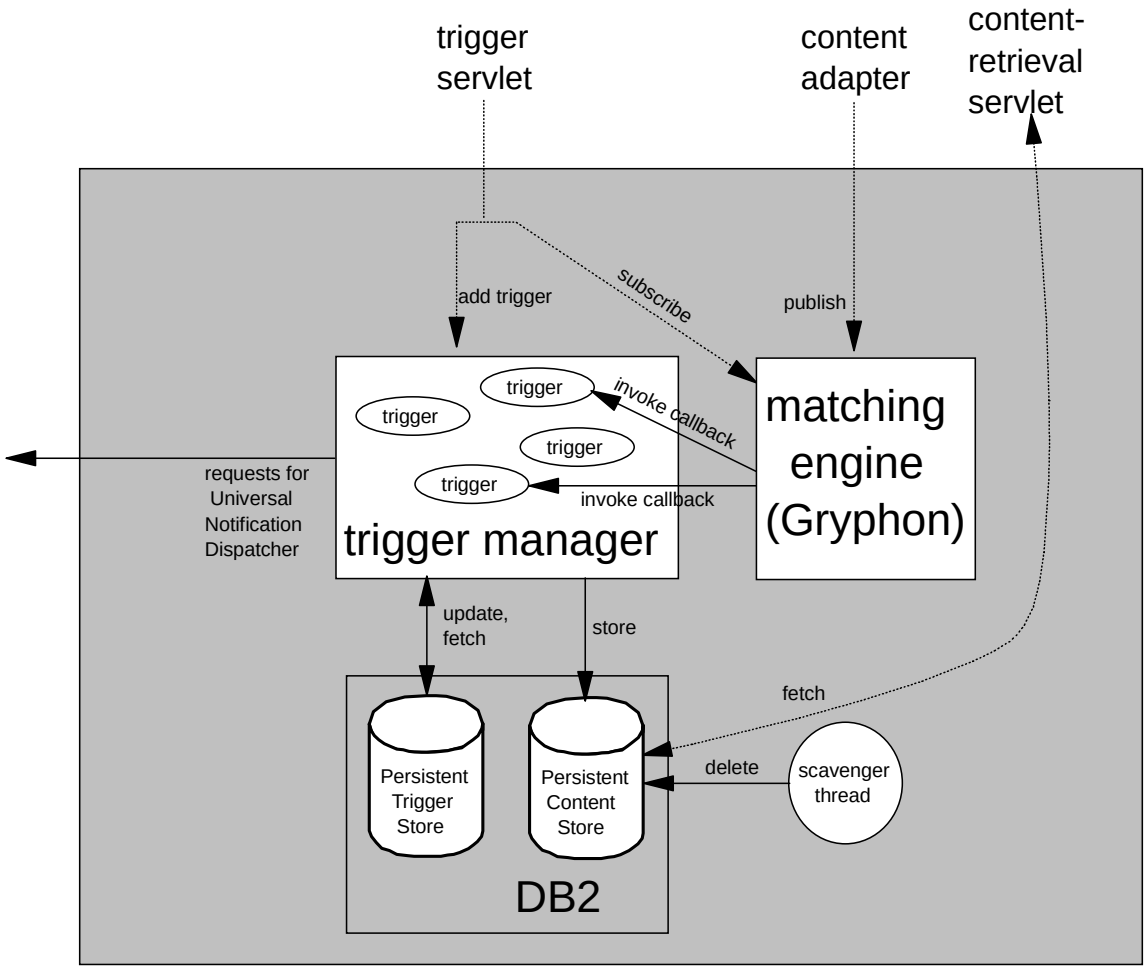

Figure 2. The architecture of the Trigger-Management Service.

3.2 Universal Notification Dispatcher

The Universal Notification Dispatcher (UND) is responsible for dispatching messages to subscribers based upon their preferences. The UND enables subscribers to have a single address through which they can receive messages (or notifications) on multiple devices. Messages are dispatched based on the preferences of the subscriber. The UND is capable of dispatching messages to a variety of devices including telephones, cellular phones, WAP-enabled phones, and instant messaging services.

The UND accepts messages from the *sender* of that message or from an application that is submitting the message on behalf of the sender. Each message is for an individual, who is called the *intended recipient* (or simply *recipient*). The UND assumes that it is operating in a closed environment and that every sender has already been authenticated by the enclosing environment. Further, all recipients must be *subscribers* of the UND.

During the subscription process, users must configure the system by specifying their delivery preferences and device configurations. Users specify their delivery preferences using role-based access control. To facilitate this process, users must first assign authorized senders to groups (e.g., friends, family, managers, colleagues). In addition, each user is automatically given a virtual *anonymous* group. All senders not explicitly assigned to another group receive the authorizations given to this anonymous group. (By default, the anonymous group has no permission.) After defining groups, the user must assign authorizations to each group. For example, the user may only allow certain groups of people to use certain devices or the user may only allow certain types of messages (e.g., urgent, normal, and FYI) to be sent to a particular device. Further, if users choose to use calendar context information, they must configure their contact preferences for the various calendar contexts using a context-enabled calendar tool. These contact preferences include which groups may send notification in which contexts. For example, while traveling, a user may allow her colleagues to send urgent messages but may allow her line management to send both urgent and normal messages.

Our system gives the recipient complete control over the method of delivery. The UND honors the recipient's preferences by dispatching messages only to devices that the recipient has authorized the sender, or more correctly one or more of the groups to which the sender belongs, to use under the current context. It releases no information about which device was chosen to the sender.

Figure 3 shows the architecture of this component. The client submits requests on behalf of senders. Clients can be GUIs, such as the *notification servlet* shown in Figure 1, that allows a user to submit a notification request directly or they can be complex programs, such as the Trigger Management Service discussed previously, that submit notification requests when complex events have occurred.

A *notification controller* uses the recipient's preferences, as received from the *Preference Engine* and the recipient's context, as received from the *Secure Context Service*, to determine the preferred device via which to notify the user. Once a device is chosen, the notification controller sends the message to the appropriate gateway for dispatch. The gateways have the responsibility of transforming the message based upon the capabilities of the intended device (if necessary) and then dispatching it for subsequent delivery. Each gateway is responsible for dispatching messages to devices of a particular type. For example, the *WAP Gateway* is responsible for delivering messages to WAP-enabled devices.

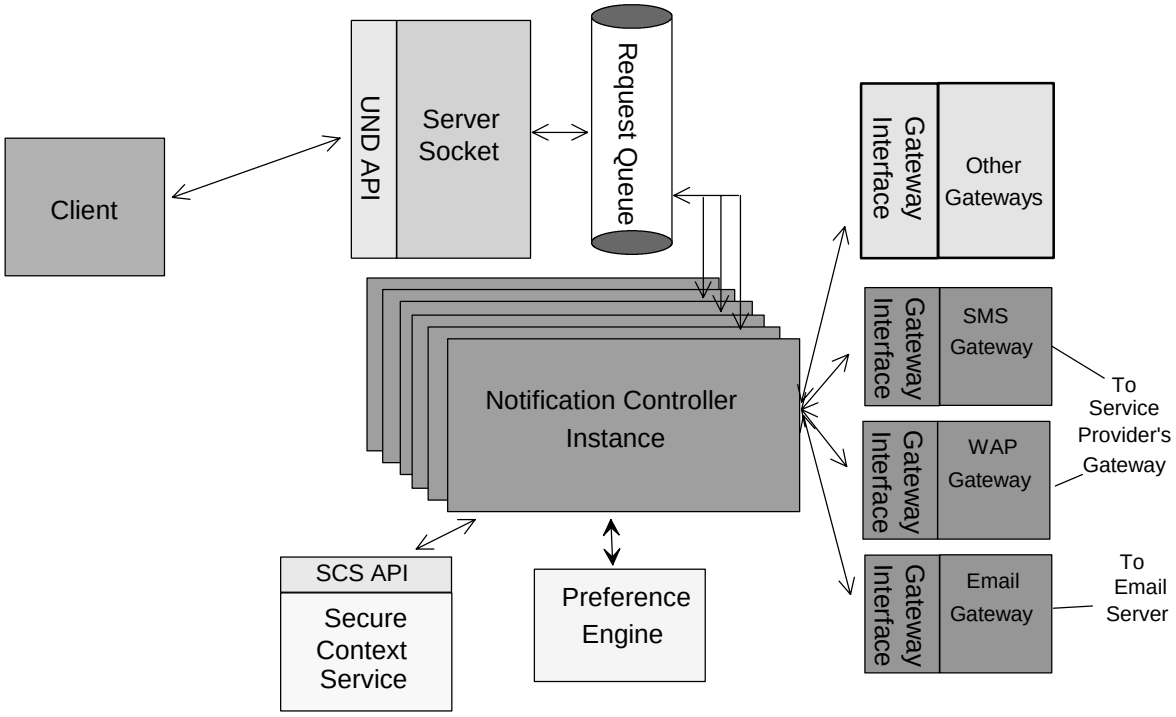

Figure 3: Architecture of the Universal Notification Dispatcher

The UND is designed to support new, unanticipated devices as they are released into the marketplace. The common gateway interface is the key to this extensibility. To support a new device, an outgoing gateway that uses the common gateway interface must be implemented. In addition, configuration parameters must be refreshed to allow the UND to dispatch messages intended for the new device type to the newly implemented gateway.

3.3 Secure Context Service

A context service provides standardized support to context-aware applications. Applications interact with the context service to obtain required information without worrying about the details of context management. We call our context service the Secure Context Service (SCS) because one of our main design goals is to protect the privacy of context subjects.

The overall SCS architecture is shown in Figure 4. It consists of a mediator, a configurable set of context drivers, and a collection of utility components. In addition, there are two programming interfaces: the SCS API and the Context Driver Interface. The SCS API allows applications to submit requests. The Context Driver Interface, used internally, allows the mediator to communicate with the various context drivers.

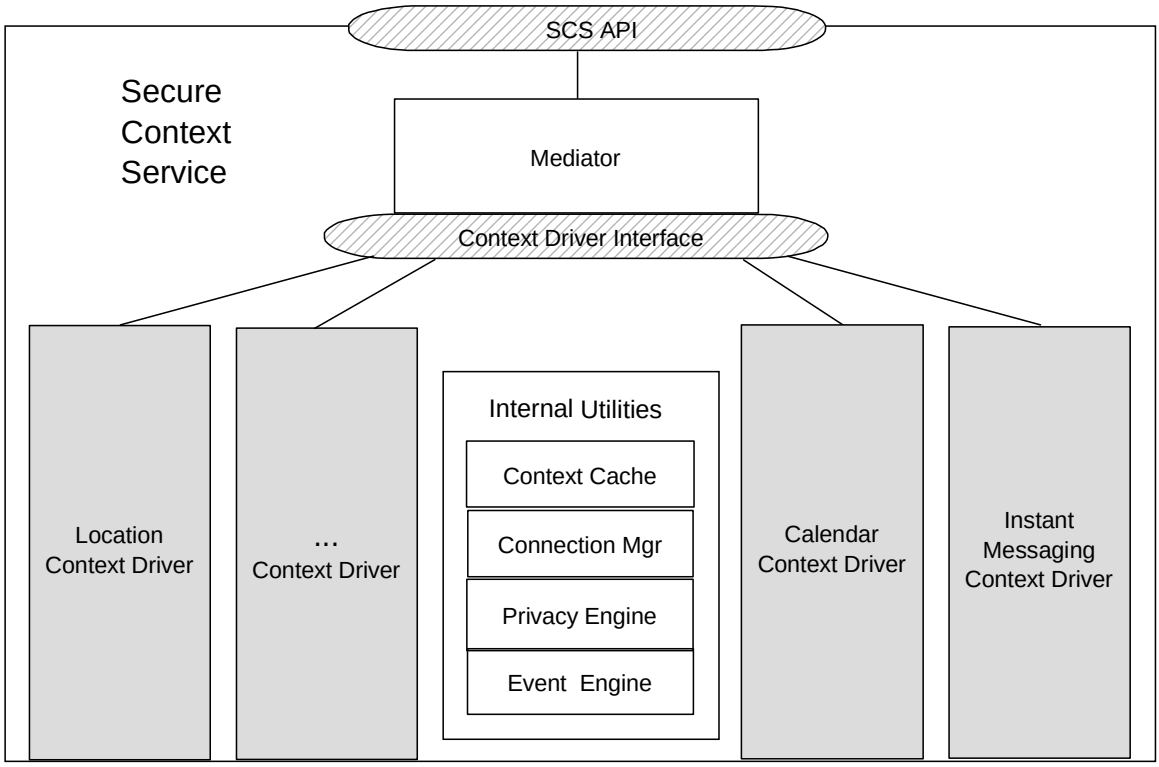

Figure 4. Architecture of the Secure Context Service.

Applications request context information through the SCS API. The API uses a forms metaphor: An application partially fills out a form, identifies the requested form fields, and optionally specifies the desired quality of information such as freshness and confidence; the SCS in turn responds with the forms that match the application specification. An application interacts with the SCS in one of three ways. First, it may issue a synchronous context query. Second, it may request a one-time notification when a particular context event occurs. Third, it may subscribe to a context condition so that it will be notified whenever the specified condition is satisfied.

The mediator dispatches application requests to the appropriate context drivers, through the uniform Context Driver Interface. Each context driver handles one type of context information and encapsulates the details of interaction with the context source. A context driver may *pull* information from context sources, either periodically or on demand. Alternatively, it may simply allow the context source to asynchronously *push* updated information.

Four utility components are available to context drivers: a context cache, a connection manager, a privacy engine and an event engine. The context cache retains recently accessed context information in main memory for performance reasons. The connection manager maintains persistent connections with various context sources to minimize the costs of constantly reestablishing connections with the same source; it is meant to be used by pull-based context drivers. The privacy engine authorizes access to context information based on policies defined by

individual information owners, again using role-based access control. The event engine matches context events with registered application interests.

Figure 4 shows a number of context drivers. The location context driver pulls a user's location information from a variety of sources. The other two drivers are push-based: the calendar context driver derives information such as a user's current activity and contact means from his or her calendar entries; the instant messaging context driver provides information on a user's instant messaging online status.

4 Current status and future plans

We have implemented a substantial amount of the function envisioned in the scope of the Intelligent Notification System. In fact, our system is driving the development of a product. In this section, we discuss the current status and future plans for each component.

The Trigger Management Service supports simple triggers with a single content source and a simple SQL match condition on the content. The service allows persistence of matched content to be retrieved later by the user. The TMS API is, however, extensible to multiple content sources as well as complex filters on those sources. Future versions of TMS will support such operations. We plan to support additional data sources for programmable aggregation. The Trigger Management Service is a first step towards a system addressing the many challenges in distributed aggregation [Coh01].

The Universal Notification Dispatcher allows for messages to be sent out using a wide variety of physical means including regular telephone, fax, email, instant messaging, SMS, and WAP. The service also supports notification transcoding which appropriately tailors the outgoing notification to the destination device characteristics. We want to extend the notification dispatcher to scalably support group notifications and to support a wider range of notification mechanisms.

The Secure Context Service currently supports both synchronous queries and asynchronous one-time notifications. It incorporates context from two different sources: the user's calendar and instant messaging status. We are in the process of incorporating additional context sources, such as various forms of location, and of supporting continuous monitoring of interesting context events. We would like to explore the notion of persistent context by detecting long-term trends and patterns from historical user context.

5 Related work

An alternative model for monitoring changing data is the *continuous query*, which was introduced in the Tapestry system [Ter92] and also used by systems such as OpenCQ [Liu99] and COUGAR [Bon00]. In this model, a stream of incoming data values is treated as a sequence of rows in a constantly growing relational-database table. The addition of new rows satisfying a particular query triggers some programmed action. In contrast, the Trigger Management System does not retain a full history of past events. Instead, our system supports triggers that retain just enough state about previous events to recognize when a situation of interest to the end user has

occurred. In addition, unlike a continuous query which specifies a static matching criterion, a trigger can adapt its matching criteria based on past history: A given incoming data value might trigger a notification given one history of past events, but not with another.

Active databases [McC89] recognize patterns of incoming events and react to recognized patterns by executing *event-condition-action rules*. Ode [Geh92], Snoop [Cha94], and SAMOS [Gat94] are examples of such rule-based active databases. Similarly, Amit middleware [Adi00] has its own XML-based language for defining patterns of events as *situations* and reacting when a particular situation is detected. In contrast to systems that define their own rule languages, the Trigger Management Service has application triggers programmed in Java. Although triggers can perform arbitrary actions, TMS makes it particularly convenient to react to situations by issuing notifications. Further, our support for persistent retention of content associated with an event is beyond the scope of these other systems.

The goal of the Mobile People Architecture project at Stanford University is to make people, rather than devices, the addressable units in the network [Man99]. To that end, they have proposed a system and have built a prototype of that system that allows people to receive communications through any network, device, or application they choose. Their system includes a tracking agent that monitors the network, device, or application users are actively using. This tracking agent corresponds roughly to one of our context drivers. Unlike their system, our design is not application specific.

The Universal Inbox project at UC Berkeley includes a personal activity coordinator that tracks the activities of users [Ram00]. This information includes a user's location, the status of a user's device, who the user is currently talking to (as well as the importance of that other person). Again, our system offers a more general context service.

The Cricket Location-Support System [Pri00] provides support for mobile devices to determine their own location within a building. Cricket explicitly does not use location tracking, purportedly to "address" the privacy problem inherent in such approaches. By doing so, however, Cricket precludes certain types of applications unless an application on the mobile device makes its location known externally. The existence of such an application lands Cricket back at square one with respect to privacy. In contrast, the Secure Context Service addresses the privacy question head on by giving users control over who can access their context information. Further, our context service supports a general notion of context, not just location.

Hull and his colleagues describe a system that, like SCS, is intended to handle many diverse sources of contextual information, though the only source with which they have experience is location data from their custom Pinger device [Hul97]. Because their system is intended to serve just a single individual using a wearable computer, privacy was not considered a design goal and was not explicitly addressed.

The TEA and Mediacup projects [Gel00] explore an architecture for obtaining and using context in everyday devices. In contrast to many context-based projects, context information in these projects is sensed on the devices themselves. The Mediacup broadcasts the context it detects. These broadcasts could be monitored and stored in a system such as ours. The TEA phone demonstrates a novel use for context information. It shows how context can be exploited to give users, in this case the caller, more information about the current activities of the person

they are trying to reach. Though the TEA phone uses self-sensed context, similar functionality could be provided through the use of our system. Neither of these projects consider the problem of privacy – in both cases, context information is treated as openly available and no restrictions are placed upon its dissemination.

6 Conclusions

The attention of the user is fast becoming the most precious resource on the Internet. The Intelligent Notification System helps the user manage the growing volumes of available information in two ways: First, the system manages triggers that monitor the flow of voluminous information and notifies the user only when situations of interest have arisen. Second, the system provides context-based universal notification to allow the user to be notified in the mode most appropriate to the user's current context and preferences. Our extensible architecture addresses the wide variety of existing and yet-to-emerge mobile information devices. We turn the information glut to our advantage by analyzing data about the user's context and using it to deliver information in the most effective way, while respecting and safeguarding the user's privacy through role-based access control. The Intelligent Notification System turns the monitoring of Internet data from a time-consuming focus of human attention to an automated activity. The user receives notifications at a rate and in a manner that lets him concentrate on his primary tasks.

References

[Adi00] Adi, A.; Botzer, D.; Etzion, O.; Yatzkar-Haham; T. Push technology personalization through event correlation. In El Abbadi, A.; Brodie, M.L.; Chakravarthy, S.; Dayal, U.; Kamel, N.; Schlageter, G.; Whang, K-Y., eds., *VLDB 2000, Proceedings of 26th International Conference on Very Large Data Bases, September 10-14, 2000, Cairo, Egypt*, Morgan Kaufmann, San Fransisco, 2000, 643-645.

[Agu99] Aguilera, M. K.;.Strom, R. E.; Sturman, D. C.; Astley, M.; Chandra, T. D.. Matching events in a content-based subscription system. Proceedings of the Eighteenth Annual ACM Symposium on Principles of Distributed Computing, May 4-6, 1999, Atlanta, Georgia, 53-61.

[Bon00] Bonnet, P.; Gehrke, J.; Seshadri, P. Querying the physical world. *IEEE Personal Communications* **7**, No. 5 (October 2000), 10-15.

[Cha94] Chakravarthy, S.; Krishnaprasad, V. Anwar, E.; Kim,S.-K. Composite events for active databases: semantics, contexts and detection. In Jorge B. Bocca, Matthias Jarke, and Carlo Zaniolo, eds., *VLDB '94, Proceedings of 20th International Conference on Very Large Data Bases, September 12-15, 1994, Santiago de Chile, Chile*, Morgan Kaufmann, San Fransisco, 1994, 606-617.

[Coh01] Cohen, N.H.; Purakayastha, A.; Turek, J.; Wong, L.; Yeh, D. Challenges in flexible aggregation of pervasive data. IBM Research Report RC 21942, January 23, 2001.

[Gat94] Gatziu, S.; Dittrich, K.R. Detecting composite events in active database systems using Petri nets. Proceedings, Fourth International Workshop on Research Issues in Data Engineering, Houston, Texas, February 14-15, 1994, 2-9.

[Geh92] Gehani, N. H.; Jagadish, H. V.; Shmueli, O. Event specification in an active object-oriented database. Proceedings of the 1992 ACM SIGMOD International Conference on Management of Data, San Diego, California, June 2-5, 1992, 81-90.

[Gel00] Gellersen, H.-W.; Schmidt, A.; Beigl, M. Adding Some Smartness to Devices and Everyday Things. In the Proceedings of the Third IEEE Workshop on Mobile Computing Systems and Applications (Monterey, CA, Dec. 2000), ACM, 3-10.

[Hul97] Hull, R.; Neaves, P.; Bedford-Roberts, J. Towards Situated Computing. In the Proceedings of the 1st International Conference on Wearable Computing (1997), IEEE, 146-153.

[Liu99] Liu, L.; Pu, C.; Tang, W. Continual queries for Internet scale event-driven information delivery. *IEEE Transactions on Knowledge and Data Engineering* **11**, No. 4 (July/August 1999), 610-628.

[Man99] Maniatis, P.; Roussopoulos, M.; Swierk, E.; Lai, K.; Appenzeller, G.; Zhao, X.; Baker, M. The Mobile People Architecture. *ACM Mobile Computing and Communications Review (MC2R)*, July 1999.

[McC89] McCarthy, D.; Dayal, U. The architecture of an active database management system. Proceedings of the 1989 ACM SIGMOD International Conference on Management of Data, Portland, Oregon, May 31 - June 2, 1989, 215-224.

[Pri00] Priyantha, N.; Chakraborty, A; Balakrishnan, H. The Cricket Location-Support System. In *Proceedings of the Sixth Annual International Conference on Mobile Computing and Networking* (Boston, MA, August 2000), ACM, 32-43.

[Ram00] Raman, B.; Katz, R.; Joseph, A. Universal Inbox: Providing Extensible Personal Mobility and Service Mobility in an Intergrated Communication Network. In *Proceedings of the Third IEEE Workshop on Mobile Computing Systems and Applications* (Monterey, CA, December 2000), IEEE Computer Society, 95-106.

[Ter92] Terry, D.; Goldberg, D.; Nichols, D.; Oki, B. Continuous queries over append-only databases. Proceedings of the 1992 ACM SIGMOD International Conference on Management of Data, San Diego, California, June 2-5, 1992, 321-330.