# The Design and Applications of a Context Service

**Hui Lei** *hlei@us.ibm.com*
**Daby M. Sow** *sowdaby@us.ibm.com*
**John S. Davis II** *davisjs@us.ibm.com*
**Guruduth Banavar** *banavar@us.ibm.com*
**Maria R. Ebling** *ebling@us.ibm.com*

IBM T. J Watson Research Center, Hawthorne, NY 10532, USA

*Context awareness enables applications to adapt themselves to their computing environment in order to better suit the needs of the user and the tasks. This paper describes a general middleware infrastructure for context collection and dissemination, realized as a Context Service. By way of two example applications, this paper also illustrates how context information provided by our context service can be exploited to enhance the user experience. These two applications are built upon the abstraction provided by the Context Service and thus help validate the design of this service. The first application, a Notification Dispatcher, uses context to route messages to the most appropriate communication device for a recipient. The second application, a context-aware content distribution system, uses context to predict users' access to web content, and uses these predictions to pre-process and pre-distribute content in order to reduce the access latency.*

## I. Introduction

Just as the discourse that surrounds a sentence can throw light on the meaning of that sentence, the environment in which a computation takes place can throw light on its meaning or on the intentions and needs of the user that the computation is supporting. We call the environment in which a computation takes place its context. We define context in the broadest possible sense, largely adopting Dey's definition [7].

Context awareness [13, 22] enhances existing applications and even enables a new class of applications in pervasive computing. These applications can help users navigate unfamiliar territory, find nearby restaurants, receive messages in the most useful and least intrusive manner, and the like. The main benefit of context awareness is that applications can adapt themselves to better suit the needs of the user and their task. Ultimately, context awareness has the potential to demand less from the user and significantly enhance the user experience by increasing productivity and satisfaction. To illustrate, consider two examples of how middleware can exploit context to improve the user experience.

The first example is a context-aware Notification Dispatcher, which dispatches messages to its subscribers based upon their delivery preferences. By using context information, such a dispatcher can direct notifications to the device that is most useful and least intrusive. For example, if the user is currently attending a meeting, the dispatcher might choose to deliver the message without an audible alert.

The second example is a context-aware content distribution system that uses context information to prepare content in anticipation of its use. One frequent source of user frustration while accessing web content is latency. Pervasive computing environments exacerbate latency problems for several reasons. Device heterogeneity often requires content transformation, which imposes an additional delay over that required to deliver the content to a standard desktop computer. Wireless network technologies typically add further delays due to lossiness and retransmission. Furthermore, traditional caching schemes have become less effective due to the dynamic nature of increased personalization and the reduction of access locality due to mobility. Existing content distribution systems [27] aim to reduce access latency by pre-fetching content close to the user's computing device. We are developing a context-aware extension of content distribution, which we call Pervasive Content Distribution. This system uses context to predict a user's access behavior, and reduces latency by pre-transcoding, preprocessing, and pre-distributing content to a location close to the user.

There are myriad applications that can use context information to enhance the user experience. To make such context information easily and widely available,

there is a need for context collection and dissemination at the infrastructure level. Many existing context-based applications [24, 4, 9] have built custom infrastructures for their context needs, and are typically limited in the kinds of context used, (e.g., location awareness.) New applications are hard to develop, because application developers may spend significant programming effort on deriving and managing context data. In addition, we are not aware of any prior work that has addressed the issue of giving users control over the release of their own context information [12]. To address these issues, we are developing a context service as an integral part of the pervasive computing infrastructure.

The main contribution of this work is a service-based approach to enable context awareness, which results in a higher level of abstraction for application developers. Applications can interact with the Context Service to obtain the information they need, without worrying about the details of context management. Furthermore, the costs associated with introducing new context sources can be amortized across many applications. Our context service provides the additional benefits of an extensible architecture to support heterogeneous context sources and integrated support for privacy. We also present in this paper two applications (that are themselves novel middleware systems) that leverage and validate our context service: a notification dispatching system and a context-aware content distribution middleware.

This paper presents the salient aspects of our context service in Section II and describes the two applications mentioned above, notification dispatching in Section III.A and Pervasive Content Distribution in Section III.B. Related work to this paper is then provided in Section IV before presenting a few concluding remarks in Section V.

## II. Context Service

Dey [7] defines context as ”any information that can be used to characterize the situation of an entity. An entity is a person, place, or object that is considered relevant to the interaction between a user and an application, including the user and applications themselves.” We have adopted this definition of context in this work. We do, however, extend this notion of context to include persistent context information as well as the transient information used in his discussion. Whereas transient context reflects the environment at a single point of time, persistent context represents a recurrent pattern of transient context. For example, the fact that Alice is attending a meeting that started at 9:30 AM is *transient* context and the fact that she normally attends a meeting at 9:30 AM each morning is *persistent* context. Although persistent context is implicitly covered by Dey’s broad definition, it is not explicitly considered in his discussion.

A context service collects and maintains context information about numerous *subjects*. Subjects may be users or objects (e.g., equipment or packages). The Context Service obtains information about subjects from a variety of context *sources*. Sources are typically third parties that collect and provide information. For example, information about subjects, Alice and her computer, may be provided by the source, ACME Corp., her employer. Or, information about Alice’s location may be provided by her mobile phone service provider. *Clients* may request context information about one or more subjects. Clients may be other users (e.g., a secretary, colleague, or spouse) or programs (e.g., a service or application). Clients may query for the current context information or may submit a request to be notified when a particular condition is met. Access to context information is controlled by a *controller*, which may be either the subject about whom the information pertains or the owner of the object about which the information pertains.

In this section, we present the design rationale for our context service, focusing on those aspects unique to context services. We then present the architecture of our context service, including its programming model and its support for privacy management.

### II.A. Design Considerations

In this section, we consider three issues critically relevant to the design of a context service. First, context services handle extremely sensitive data and, consequently, must protect the privacy of their subjects to the greatest extent possible. Second, context services report data collected in the real world and should therefore offer quality of information measures along with that data. Third, context services cannot possibly anticipate every possible source of context, and so must be easily extensible to cope with new and unexpected forms of context. Issues common to distributed systems in general (such as performance, availability, and scalability) and issues related to the economics of context collection and use (such as relevance) are outside the scope of this discussion.

### *II.A.1. Privacy*

People's reaction [6] to previous research in this area, as well as emerging standards [1] to protect privacy, lead us to the conclusion that any context service that does not respect the privacy rights of its subscribers would not be trusted. Our guiding principle in designing the privacy management system of our context service is that users should maintain control of their context information. Fundamentally, this principle implies that the Context Service cannot release context information without validating the permissibility of the request from the controller of that context information. The implication of this decision is a potential latency cost for each request for context information.

In addition to respecting the privacy of context subjects, context services should also consider the privacy of context sources. The identity of context sources should not be released without their explicit permission. This raises the issue of what happens when the privacy policy of the context subject and that of the source conflict. In such instances, we propose to prioritize the privacy policy of subjects above that of the context source. This decision implies that the subject of the context data can learn the identity of the source, but that other requesters cannot.

Context services must also consider whether to give subjects control over what context data gets collected about them. As a service intended to support applications running on behalf of the user, our opinion is that the subjects should control which sources collect and supply data about them.

### *II.A.2. Quality of Information*

Context information often involves real world entities. Thus, it makes sense to measure the quality of context information (QoI), or the extent to which the data corresponds to the real world [3]. The quality of context information can vary, perhaps substantially, depending on the context source. A context service must therefore have a clear interface with context sources to allow them to express inaccuracies and uncertainty in the data, via QoI metrics such as freshness, confidence, error, and the like.

A context source that aggregates different sources of data must address the issue of resolving the different levels of QoI from its sources and providing an aggregate QoI. For example, a context source providing wind chill information may use two other context sources providing current temperature and wind velocity, with different QoI measurements. The aggregated context source must calculate a reasonable aggregate QoI based upon standard scientific practice.

It is important to note that the goal of our context service is not to resolve ambiguities or inconsistencies in the context information provided by sources, but rather to pass along QoI data from sources to clients. For example, a context service may obtain GPS location data, 802.11 location data, and cellular location data from different sources. These may be consistent, but are likely to be specified in different ways (e.g., latitude-longitude-altitude, access point, cellular tower, zip code, etc.). In this case, the context service returns the requested location data along with QoI information, and allows the caller to resolve the quandary. If aggregate data is desired, an aggregate context source must be configured to obtain the raw location information, possibly via the Context Service, from various sources and determine the best location to report.

### *II.A.3. Extensibility*

Context-aware computing is relatively new to the computing world. Research is ongoing and products are still rare. Consequently, we expect new and unanticipated sources of context information to be developed in the coming years. A general context service, intended to support diverse applications and services, will need to accommodate these new context sources easily.

Further, context services must be flexible about the interaction method used with the context source. For example, the service might need to pull data from one source and accept a push of data from another source. Similarly, it might need to use different protocols to interact with different sources (e.g., Java RMI to access context from one source and SOAP to access context from another source). The service must be capable of coping with a variety of interaction methods.

## II.B. Architecture Overview

The overall architecture of our Context Service is shown in Figure 1. It consists of a dispatcher, a configurable set of context drivers, and a collection of utility components. In addition, there are three programmatic interfaces: the Client API, the Context Push Interface, and the internal Context Driver Interface. The dispatcher routes application requests to the appropriate context drivers, through the Context Driver Interface. Each context driver handles one type of context information and encapsulates the details of interaction with context sources for that infor-

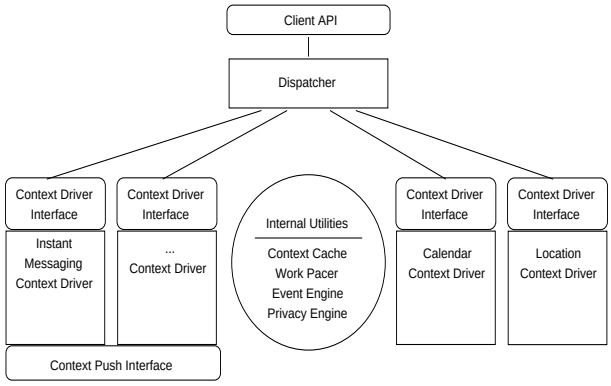


Figure 1: Architecture of the Context Service.

mation. A context driver may either pull information from context sources or let context sources push updated information to it via the Context Push Interface. Context drivers can make use of four utility components: a context cache, a work pacer, an event engine, and a privacy engine. The context cache retains recently accessed context information in main memory for performance reasons. The work pacer schedules the pulling of information from context sources to avoid overloading the context sources. The event engine matches context events with registered application interests (callback requests). The Privacy Engine controls access to context information based on policies defined by context controllers.

Our current design is targeted at relatively contained domains such as smart homes, smart offices, and enterprises. To support context awareness in a larger scale, we envision that multiple instances of the Context Service will be deployed, one for each local domain. A service discovery and inspection mechanism will allow applications to dynamically bind to the right service instance. Mechanisms for advertising and querying about server instances are topics of our on-going research.

In the remainder of this section, we examine more closely the programming model, the context drivers, and the privacy control mechanism.

### *II.B.1. Programming Model*

The programming model of our Context Service supports many types of context information based on a forms metaphor. Each context *form* describes one type of context information and embodies a collection of related context attributes (i.e., form fields). For example, one form describes current user activity, contact means, and location, while another form describes users' online presence and availability. A form is further associated with meta-data on the quality of information contained in the form. Currently two QoI metrics are used: timestamp for indicating data freshness, and confidence as asserted by the data source.

To obtain context information, a client partially fills out a form template, selects the requested fields, and specifies the desired QoI, if any. The Context Service then responds with the form data that match the application's query template. Applications may query for the current context information synchronously or may submit a callback request to be notified asynchronously when a particular condition is met. A callback request can additionally specify the number of callbacks desired and the frequency of notifications. Both queries and callback requests are subject to an application-supplied or default expiration time.

A context source may push new or updated context information into the Context Service by passing the latter a context form filled with the latest information. The context source also indicates whether the update is partial or complete. A partial update will substitute information in the filled form fields only, keeping information that arrived prior to the current update that corresponds to the unfilled fields. In comparison, a complete update will nullify information that corresponds to the unfilled form fields.

When designing the programming model, we realized that, although context data are heterogenous, the ways in which applications would like to obtain them are often similar. Therefore, our API encompasses only a small number of operations such as queries and callbacks. Data heterogeneity is accommodated by allowing for an extensible set of form types on which the operations may be performed. We have also been investigating how to augment the expressiveness of our programming model to allow applications to aggregate context data from multiple sources [5].

A forms-based programming model offers a number of benefits. First, the forms metaphor provides a uniform abstraction for heterogeneous context data. Second, new context sources can be easily integrated into our system by defining a new context form and writing a context driver that handles the form. Third, a forms data model offers flexibility in specifying the selection condition for the desired context.

### *II.B.2. Context Drivers*

A context driver is a "plug-in" component that encapsulates the details of interaction with context sources to gather and distribute a specific type of context information. The context driver implements the Context Driver Interface and uses internal utilities such as the context cache, work pacer, event engine, and Privacy

Engine.

The Context Driver Interface provides a uniform abstraction for context drivers. The dispatcher interacts with each of the context drivers through this interface, without having to understand the details of interaction between context drivers and context sources. This allows new context drivers to be easily configured into the Context Service and existing context drivers to be modified without changing the rest of the Context Service. The Context Driver Interface declares methods for starting and terminating the driver, as well as methods for determining the driver capability in terms of what context forms it can handle. Furthermore, it declares methods for handling context queries and callback requests that arrive via the API as well as methods for handling context update requests that arrive via the Context Push Interface.

A context driver may interact with a context source based on either a push approach or a pull approach. Furthermore, a context driver may treat the rest of the Context Service as a context source and, via the client API, acquire information supplied by other context drivers. This effectively allows raw context data to be filtered, combined, and aggregated so that higher-level context information may be derived. Such a mechanism is important to support the notion of persistent context.

The current implementation of the Context Service includes context drivers for instant messaging (IM) online status, calendar events, user location, and desktop activity.

- The IM context driver gathers information on whether and how long a user has been active on instant messaging. It receives such information from external IM status feeders. Our implementation features a Sametime feeder that monitors user presence in Lotus Sametime [14], using the Sametime Toolkit v1.5, and pushes the presence information to the Context Service.

- The calendar event context driver provides information on a user's activity and contact means, as derived from the user's calendar entries. Currently, Lotus Notes is the only supported calendar system. A Notes calendar agent downloads calendar entries into a raw calendar table. This raw calendar data are then mined to derive higher-level calendar events. The calendar events are stored in a calendar context table accessible to the calendar context driver. Both the raw calendar table and the calendar context table are stored in a DB2 database accessed using JDBC.

- The location context driver pulls and reports a user's location information from various sources. It is designed to support multiple types of location sources: cellular carriers, GPS devices, RIM Blackberry devices, and 802.11 networks. It resolves conflicts and ambiguities in data from different sources using a mechanism developed by our colleagues J. Myllymaki and S. Edlund [17].

- The desktop context driver obtains data from an agent running on the user's computer. This agent monitors what application the user is actively using. It reports that data back to the Context Service. It uses XML to specify which applications are of interest and only monitors those applications specified.

### *II.B.3. Privacy Model*

The Privacy Engine is used to specify, store, and retrieve privacy policies. Each privacy policy specifies whether a proxy *application* acting on behalf of a *requestor* is granted access to *information* about a particular *subject*. An example privacy policy might state that Steve can access Paul's location via LocationFinder. In such a policy, Steve is the requestor, Paul is the subject, location is the information and LocationFinder is the application. Note that we assume a closed, secure system in which the identity of all participants is known by the system.

A key feature of the Privacy Engine is its ability to specify policies for organizations (e.g., business enterprises) as well as individuals. For example, an organization-level policy might state that in the distribution center group, the employees are able to view the location of every truck. Such organization-level policies may typically be specified by system administrators.

The privacy protection mechanism used by the Privacy Engine is based upon Role Based Access Control (RBAC) [23]. We have chosen RBAC because studies [8] indicate that RBAC reduces the cost of administering security policies. RBAC separates the association between users and groups from the associations between groups and privileges. Since the number of groups is typically much smaller than the number of users, RBAC reduces the number of associations that must be managed in most cases and hence the administrative cost. In addition, because subject-based policies align closely with existing business practices and can be expressed naturally in terms of roles, RBAC may make the specification of security policies less prone to human error.

## III. Applications of Context

In this section we present two examples of middleware that exploit context to enhance the user experience. The first example, a Notification Dispatcher, makes relatively simple use of "raw" context to dispatch messages to recipients. The second example, a pervasive content delivery system, derives aggregate context from raw context via learning techniques, and uses it to preprocess and pre-distribute web content.

### III.A. Notification Dispatcher

The Notification Dispatcher is a context-aware system that routes messages to one of several possible communication devices owned by message recipients based on the context of the recipient and the urgency of the message (FYI, Normal or Urgent). Recipients register their communication devices, specify groups of senders, and declare which devices members of a particular group are allowed to send messages to. For example, a subscriber might say that all FYI messages received from a sender in the "golf buddies" group should be routed to both the user's cell phone and email account. The recipient may also specify a preference declaring inappropriate times in which to send messages.

To use the system, a sender addresses a message to a particular recipient and declares the urgency of the message (FYI, Normal, Urgent). Upon receiving the message, the Notification Dispatcher considers all the recipient-specified rules to determine the appropriate device. If the current time is inappropriate, the Notification Dispatcher delays the message so that it can be sent at a later time.

An illustration of the Notification Dispatcher is shown in Figure 2. Clients submit synchronous and asynchronous requests to the Notification Dispatcher. The requests are queued before being allocated to a Notification Controller (NC) instance. The NC instance first queries the Directory Services Engine to determine what user groups are associated with the message recipient and whether the message sender is a member of any of the groups. Each group specifies the recipient devices, if any, that a message should be directed to from senders contained in the group. Once the recipient group and devices are determined the NS instance queries the Context Service (CS) with message context to determine additional delivery specifications. The Context Service may have several context drivers installed but the Services Preference Manager (SPM) plays an important role with respect to the Notification Dispatcher. The Context Service uses SPM to determine any constraints specified by the recipient such as times at which no messages should be sent. Messages that cannot be sent at a particular time may be delayed for up to 24 hours in our current implementation. Once a message is ready for delivery, the NC instance uses the appropriate gateway as shown on the right of the figure.

Today, the Notification Dispatcher relies on two primary forms of context: instant messaging online status and calendar events. It uses the online status in determining the availability of the Sametime Instant Messaging system as a means of delivering the message. It uses calendar information to determine the recipient's current calendar entry so that it may check the recipient's preferences for receiving messages during that type of calendar event. For example, if the recipient is currently in a meeting, the Notification Dispatcher uses the recipient's notification preferences for meeting events. We are also considering how location could be used. Past research [28] has used telephones near the recipient, not necessarily "belonging" to the recipient. An alternative that would avoid the usability problems encountered in those projects would be to use the recipient's stationary phone for notification only if the recipient is currently located near that phone.

The Notification Dispatcher has been used primarily in research prototypes, both within our own research group and by other research groups at IBM. One project [2] uses the Notification Dispatcher to inform users of registered events (e.g., when a stock hits $150 or when a news service publishes an article with a particular topic). Another ongoing project is using the Notification Dispatcher within its demonstration of a location-based telematics service to inform its users when an event occurs (e.g., a preferred restaurant is nearby). We are currently planning to deploy a production version of the system within the lab.

The Notification Dispatcher application evolved concurrently with our Context Service infrastructure. This joint development offered important insight into the design of the Context Service. First and foremost, the Notification Dispatcher validated our design of the Privacy Engine. The incorporation of the Privacy Engine into the Context Service freed the Notification Dispatcher's design from concerns about the accessibility of context information. Instead, the Notification Dispatcher is simply able to request context information from the Context Service and context is returned only if the Context Service's Privacy Engine grants access. The result of this separation of concerns is that as the Context Service's Privacy Engine evolved from

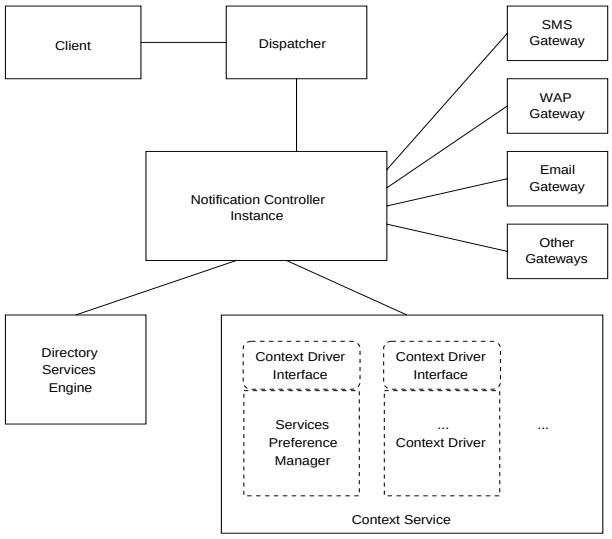


Figure 2: Architecture of the Notification Dispatcher.

a rudimentary access control mechanism to a refined role-based access control system. The process of this evolution did not require changes to the architecture of the Notification Dispatcher application.

In addition to leveraging the Privacy Engine, the Notification Dispatcher took advantage of the extensibility offered by the Context Service. While the initial version of the Notification Dispatcher did not require the context offered by the Services Preference Manager (SPM), we recognized the need for such context in the second generation of the Notification Dispatcher. Fortunately, the extensibility of the Context Service made the task of adding an SPM context driver very easy with little impact to the Notification Dispatcher's design.

The Quality of Information (QoI) features of the Context Service were validated by the Notification Dispatcher in its use of location context. As mentioned in Section II.B.2 the results of multiple location sources are resolved by the Context Service based on the respective QoI metrics. This facility is completely hidden from the Notification Dispatcher so that the ND is not burdened by this resolution process.

### III.B. Pervasive Content Distribution

The Pervasive Content Distribution (PCD) system is a context-aware middleware system that aims to reduce latency and thus improve user experience while accessing content from pervasive devices. Access to content in pervasive environments is affected by several factors above and beyond those in traditional networked environments. These include the characteristics of the last wireless hop (i.e., bit error rates, high packet loss probabilities, delays, etc) that increase the end-to-end perceived latency. Mobility of users reduces the access locality in caching proxies. Further, the heterogeneity of client devices introduces additional transcoding delays.

The approach used in PCD to reduce latency is to pre-process and pre-distribute content to a network location close to the user device. The preprocessing and pre-distribution is driven by a prediction of the user's location and of the task that the user might perform. For example, we can use past history to determine the likelihood that a user will access particular web applications at particular times. We can also use context information, such as location and calendar information, to determine the kinds of tasks and activities the user is most likely to perform.

Once the access pattern of a user is predicted, the PCD system translates the pattern into a set of instructions to an underlying content transformation and distribution system. The content transformation system pre-transforms the content to suit the user's device characteristics such as its markup language, its bandwidth and multimedia capability, and so forth. The content distribution system in essence replicates the content to a proxy server close to the user's location. To do so, the PCD system is build on top of widely available content transformation and distribution middleware [11], consisting of transcoding proxies performing the content transformation and proxy servers at the edge of the network, which we call *edge servers*.

There are three types of instructions produced by the PCD system.

- *Pre-fetching and pre-processing directives:* These directives instruct edge servers to fetch and prepare content ahead of time, in anticipation of client requests. Preparation operations are typically performed by a transcoding proxy [11] that adapts the content to the capabilities of client devices, or by a portal server [11] that generates and aggregates content from different sources according to user preferences. Clearly, in order to perform these fetching and processing operations ahead of time, the requested applications must be static, or dynamic with a relatively small rate of change.

- *Subscription directives:* For dynamic content changing at the source (e.g., stock quotes), the system produces subscription directives. These directives set up edge server subscriptions for content on behalf of users. Such subscriptions define channels between subscribers and providers. Newly produced content is pushed

by content providers on these channels to edge servers in anticipation of user requests.

- *Application off-loading directives:* For dynamic content that also depends on user inputs or query parameters that cannot be predicted (e.g., queries for the state of services and devices in a private home network), the system generates application off-loading directives. In this case, an edge server running close to the user attempts to bring the entire application to the edge of the network and run it there. This may not be possible in all cases (e.g., off-loading large databases).

In addition to context sources, the PCD system uses metadata consisting of device profiles and application models. Device profiles include standardized descriptions of the computational capabilities of all the devices encountered by the system [19]. Typical descriptions include attributes such as screen size, color depth, storage capacity, and CPU speed. The application model represents the application structure. It consists of a graph that captures the relationship between interface components (e.g., HTML or JSPs) as well as the connection to back-end services (e.g., Enterprise Java Beans or Web Services). Application models are typically provided by application developers.

The PCD system requires two kinds of context drivers from the Context Service described in Section II:

- *Transient context drivers* This set includes many of the drivers discussed in Section II.B.2, such as the Location Context Driver and the Calendar Event Context Driver. It also includes a new context driver, the Access History Context Driver that monitors the actions of a user on each of her devices.

- *Persistent context drivers* As mentioned in Section II, persistent context is a recurrent pattern of transient context. Persistent context used by the PCD middleware is either derived from transient context history logs or obtained directly from users. In the former case, we are developing data mining techniques to extract meaningful information from the large amount of data available in these logs. Accordingly, the result of such mining operations are stored and provided to the PCD system by an Access Pattern Driver. In the latter case, users are given the ability to tell the system how they expect content to be delivered to them by specifying their preferences via a User Preferences Context Driver.

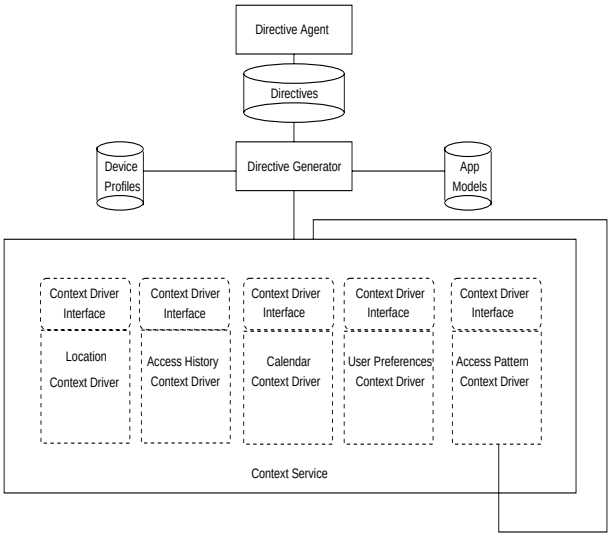


Figure 3: Architecture of the Pervasive Content Distribution Middleware.

The overall PCD architecture is shown in Figure 3. The Access Pattern Driver is an aggregate context driver as described in Section II.B.2, that obtains transient context information via the Context Service Client API. The directive generator generates content distribution directives using general context obtained via the Context Service Client API, device profiles obtained from the device profile repository and application models obtained from the application models repository. The resulting directives are stored in the directives repository. The directive agent reads these directives and transmits them to different parts of the end-to-end delivery system, including caching proxies and transcoding proxies. The PCD system is being built after the original development of the Context Service.

PCD leverages the extensibility of the Context Service. To build the PCD system, we are extending the Context Service with the Access Pattern Context Driver. This new driver focuses only on the mining operations involved in the access pattern generation and leaves the task of distributing and managing these patterns to the Context Service framework. The data generated by this new driver are made available to the Directory Generator through the Context Service Client API. Furthermore, the Context Service shields the PCD middleware from privacy issues.

## IV. Related Work

Researchers have long realized the need to separate context-aware applications and the underlying context gathering and distribution infrastructure. One early effort is the mobile application customization system

developed by Schilit et al [25]. Their system consists of a collection of environment servers, each providing information on one aspect of the context (e.g., a room, a meeting, a user, a work group). Context data from each server is organized as a set of name-value pairs. Applications may request the entire set of name-value pairs from a server or subscribe to any value changes in the set. The authors did not consider the interfacing between their system and context sources. Nor do they address the problem of privacy; context information is treated as openly available and no restrictions are placed upon its dissemination.

Spreitzer and Theimer discuss architecture alternatives for disseminating location information [26]. These include user agents which maintain individual users' location information and may be queried directly by applications, the Location Query Service that brokers the user agents within a region, and multicast groups, members of which may listen to application queries and answer the queries if they have a match. Most remarkably, the authors identify the importance of information accuracy and user privacy in a location-aware system.

Hull and his colleagues describe a system that, like ours, 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 [10]. Because their system is intended to serve just a single individual using a wearable computer, privacy was not considered a design goal and is not explicitly addressed.

A widely cited project is Georgia Tech's Context Toolkit [22], which comprises building blocks for context-aware applications. These building blocks are context widgets that abstract context sensing and acquisition, context interpreters that map context between different semantic levels, context aggregators that group related context for particular entities, and discoverers that help applications locate context components of interests. Although the toolkit shares our design goal of providing infrastructure support for context awareness, it provides a lower-level programming abstraction. Application developers are burdened to discover the building blocks, and are exposed to differences in context derivation (whether they are sensed, interpreted or aggregated). Furthermore, since the context components are treated as separate entities, it is not clear how the functionality of dynamic data composition from multiple components may be integrated into the toolkit metaphor.

In ubiquitous computing, privacy has been identified for some time as one of the key problems that must be addressed. Despite its importance, only a small amount of work has so far been performed [12].

Like our Notification Dispatcher, a number of projects also address the issue of user mobility to support unified messaging. For instance, the Universal Inbox project at UC berkeley presents an architecture that enables redirection of in-coming communication based on user preference profiles [21]. Another effort is the Mobile People Architecture project at Stanford University, which uses a personal proxy to maintain person-to-person reachability [16]. The personal proxy makes routing decisions based on registration of the user's connectivity state. In comparison, our Notification Dispatcher is empowered by a variety of dynamic context information provided by the Context Service.

The novelty of the Notification Dispatcher with respect to both of these projects is that ND uses a very broad notion of context that goes beyond the location of the recipient or the type of communication device. In addition, the use of a context service enables ND to expect its use of context to be broadened even further in future versions.

The Cricket Location-Support System flips the responsibility for location awareness around [18], providing 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, our context service addresses the privacy question directly by giving users control over who can access their context information. Further, we support a general notion of context, not just location.

The general area of content distribution in traditional networked environments is a well-studied area [27, 20]. Typical content distribution systems use strategies such as caching and replication to optimize overall system and network performances (i.e., reduce average latency and bandwidth usage). These optimizations do not take into account any information about users. In contrast, the PCD system aims to exploit context information to address the specific characteristics of pervasive environments described earlier. Context-aware content distribution is a new research area where little work has been performed. The iValet [15] system also uses context information about users to predict from user's past interactions

where and how to send information. To describe the goal of this system Macskassy et. al [15] propose the metaphor of a valet or secretary assisting the end user while accessing information. In contrast to PCD, the iValet does not address issues of prefetching and caching of prefetched and pre-processed content at the edge of the network. iValet delivers application to client devices and is not transparent to end users.

## V. Conclusions

In this paper, we have described a middleware infrastructure that enables the collection of context information from a large number of disparate sources and the dissemination of that information to interested clients. In order for this middleware to be widely deployed, it needs to address the issues of privacy, extensibility, and quality of information, as we describe in the paper. To illustrate the utility of this infrastructure, we describe two middleware applications that exploit context information to enhance the user experience. One of them is a Notification Dispatcher that uses context to determine how best to notify recipients, and the other is a content distribution system that uses context to pre-process and pre-distribute content ahead of use. These applications exploited the key characteristics of the design of the Context Service: extensibility, privacy and Quality of Information. Both the Notification Dispatcher and the PCD middleware introduce new drivers for new types of context. They both rely on the Context Service to protect and manage user privacy issues. The Notification Dispatcher also leverages the support of Quality of Information by the Context Service.

As context-aware computing becomes more of a reality, context infrastructure such as the one we describe will play a key role in enabling the easy construction of applications. Individual applications can depend on this infrastructure to obtain diverse information about users without having to painstakingly access numerous context sources. Context sources, on the other hand, can easily make their context available to a wide range of applications by building context drivers and plugging it into the context infrastructure. Third parties will likely derive aggregate context from raw context and make that available as context drivers as well.

For this infrastructure to be widely used, there are thorny economic and legal issues that will also come into play. Ultimately, there will be at least five parties involved: the subject (e.g., Alice's mobile phone), the source (e.g., her wireless service provider), the Context Service provider (e.g., a new entity that hosts and maintains the Context Service), the controller (e.g., Alice) and the client (e.g., Bob). Legal and economic contracts between these parties must be developed to obtain the desired context information while honoring the privacy of both the subjects and the sources.

## References


[1] Platform for Privacy Preferences (P3P) Project. *http://www.w3.org/P3P*.

[2] V. Bazinette, N.H. Cohen, M.R. Ebling, G.D.H. Hunt, H. Lei, A. Purakayastha, G. Stewart, L. Wong, and D.L. Yeh. An Intelligent Notification System. *IBM Research Technical Report TR 22089*.

[3] P. Castro and R. Muntz. Managing Context Data for Smart Spaces. *IEEE Personal Communications*, pages 44–46, October, 2000.

[4] K. Cheverst, N. Davies, K. Mitchell, and A. Friday. Experiences of developing and deploying a context-aware tourist guide. *Proceedings of the sixth annual international conference on Mobile computing and networking*, August 2000.

[5] N. Cohen, H. Lei, P. Castro, J. Davis II, and A. Purakayastha. Composing Pervasive Data Using iql. *Fourth IEEE Workshop on Mobile Computing Systems and Applications, (Callicoon, NY)*, pages 94–104, June 2002.

[6] P. Coy. Big Brother Pinned to your Chest. *Business Week*, 3279, August 17, 1992.

[7] A. K. Dey. Understanding and Using Context. *Personal and Ubiquitous Computing*, 5(1):4–7, 2001.

[8] D.F Ferraiolo, D.M. Gilbert, and N. Lynch. An Examination of Federal and Commercial Access Control Policy Needs. *NIST-NCSC National Computer Security Conference, (Baltimore, MD)*, pages 107–116, September 1993.

[9] H. Gellersen, A. Schmidt, and M. Beigl. Adding context-awareness to mobile artifacts. *IEEE Workshop on Mobile Computing Systems and Applications (WMCSA2000), Monterey, CA*, December 2000.

[10] R. Hull, P. Neaves, and J. Bedford-Roberts. Towards situated computing. *1st International Symposium on Wearable Computers, (Cambridge, MA)*, Oct 1997.

[11] IBM. Websphere Product Family. *http://www.ibm.com/software/websevers/*.

[12] M. Langheinrich. Privacy by Design – Principles of Privacy-Aware Ubiquitous Systems. *Proceedings of Ubicomp 2001*, pages 273–291, 2001.

[13] H. Lieberman and T. Selker. Out of Context: Computer Systems That Adapt to, and Learn From, Context. *IBM Systems Journal*, 39:3, 39:4:617–632, 2000.

[14] IBM / Lotus. Lotus Sametime. *http://www.lotus.com/products/lotussametime.nsf/wdocs/homepage*.

[15] S.A. Macskassy, A.A. Dayanik, and H. Hirsh. Information Valets for Intelligent Information Access. *AAAI Spring Symposia Series on Adaptive User Interfaces, (AUI-2000)*, 2000.

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

[17] J. Myllymaki and S. Edlund. Location Aggregation from Multiple Sources. *International Conference on Mobile Data Management, (Singapore)*, Jan 2002.

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

[19] Composite Capabilities / Preferences Profiles. Requirements and Architecture. *http://www.w3.org/TR/2000/WD-CCPP-ra-20000721/*.

[20] M. Rabinovich and O. Spatscheck. *Web Caching and Replication*. Addison-Wesley, 2002.

[21] B. Raman, R. Katz, and A. Joseph. Universal Inbox: Providing Extensible Personal Mobility and Service Mobility in an Integrated Communication Network. *In Proceedings of the Third IEEE Workshop on Mobile Computing Systems and Applications (Monterey, CA)*, pages 95–106, Dec 2000.

[22] D. Salber, A.K. Dey, and G.D. Abowd. The Context Toolkit: Aiding the Development of Context-Enabled Applications. *In Proceedings of ACM CHI 99, Pittsburgh, PA*, pages 434–441, May, 1999.

[23] R.S. Sandhu, E.J. Coyne, H.L. Feinstein, and C.E. Youman. Role-based Access Control Models. *IEEE Computer*, 29:2:38–47, February, 1996.

[24] B. Schilit, N. Adams, and R. Want. Context-aware computing applications. *Workshop on Mobile Computing Systems and Applications*, December 8-9, 1994.

[25] B. Schilit, M. Theimer, and B. Welch. Customizing mobile applications. *USENIX Mobile and Location-Independent Computing Symposium, (Cambridge, MA)*, Aug 1993.

[26] M. Spreitzer and M. Theimer. Providing location information in a ubiquitous computing environment. *14th ACM Symposium on Operating System Principles, (Asheville, NC)*, Dec 1993.

[27] D. Verma. *Content Distribution Networks: An Engineering Approach*. Wiley Inc, 2001.

[28] R. Want, A. Hopper, V. Falcao, and J. Gibbons. The Active Badge Location System. *ACM Transactions on Information Systems*, 10:1:91–102, 1992.