Managing Context Information in Mobile Devices

A new software framework simplifies the development of context-aware mobile applications by managing raw context information gained from multiple sources and enabling higher-level context abstractions.

Panu Korpipää, Jani Mäntyjärvi, Juha Kela, Heikki Keränen, and Esko-Juhani Malm
VTT Technical Research Centre of Finland

Mobile device users want to be able to access and manipulate information and services specific to their location, time, and environment. Context information gathered from sensors, networks, device status, user profiles, and other sources can enhance mobile applications' usability by letting them adapt to conditions that directly affect their operations.

To achieve true context awareness, however, mobile systems must produce reliable information in the presence of uncertain, rapidly changing, partially true data from multiple heterogeneous sources. Mobile devices equipped with low-cost sensing elements can recognize some aspects of context. However, extracting relevant context information by fusing data from several sensors proves challenging because noise, faulty connections, drift, miscalibration, wear and tear, humidity, and other factors degrade data acquisition. Extracted contexts overlap, change with time, and yield only partially reliable approximations.

Furthermore, mobile devices' dynamic environments require that we learn context descriptions from multidimensional data. Learning systems can't easily generalize beyond training data, however. Using even sufficiently reliable derived contexts directly to control mobile applications poses problems because users with different ideas of "context" might find application behavior irritating.

To address these challenges, we present a uniform mobile terminal software framework that provides systematic methods for acquiring and processing useful context information from a user's surroundings and giving it to applications.

A context management framework

The framework we present permits recognizing semantic contexts in real time in the presence of uncertain, noisy, and rapidly changing information and delivering contexts for the terminal applications in an event-based manner. Our application programming interface (API) for using semantic context information uses an expandable context ontology to define contexts that clients can use.

We chose a blackboard-based approach[1] as the underlying communication paradigm between framework entities. Our approach focuses on mobile terminal capabilities rather than infrastructure. Accordingly, we designed the framework for the Symbian platform (www.symbian.com) to achieve true device mobility, high performance, and a broad user base.

Four main functional entities comprise the con-

Published by the IEEE CS and IEEE ComSoc ■ 1536-1268/03/$17.00 © 2003 IEEE

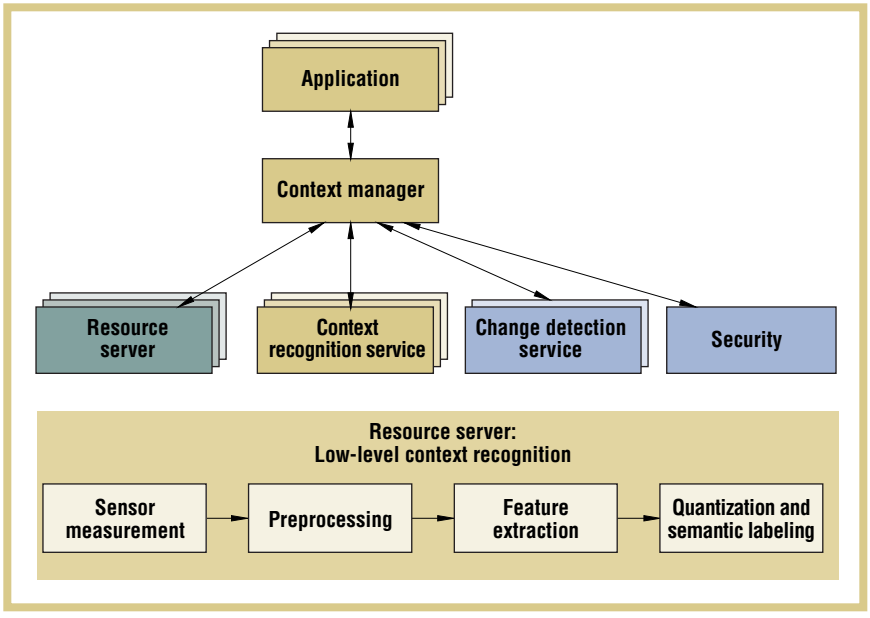


Figure 1. Entities and information flow (arrows) in the context framework. Clients (application, resource server, recognition service) can add, subscribe to, and request context information. The context manager stores contexts and delivers responses and change notifications to the clients. The figure details resource-server information flow to illustrate the phases required to transform the raw sensor data into human-interpretable context information, which applications can further process and use in an event-based manner.

text framework: *context manager*, *resource server*, *context recognition service*, and *application*. When entities communicate, the context manager functions as a central server while other entities (except security) act as clients and use services the server provides. The context manager, any resource servers, and applications run on the mobile device itself, and the services are either distributed or local.

At the heart of the mobile terminal context system is the blackboard-based context manager. This central node stores context information from any source available to the terminal and serves it to clients in three ways:

- Clients can directly query the manager (as a context database) to gain context data.
- Clients can subscribe to various context change notification services.
- Clients can use higher-level (composite) contexts transparently, where the context manager contacts the required recognition services.

The resource servers connect to any context data source and post context information to the context manager's blackboard, which further processes the data if needed and delivers it to the clients according to their subscriptions. The delivered data's abstraction level should be high enough and the frequency low enough to be useful yet not overwhelming. With raw high-frequency (sensor) data, the resource server should perform low-level recognition before delivering the data to the context manager. Figure 1 shows the low-level context recognition process flow in the resource server. The measurement phase reads the sensors and outputs raw data. The preprocessing phase builds measurement data arrays that contain a certain number of samples (quantization of time dimension) and calculates generic features for each time interval.

Feature extraction calculates more specific context features. The quantization phase binds the feature values to the real-world context, which has a meaning for a person or an application according to a predefined ontology. Fuzzy sets or crisp limits are used for quantizing extracted features. The quantization phase outputs context atoms (base context units) that applications can use or recognition services can further refine. The resource servers use one of two methods for quantization:

- Set crisp limits, resulting in a true-false labeling for context atoms represented with fuzzy membership functions denoted as $\mu(x)$. For example, in quantizing environment sound intensity, the quantization divides the processed feature into three quantities—Silent, Moderate, and Loud—corresponding to the three membership functions Figure 2a shows. If one of these is true, the others are false.
- Apply a fuzzy set for features, result-

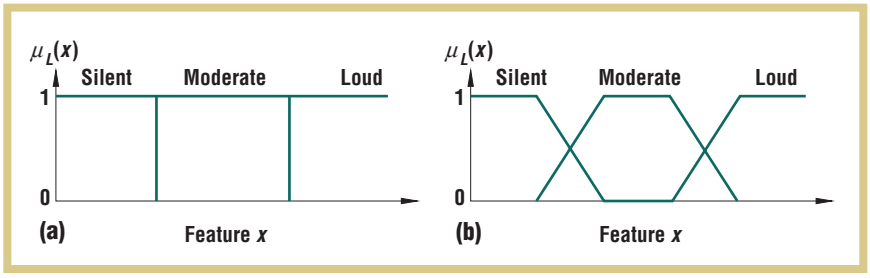


Figure 2. Examples of (a) crisp and (b) fuzzy quantizations.

TABLE 1
Sensor-based context ontology vocabulary example.

Context type	Context value
Environment:Sound:Intensity	{ Silent, Moderate, Loud }
Environment:Light:Intensity	{ Dark, Normal, Bright }
Environment:Light:Type	{ Artificial, Natural }
Environment:Light:SourceFrequency	{ 50Hz, 60Hz, NotAvailable }
Environment:Temperature	{ Cold, Normal, Hot }
Environment:Humidity	{ Dry, Normal, Humid }
User:Activity:PeriodicMovement	{ FrequencyOfWalking, FrequencyOfRunning, NotAvailable }
Device:Activity:Stability	{ Unstable, Stable }
Device:Activity:Placement	{ AtHand, NotAtHand }
Device:Activity:Position	{ DisplayDown, DisplayUp, AntennaDown, AntennaUp, SidewaysRight, SidewaysLeft }
Context type (higher-level)	**Context value**
Environment:Location:Building	{ Indoors, Outdoors }

ing in continuous valued fuzzy labeling (Figure 2b). For example, $\mu_L(x)$ = 0.7 / Silent + 0.3 / Moderate + 0 / Loud.

The manager's recognition service table registers plug-in context recognition services, which lets applications share the recognized higher-level contexts. Developers or processes can add and remove recognition services from the system online. The registration process defines each recognizer's input and output, so that any change in a recognizer's input data set automatically spurs the recognition of higher-level context. The application can operate by using the higher-level contexts without needing to know about the underlying processing. Context recognition services use as input either a set at a certain time instant or a time series of context atoms, and return single higher-level contexts for the context manager.

In other words, the resource server and recognition service convert an unstructured raw measurement data flow into a representation defined in the context ontology,[2] which permits serving the human-interpretable context information for the applications in an event-based manner. The framework thus provides a semantic interface that enables more systematic and rapid application development and facilitates more efficient reuse of context information compared to using raw measurement data.

The framework in Figure 1 also contains blocks for change detection and security, but we don't focus on these functionalities here. Briefly, any change detection services facilitate alternative ways to detect context change. The security module checks the trustworthiness of incoming contexts, which is especially important for contexts received from outside the terminal.

Information sources

A mobile device can acquire context information from many possible sources. In our experiments, we used sensors including a microphone, three accelerometers, two channels for light, and sensors for temperature, humidity, and touch. Sensor data provided the starting point—we could also use other context sources available for the terminal—and we designed the framework to handle contexts from such sources as

- Device processes that represent mobile devices' internal information, such as applications currently running
- The Internet and other networked resources
- The Global Positioning System, among the most important sources of location information
- Internal device processes, which provide explicit information on device application use and users' scheduled tasks, preferences, and social networks
- Time information, used to associate certain events with others and form event sequences that might relate to a current higher-level context or predict a future event

Mobile devices that support various wireless networks, such as GSM and GPRS, and short-range ad hoc networks, such as Bluetooth, provide continuous and ad hoc connectivity to local and remote data.

Ontology for sensor-based context information

To manage the context information systematically, the framework entities must have a common structure for representing information. We therefore designed an ontology for representing sensor-based context information.[2] The ontology (Table 1) consists of a schema—which represents the structure and the properties for all the ontology's concepts—and a client-usable, extendable vocabulary that presents the terms for describing context information. To facilitate ontology sharing and communication, we used the Resource Description Framework (www.w3c.org) as the description syntax. RDF offers common properties and syntax for describing information and permits sharing the ontology among different information providers or sharing context between devices communicating collaboratively.

TABLE 2

Audio-based context ontology vocabulary example.

Higher-level contexts are presented separately at the bottom of the table; the rest are atoms the resource server extracts from sensor data.

Context type	Context value
Environment:Sound:Harmonicity	{ Low, Medium, High }
Environment:Sound:HarmonicityUpperLimit	{ None, Low, Medium, High }
Environment:Sound:SpectralCentroid	{ UltraLow, Low, Medium, High }
Environment:Sound:Transients	{ None, Transients }
Environment:Sound:CombFilterLength	{ Low, High }
Environment:Sound:HighHarmonicityRatio	{ Low, Medium, High }
Environment:Sound:UpperLimitRatio	{ Low, Medium, High }
Environment:Sound:HistoryF0Ratio	{ Low, High }
Environment:Sound:HistoryTransients	{ None, Medium, High }
Environment:Sound:HistoryUpperLimitRatio	{ Low, High }
Environment:Sound:SpectralSpread	{ Low, Medium, High }
Environment:Sound:SpectralSpreadRatio	{ Low, Medium, High }
Environment:Sound:SpectralFlatnessOfFourthBand	{ Low, High }
Environment:Sound:SpectralCentroidDeviation	{ Low, Medium, High }
Environment:Sound:FundamentalFrequencyRatio	{ Low, High }
Environment:Sound:HistoryHarmonicityDeviation	{ Low, Medium, High }
Environment:Sound:HistoryCombFilterLength	{ Low, Medium, High }
Environment:Sound:ConsecutiveFundamentalFrequency	{ Low, High }
Environment:Sound:LowEnergyRatio	{ Low, High }
Context type (higher-level)	**Context value**
Environment:Sound:Type	{ Car, Elevator, RockMusic, ClassicalMusic, TapWater, Speech, OtherSound }

The framework describes each context using six properties. Each context expression must contain at least *Context type* and *Context value*:

- *Context type* refers to the context category. All subscriptions and queries must have context type or source as the primary parameter. Context type concepts form a tree structure.
- *Context value* refers to the semantic or absolute "value" of context type and is usually used together with *context type*, forming a verbal description (Tables 1 and 2). In some cases, *context value* might contain an absolute numerical value or feature describing context.
- *Confidence*, an optional property, describes the context's uncertainty, typically as a probability or a fuzzy membership of context depending on the source.
- *Source* describes the context's (semantic) source. A client interested in contexts from a specific source can use this property.
- *Timestamp* denotes the latest time the context occurred.
- *Attributes* specify the context expression freely and might contain any additional details not included in the other properties.

Higher-level context and context atoms differ only in that the former's *Context type* property links to a recognition service description that contains the *Input context type set* for recognizing the higher-level context. Hence, the following three properties describe recognition services:

- *Context type*: a client can use the category of the higher-level context as a normal context atom or specifically request the recognition for the type.
- *Context recognition service*: the recognition service takes as input the contexts defined in the *Input context type set*, performs the classification, and returns the higher-level context description.
- *Input context type set*: the recognition service subscribes to the input context type set to be notified of the changes. Upon change in input type set, the recognition service infers the higher-level context.

The structure permits creating and handling composite contexts and context hierarchies. The client can use all contexts similarly despite the possible underlying hierarchy, making recognition processing transparent for the client. Table 1 shows the fusion of the two higher-level contexts (last row of the table) using naive Bayesian classification from a set (vector) of 14 atoms describing the environment's light, humidity, and temperature. In Table 2, the seven context values in the last row describe higher-level contexts fused from a set of 47 audio-based context atoms.

Both Table 1 and Table 2 examples form part of the sensor-based context vocabulary, which we designed to be expandable. We've identified and can measure or recognize about 80 contexts that currently consist mostly of context atoms. The contexts result from measuring and processing audio, light, three-axis acceleration, temperature, humidity, and touch. The audio context atoms

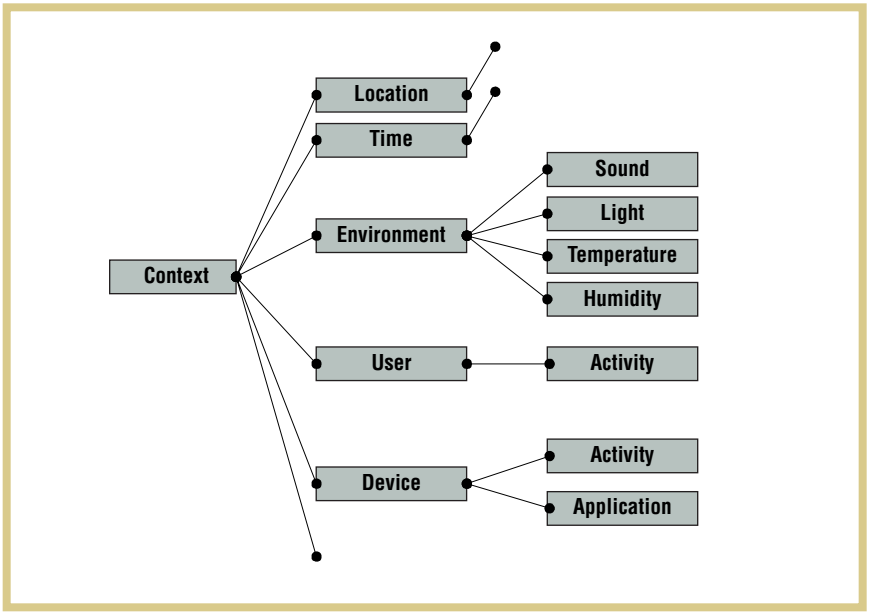


Figure 3. The expandable context vocabulary's main categories.

use algorithms from the upcoming MPEG-7 standard. We chose context atoms for the vocabulary based on their ability to describe some potentially useful real-world properties. Other criteria include the feasibility of measuring or recognizing the chosen context as accurately and unambiguously as possible.

Most of the context atoms are accurate in that they represent more or less direct low-level descriptors of the measured data. However, this isn't always the case. Some context types, such as semantic temperature, are subjective and their meaning depends on the observer. Furthermore, higher-level contexts always incorporate some uncertainty in the form of inferred probability based on previously learned evidence. Even worse, both low- and high-level contexts often ambiguously reflect real-world conditions. Hence, the use of recognized contexts is not always straightforward.

The extendable vocabularies let developers add both context types and their values. Figure 3 shows the context vocabularies' main concept categories. The representation corresponds to the list representation in Tables 1 and 2, and colons separate subcategories. Current vocabulary sets are also available online (www.iie.fi/tie/publications/publications_index.htm). We address naming conventions elsewhere.[2]

Among its most important tasks, the framework manages uncertainty of sensor-based information. As mentioned earlier, the *Confidence* property describes context uncertainty: a context derived from multiple sensory input only holds true with a certain probability. Similarly, a context might be only partly true when the boundary between two contexts is not clear and discrete, in which case a fuzzy membership associated with the context indicates its partial truth.

Context recognition uses confidence properties of the input data set to form the classifier input vector. If no confidence property value exists, the framework uses either zero or one on the basis of the corresponding context atom's presence.

Inference mechanism concept

The conceptual blackboard-based system allows the use of any inference framework for reasoning new contexts or actions. Context uncertainty described in the confidence property can prove useful in such inference frameworks as probabilistic networks, clustering, or case-based reasoning.[3] The central blackboard manager distributes the inference and makes it transparent to the client. The recognition services take information from the common dataspace, process it, and return more abstract information. Developers don't need to restrict the services' internal processing methods if the services return information as defined by the common ontology. The recognition services produce higher-level contexts from either a set of contexts at a certain moment or a context history.

Bayesian reasoning for higher-level contexts

A naive Bayes classifier recognizes higher-level contexts (Tables 1 and 2) from lower-level context atoms.[4] In the framework, the classifier could serve as a plug-in context recognition service that takes a set of context atoms as input from the context manager, performs the classification, and returns a higher-level context to the manager. The manager stores it and delivers it to clients based on requests and subscriptions. The naive Bayes classifier works well for online context inference and sensor fusion because

- It has proven robust even with missing, uncertain, and incomplete information.[5]
- For input, it can use context data described by the ontology, a vector of context atom confidence values. Fuzzy membership values can be applied as virtual evidence.[5]
- It is computationally efficient. Training and inference both have a linear complexity in input data size.
- It requires no background information modeling except for choosing the relevant network inputs. For instance, background knowledge tells us it is irrational to try to infer the type of ambient music from the context atoms describing device stability, even though it might happen to be a discriminating factor based on the data set.

Table 3 presents examples of contexts

TABLE 3
Examples of contexts from various situations as they appear on the context manager blackboard. Clients employ the contexts by using the API.

Example context	Context type	Context value	Confidence	Source	Attributes
1	Environment: Sound: Type	Car	1.0	Recognition Service 1	Confidence = Probability
2	Environment: Sound: Type	Elevator	0.9	Recognition Service 1	Confidence = Probability
3	Environment: Sound: Type	Speech	0.8	Recognition Service 1	Confidence = Probability
4	Environment: Location: Building	Outdoors	1.0	Recognition Service 2	Confidence = Probability
5	Environment: Temperature: Absolute	21	1.0	Device Sensor	ValueUnit = Celsius
6	Environment: Humidity	Dry	0.7	Device Sensor	Confidence = Fuzzy membership
7	Device: Activity: Placement	AtHand	1.0	Device Sensor	Confidence = Crisp
8	User: Activity: PeriodicMovement	Walking Frequency	0.6	Device Sensor	Confidence = Fuzzy membership; Speed = 5; SpeedUnit = km/h

that a naive Bayes classifier can recognize. The context manager blackboard contains contexts in the format shown, with only security code and timestamp omitted. Context atoms and higher-level contexts both use the same representation. The ontology defines context structure and vocabulary.

Two separate naive Bayesian networks produce the first four examples Table 3 shows. Audio context recognition is based on 47 lower-level context atoms. Within the framework, whenever any input-set atoms change, the recognition service performs the classification for subscribed clients. In the first four examples, where the context source is the Bayesian network recognition service, the context confidence attribute states the context probability as shown by the free attributes field. In examples 5, 6, and 8, confidence states the context's fuzzy membership value, and example 7 results from a crisp quantization.

Figure 4 shows the formation of the

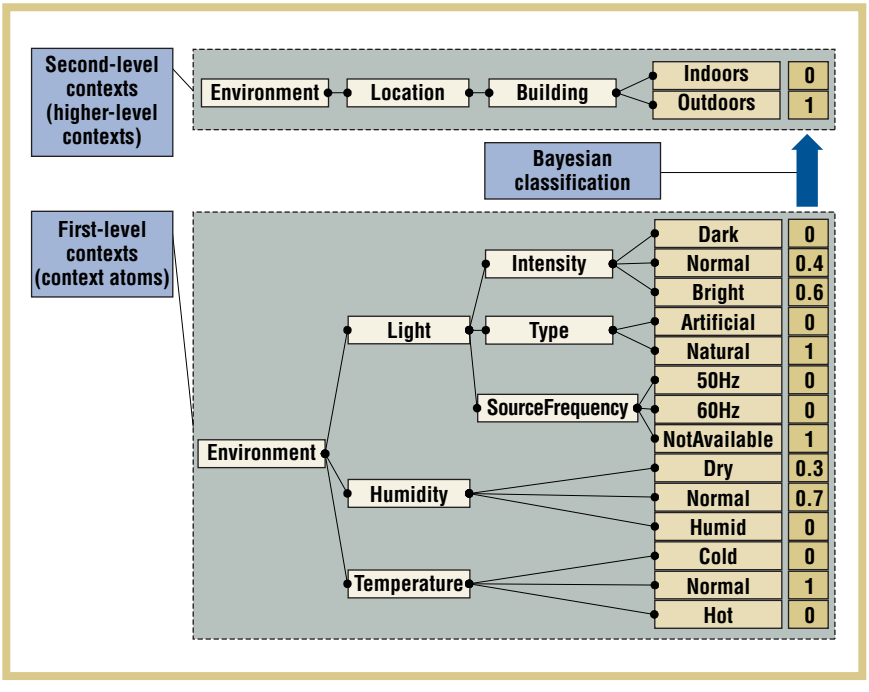


Figure 4. The formation of the higher-level context *Outdoors* from the context atoms. White rectangular boxes represent types and light tan boxes represent context values. Dark tan boxes contain the corresponding confidence instance values for the current situation. A naive Bayesian network classifies the confidence instance values into one of the previously defined output classes, *Indoors* and *Outdoors* in this network. The audio network uses a similar principle but with an input vector length of 47 and seven output classes.

higher-level context *Outdoors* (example 4) from the context atoms. The Indoor/Outdoor Bayesian network (recognition service 2) classifies the current situation's context atom confidence values (vector). The classifier associates each context atom with a conditional probability indicating each input's probability given the output. The classifier learns conditional probabilities from the training data. The classification applies the Bayes theorem to calculate the output class given current input values and conditional probabilities.

Our earlier study[4] applied naive Bayesian networks to classify nine contexts of a mobile device user in their normal daily activities. We based these contexts on an extensive set of audio context atoms derived partly from the upcoming MPEG-7 standard's algorithms and partly from other sensors' context atoms. We sought to recognize the nine contexts (Table 1 and 2, higher-level contexts) measured from a continuous real-life scenario containing different activities such as driving a car, running, walking, using an elevator, listening to different kinds of music, and speaking. The measurement system hardware consisted of a small sensor box attached to the shoulder strap of a backpack containing a laptop. When collecting scenario data, the user carried the backpack.

The classification results indicated that the naive Bayes classifier can extract situations fairly well, even from continuous input, but also showed that most results will likely be valid only in a restricted scenario. Under controlled conditions, recognition accuracy for the nine contexts measured 96 percent true positives and 100 percent true negatives. In real-world conditions, where the classifier didn't know context transitions and where disturbances, undefined phenomena, and scenario phase transitions decreased the performance, the overall recognition accuracy fell to 87 percent true positives and 95 percent true negatives. These were averaged over nine 8-minute scenarios and nine different contexts, and measured by four testees. However, the classifier recognized some audio-based contexts, such as *Car*, *Elevator*, and *Tapwater*, with nearly 100 percent accuracy.

From the machine-learning viewpoint, the study took the supervised learning approach, which defines the target classes and their inputs before training.

From the machine-learning viewpoint, the study took the supervised learning approach, which defines the target classes and their inputs before training. After training, the classifier could be used to recognize the trained contexts online as a registered recognition service. However, we still faced the traditional machine-learning problem of generalization beyond training data, in addition to possible ambiguities, because in a wider perspective some features might refer to multiple real-world situations.

Application programming interface

The framework lets clients put context information to practical use once the resource servers and recognizers have processed it to a suitable level of abstraction. The context manager API, which we've implemented on the Symbian platform, offers several services.

Adding context

Any secure client can add context to the context blackboard. Resource servers process and collect the context properties into context objects, then send them to the context manager. Device-external sources should use a message syntax such as RDF that facilitates information sharing. The following example shows the method for adding context:

```
ContextAdd( ContextObject )
```

Requests and responses

The client may request context information directly from the context manager database, or specifically request context recognition. The application developer only needs to know the context ontology and the methods. The client may also request context information by the context ontology tree subbranch (shown below), in which case the response will contain all contexts under the subbranch. The following examples denote responses to requests with an arrow (→) and show only the *Context value* attributes of any response objects for clarity.

```
ContextRequest( Environment:Light:Intensity )
→ Bright
ContextRequest( Environment:Light )
→ {Bright, Natural, NotAvailable}
ContextSetRequest( {Environment:Humidity,
    Device:Activity:Position}
→ {Dry, AntennaUp}
ContextSetRecognitionRequest(
    Environment:Sound:Type )
→ Car
```

The context set request lets the application developer get all the required contexts by using one command instead of many. The recognition register defines the input context types used to recognize the *Environment:Sound:Type* higher-level context.

In addition to the examples, the API

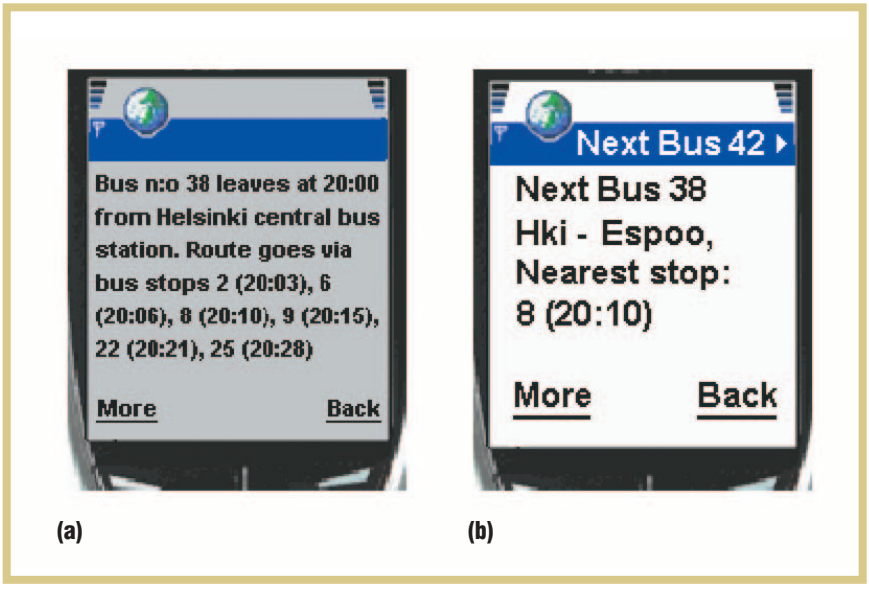


Figure 5. Adaptive information representation on the display of a mobile terminal during service browsing. Current contexts in screenshots are (a) *Environment:Light:Intensity Normal; User:Activity:PeriodicMovement NotAvailable;* and (b) *User:Activity:PeriodicMovement RunningFrequency*.

contains methods for requesting several latest contexts or a time interval of a certain context type.

Subscriptions and indications

The subscription-indication service gives applications the required context information in an event-based manner. Applications can subscribe to context change notifications—clients essentially tell the context manager, "when something about this happens, let me know." The following examples illustrate how applications use this service (again showing only the indication object's *Context value* attribute for clarity).

```
ContextChangeSubscribe( Device:Activity:Placement )
→ AtHand
ContextChangeSubscribe(
    Environment:Temperature:Absolute )
→ 21
ContextStartSubscribe( Location:Facility, MovieTheatre )
→ MovieTheatre
ContextChangeSubscribe(
    Environment:Sound:Type )
→ Elevator
```

The first example simply notifies the client when device placement changes. The next example, subscribing to context's absolute value, only proves suitable for context types properly treated for change by the source because this service indicates every time the value changes. The next example, subscribing to start and end notifications for a specific context, can prove especially useful for a large context set, such as location. For instance, an application might only need the context *MovieTheatre* to change the device profile to silent upon arrival and back to previous after leaving. Subscribing to context start and end relieves the application from receiving unnecessary indications about all changes in the context type—the application will only get the relevant messages.

In the final example, the client has subscribed to a recognition from a context set. The client gives the resulting higher-level context type. The recognition register defines the set of recognition service input subcontexts. Hence, whenever any context in the input set changes, recognition occurs automatically and, if the recognized higher-level context changes, the client receives the result. Similarly, methods exist for recognition based on the sequence of previous values for a certain type.

Example application

We've used the context ontology to demonstrate the use of sensor-based context information in mobile terminal applications such as browsing and information presentation in the user interface.[6] The approach uses part of the ontology vocabulary, particularly the context types *Environment:Light:Intensity*, *Environment:Sound:Intensity*, and *User:Activity:PeriodicMovement*. Environment sound intensity controls the volume of operating tones. Font size, screen brightness, and service content adapt according to user activity and ambient light level. We've chosen these context types and variables empirically based on tests with real sensor measurements. Figure 5 shows context-based information representations of a bus timetable service.

Fuzzy controllers, designed using fuzzy rules that are basic operations for fuzzy sets, help guide the application adaptation. For example, the fuzzy rule definition for adapting font size using the logical operation AND uses the membership of the intersection between two fuzzy sets—*Environment:Light:Intensity* and *User:Activity:PeriodicMovement*:

$$\mu_{FONTSIZE}(x) = \min(\mu_{USER:ACTIVITY:PERIODICMOVEMENT}(x), \mu_{ENVIRONMENT:LIGHT:INTENSITY}(x)).$$

Fuzzy rule definition also uses the logical operations OR and NOT, which permit conversion of multivariable rules into single continuous control signals for applications.

We employed context-based application adaptation, validated with users, to demonstrate our sensor-based context ontology.[6] User feedback revealed that direct adaptation of minor application features such as display illumination is

acceptable, but the device must ask the user before adapting major application features such as information presentation. Further studies will let us more extensively validate the whole framework from multiple viewpoints such as application developer, recognition service developer, and application user.

Discussion

Our framework and context ontology provide a semantic application programming interface for handling imprecise information from multiple sources. Many obstacles remain, however, to the rapid use of many contexts derived from sensor data. We need more sensors and a large data collection to gain enough information to discriminate reliably between higher-level contexts in a general mobile device usage setting. Also, some contexts are ambiguous or subjective and, as such, application-specific.

The blackboard mechanism is widely used for communicating information between entities and has proven suitable for many problem domains. However, it isn't the most efficient method because every communication through a blackboard requires at least two hops. We must therefore add resource server information processing requirements. The sources should process the context data to a sufficient abstraction level before adding it on the blackboard and shouldn't add context data too frequently.

Even though the context manager can handle the continuous flow of raw data from source to client, several factors argue against this. First, event-based communication of high-abstraction-level data is optimal for most context-aware applications. Applications can subscribe to context change notifications instead of having to

Related Work on Context Extraction and Frameworks

Numerous studies discuss context extraction in mobile computing, and research exploring wearable-computer context recognition has used such methods as wearable cameras, environmental audio signal processing, and Hidden Markov Models.[1] Executing context recognition from multidimensional sensor signals, for example, combines the benefits of neural networks and Markov models.[2] Processing multiple sensors' information as multidimensional context vectors permits extracting and compressing relevant context information using statistical methods.[3] Time series segmentation can be used to extract higher-level context descriptions.[4]

Context recognition itself is not enough; we must also deliver the extracted information to the applications in a practical manner. Context frameworks can facilitate the systematic development of context-aware applications. The Context Toolkit[5] offers one solution for supporting context-aware application prototyping. Its design separates context acquisition from use and supports context interpretation, distributed communication, constant context availability, context storage, and resource discovery. The Context Toolkit represents one of many ways to approach the context framework problem.

Researchers have proposed many models for coordinating multiple interoperating components. Terry Winograd[6] divides these models into three groups—widget, client–server, and blackboard models—with the latter proposed as an alternative context framework model.[7] Blackboard architecture is a heritage from AI research.[8]

REFERENCES

1. A. Schmidt et al., "Advanced Interaction in Context," *Proc. Int'l Symp. Handheld and Ubiquitous Computing, LNCS 1707*, Springer-Verlag, 1999, pp. 89–101.
2. K. Van Laerhoven and O. Cakmakci, "What Shall We Teach Our Pants?" *Proc. 4th Int'l Symp. Wearable Computers*, IEEE CS Press, 2000, pp. 77–83.
3. J. Himberg, J. Mäntyjärvi, and P. Korpipää, "Using PCA and ICA for Exploratory Data Analysis in Situation Awareness," *Proc. IEEE Conf. Multisensor Fusion and Integration for Intelligent Systems*, IEEE CS Press, 2001, pp. 127–131.
4. J. Himberg et al., "Time Series Segmentation for Context Recognition in Mobile Devices," *Proc. IEEE Conf. Data Mining*, IEEE CS Press, 2001, pp. 203–210.
5. 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*, vol. 16, nos. 2–4 (special issue on context-aware computing), Dec. 2001, pp. 97–166.
6. T. Winograd, "Architectures for Context," *Human-Computer Interaction*, vol. 16, nos. 2–4 (special issue on context-aware computing), Dec. 2001, pp. 401–419.
7. A. Fox et al., "Integrating Information Appliances into an Interactive Workspace," *IEEE Computer Graphics & Applications*, vol. 20, no. 3, July/Aug. 2000, pp. 54–65.
8. R. Engelmore and T. Morgan, eds., *Blackboard Systems*, Addison-Wesley, 1988.

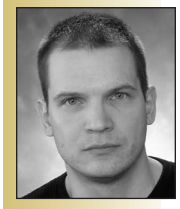
process a continuous flow of low-level measurement data. Second, communicating low-level data up to a client continuously consumes more processing power and, correspondingly, the limited battery resources. Third, the raw data flow would quickly fill the history space reserved for each context type in the context database. Optimal use of the context manager therefore requires defining practices for developing the sources allowed to add contexts.

Using the widely adopted Symbian platform and context manager data storage capabilities, mobile systems will soon be able to effortlessly collect real-world context data from multiple users. Analyzing the collected data will also help us develop more reliable, generic higher-level context recognition engines capable of delivering more accurate context information to applications. Moreover, the framework will make application development itself more systematic and facilitate reuse of contexts and the modules that produce them. Creating real mobile context-aware applications using the API will validate the approach and evaluate the framework's suitability for different application domains.

REFERENCES

1. R. Engelmore and T. Morgan, eds., *Blackboard Systems*, Addison-Wesley, 1988.
2. P. Korpipää and J. Mäntyjärvi, "An Ontology for Mobile Device Sensor-Based Context Awareness," *Proc. Context '03*, *LNAI no. 2680*, Springer-Verlag, 2003, pp. 451–459.
3. T. Mitchell, *Machine Learning*, McGraw-Hill, 1997.
4. P. Korpipää et al., "Bayesian Approach to Sensor-Based Context Awareness," *Personal and Ubiquitous Computing J.*, vol. 7, no. 4, 2003.
5. J. Pearl, *Probabilistic Reasoning in Intelligent Systems*, Morgan Kaufmann, 1988.
6. J. Mäntyjärvi and T. Seppänen, "Adapting Applications in Handheld Devices Using Fuzzy Context Information," *Interacting with Computers J.*, vol. 15, no. 4, 2003, pp. 521–538.

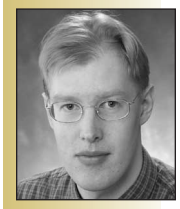
the AUTHORS



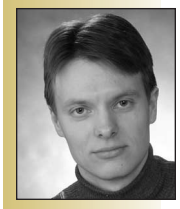
Panu Korpipää is a research scientist at the Technical Research Center of Finland, VTT Electronics, in the Knowledge Engineering group. His current professional interests include research for increasing the degree of context awareness in mobile devices and developing visual interfaces for multidimensional data. He received his MSc in electrical engineering from the University of Oulu and is working toward a PhD in the Department of Electrical and Information Engineering at the University of Oulu. Contact him at VTT Technical Research Centre of Finland, VTT Electronics, P.O. Box 1100, FIN-90571 Oulu, Finland; panu.korpipaa@vtt.fi.

Jani Mäntyjärvi is a senior research scientist at the Technical Research Center of Finland, VTT Electronics, in the Knowledge Engineering group. His current professional interests include pervasive and context-aware computing for handheld devices and technologies for adaptive user interaction. He received his MSc in biophysics from the University of Oulu and is working toward a PhD in the Department of Electrical and Information Engineering at the University of Oulu. Contact him at the VTT Technical Research Centre of Finland, VTT Electronics, PO Box 1100, FIN-90571 Oulu, Finland; jani.mantyjarvi@vtt.fi.

Juha Kela is a research scientist at the Technical Research Center of Finland, VTT Electronics, in the Knowledge Engineering group. His research interests include multimodal and mobile interaction. He received his MSc in electrical engineering from the University of Oulu and is working toward a PhD in the Department of Electrical and Information Engineering at the University of Oulu. Contact him at VTT Technical Research Centre, VTT Electronics, PO Box 1100, FIN-90571 Oulu, Finland; juha.kela@vtt.fi.



Heikki Keränen is a research scientist at the Technical Research Center of Finland, VTT Electronics, in the Knowledge Engineering group. His research interests include usability, user interface software, and software engineering. He received his MSc in information engineering from the University of Oulu and is working toward a PhD in the Department of Information Processing Science at the University of Oulu. Contact him at the VTT Technical Research Centre, VTT Electronics, PO Box 1100, FIN-90571 Oulu, Finland, heikki.keranen@vtt.fi.



Esko-Juhani Malm is a research scientist at the Technical Research Center of Finland, VTT Electronics, in the Knowledge Engineering group. His professional interests include embedded software engineering and pervasive computing. He received his MSc in electrical engineering from the University of Oulu. Contact him at the VTT Technical Research Centre, VTT Electronics, PO Box 1100, FIN-90571 Oulu, Finland; juhani.malm@vtt.fi.