Context Watcher – Sharing context information in everyday life

Johan Koolwaaij[1], Anthony Tarlano[2], Marko Luther[2],
Petteri Nurmi[3], Bernd Mrohs[4], Agathe Battestini[5] and Raju Vaidya[4]

[1] Telematica Instituut, PO Box 589, 7500 AN Enschede, The Netherlands,
koolwaaij@telin.nl
[2] DoCoMo Euro-Labs, Landsberger Strasse 312, 80687 Munich, Germany,
{tarlano,luther}@docomolab-euro.com
[3] Helsinki Inst. for Information Technology HIIT, PO Box 68, FI-00014 Helsinki, Finland,
petteri.nurmi@cs.helsinki.fi
[4] Fraunhofer Fokus, Kaiserin-Augusta-Alee 31, 10589 Berlin, Germany,
{bernd.mrohs,raju.vaidya}@fokus.fraunhofer.de
[5] Nokia Research Center, PO Box 407. FI-00045 Helsinki, Finland,
agathe.battestini@nokia.com

ABSTRACT
We present the Context Watcher, a mobile application that enables mobile phone users to easily and unobtrusively share personal context data such as their location, heart rate, speed, or view, with their mutual consent. Not only can the data be shared, it can also be used as input for information services, to adapt applications to the context, or to automatically derive daily patterns and situational information, such as 'meeting with a supervisor', 'is a regular visitor' or 'having the best condition of all your friends'. The application is built atop of the MobiLife context management framework, a generic approach enabling context discovery, exchange and reasoning. In the context management framework, different entities, context providers, are exposed to and interact via the internet. The context management framework and its principles will be discussed in detail, as well as the different context providers that play a role in the Context Watcher application. The Context Watcher has a large user base of 100+ users, some of which are 24x7 power-users taking the Context Watcher wherever they go, enabling the research on daily patterns and situations to work with real-life data, resulting in powerful algorithms and better understanding of context and context-awareness.

KEY WORDS
Context-aware computing, context reasoning, context tagging, mobile applications

1 Introduction

When a mobile user will be late for a scheduled event, such as a dinner party at a friend's house, the easiest way to resolve this unanticipated situation is to call their host, apologize for the delay, and provide additional information concerning their estimated arrival time. A drawback of this resolution is that this requires synchronous interaction between all parties, which causes the cooking party to leave the stove for a critical minute or to juggle the phone in one hand and the spoon in the other hand.

It would be much easier if the host could check asynchronously where his guests were located, and get an alert in case they would not make it in time for the scheduled dinner time? With today's current state of technology, scenarios such as this are not at all futuristic, but are possible to be implemented in your everyday life. Leveraging on these possibilities is exactly the goal of the IST project MobiLife [33], which brings recent advances in mobile applications towards users and group of users for making their everyday life easier. The main target user group of the MobiLife project is the family, especially families with members living in different locations, e.g., with children going to college. One of the project's main technology aims is to design a general framework that can support the provisioning of applications and services that are capable of adapting their behavior and functionality to information that characterizes the situation of the user. In short, we refer to these types of applications as context-aware applications. MobiLife will develop and demonstrate context-aware applications that are easy to use, targeted to be used within the family group, and that satisfy real needs of the family members.

The rest of this paper is structured as follows. Section 2 presents the MobiLife context management framework together with a short overview and specification of the interfaces and messages of the key components required for delivering context-aware functionality. Section 3 presents the Context Watcher application as an example of a context-aware application that is built atop the context management framework enabling family members to stay better informed about each other and their friends, without the need for synchronous interaction by using automatically derived context information. Section 4 presents highlights of context providers that are

used by the Context Watcher application, and focuses on exchange and reasoning of context information. Then, section 5 zooms in on related work and compares the Context Watcher with other (research initiatives). And finally, Section 6 concludes the paper with some of our experiences and findings and an outlook to the future.

2 Context management framework

The MobiLife context management framework (CMF) [1] is MobiLife's approach to discovery, exchange, and reasoning with context information. Its goal is to enable context information to easily flow from one provider to multiple consumers, and from multiple providers to a single consumer, in order to build smart constellations of context providers that finally can produce high-level situational information. This high level information is then built upon tiny bits of context information from heterogeneous sources, where heterogeneity has different dimensions, from syntax and semantics to transport, security, protocols and quality of context. The main tasks for the MobiLife CMF are:

- Enabling the discovery of context providers,
- Standardizing context exchange between providers and consumers,
- Supporting easy context reasoning by allowing reasoning components, e.g., a recommender, to be added to an application in a plug and play manner,
- Supporting the construction of different constellations of context providers to provide high-level situational information, e.g., the pipe-and-filter approach.

The realization of these tasks, however, is not trivial or necessarily straight-forward, especially if the solutions to be provided must be generic enough to work across different domains and application areas. We define the MobiLife CMF as a set of components, which are connected dynamically at run time, that together provide the relevant context information using sensing and interpretation mechanisms. The pipe-and-filter approach enables context providers to re-use existing context information, where context information is exchanged and enriched when it flows through the hierarchy, until we are able to provide more qualitative situational information about the subject, be it users, vehicles, or rooms. In the CMF, context providers (CP) can encapsulate context-information sources, and many different context providers can co-exist. A component might often be both a context provider and a context consumer (CC) at the same time And, hence, a generic consumer/ producer model, where every component can both publish and consume context, is needed. Fig. 1 illustrates the components of the CMF and highlights our view that the context provider is the central actor in the system. To find context providers, the consumers use a context broker and the internal working of a context provider might range from simply wrapping a body sensor for a single user to a full fledged inference reasoner that combines information gathered from other context providers for multiple users. What binds them together is that they all implement a minimal set of common interfaces, described in XML-based definitions, which make use of a context ontology describing the logical relations between the different context concepts in OWL-DL. A minimal set of interfaces describes the representation for context. The main task is to provide a published agreement or interface contract between context providers and context consumers. The contract is the only information a context consumer has prior to the binding with the context provider in order to utilize the context service functions. The most important messages in the representation include the context provider advertisement, the context query, the context element and the context subscription. With these messages, a context provider can implement an interface with a minimal set of operations to request an advertisement, to retrieve or push context information, or to subscribe to context changes. All services may support SOAP as well as HTTP GET and POST bindings, and are described in WSDL to ease the integration efforts.

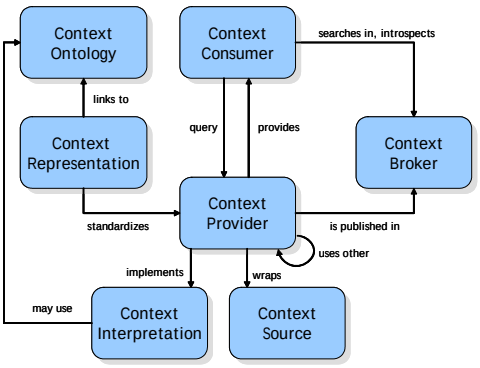


Fig. 1. The components of the CMF

To provide more insight about the functionality of the CMF, we shortly describe the most important messages that are needed for interacting with a context provider. More details on the design issues are provided in [1].

Context provider advertisement: Each CP should be described by an advertisement that uniquely describes the functionalities of the provider, the types of context information it can provide and the relevant entities playing a role in this information. Examples of entities are agents (person or groups), locations (building, floor, or room) and objects (vehicle, car, or plane). With the advertisement, we can describe providers that can deliver the temperature of a collection of rooms, or the location of a specific group of users, together with a URI pointing to the service that can deliver this information. One CP may deliver different types of context parameters, which can be nested to create structures of context parameters, and which can be linked to the central context ontology to gain common understanding and enable ontology reasoning.

Context query: A context query describes what type of context information is requested from which CP, relating the information to specific entities. If not stated otherwise, the query refers to the last known context state, e.g.,

someone's actual location. However, a consumer may also specify a filter, to select specific historic information from a CP with some kind of caching. These filters can be posed on all attributes of the context parameters. A summary can be added to the query to perform operations on the result set of the filtering. The result of this operation is then provided as a context element to the context consumer.

Context element: A context element is an elementary piece of context information. Each context element has metadata describing the context provider that delivered the data, a name, the entity that is the subject of this context element, and a timestamp denoting on what point in time this context information was captured. The value of the context element is again a parameter tree, as in the advertisement, this time containing the values of the parameters. A value can be accompanied by a measure for accuracy or other quality-of-context data.

Context subscription: A context subscription is used by consumers to subscribe to a provider for notifications about context changes or for updates of context information at regular time intervals. The conditions and/or the length of the time interval can be specified by the consumer, as well as the location of its pushContext interface it uses to receive the context elements, when the conditions of the subscription are met. The context subscription contains a filter and a summary that specify *what* context data will be sent, and a condition that specifies *when* the context data will be sent to the specified interface. This means that the CP should dynamically bind with the CC's pushContext interface when the given conditions are evaluated positively.

Filter, conditions and summary: The conditions specify a certain situation in terms of a set of logical rules, based on the attributes and values of the context parameters. A simple condition works on a specific parameter (e.g., speed) and a specific attribute of this parameter (e.g., the value or the timestamp). This attribute can be logically compared with a value, which results in a binary (true or false) decision. Complex conditions are of course compounds of simple conditions. A summary can be added to the query or to the context subscription. The effect is that a summary operator will be performed over the result set of the filtering, e.g., to compute the average of a context parameter in a certain time period.

The next section describes an application that uses these interfaces and messages to obtain or provide the latest contextual information about the members in a social network.

3 The Context Watcher application

The Context Watcher is a mobile application developed in Python, and executes on Nokia Series 60 mobile phones. The aim of the Context Watcher is

1. to enable end-users to automatically record, store, share and use context information in a way that makes their life easier, e.g., because weather services can work automatically when location context is already known, because they can unobtrusively check the whereabouts and well-being of their friends, or because context tags enable them to find photos of a specific situation with ease
2. to gather real-life context data from a large user group. The data can be used to develop and test reasoning algorithms and to train reasoning engines that produce situational information from raw context data. The derived situational information and the effectiveness of the algorithms can then be tested in a real setting with the Context Watcher
3. to create awareness in a large audience about the possibilities of context-aware computing in the everyday life of now, e.g., by demonstrating the ease and usefulness of life blogging and picture tagging using automatically recorded context information

The available information depends on the context providers that are connected to the Context Watcher application, and is certainly not limited to location only. Not all users have exactly the same set of context providers, depending on availability of sensors, situation, or user requirements. The Context Watcher application has been implemented using a modular architecture that allows dynamic configuration of the used components..

The Context Watcher application is able to record information about the user's

- Location (based on GSM cell and optionally on GPS)
- Mood and other subjective pieces of context information based on user input, e.g., availability or perception of safety
- Activities, meetings, and daily patterns (based on clustering and information fusion)
- Body data (based on heart and foot sensors)
- Weather (based on a location-inferred remote weather context provider)
- Visual data (pictures tagged with contextual data)

When the user starts the Context Watcher it, depending on its configuration, automatically connects to the available local sensors, e.g., via BlueTooth, and to the remote context providers over the 3G network. All connections may also be manually activated by the user. E.g., when the user has a GPS receiver, the Context Watcher application will send the GPS data together with GSM cell information to the remote location provider, which will enrich the data (described in Section 4.1). Enriched information is returned to the user, and they receive an alert when they enter a known cluster (such as their Home, Office, or Hotel) or when they are close to one of the persons in their buddy list.

Social networks: Users can invite other users to become their buddies. Initially, new buddies will be shown in their contact list as 'requested' with no extra information. As soon as a buddy approves the request from their contact list, a relation is established and context information can flow over that relation.

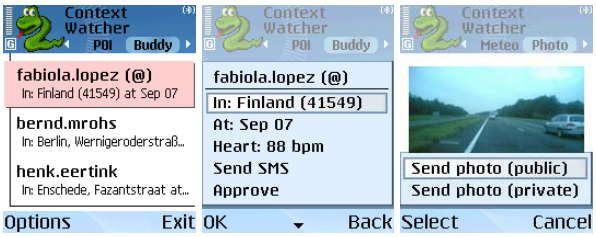


Fig. 2. Contact list with near real-time sharing of contextual data and interaction possibilities, including sharing of context-enriched pictures between buddies or to the general public.

Fig. 2 shows an example of a contact list with near real-time context information, that is, actual information when both parties have the Context Watcher application running, or the last known information together with a time stamp when the other party is off-line. The information shown is best-effort. This means that –in the example- Fabiola has no GPS attached or is inside a building, her cell information cannot be resolved, so the best location determination is at country level. The other two contacts have a GPS running or are in a cell of which the location can be resolved from the cell-id database [4]. The location, in terms of latitude-longitude, is resolved into street and city information using the MapPoint web service [7], and this information is shown in the contact list, together with readings from other context providers, including heart rate and walking speed information from the wellness provider (see Section 4.4), resulting in the 88 beats per minute heart rate of Fabiola. Our next steps include transformation of these context bits into situational descriptions like: "Fabiola is being relaxed at home" or "frantically typing in her office".

Context tagging: All gathered context information can be used 1) to adapt the behavior of application running on the mobile phone (e.g., prioritizing the favorite applications depending on location cluster), 2) to serve as input for information services (maps, points of interest) or other context providers (described in Section 4), and 3) to tag multimedia content recorded with the mobile device.
When pictures are submitted to the photo context provider from the Context Watcher application, a large part of the descriptive text can be written automatically using present context information. Fig. 2 & 3 show how a picture can be submitted, and how this picture is tagged with automatically recorded street and city information, the geo position, the speed and direction of movement, the name of the location cluster (is this a home or an office picture), and the people who are nearby.

We have integrated the photo context provider with Flickr [31], one of the largest public image servers of this moment, where the context information is submitted as tags, and the descriptive text is automatically generated, e.g., "I was on [business trip] together with [Henk] and [Bernd] in [Oulu] and I made this picture of the [Alexanderkatu] while traveling [with public transport] to the [summer school]", where all the information between brackets is auto-generated. This means that one action on the mobile phone is enough to send a richly described picture to a remote image server, enabling others to easily find pictures of their liking, e.g., by browsing the context tags to separate the home pictures from the office pictures. Programmatic interaction is also a possibility: because all pictures are geo-tagged, they also show up in experiments where other parties integrate Flickr and Google Earth, including [6] and [15]. The collection of all public context-enriched pictures from different users can be found at [16]. The user can choose the image quality, and when bandwidth will increase and costs will decrease (in The Netherlands, one operator offers flat fee GPRS for 10 Euro per month), users will be bound to submit higher quality images over time.

Life blogging: One of the next steps is to generate daily reports from the different streams of context information and the pictures taken during the day, and to present them in a format which is enjoyable and informative to a human reader, with cross links between the summaries for easy navigation and categorization. This way it is possible to easily browse your own life, e.g., by finding those other days that you met buddy Marko, and recollecting what you did together. An example is provided in [32].

Owner (rights) Date	Picture	Context Data
Johan (private) 9/14/2005 1:35:45 PM		*cell id:* 10571 *altitude:* 59.4 *speed:* 115.1 km/h *course:* 246.6 *pos:* (52.279,6.503) *range:* 1 m *street:* E30 *postal code:* 7462 *city:* Rijssen (NL)
Johan (public) 12/1/2005 6:15:25 PM		*cell id:* 17404 *pos:* (45.189,7.644) *range:* 1 m *street:* SP2 *postal code:* 10072 *city:* Caselle Torinese (IT) *buddy:* wagner *flickr:* link

Fig. 3. Sample overview of the submitted pictures, together with relevant context data of that moment. The first example shows GPS based location data, the second one show cell-id based location data together with automatically derived cluster names and nearby buddies (if any).

4 Context providers

The Context Watcher application itself is in fact a very thin application that interacts with a wide spectrum of local and remote context providers and information services. In this section, we give information about the functionality of underlying context providers, and show that despite the differences, in for example complexity or in intelligence, these context providers together can deliver a new and innovative mobile application that

enables the end-user to share, store, and re-use his personal information with more ease, and to enjoy the use of an application that takes work out of his hands and better adapts to his needs and wishes. All context providers offer both a web interface for visual inspection and maintenance, and a web service interface for programmatic interaction, standardized to the representation discussed in Section 2. Each context provider implements the CRF interfaces, which is done by easily generating the server class (named contextProvider) from the WSDL. The actual implementation of the context provider inherits from this abstract contextProvider class and overrides all methods in this class, e.g., getAdvertisement and getContext. The interaction between the Context Watcher and the context providers is shown in Fig. 4.

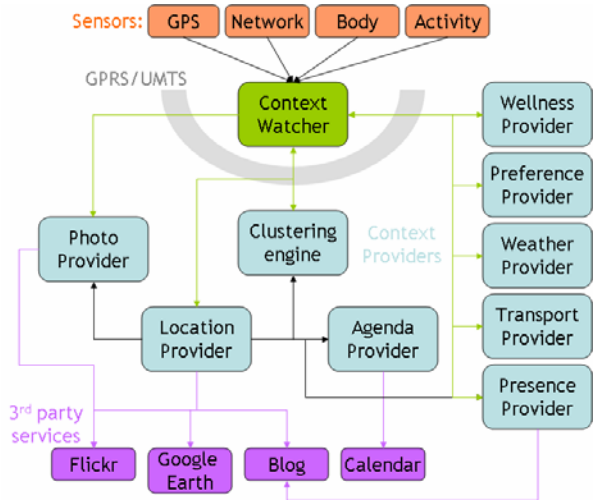


Fig. 4. Interaction of the Context Watcher with sensors, context providers and 3rd party services.

4.1 Location provider

The main tasks of the location provider are threefold. First of all, the location provider enriches and refines coarse location information. For example, the location provider attempts to deduce the current city and street from GSM cell and GPS latitude-longitude information. For resolving cell ID into location it uses a large cell database [4]. Secondly, the location provider acts as a repository for location information and, thirdly, exchanges the information with authorized parties. For now, only the individual user and his/her buddies are authorized to request location information. Using historic location information, the location provider can also find frequently visited places for a specific user (see also Section 4.2), and provide information about trajectories between those frequently visited places, e.g., the daily pattern from home to office. It can detect spontaneous meetings with known buddies and this information can serve to alert the user in the Context Watcher application (e.g., about nearby buddies), or to exchange with buddies (e.g., about current activity).

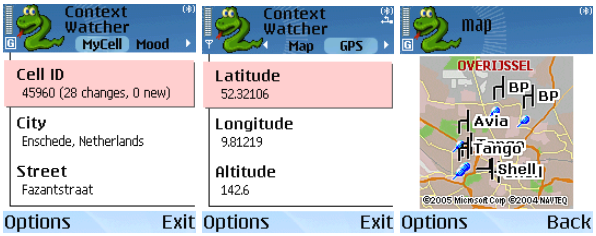


Fig. 5. Enriched location data on the mobile phone, also used as input for e.g., point of interest and map services.

Fig. 5 shows how location information from the location CP is displayed and used in the Context Watcher application. In the example above, GPS data is used as input, but the system works exactly the same based on cell-id information, only the accuracy goes down substantially, which results among others in less detailed maps, and less detailed location information to be sent to the buddies.

4.2 Location clustering

The GSM cell information and the GPS coordinates are typically not meaningful to a user nor do they normally carry any semantic meaning for the user. Instead of giving information about the location in exact form, it is often more useful to identify abstract locations such as ‘Home’ or ‘Work’. In order to do the step from non-meaningful data to personally important locations, we employ a clustering algorithm that uses information from GSM cell transitions and the GPS latitude-longitude coordinates at the transition points to identify clusters of cell identifiers that are likely to belong to the same abstract location.

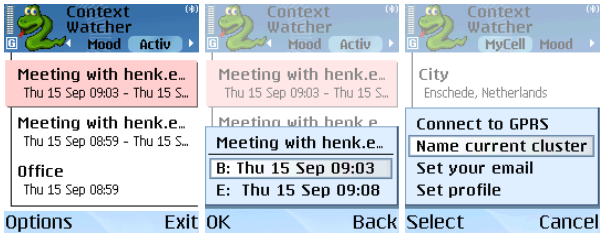


Fig. 6. This example shows the automatically derived clusters and meetings. Each event has its own begin (B) and end (E) time, and is also published in the agenda CP.

Whenever a cell transition occurs, the Context Watcher application submits all location information to the location provider, which then processes the data. The clustering is not performed each time data is sent to the server, but instead we run the algorithm periodically, typically overnight. Alternatively, the user can manually trigger the clustering. Whenever the current GSM cell is identified to belong to a cluster, information about the cluster is submitted back to the phone.

Details of the applied clustering algorithms are described in [14] and [17]. Here, it suffices to say that the currently applied algorithm builds a first-order Markov matrix of

the different states of the user, based on cell-id and geo-location, detects the short-range loops in that matrix, and assigns all states in the loop to a cluster. Clusters are numbered by the system, and can be named by the user. Fig. 6 shows the cluster naming in the Context Watcher. If the cluster has a name, the activity tab of Context Watcher shows that as the current activity. On the other hand, if the cluster has no name yet, the user is informed that (s)he is in an unknown cluster, which can then be named. Cluster management can also be done via a personal web site that provides on overview of all clusters of a user. The clusters are also used by other context provider, e.g., they are used in the personal blogs, and the agenda provider uses them to automatically name activities.

4.3 Agenda provider

The agenda context provider attempts to simplify calendaring and scheduling by exposing temporal context information, from the time domain, using standard internet technologies such as the Internet Calendaring and Scheduling Core Object Specification (RFC 2445), HTTP, and WSDL. Context information can be used both programmatically, as a web service, by context consumers or the Context Watcher application, as well as manually by individual users using any application that supports the iCalendar specification, such as the Mozilla Calendar [13] application.

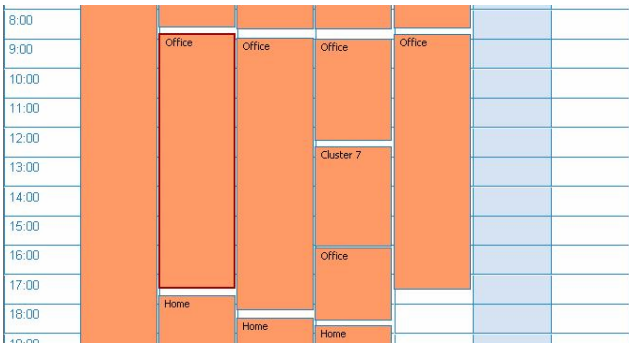


Fig. 7. Dataview of the Agenda Provider in Mozilla calendar: automatically obtained in a regular workweek with the Context Watcher running.

Although the tasks of calendaring and scheduling are often viewed as synonymous, this is incorrect. Both calendaring and scheduling represent unique temporal data management concepts. *Calendaring* involves the creation and manipulation instances of temporal data on a calendar, such as an Event or a Todo, while *scheduling* involves the communication and negotiation between calendars containing temporal data instances for the manipulation of new or existing related temporal data instances to achieve a goal. To handle both calendaring and scheduling the agenda package exposes all data objects, whether they be Events or Todos, as a web service. Through the use of standard web service technology, such as HTTP or SOAP, individual and group calendaring and scheduling task can be achieved by context consumers. One example – illustrated in Fig. 7 – of both calendaring and scheduling is the calendaring of the most frequently visited locations, originating from the Location Context Provider, which can be used by a reasoning context consumer to generate resulting scheduling data.

4.4 Wellness provider

A personal condition context provider typically stores information about the mental and physical state of a user. This information may come from different body sensors that collect information about body movements, heart activity, brain activity, eye movements, et cetera. The wellness provider is an example of a personal condition context provider that collects data gathered from a Suunto T6 watch equipped with an additional footpod (which measures walking speed and distance) and a heart rate belt (which measures heart rate). These three devices communicate using a proprietary low-energy protocol called ANT [5]. To connect these devices to a phone, a Bluetooth-to-ANT module is used. This module receives ANT messages from the sensors and exposes the information for other parties via a Bluetooth interface that implements the SSI protocol [9]. The Context Watcher application collects the information from the Bluetooth interface and updates the remote wellness provider with the information using a GPRS connection. The wellness provider provides different kinds of information, e.g., performance charts (heart rate against speed) and statistics over past heart rate information:

Heart rate for	Average	Deviation	Minimum	Maximum
johan	75.3	15.5	46	153
anthony	76.8	11.7	66	98
agathe	85.5	12	75	109

This information can be shared with the buddies running the Context Watcher application, as depicted in Fig. 2. This wellness data is mostly used to compare physical condition and performance with the buddies, or to generate personal training advice.

4.5 Transport provider

Many public transportation companies offer web-based interfaces to access transportation routes and timetable information and to provide users with advanced search features, such as maps, to plan their trip. The transport provider application demonstrates the use of a symbolic cluster map (SCM) based context recognition module [2] and provides a personalized access to public transportation information from a mobile device. The initial version provides access to the public transportation

system of two different locations, the city of Helsinki in Finland [11] and the region of Niedersachsen and Bremen in Germany [12].

The SCM module clusters tuples of strings representing the input context data. The context data used in our demonstration is GSM Cell Id and time. Strings are dynamically associated with the GSM cell information based on (cell-id, lac, mnc, mcc). Both the location and time data are discretized, and this discretized form will be given as input to the SCM, e.g., as (a, c, b, p). The SCM module picks the winning node of the grid for the given input, and updates the internal representation of the values associated with this node. We call recognized context the winning node for a given input data. The size of the grid depends partially the context recognition granularity. Our working implementation uses a grid of ten nodes, named 1-10. Moreover the configuration of the grid is stored into a file and retrieved from it when the application restarts.

The transport provider application keeps an SCM module that senses and clusters data periodically, i.e., every 30 seconds, and uses the recognized context to index the search queries entered by the user. For example, if his current context is 3, the user has access to the list of queries he has previously entered when in context 3. At best, the user reuses a stored query that is automatically updated with the current date and time. At worse, the user enters other parameters. If new, the query will be added to the list of queries for the current context. The queries are transformed into URLs that are sent to the chosen transportation internet servers. The HTML page retrieved is simply displayed by the default browser application. This way, it is very easy for the user to obtain relevant personal traveling information based on context: e.g., at home the recommended query is for the next bus to the office, and in the office the recommended query is for the next bus home, all with one click action of the user. Of course, this is a simple case, but also more complex and unpredictable cases have shown useful results.

4.6 Weather provider

The weather provider is an example of how public information service can be wrapped into the MobiLife CMF structure. It delivers environmental information for the given location, identified by a tuple that specifies a city and a country (Fig. 8). If only GPS latitude-longitude information or a cell-id is available, a corresponding translation (such as the one implemented by the location provider described in Sect. 4.1) has to be applied before the invocation.

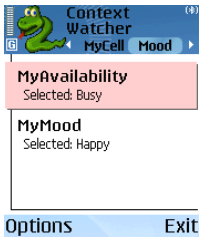
The contextual data provided comprises a description of the current situation as well as a forecast for the next few days, covering all major regions such as Africa, America, Asia, and Europe. Besides actual weather information such as temperature (measured and RealFeel), wind (speed and direction), precipitation (snow and rain), UV index, humidity, pressure, visibility and a textual description of the current conditions, one can also retrieve prognosis of those elements (e.g., the expected maximum temperature for tomorrow).

Fig. 8. The results of the weather provider in the Context Watcher application

Forecast information is available for today as well as for the next few days. While actual weather data can be retrieved using the simplified getLastContext interface, weather forecasts have to be accessed by using the general getContext function with an appropriate filter element. Of course, the accuracy values for context elements describing predictions are low for long-range forecasts. In additional to pure weather information, basic localized information about the moon and the sun (i.e., moonset, moonrise, moon phase, sunset and sunrise) are available too. This data can be used, for example, to estimate the actual outdoor light conditions at a given location. All qualitative values (such as the weather condition, the moon status and the wind direction) are complemented by references to corresponding ontology elements to enable ontology-based situational reasoning [10]. Future work includes the implementation of the CMF publish and subscribe interface to warn consumers about major weather changes like falling temperature or rain in the afternoon.

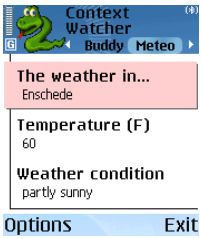
4.7 Presence provider

The presence provider enables users to share their current presence status with others. Users can publish their availability for being contacted and their current mood for all their buddies. Using the Context Watcher application, users can manually set their current availability and change their mood and send this information to the presence context provider that offers this information to interested context consumers. Users can select ‘Online’, ‘Busy’ or ‘Away’ as availability and ‘Happy’ or ‘Sad’ as mood. If desired, users can also add custom moods or availabilities. The presence information could also be changed automatically by a reasoning system that infers user’s presence status based on other context information like location, time and people in proximity. This way, it could automatically be inferred that users are not available for being directly contacted at the moment, because they are currently in a business meeting. The availability will be seen by the buddies when they try to call, which can make



them to decide to call later. Knowing about the presence status of users is one of the key factors in realizing situation based applications. Acting as context consumer, an intelligent service provisioning application can offer services to the user based on the current presence status. Services may also be automatically triggered when the presence status changes. For example, a phone call can be automatically redirected to the secretary when user's availability is set to 'busy'.

4.8 Preference provider



The preference context provider offers personal and group-related profile and personalization data. This includes users' personal information or users' global preferences shared by all services or service specific preferences. Preferences can be activated related to certain user situations. Based on a high-level situation description the most appropriate set of personalization data will be selected. For example, the situation "being at home" may lead to a different suggestion which music should be played compared to the situation "driving in my car". The preference context provider can be used by any application for storing, reading and managing situation-based group and user preference data. The management of such information can be done via web. A simple GUI allows users to create or update their preferences. Users can specify their personal records and specify their preferences for different services and profiles. Users can also activate certain profile or specify when to activate or deactivate certain profiles.
The Context Watcher application takes advantage of the preference context provider and enables users to personalize certain features. One simple example is to select the preferred measurement unit for the weather provider, which can be set to Fahrenheit or Celsius. The Context Watcher application displays the temperature value of the weather information based on user's setting.

4.9 Context Visualization with Google Earth

As the use of the Context Watcher application grows, so does the context information. Visualization of such information important to have an overview of the information collected. The location provider collects location information of Context Watcher users. The Google Earth application provides a powerful interface to visualize the location of a user's buddies in the world map using the location information from the location provider. Moreover, it is also possible to show any other context information associated with the user, coming from different context providers. Of course, you will see only those pieces of information the users allows you to see.

The Google Earth application uses the KML data file in XML format for showing geographic features like points, lines, and images in the Google Earth Client. This file format provides ability to specify images and labels to identify locations in the world map as well as to dynamically get such location information from the remote or local network locations at certain intervals.
Taking advantage of such features, a simple KML file has been created which at regular time intervals accesses the CMF to get the location of all buddies, resulting in a visually appealing way to see all your buddies moving over the globe as depicted in Fig. 9.

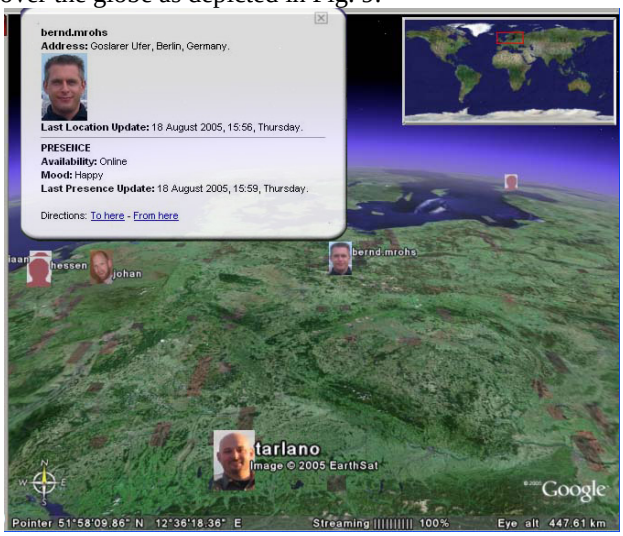


Fig. 9. Near real-time buddy locations integrated in Google Earth

Additional buddy information includes the postal address and the user's current presence information retrieved from the presence context provider, and an image of each user. This way it is possible to show the location and additional context information of users in near real-time.

5 Related work

This section shortly describes related work in context aware applications and architectures.

Architectures and Frameworks: A substantial number of previous research efforts have proposed architectures and middleware for context-aware applications. In this section, we shortly review and discuss these efforts in relation to the MobiLife CMF. According to the overview Moran and Dourish [21], most approaches are implemented in the form of middleware and its services, or in the form of application frameworks.
Examples of the middleware approaches include, e.g., the Reconfigurable Context-Sensitive Middleware (RCSM) [22], and the CORTEX middleware [23]. For our purposes the CORTEX approach is more interesting as it introduces special entities, called *sentient objects*, which are responsible for receiving, processing, and providing

context-related information. Sentient objects are defined as autonomous objects that are able to sense their environment and act accordingly [23]. The advantage of this approach is the possibility to re-organize them, for instance depending on their primary task.

Another closely related middleware approach is Gaia, which also uses the concept of providers and consumers as the basis for data exchange. Another similarity with the Gaia is that it is flexible in terms of the supported reasoning functionalities [30]. Also some application frameworks that support context-awareness have been proposed. For example, [20,24] describes the implementation of a framework that supports management of context information on mobile terminals. The structure of this framework is centered on the blackboard paradigm for communication, which is handled by a context manager. Most components that use this framework, including the applications, act as clients for the context management system on the device itself. Other services can potentially run also in a distributed environment. Another framework approach is the Context Toolkit [18,25], which separates acquisition and presentation of context information from the application that requires it, by using so-called widgets. The focus of this work lays in the automatic inference of higher-level context information from lower-level sensor data.

Tools for data gathering and sharing: Various tools for gathering and sharing contextual data have been proposed in the literature. Many of these are using Bluetooth for collecting the data, but also more generic tools that attempt to get information about the network and/or device capabilities have been developed. Closest to our work is the ContextPhone [3]. The ContextPhone has been implemented for Nokia Series 60 mobile phones using C++. The basic set of information that is available via the ContextPhone is similar to our setting, and the ContextPhone also supports sharing information and uploading context tagged photos to Flickr, but with a limited context data set. The main difference is that ContextPhone is a platform that runs on the phone almost entirely with communication pipes to the outside world, whereas the Context Watcher is only a thin application that interacts with a network of remote Context Providers in the CMF, enabling more powerful reasoning, longer data storage, and easy integration with 3rd party services such as the context blogs.

Another closely related approach is to integrate several sensors directly to a terminal device. For example, Muffin [26] integrates, e.g., an air temperature sensor, a humidity sensor, an alcohol gas sensor, a pulse sensor, a compass and a linear 3D accelerometer, to a terminal device. The Bodymedia SenseWear PRO_2 Armband [27], on the other hand, allows obtaining information, e.g., about energy expenditure, sleep/wake states and duration of physical activity. A related approach is the Intel Mote [28] technology, which allows several external physiological sensors to be attached to a small chip that has Bluetooth capabilities. In terms of data gathering, the work that is closest to our work is the IBM Mobile Health Toolkit [29], which contains a MIDP application that runs on a mobile phone and which uses a specific set of health sensors (part of the kit) to gather remote medical data.

Life blogging was studied in [21]. In the study, ContextPhone was used to gather data about communication behavior of users and the strength, dynamics and evolution of social networks were studied. In addition, the authors used the concept of entropy to estimate the predictability of daily routines and formulated the use of so-called eigenbehaviors from the data.

6 Experiences & conclusions

We have shown that a mobile phone can truly serve as an application platform for sharing context information in everyday life. Experience using our application has shown that context-awareness does not force people to adapt or change significantly their daily patterns in their normal environments, but accompanies them throughout everyday life. As such, the Context Watcher application has proven to be a perfect platform for large scale and cheap user experiments with mobile, context-aware applications in a world where reality can be the laboratory for user tests.

The Context Watcher application as a prototype was built on top of the MobiLife CMF. It integrates with a substantial number of remote and local context providers, and has a user base of more than 100 persons. In about one year time, these users have formed many relationships, and by observing their movements through out everyday life, the system has found about 600 frequently visited places and detected numerous meetings between buddies in a total of 200.000 location measurements. In the several thousands of submitted pictures, we even proposed conventions for context tagging that were adopted as de-facto by a much larger user community.

The functionalities most valued by the users were 1) the possibility to know where the others are and to keep in touch without having to approach them directly, 2) easy access to services because the input parameters are automatically provided context parameters (e.g., local weather with one click, rich picture submission with one click, easy public transport info, et cetera), and 3) remote logging of activities and preferences, and the sharing of this information across different (mobile) applications and web sites, including Flickr.com and their personal blogs. Sharing information via these channels with non-users of the Context Watcher (e.g., parents) was perceived as added value. Given their experience with the Context Watcher most users could easily imagine new applications that involved sharing of context data in specific niches, like body performance data when doing sports, or road quality information when skating.

The development of the Context Watcher application also evaluated the specification of the CMF and in

particular the specifications describing its interfaces. Feedback from the different developers, using a variety of languages (Java, C# and Python) and development tools demonstrated the specification to be at a mature level. Additionally, we have used the Context Watcher application many times to explain the principles behind the CMF and to improve understanding about context-aware applications to a broad audience.

The Context Watcher application is freely available [8], and has proven to be the right basis for easy extension with new wishes of users and offerings of new context providers.

Acknowledgements

The authors would like to thank the MobiLife project, and Work Package 4 in particular, as well as Fabiola Lopez from Suunto for her support in supplying the body sensors and Bluetooth integration. And special thanks goes to all Context Watcher users for providing real-life experimental data and valuable feedback!

References

[1] P. Floreen, M. Przybilski, P. Nurmi, J.Koolwaaij, A. Tarlano, M. Wagner, M. Luther, F. Bataille, M. Boussard, B. Mrohs and S. Lau, Towards a Context Management Framework for MobiLife, *Proc. of the 14th IST Mobile & Wireless Communications Summit*, Dresden, Germany, June 19-23, 2005.

[2] J. Himberg, J. A. Flanagan, and J. Mäntyjärvi, Towards Context Awareness Using Symbol Clustering Map, *Workshop for Self-Organizing Maps 2003 (WSOM2003)*, pp. 249-254, Kitakyushu, Japan, 2003.

[3] M. Raento, A. Oulasvirta, R. Petit, and H. Toivonen, ContextPhone - A prototyping platform for context-aware mobile applications, *IEEE Pervasive Computing*, 4 (2): pp. 51-59, 2005.

[4] J.W. Koolwaaij, Mapping the GSM landscape, *Proc. of the 4th SVG Open conference*, Enschede, The Netherlands, 2005.

[5] ANT, *http://www.thisisant.com*

[6] Map Flickr photos with Google Earth *http://www.lifehacker.com*

[7] MapPoint web services, *http://www.mappoint.net*

[8] Context Watcher manual *http://www.lab.telin.nl/~koolwaaij/showcase/crf/cw.html*

[9] Simple Sensor Interface, *http://www.ssi-protocol.net*

[10] Situational reasoning – a practical OWL use case, M. Luther, B. Mrohs, M. Wagner, S. Steglich, and W. Kellerer, *Proc. of the 7th International Symposium on Autonomous Decentralized Systems (ISADS'05)*, Chengdu, China, April 2005.

[11] Public transport information system Helsinki *http://aikataulut.ytv.fi/reittiopas*

[12] Public transport information system Hannover *http://www.efa.de/gvh*

[13] Mozilla Calendar, *http://www.mozilla.org/projects/calendar*

[14] Self-Mapping in 802.11 Location Systems, A. LaMarca, J. Hightower, I. Smith and S. Consolvo, Placelab white paper.

[15] Geobloggers, *http://www.geobloggers.com*

[16] The picture collection in Flickr made with the Context Watcher, *http://www.flickr.com/photos/tags/mobilife*

[17] P. Nurmi and J. Koolwaaij, Identifying meaningful locations, Submitted to Conference on Mobile and Ubiquitous Systems: Networking and Services, 2006.

[18] A.K. Dey, G.D. Abowd, The Context Toolkit: Aiding the Development of Context Aware applications, *http://www.cs.cmu.edu/~anind/context.html*

[19] N. Eagle, Machine Perception and Learning of Complex Social Systems, *Ph.D. Thesis*, Massachusetts Institute of Technology, 2005.

[20] P. Korpipää, M. Koskinen, J. Peltola, S.-M. Mäkelä and T. Seppänen, Bayesian approach to sensor-based context awareness, *Personal Ubiquitous Computing*, 7(2): pp. 113 – 124, 2003.

[21] T.P. Moran and P. Dourish, Introduction to special issue on context-aware computing, *Human-Computer Interaction (HCI)*, 16(2-3), pp. 87 – 96, 2001.

[22] S.S. Yau, F. Karim, Y. Wang, B. Wang, S.K. Gupta, Reconfigurable context-sensitive middleware for pervasive computing, *Pervasive Computing*, 1(3), pp.33-40, 2002.

[23] H.A. Duran-Limon, G.S. Blair, A. Friday, P. Grace, G. Samartzidis, T. Sivaharan, and M. Wu, Context-aware middleware for pervasive and ad hoc environments, *Technical Report*, Lancaster University 2004.

[24] P. Korpipää, J. Mäntyjärvi, J. Kela, H. Keränen and E.-J. Malm, Managing context information in mobile devices, *Pervasive Computing*, 2(3), pp. 42 – 51, 2003.

[25] A. K. Dey, G. D. Abowd and D. Salber, A conceptual framework and a toolkit for supporting the rapid prototyping of context-aware applications, *Human-Computer Interaction*, 16(2,3 & 4), pp. 97 – 166, 2001.

[26] T. Yamaba, A. Takagi and T. Nakajima, Citron: A context information acquisition framework for personal devices, *Proc. of the 11th International Conference on Embedded and Real-Time Computing Systems and Applications*, IEEE; 2005.

[27] Bodymedia, Bodymedia Health Monitoring System, *http://www.bodymedia.com/main.jsp*

[28] Intel, Intel Motes, *http://www.intel.com/research/exploratory/motes.htm*

[29] IBM, IBM Mobile Health Toolkit, *http://www.zurich.ibm.com/mobilehealth*, 2006.

[30] A. Ranganathan and R.H. Campbell, A Middleware for Context-Aware Agents in Ubiquitous Computing Environments, *Proc. of the ACM/IFIP/USENIX International Middleware Conference*, 2003.

[31] Flickr, a popular image sharing service with open interfaces, *http://www.flickr.com*

[32] Example context blog: *http://koolwaaij.blogspot.com*

[33] MobiLife web site: *http://www.ist-mobilife.org*