Supporting Context-Aware Mobile Applications: An Infrastructure Approach

Marten J. van Sinderen and Aart T. van Halteren, University of Twente
Maarten Wegdam and Hendrik B. Meeuwissen, Lucent Technologies
E. Henk Eertink, Telematica Instituut

ABSTRACT

Mobile phones and PDAs are converging into mobile lifestyle devices that offer a wide range of applications to end users. Many of these applications will have the ability to adapt themselves to the user's situation, commonly referred to as context awareness. We argue that an infrastructure is needed to enable wide deployment of context-aware applications. A major benefit is interoperability between heterogeneous context sources and applications in a privacy-sensitive way. We identify three main technical challenges to realize such an infrastructure:

- Reasoning to infer higher-level and better quality context information
- Efficient exchange and distributed processing of context information in dynamic and pervasive environments
- End-user-controlled handling of the privacy aspects

This article explains how we address these challenges with regard to the realization of an infrastructure that supports context-aware mobile applications. We use this infrastructure to support several mobile healthcare applications.

INTRODUCTION

The combination of mobility, context-awareness, and proactiveness enables a new class of user-centric and mobile lifestyle applications that become increasingly important.

Mobile applications provide their services in a seamless and uninterrupted way to end users that are "on the move." Context-aware applications exploit context to adapt the timing, quality and functionality to the users' situation and resources availability. Proactive (or attentive) applications provide services on their own initiative (e.g., based on context-dependent conditions).

According to [1]: "Context is any information that can be used to characterize the situation of an entity." As in [1], we see an entity as something with a real existence, which is relevant for the service delivery of applications (e.g., a person, place, physical component, or computational component). We add the concept of "quality of context" (QoC) [2] to express the quality characteristics of the context information. Examples of context are user activity, geo-location, GSM cell ID, speed and direction, battery power, idle time, available networks, favorite places, and method of transportation.

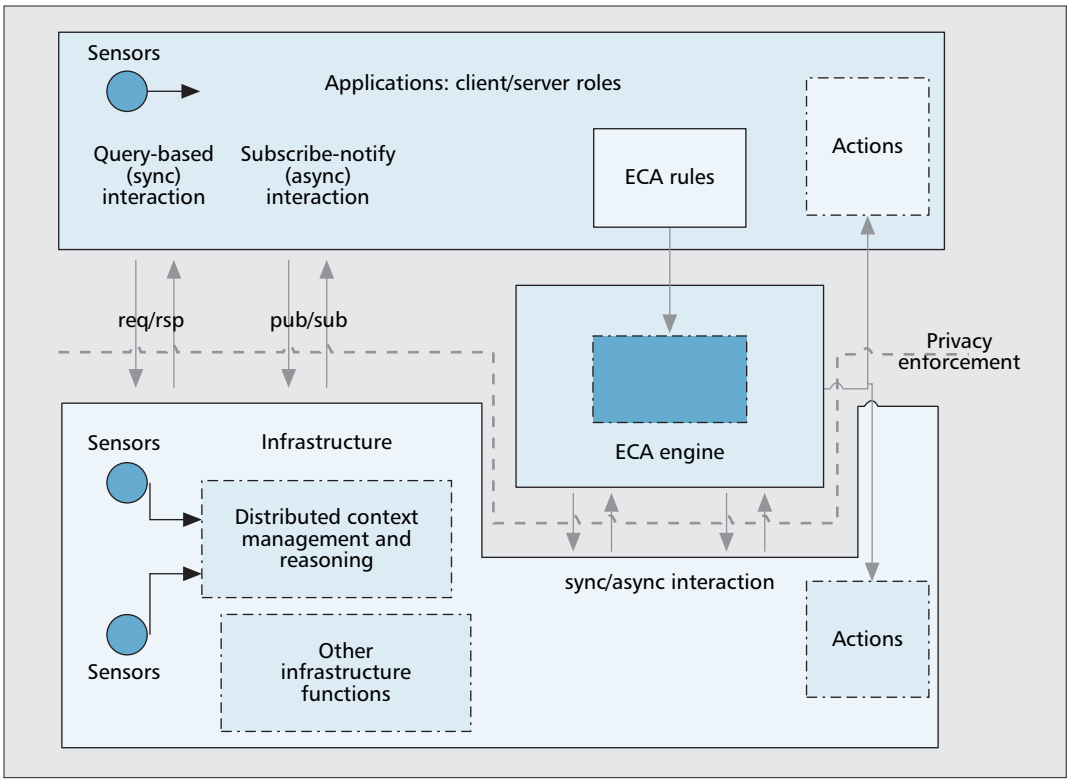
The main objective of this article is to present the required and desirable functionality of an infrastructure that supports context-aware applications. The reasons for such an infrastructure include:

- Cost reduction, by sharing context sources and context management functionality
- Handling complexity, by taking care of context processing tasks that are too "heavy" for lightweight application devices
- Efficiency, by timely and resource-efficient aggregation of information from several physically distributed context sources
- Richer functionality, by providing dynamic discovery of (new) context sources, and supporting information exchange between such sources and applications
- Trust, by acting as the single entity for end users to enforce their privacy policies across the environments in which they roam

Figure 1 shows the basic layered architecture of the proposed infrastructure, as detailed below.

The application layer consists of context-aware, proactive, and mobile applications. It also comprises sensors that provide application-spe-

0163-6804/06/$20.00 © 2006 IEEE



The infrastructure layer entails functionality for supporting applications. The heart of the infrastructure layer is the functionality for distributed context management and reasoning, but also other functions such as identity management, storage, and discovery are supported.

Figure 1. *Basic layered architecture.*

cific context information to the applications. Applications generally are divided in client and server components that can interact with each other and with the infrastructure using one of the interaction patterns supported by the infrastructure. Applications may delegate context-dependent behavior to the infrastructure, and application-specific actions may be invoked from the infrastructure as part of the execution of delegated behavior.

The Event-Condition-Action (ECA) engine is responsible for delegated application behavior. This includes interaction with the infrastructure layer to become informed of context events, and invocation of either application-specific or infrastructure-provided actions triggered by context events, according to the delegated behavior specification. Each unit of delegated application behavior executes within its own security context to ensure that context information is only communicated with consent of the application.

Privacy enforcement ensures that privacy-sensitive information that is trusted to the infrastructure can only be accessed according to the privacy policies that have been defined or agreed upon by the end user (owner or subject of the information). The entity to which the context information refers can be anonymised or the context information itself can be obfuscated (reducing the QoC).

The infrastructure layer entails functionality for supporting applications. The heart of the infrastructure layer is the functionality for distributed context management and reasoning, but also other functions such as identity management, storage, and discovery are supported. The infrastructure layer provides interaction patterns to the application layer and the ECA engine to allow access to this functionality and to support end-to-end communication. It also comprises sensors that provide infrastructure-specific context information to the context management and reasoning functionality. The infrastructure layer internally uses any combination of wireless and wired network technology to accomplish the required connectivity for communication.

To validate and demonstrate the architecture, we developed applications for telemonitoring of epileptic seizures, telemonitoring of spasticity, and teletreatment of chronic pain. This article focuses on the infrastructural support of these applications (see [3], D1.3v2, for details).

A recent comprehensive overview of other approaches toward context management can be found in [5]. Our architecture addresses several weaknesses identified in [5]. In particular we mention here the integration of privacy and trust, the support of personal mobility and fault-tolerance through ad-hoc discovery of context sources, and reasoning based on quality attributes of context information. Another approach that is similar to ours has been developed in MobiLife [9]. The MobiLife architecture differs in the choices of communication protocols, has less focus on integrating security and privacy mechanisms, and lacks rule-based interfaces towards applications.

This article elaborates the basic layered architecture (Fig. 1) while successively addressing the identified research challenges:

- Reasoning to infer higher-level and better quality context information
- Efficient exchange and distributed processing of context information
- End-user-controlled handling of the privacy aspects

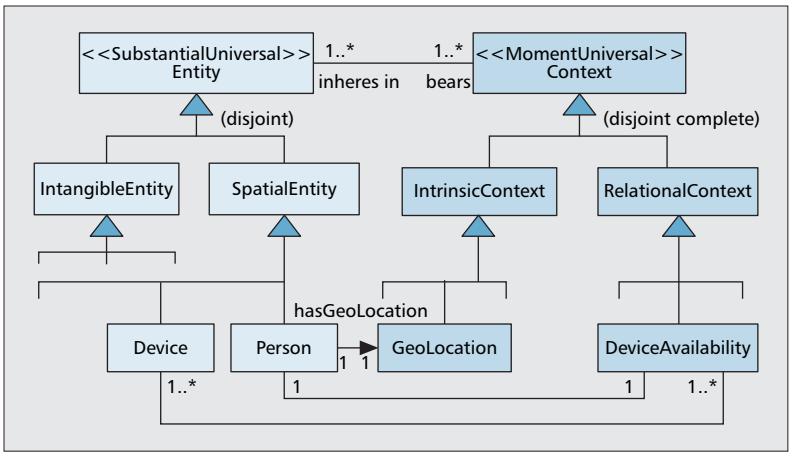


Figure 2. *Entity-context relationship.*

Context Modeling and Reasoning

Various sources provide context information, including body sensors in a Body Area Network, sources on mobile devices, application servers, and network servers. Our infrastructure comprises a general context model to facilitate interoperability of components that handle context information. This model not only addresses semantics, but also qualitative aspects of context information.

Context Modeling

A context model is an information model designed to represent context information. It describes context attributes, their mutual relationships, and their values. The general context model adopted for our infrastructure distinguishes between the concepts of entity and context (see Fig. 2), classified as SubstantialUniversal and MomentUniversal, respectively, which is consistent with the results from conceptual modeling theory [14]. An entity "bears" one or more contexts and a context "inheres in" one or more entities. Context is further classified according to its nature: intrinsic or relational context [12]. Intrinsic context is intrinsically part of a single entity and does not depend on the relationship to other entities (e.g., the geo-location of a person, or the battery power of a device). Relational context is a condition that determines the relationship of multiple entities (e.g., the availability of a collection of devices to a person, or the containment of one or more persons in a room).

Components of the infrastructure and context-aware applications will specialize and extend the general context model. So the common model aims to facilitate interoperability between applications and the infrastructure. Here we use UML class diagrams to describe the general context model, but ontologies provide an alternative means to model context.

When different components exchange context information, a common context data format is needed to which the context information can be mapped. We adopt two alternative formats for this purpose:

- An extension of the existing IETF XML data format PIDF (Presence Information Data Format) with new XML elements and attributes ([3], see D3.12)
- W3C's Resource Description Framework (RDF), embedding the meaning of context information in an ontology

Quality of Context

The contexts of an entity change dynamically. Therefore, not only the value of a context attribute is of interest, but also the moment in time that this value was determined. Moreover, context sources are generally restricted in their ability to estimate the true and precise value of a context attribute. For example, a context source that maintains a decision tree inherently has a particular probability that it will inadvertently yield a wrong value, and a context source that relies on an electronic calendar may not be fully trusted when users do not maintain the calendar properly. To deal with such aspects within the infrastructure, we adopt the notion of QoC, which is meta-information defined as "any information that describes the quality of information that is used as context information" [2]. We focus on the following QoC parameters:

- Accuracy (i.e., the difference between the value of the context attribute and the aspect of reality it represents)
- Probability of correctness (i.e., the probability estimated by the context source that it provides the correct value)
- Trustworthiness, (i.e., the probability estimated by the receiver of a context attribute value that the context source provided the correct value)
- Up-to-dateness (i.e., the age of the context attribute value, typically represented by a time stamp)

We also consider cases in which the infrastructure can only provide the context information to an application at the required quality level by intelligently combining the information from multiple context sources.

Context Reasoning

Some context information of interest to particular context-aware applications cannot be measured or obtained directly, but can only be derived. For example, the information whether or not a person is traveling and, if yes, with which method of transportation, can generally not be sensed directly and therefore has to be inferred from other context information such as GSM cell IDs, WiFi access point IDs, calendar information, and whether or not the person's device is AC connected. This inference of new higher-level context information, as well as predicting future values of context attributes to support proactive applications, is the domain of context reasoning.

The reasoning techniques we apply include ontology reasoning and machine learning. Ontology reasoning encompasses deriving new facts, given both a knowledge base of specified facts and an ontology that formally describes classes and their relations within an application domain. Machine learning techniques (e.g., Bayesian networks, artificial neural networks, or decision trees) can be used to construct models of higher-

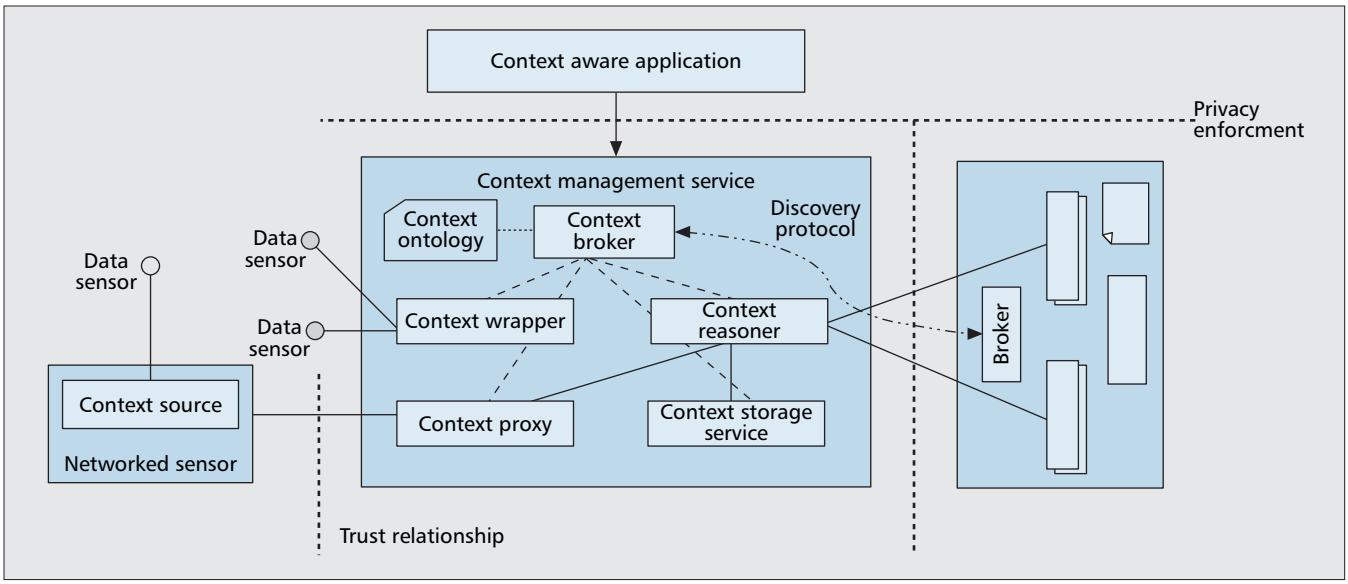


Figure 3. *Context management service.*

level context attributes with corresponding QoC levels, given lower-level attributes.

As an example of combining the strengths of both reasoning techniques, we study the combined use of

- Bayesian network modeling, fuzzy clustering in particular, to convert initial sensor data into a useful context attribute
- Ontology reasoning to obtain higher-level context information

In fact, lat/long coordinates and GSM cell IDs are first used to deduce clusters of GSM cells and estimate a person's location. This location information is then used by an ontology reasoner to derive either other semantics of the person's location, or another context attribute of interest (e.g., by combining it with presence and resource information). The specific ontology in this prototype describes an indoor office environment in detail. The ontology reasoner can be consulted by a context-aware application that is interested in events such as a particular person entering or leaving the office, or a particular person getting close to a certain resource (e.g., a screen in a "public" office space near the coffee machine). Another case study of reasoning is elaborated in [4].

Context reasoning can be performed at different places within our infrastructure (e.g., on the mobile device, in the enterprise domain, or in the operator domain), depending on the availability of the required context information and computing resources. As input for the context-reasoning algorithms, the infrastructure is not only able to keep the most recent values of context attributes, but also their history.

Context Management and its Distribution

The Context Management Service (CMS) is responsible for the secure delivery of context information to other context-aware services and applications. Particular design issues include the distribution of functionality across components (mobile phones, networked sensors) and across different domains (e.g., personal domain, home domain, public network operator domain). We explain the core capabilities of the CMS and discuss distribution aspects in more detail.

Context Management Services

The CMS offers two distinct services: discovery of the supported types of context information, use of a ContextBroker and the access to context information, and use of an abstract ContextSource interface.

ContextBroker Interfaces — The ContextBroker supports three types of discovery interfaces:

- Discovery of context types returns the types of context information that are supported by the CMS. The types are references to a particular ontology.
- Discovery of context sources returns references to the ContextSources.
- Registration of context sources allows applications and services to register their own context sources (together with their types) at the CMS.

The protocol mechanisms used for discovery are not necessarily the same as the protocols used for utilization of the context sources or getting access to context information. The context broker can be seen as a specific type of discovery service (see the "Application Interaction and Adaptation" section).

ContextSource Interfaces — The ContextSource interface offers methods for subscription asynchronous (or event-based) interactions between context-aware applications and the CMS, and query methods.

- In asynchronous methods, applications and services can subscribe to changes to particular context information. The CMS supports Session Initial Protocol (SIP)-compliant interactions and a callback registration function.

The networked sensor is a shrink-wrapped version of a full CMS system that lacks broker-functionality: it is not able to discover other context sources, and context information is only made available through the ContextProxy assigned to it.

- Synchronous (query) methods provide the actual context value. The ContextStorageService supports both SQL queries and RDF Data Query Language (RDQL) queries.

Context Management Components

Figure 3 depicts the main components of the CMS in the "Context Management Service" block. The CMS shows four different types of ContextSources: the ContextWrapper (responsible for obtaining sensed data from data sensors, and making these available in the CMS-managed context information formats), the ContextReasoner (a component that enriches or aggregates context information by combining various other types of context information), the ContextProxy (operational in hiding distribution aspects of ContextSources that logically belong to the same domain, but have constrained resources), and the ContextStorageService (provides persistency, history, and advanced query services; our current implementation is based on a relational database that offers both direct access and a Jena-based reasoning interface).

ContextSources are registered at the ContextBroker, and the types of context information are registered in a context ontology. The ContextBroker is responsible for the discovery of, and access to, the context sources. These components are described in more detail in [5] (see D2.10) and [13].

Distribution of the Context Management Service

We assume that each domain has its own "home" CMS, and that communication between components within a domain occurs over trusted connections. This "domain" concept is a logical notion. For example, particular context sources running on personal devices of employees may logically belong to the corporate CMS or a home-based CMS, or be made available only locally to local context-aware applications (such as a navigation application that uses the CMS to obtain location information obtained from the GPS sensor attached to a personal device). The networked sensor is a shrink-wrapped version of a full CMS system that lacks broker-functionality: it is not able to discover other context sources, and context information is only made available through the ContextProxy assigned to it.

Our architecture may operate in a peer-to-peer fashion. This is supported by means of an interContextBroker discovery protocol, through which particular ContextSources can be discovered that may be able to provide context information about an entity. An example peer-to-peer scenario is one where an individual enters a smart environment, and discovers the local context sources that the personal device will be able to share with its own "home" CMS.

We are currently working on strategies that allow for dynamic distribution of functionality. For instance, in most situations it might be more efficient to perform reasoning functionality close to the ContextStorageService. However, with frequently updated information it might be more efficient to perform (e.g., aggregation) on a personal device. This requires resource management functionality in the architecture depicted in Fig. 3.

The security and privacy issues involved with these distributed aspects are discussed in the "Privacy and Trust" section.

Application Interaction and Adaptation

Our infrastructure provides context-aware applications with support for application interaction and application adaptation. Application interaction support enables (remote) interaction among application components, taking context information into account. The infrastructure uses service discovery to enable the run-time binding of application components before interaction takes place. Application adaptation support is the means to adapt application components and their interactions in accordance to some given context.

Service Discovery

The Service Discovery Service (SDS) is responsible for the context-aware discovery of services. A service can be found by matching the discovery request of a client with published service descriptions. Our infrastructure incorporates the context information of both the client and published services to provide more accurate and relevant matchmaking results, thus enhancing existing service discovery solutions (such as UDDI and Jini). This is particularly important in pervasive computing environments where the number of services may be very large, since each computing device in such environments is a potential (context) service provider.

The SDS can be used to discover both application-layer services and infrastructure-layer services. Specifically, discovery of context sources by the context broker (Fig. 3) can be delegated to the SDS. The SDS distinguishes between instant (or active) discovery and subscribed (or passive) discovery. With instant discovery, the SDS returns zero or more results of the search in a response following the discovery request. With subscribed discovery, the SDS notifies the client of search results during a specified period of time. Notifications are only sent when a first match or better matches than previous matches are found.

Application Interaction

After a client has discovered a suitable service, interaction with this service can be synchronous (query-based), asynchronous (subscribe-notify), or delegated. We focus on the latter type of interaction, as the former two are discussed in the "Context Management and its Distribution" section.

Delegated application interaction presents a flexible way to offload complexity from applications, and to facilitate the introduction of new application functionality at run-time. It entails the specification of application behavior as a set of rules following the Event-Condition-Action (ECA) pattern [6, 7], which is given to the infrastructure for execution. The ECA ingredients are as follows: an *Action* is a (composed) service, an *Event* models an occurrence of interest (e.g., a change of context), and a *Condition* specifies what events must have happened prior to the performance of an Action. The ECA engine is responsible for the execution through its ECA

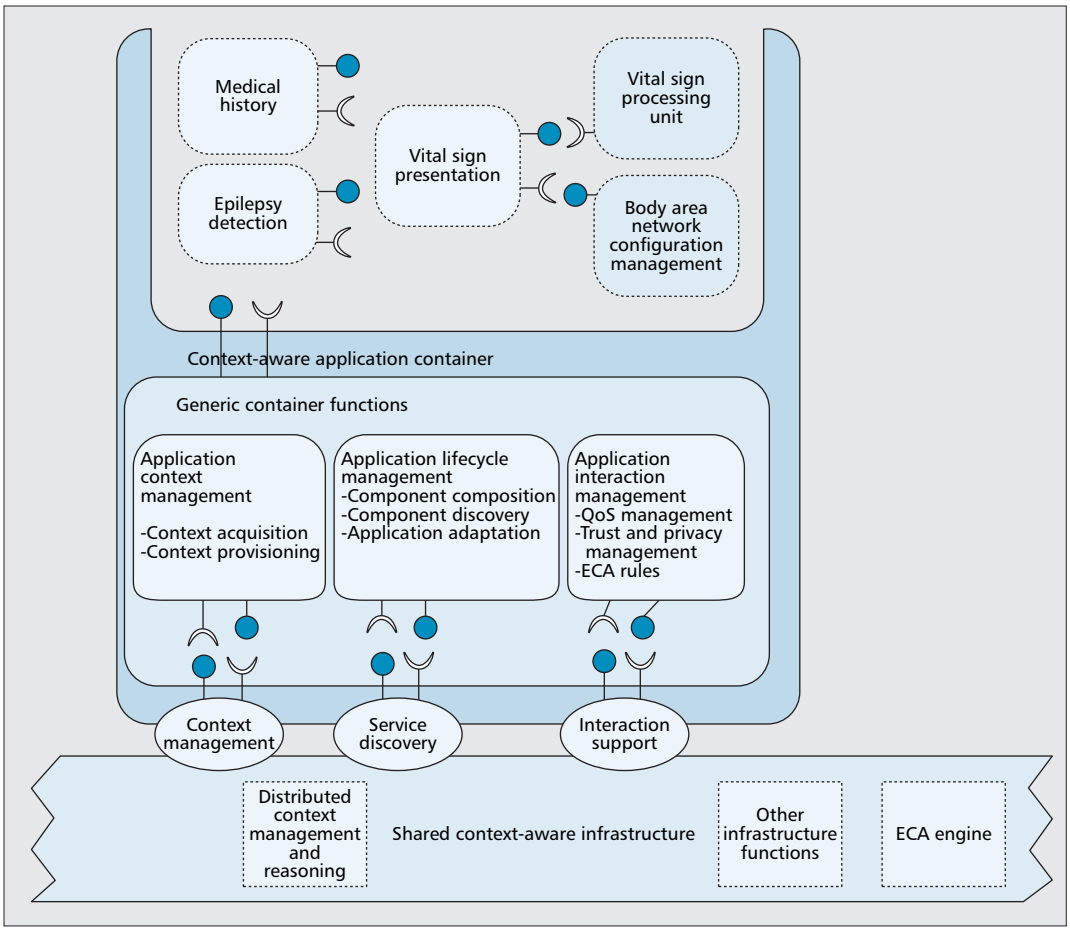


Figure 4. *Context-aware application container.*

To support application adaptation, our infrastructure employs a context-aware application container. The application container facilitates the deployment of context-aware application components and provides an execution environment for these components.

Controlling Service (ECA-CS). The ECA-CS can invoke the SDS to discover context sources for the Events as well as services corresponding to the Actions in the ECA rules specification.

The interactions of ECA-CS with other parts of the infrastructure are subject to the same security (privacy enforcement) measures as the interactions between the application layer and the infrastructure layer (see the "Privacy and Trust" section).

Application Adaptation

To support application adaptation, our infrastructure employs a context-aware application container. The application container facilitates the deployment of context-aware application components and provides an execution environment for these components. A container provides the local environment to the application components either on a (mobile) client device or on a (fixed) application server in the back-end.

Figure 4 depicts an instance of such a container, which provides an execution environment to three example application components (i.e., medical history, epilepsy detection, and vital sign presentation) and two domain specific application components (i.e., a vital sign processing unit and BAN configuration management). Generic container functions are application context management (to exchange application-level context information with infrastructure services), application lifecycle management (to start/stop and activate/deactivate application components), and application interaction management (to support the binding to components deployed in other containers taking the QoS, trust, and privacy requirements for these bindings into account).

This architecture supports application adaptation during several phases of the application lifecycle, through design time and deployment (or configuration) time adaptation, and during runtime.

Privacy and Trust

Context-aware mobile applications face similar security threats as distributed and mobile applications in general, but the privacy and trust aspects are more prominent due to automated collection and processing of context information and the privacy sensitive nature of context. Reasoning about context makes it possible to infer information about the user by combining context information from different context sources, and thus increases the privacy risks. For example, locations from different people can be combined to show with whom somebody meets and at which locations. On the other hand, reasoning capabilities can also reduce the privacy issues by releasing only specific inferred, higher-level, context information that is really needed for the purpose at hand. For example, for certain purposes it may suffice to know whether someone is working or not instead of indicating his location.

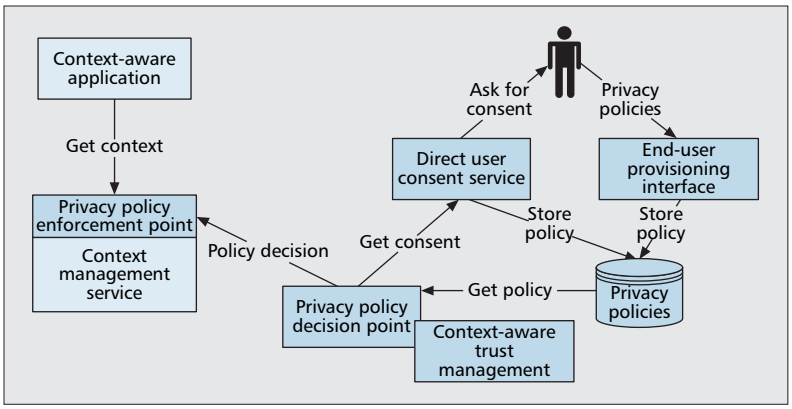


Figure 5. *Direct consent and context-aware trust management.*

Both for achieving user acceptance and for legal reasons, the user needs to be in control of the release of his context information. A context-aware infrastructure should provide the end user with a (logically) central place of privacy control and trust management, contrary to point solutions within different, possibly not-trusted, applications. This does require the end user to trust the infrastructure (context provider). And even with user consent to collect and release context information, both the infrastructure and the applications have to be designed with privacy in mind, and communicate, store, and release no more context information than is needed.

Context-aware Trust Management

Due to its privacy-sensitive nature, context information should only be provided to trusted components. This requires management of trust relationships, which is especially hard in a pervasive environment because users are supposed to interact with components unknown beforehand. Current trust-management solutions are not very suitable for this dynamic environment, as they require previously defined trust relationships in order to operate. In addition to using trust relationships to decide which components can get access with what context information, we use trusted context information to establish trust relationships. In addition, we use an explicit trust concept as part of end-user-controlled privacy policies. By enabling users to define their privacy policies in terms of trust, and dynamically calculating this trust, these privacy policies become both easier to understand and more dynamic (discussed in more detail in [3], D2.10).

Privacy Consent through Direct User Interactions

An end user has to provide detailed privacy policies on what context information can be shared with whom, with what QoC, and under what conditions. Although our context-aware trust management solution described above makes the privacy policies easier to understand and more dynamic, there is still a great deal of configuration needed from the user, for example, for very privacy-sensitive information which the user only wants to release on a case-by-case basis, or for releasing context information for which the trust cannot be calculated.

Existing work on privacy policies typically assumes that the user preprovisions his or her privacy policies. However, users are reluctant to make this effort [10], and in addition preprovisioning is too static for context-aware applications. We therefore propose to extend a preprovisioning approach with a direct user-consent approach in which the context-aware infrastructure interacts with a user when a request for context information is received for which there is not (yet) a policy, or for cases in which the end user's policy requires per-case user consent.

Figure 5 depicts the solution for privacy consent and context-aware trust management that we propose for our infrastructure.

Since the interactions in this solution are intrusive to the end user, we make these interactions context-aware themselves in order to determine the right moment, modality (SMS, Instant Messaging, voice-response, etc.), and device (mobile phone, desktop, etc.) with which to interact with the user.

Scalable Enforcement of Privacy Policies

Interactions between applications and infrastructure services are subject to access-control enforcement, specifically to enforce end-user privacy policies and context-provider access control policies. Because of the sheer number of these interactions and the real-time requirements for these interactions, this enforcement has to be done in a scalable manner and delay should be minimized. The approach we take is to enforce the static, request-independent parts of the access policy during service discovery, and to use signed access tokens that embed the request-dependent part of the access policy to do policy enforcement for each subsequent service request locally and hence efficiently.

Our architecture is scalable because the policy enforcement is done in a completely decentralized manner in which the policy-enforcement functionality does not need to interact with other components in the architecture. In addition, we improve the performance of the policy enforcement since the evaluation of the static parts of the access policy is only done during discovery, contrary to for each interaction. We have prototyped this architecture for Web Services [11].

Concluding Remarks

In this article we have addressed infrastructure support for context-aware applications. Proper support allows for faster and cheaper development of context-aware applications, which are easier to manage and maintain, and can rely on a scalable, reliable, and secure substrate for distributed context management.

Our proposed infrastructure has the following salient features:

- Context modeling and reasoning, which provide a generally applicable context model with the notion of QoC, thus addressing the problem of quality- and time-critical context reasoning.

CONTEXT-AWARE MOBILE HEALTHCARE APPLICATION SCENARIO

Sophie is a young woman who suffers from epilepsy, a neurological disorder in which nerve cells of the brain occasionally release abnormal electrical pulses, leading to so-called seizures. Because the occurrence of a seizure is often sudden and unexpected, patients may feel limited in their daily life and insecure when they are alone. Sophie has recently been provided with an ambulant 24-hour monitoring device, which is part of the Epilepsy Safety System (ESS). Thanks to the ESS, Sophie is no longer bound to traditional institutional care and can live a more normal life without losing her feeling of safety.

ESS uses context information, such as Sophie's location and the location and presence of her caregivers, to deliver personalised care to Sophie. ESS uses information from context sources that wrap various positioning techniques (e.g., a GPS receiver, GSM cell IDs, and WiFi beacon spotting) and infrastructure-provided reasoning services to determine the location of Sophie and her caregivers. In case of a seizure, ESS directs voluntary or professional caregivers to Sophie's location.

ESS uses ECA rules to specify how to react to a seizure event. This includes rules that determine, depending on the severity of a seizure, if only voluntary caregivers or also professional caregivers must be contacted. Other rules determine how to contact a caregiver, for example, whether to send an MMS message or voice-directed instructions about how to reach Sophie.

The infrastructure protects the privacy of the patient's context information and of the voluntary and professional caregivers. The trust and privacy functions ensure that the location and presence information of the caregivers is only provided to the ESS if they are near to the patient and are available to help.

Proper support allows for faster and cheaper development of context-aware applications, which are easier to manage and maintain, and can rely on a scalable, reliable, and secure substrate for distributed context management.

- Context management, which makes the delivery of context information efficient in a highly distributed and dynamic environment with many context sources and many interested components.
- Application interaction and adaptation, which provide multiple interaction patterns, delegation of application behavior to the infrastructure, context-aware service discovery, and support for application adaptation.
- Privacy, which provides end users with dynamic privacy control over their context information.

We use mobile healthcare as the main application area in which to demonstrate and validate our infrastructure. The sidebar presents a typical use of the infrastructure in a mobile healthcare application scenario (based on [8]).

ACKNOWLEDGMENT

This work is part of the Freeband AWARENESS project (http://awareness.freeband.nl/). Freeband is cosponsored by the Dutch government under contract BSIK 03025.

REFERENCES

[1] A. K. Dey and G. D. Abowd, "Towards a Better Understanding of Context and Context Awareness," *Proc. Conf. Human Factors in Comp. Sys.*, The Hague, The Netherlands, Apr. 2000.
[2] T. Buchholz, A. Küpper, and M. Schiffers, "Quality of Context: What It Is and Why We Need It," *Proc. Wksp. HP OpenView Univ. Assn.*, Geneva, Switzerland, July 2003.
[3] AWARENESS project results (see All Results); http://awareness.freeband.nl
[4] H. van Kranenburg *et al.*, "Grounded Contextual Reasoning enabling Innovative Mobile Services," *Proc. 5th Wksp. Apps. and Services in Wireless Networks*, Paris, France, June/July 2005.
[5] H. Henricksen *et al.*, "Middleware or Distributed Context-Aware Systems," *Proc. Int'l. Symp. Distrib. Objects and Apps.*, Springer LNCS 3760, 2005, pp. 846–63.
[6] D. Ipina and E. Katsiri, "An ECA Rule-Matching Service for Simpler Development of Reactive Applications," *IEEE Distrib. Sys. Online*, vol. 2, no. 7, Nov. 2001.
[7] P. Dockhorn Costa, L. Ferreira Pires, and M. van Sinderen, "Architectural Patterns for Context-Aware Services Platforms," *Proc. 2nd Intl. Wksp. Ubiquitous Comp.*, Miami, FL, May 2005.
[8] T. Broens *et al.*, "Towards an Application Framework for Context-aware M-health Applications," *Proc. 11th Open Euro. Summer School*, Madrid, Spain, July 2005.
[9] P. Floreen *et al.*., "Towards a Context Management Framework for MobiLife," *Proc. 14th IST Mobile & Wireless Summit 2005*, Dresden, Germany, June 2005.
[10] S. Lederer *et al.*, "Personal Privacy through Understanding and Action: Five Pitfalls for Designers," *Pers. and Ubiquitous Comp.*, vol. 8, no. 6, Nov. 2004, pp. 440–54.
[11] J. van Bemmel, M. Wegdam, and K. Lagerberg, "3PAC: Enforcing Access Policies for Web Services," *Proc. IEEE Int'l. Conf. Web Services*, Orlando, FL, July 2005.
[12] P. Dockhorn Costa *et al.*, "Controlling Services in a Mobile Context-Aware Infrastructure," *Proc. 2nd Wksp. Context Awareness for Proactive Sys.*, June 2006.
[13] H. van Kranenburg and H. Eertink, "Processing Heterogeneous Context Sources," *Proc. Next-Generation IP-Based Service Platforms for Future Mobile Sys. Wksp. 2005*, Trento, Italy, Jan. 2005, pp. 140–43.
[14] K. Mulligan and B. Smith, "A Relational Theory of the Act," *Topoi*, vol. 5, no. 2, 1986, pp. 115–30.

BIOGRAPHIES

MARTEN J. VAN SINDEREN (m.j.vansinderen@utwente.nl) received his Master's degree in electrical engineering and Ph.D. degree in computer science from the University of Twente, The Netherlands. He is an associate professor at the University of Twente, and manager of A-Services Internet, one of the strategic research orientations of the Centre for Telematics and Information Technology, the ICT research institute of the University of Twente. His research interests include design methods and architectures for networked systems, and service platforms for supporting context-aware mobile applications. He currently leads the Dutch Freeband A-MUSE project (BSIK 03025) on service design and semantic interoperability.

MAARTEN WEGDAM (wegdam@lucent.com) holds a Master's and a Ph.D. degree in computer science from the University of Groningen, The Netherlands, and the University of Twente, respectively. He is a senior member of technical staff of Bell Labs Europe at Lucent Technologies in the Netherlands. His fields of interest include context awareness, identity federation, privacy, middleware, and QoS in middleware. He currently leads the Dutch Freeband AWARENESS project (BSIK 03025) on an infrastructure to support context-aware mobile applications. He has a part-time position as assistant professor at the University of Twente.

HENK EERTINK (Henk.Eertink@telin.nl) holds Master's and Ph.D. degrees in computer science from the University of Twente. He is a senior researcher at Telematica Instituut, The Netherlands, where he leads the Intelligent Communication (INCA) group. His research interests include simplici-

ty and trustworthiness aspects in mobility management and context-aware systems.

HENDRIK B. MEEUWISSEN (erik@lucent.com) received his Master's and Ph.D. degrees in electrical engineering from Eindhoven University of Technology, The Netherlands. He is a senior member of technical staff of Bell Labs Europe at Lucent Technologies in the Netherlands. His research interests include theoretical and applied aspects of communication networks and services, with a current focus on quality of service and context-aware service platforms. He was project leader of the Dutch EQUANET project (TSIT2031) on end-to-end quality-of-service in next-generation networks. He is a member of the Dutch Electronics and Radio Society (NERG). He is a co-recipient of the 2004 Bell Labs President's Gold Award.

AART T. VAN HALTEREN (a.t.vanhalteren@utwente.nl) holds Master's and Ph.D. degrees in computer science from the University of Twente, where he is an assistant professor. His research interests include software infrastructures for (context-aware) networked applications, body area networks, and mobile healthcare applications. As a work package manager in the MobiHealth project (IST-2001-36006) he was responsible for the development of a body area network for ambulant healthcare. He is currently responsible for the architecture of the MobiHealth platform in multiple collaborative research projects.