Bridging Context Management Systems for Different Types of Pervasive Computing Environments

Cristian Hesselman[1], Hartmut Benz[2], Pravin Pawar[3], Fei Liu[3], Maarten Wegdam[3,4], Martin Wibbels[1], Tom Broens[3], and Jacco Brok[4]
[1]Telematica Instituut, The Netherlands
{cristian.hesselman, martin.wibbels}@telin.nl
[2]Twente Institute for Wireless and Mobile Communications, The Netherlands
hartmut.benz@ti-wmc.nl
[3]University of Twente, The Netherlands
{p.pawar, f.liu, m.wegdam, t.h.f.broens}@ewi.utwente.nl
[4]Alcatel-Lucent, The Netherlands
{wegdam, brok}@alcatel-lucent.com

ABSTRACT

A context management system is a distributed system that enables applications to obtain context information about (mobile) users and forms a key component of any pervasive computing environment. Context management systems are however very environment-specific (e.g., specific for home environments) and therefore do not interoperate very well. This limits the operation of context-aware applications because they cannot get context information on users that reside in an environment served by a context management system that is of a different type than the one used by the application. This is particularly important for mobile users, whose context information is typically available through different types of context management systems as they move across different environments. In this paper, we address this interoperability problem by placing bridges between different types of context management systems, in particular systems for home, mobile, and ad-hoc environments. The novelty of our bridges is that they focus on resolving functional differences between context management systems, whereas prior work in this area concentrates on resolving differences in data models. We discuss our bridging architecture and zoom in on a few selected bridges, focusing on their context discovery and exchange functions. We also outline how we implemented these bridges.

Categories and Subject Descriptors

C.2.1 [**Computer Communication Networks**]: Network Architecture and Design – *network communications;* C.2.2 [**Computer Communication Networks**]: Distributed Systems – distributed *applications*.

General Terms

Design

Keywords

Context management, bridging, interoperability

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. To copy otherwise, or republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee.
Mobilware '08, February 12-15, 2008, Innsbruck, Austria.
Copyright © 2008 ACM 978-1-59593-984-5/08/02... $5.00.

1. INTRODUCTION

The ultimate vision of pervasive computing is that of a ubiquitous intelligent system that supports users wherever they go [1]. One of the key ingredients to make this happen is a Context Management System (CMS), which is a distributed system that enables applications to obtain context information about users and other "entities" such places and devices (e.g., [2, 3]).

An important characteristic of CMSs is that they are usually very much environment-specific. For example, some CMSs are specifically geared towards home environments [4], whereas others are dedicated to large-scale mobile environment [4, 5], or to small ad-hoc networks [6]. As a result, different types of CMSs often do not interoperate very well, for instance because they use different naming schemes for their users, different discovery protocols, or different context ontologies [4, 7]. This means that applications are generally limited to what their 'native' CMS can provide, which in turn limits their operation.

As an example, consider a user Alice driving from home to work and her co-worker Bob trying to find out where Alice is, for instance because she is running late for a meeting. Bob gets Alice's location through a context-aware buddy application that uses the CMS of the office environment. This CMS does however not interoperate with other types of CMSs, such as the kind of CMS in Alice's home, the CMS of Alice's mobile network operator (serving Alice when she is driving to work), or an ad-hoc type of CMS (e.g., deployed near a construction site on the freeway). Without a system that interconnects these CMSs, Bob's application will be deprived from location information about Alice as long as she is not yet in the office.

In this paper, we address this interoperability problem and integrate three different types of CMSs: for home/small office environments, for mobile telecom environments, and for wireless ad-hoc environments. The added value of this integration is that it enables applications to obtain context information from multiple types of CMSs, which would for instance allow Bob to also get

Alice's location when she is still at home or when she is driving to work. All three types of CMSs have been developed and tested with several applications in the project AWARENESS [8].

Our approach is to place (semi-)transparent bridges between different types of CMSs. These bridges enable applications to obtain context information through one type of CMS (their 'native' CMS), including context information that originates from other types of CMSs. As a result, applications do not have to interact with these 'foreign' CMSs, which simplifies application development. This is particularly important for mobile users, whose context information is typically available through different types of CMSs as they move across different environments. Another advantage is that the bridges facilitate the incremental role-out of new types of CMSs, which is essential for the deployment of large-scale ubiquitous computing systems. A limitation of the bridges is that they are specific for each pair of CMSs and most likely need to be deployed and managed in large numbers of instances.

There are two important alternatives to our bridging approach: standardization and a unifying adapter. Standardization requires a single CMS to serve all possible types of environments. We believe that this is infeasible because in our experience, the requirements of environments vary strongly and thus require different types of CMSs to cater for and exploit the unique characteristics of the different environments and work efficiently (cf. small ad-hoc environments vs. the millions-of-users of a telecom operator). Furthermore, there is no obvious candidate for such a standard. A unifying adapter keeps the different CMS types, but unifies access to them through by a "convergence layer" that provides a single interface to applications. Such an adapter eases application development, but its complexity and size increases quickly with the number of supported types of CMSs. In addition, existing applications must be modified to benefit from the interoperability. [7] and [9] are examples of works that follow this unification approach.

Our work is novel in that it uses bridges to integrate different types of CMSs and because our bridges focus on resolving functional differences between CMSs, in particular differences regarding context discovery and exchange. Our bridging architecture also contains other bridging functions, such as identity management and privacy control. Prior work in this area concentrated on integrating the data models of different types of CMSs.

We begin our discussion with a brief description of the CMSs we use (Section 2). Next, we discuss the architecture of our bridges, focusing on a few selected types of bridges (Section 3). We then outline how we implemented them (Section 4) and compare our work to existing systems we found in the literature (Section 5). We end with conclusions and future work (Section 6).

2. CONTEXT MANAGEMENT SYSTEMS

Our Context Management Systems (CMSs) share the same basic architecture, which consists of context producers, context consumers, and context brokers. Context producers are all processes that publish context—from low-level physical sensors in a sensor wrapper process to high-level context reasoners. Context consumers are all processes that use context. Context consumers find suitable producers by querying context brokers. Consumers retrieve context from context producers either via subscription or query. Context producers register with context brokers, which may also manage the life-cycle of the producers that have registers with them (start, stop).

Many context producers are also context consumers. For example, context reasoners receive information from several context producers (e.g., several location producers), reason over it (e.g., select the most accurate location or synthesize accuracy by combining measurements), and publish the result as context. Similarly, a context storage engine subscribes to various producers, stores the information it receives from them, and publishes it as historical context.

In this section, we highlight the most important aspects of the four CMSs we have developed and express them in terms of the basic architecture outlined above. One system targets large-scale mobile operator environments (Section 2.1), two of them are meant to operate in homes and office environments (Section 2.2), and the fourth targets (mobile) ad-hoc environments (Section 2.3). Section 3 discusses the context bridges that we put between these systems in detail.

2.1 Mobile Environment

The CUMULAR Context Server (CCS) [10] is a centrally managed CMS for operators of mobile telecom networks. It fits in the IP Multimedia Systems (IMS) architecture as defined by 3GPP [11], which is widely adopted as a convergence architecture for telecom operators. The CCS can federate with other CCS instances if a trust relationship exists between the operators running the CCSs.

In terms of our basic context management architecture, the CCS consists of a large context reasoner that is distributed across nodes in the network of a telecom operator. The reasoner provides context information to CCS applications and obtains lower-level context information from various context producers. These producers may run on devices with limited resources and availability (e.g., mobile phones). The main responsibilities of the reasoner are to run sophisticated and computing-intensive reasoning algorithms, to enforce the user's privacy policies regarding the release of context information, control the use of bandwidth towards mobile phones, and select context producers based on the quality of the information they provide. We implemented the reasoner as an enhanced SQL database.

The context reasoner provides per-application views on the reasoner and can be customized per application regarding used protocols, context needs, and access control restrictions. It can also communicate with different types of context producers, for instance producers that provide a SIP/SIMPLE-based [12] interface. The architecture can be extended with new types of applications or context producers.

2.2 Home/office Environment

We developed two CMSs for homes and small office environments. The first one is the Context Management Framework (CMF) [3]. The central notion in the CMF is that of a personal context broker [13], which keeps track of the set of context producers that can currently provide context information about a particular user, including context producers in other CMF domains (e.g., when the user is visiting that domain). Applications can query a broker for a certain type of context information (e.g., information about the user's activity), after which the context broker returns a reference to a context producer that can provide this information. A personal context broker also enforces the privacy policies of a user by (1) checking if an application is

allowed to get the type of context information it asked for and (2) by placing a proxy context producer between the application and the "real" context producer. A personal context broker returns references to these proxies to the application, which enables the proxy to enforce the user's privacy policies (e.g., by reducing the quality of context information).

The second system is the Context Distribution Framework (CDF) [14]. The CDF focuses on context producers that reside on mobile devices. Its core consists of a specialized context producer called the Context Distribution Service (CDS). The CDS runs in the infrastructure and its main responsibility is to dynamically select the "best" producer for a certain type of context information, where the meaning of "best" is specified by the application in terms of its Quality of Context (QoC) requirements for the particular context type. The CDS subscribes to the CDF's context broker so that it receives an event whenever a new (mobile) context producer registers with the context broker. When this happens, the CDS compares the new producer's QoC with that of other producers that can provide the same type of context information as well as with the application's QoC requirements. If the new producer is "better" than the one currently in use, the CDS transparently switches to that producer and forwards its context information to the application. The CDS also notifies the application about the change. The QoC parameters that the CDS uses to rank context producers are freshness, spatial resolution, temporal resolution and probability of correctness. The CDS includes support for ontologies to represent context information and is implemented as a service in a Jini network.

2.3 Ad-hoc Environment

Our CMS for ad-hoc environments is called Jexci and is peer-to-peer-based. In the absence of centralized context brokers, each Jexci peer implements a part of that functionality. Context producers and consumers find each other in peer groups, which are groups of producers that can each provide a particular type of context information about a particular entity. Implicitly, each peer group defines an overlay (sub-)network connecting only related peers. Hierarchical peer groups mirroring the domain hierarchy of entities provide scalability. Jexci uses the peer-to-peer framework JXTA [15]. Jexci allows a producer and a consumer to negotiate a context exchange format and thus enables Jexci users to independently create new context types and encoding formats for context.,

To improve context discovery in fully distributed ad-hoc networks, we designed and implemented the discovery protocol Ahoy and integrated it with Jexci. Ahoy is based on attenuated Bloom filters and allows highly efficient on-demand (context) service discovery [16]. Attenuated Bloom filters (ABFs) consist of layers of bit vectors to indicate the existence of context information a certain number of hops away in the ad-hoc network. Every node stores an attenuated Bloom filter from each direct neighbor. By consulting those filters, nodes only send queries to destinations that are likely have the requested context information with a small probability of getting a false positive.

3. CONTEXT BRIGDES

A context bridge is a functional component that enables context-aware applications using one type of CMS (the native CMS) to obtain context information from a context producer in another type of CMS (the foreign CMS). Project AWARENESS has developed several bridges that perform this task for the CMSs of Section 2. These bridges are unidirectional, which means that a bidirectional bridge requires two bridges, one for each direction. The requirements for these two bridges may however differ.

In this section, we first consider the functions that any context bridge needs to provide (Section 3.1). Next, we present the overall architecture of the bridges developed in AWARENESS (Section 3.2) and consider two of a bridge's functions in more detail: context discovery (Section 3.3) and context forwarding (Section 3.4). For a few of our bridges, we also illustrate how they realize these two functions.

3.1 Bridging Functions

A bridge has to resolve the differences between native and foreign CMSs. To accomplish this, a bridge needs to support the following functions:

- Identity mapping. A bridge needs to map the entity identifiers of the native CMS to those of the foreign CMS. This is because context information is tightly bound to a user (and other entities such as devices and places) and because the identity of that user may be different in the foreign CMS. For example, Alice's identity in the CMS of the mobile operator may differ from the identity that the CMS in her office uses.
- Context discovery. To discover context producers in the foreign CMS, a bridge needs to translate the context query of a context consumer in the native CMS to the protocol and query capabilities of the foreign CMS. Further filtering of the discovery results is necessary when the foreign CMS provides less discriminatory queries than the native CMS. Since we think of context producers as services, this function is essentially about bridging different service discovery protocols [17].
- Context forwarding. A bridge needs to forward context information from the foreign CMS to the native CMS. It therefore needs to deal with differences in communication mechanisms in both CMSs, in particular when the native CMS supports a publish-subscribe mechanism, but the foreign CMS does not. A bridge also needs to translate between multiple protocols (e.g., web services and JXTA), as they are often tightly connected with the environment in which the CMSs operate.
- Context format mapping. The bridge needs to translate the context semantics (ontology), encoding, and data formats of the foreign CMS to those of the native CMS. Ontology mapping can be very difficult [4] even though the translation of standard units (e.g., longitude/latitude/projection, temperature) is straight forward. Other types of context information (e.g., 'activity', which could take the values of 'working' or 'driving') require clear definitions to be meaningful across CMSs. In order to exchange context information augmented with QoC, CMSs also need to provide QoC definitions [18]. The bridge furthermore needs to be able to translate between different data formats (e.g., SQL, RDF, PIDF, and key-value pairs).

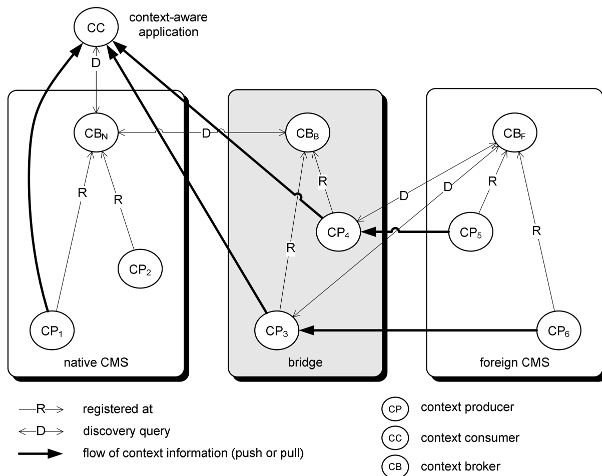

Figure 1. Basic bridging architecture.

- Context adaptation and reasoning. A bridge may need to adapt the context information passing through it, for instance when a context producer in the foreign CMS generates data at a rate that the native CMS cannot handle (e.g., when the native environment is an ad-hoc network and the foreign CMS is that of a mobile operator) and the rate of that producer also cannot be configured on a per-consumer basis. A bridge may need to reason over context information in case the foreign CMS does not directly support the type of context information that the application needs and is also unable to compute it (e.g., when the foreign CMS is an ad-hoc network that only provides low-level context information). Another reason for having context reasoners as part of a bridge may be to reduce the number of context providers published on the native side of the bridge (aggregation).
- Privacy control. If the foreign CMS does not enforce the privacy policies of its users regarding the distribution of context information to the native CMS, the bridge may need to enforce these policies on behalf of the foreign CMS. This may require the bridge to adapt the quality of the context information that it forwards from the foreign to the native CMS [18]. The foreign CMS may also have its own policies about the release of its context information to native CMSs. It could for instance indicate that the native CMS is not allowed to cache any of the foreign CMS' information.

3.2 Architecture

Figure 1 provides an overview of the basic AWARENESS bridging architecture. AWARENESS bridges all share this architecture, but they realize it in different ways depending on the native and foreign CMSs involved.

The centre of the architecture is an AWARENESS bridge, which consists of a context broker and a set of context producers. Together, the broker and the producers realize the functions of Section 3.1. The context broker is responsible for identity management and context discovery, whereas the context producers act as proxies and take care of context adaptation, reasoning, and mapping context information to another format. Both the broker and the context producers are involved in the enforcement of privacy policies. Observe that a producer of the bridge may obtain context information from multiple foreign context producers (not shown in Figure 1), for instance to reason over that information. As an example, Figure 1 shows a bridge that consists of two context producers, CP_3 and CP_4. The context broker of the bridge is labeled CB_B.

The components of an AWARENESS bridge may be distributed across the native CMS, the foreign CMS, dedicated bridging nodes, or a combination thereof. The complexity of a bridge in terms of the functions it needs to provide (see Section 3.1) depends on the differences between the native and foreign CMSs: the larger the difference, the more complex the bridge's functions. For example, if the CMSs only vary slightly, the bridge's context transfer functions can simply translate context request and responses between the native and the foreign CMS. If the differences are larger, the bridge may also need to possess context adaptation and reasoning capabilities. For example, if the foreign CMS is the CMS of a mobile network operator and the native CMS operates in an ad-hoc network, the bridge may need to put an upper bound on the rate at which it propagates context changes into the ad-hoc (native) CMS to prevent it from being overwhelmed with context updates. More advanced functions like these do however require the bridge to maintain more state, which means that the bridge may also need to manage parameters that determine its degree of scalability. Examples are the number of users that are visible through the bridge in the native CMS, the number of concurrent subscriptions on context producers in the foreign CMS, and the number of queries/notifications per second passing through the bridge. This is particularly important if the bridge runs on a resource-constrained device, such as a mobile phone in an ad-hoc network.

We distinguish two types of bridges: those that are fully transparent and those that are semi-transparent. Figure 1 shows the basic architecture of a transparent bridge, which is invisible to a context-aware application. In this case, the application continues to use its native context broker (CB_N in Figure 1) to discover native context producers (CP_1 and CP_2 in Figure 1). When the native broker finds out that it does not have the producer that the application needs, it queries the context broker of the bridge (CB_B) to discover the context producers that the bridge provides and returns any matches to the context-aware application via the native context broker. The application subsequently interacts with one of the bridge's context producers to obtain the context information it needs (push or pull). The bridge's context producers use the context broker of the foreign environment (CB_F) to obtain references to the foreign context producers they need (the producers act as a client of the foreign context broker), get the context information from these producers, and send the (processed) result back to the context-aware application of the native CMS.

With a semi-transparent bridge (not shown in Figure 1), the native broker returns a reference to the broker of the bridge when it cannot find a producer that matches the application's request. The application then resubmits its query to the bridge's broker, which forwards it to a context broker in the foreign CMS. This broker returns its results to the bridge's context broker, which instantiates context producers for the bridge and links them to the foreign context producers. The bridge's broker then returns a reference to one ore more of its context producers to the application. The application subsequently interacts with one of these producers to obtain the context information it needs. The

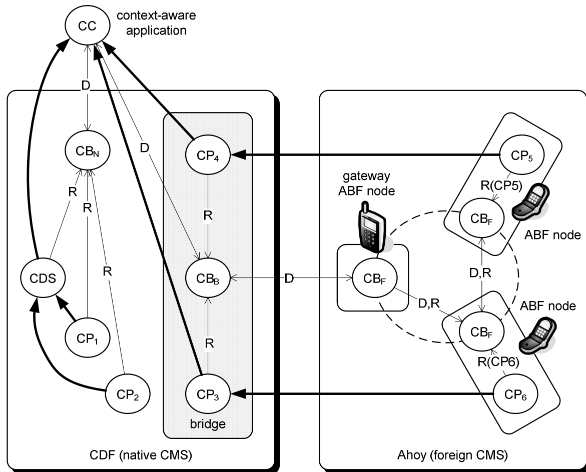

Figure 2. CDF applications using Ahoy context producers (home/office to ad-hoc environment).

context producers that are part of the bridge register with the bridge's context broker.

Note that the context brokers in Figure 1 also need to be able to discover each other. To accomplish this, the bridge's context broker is registered with the native context broker, whereas the foreign context broker is registered with the context broker of the bridge (not shown in Figure 1). A context broker may be distributed across multiple nodes (e.g., in an ad-hoc network) and does not need to be centralized.

In the following two sections, we introduce three of the bridges we have developed and discuss their design in terms of the basic architecture of Figure 1, focusing on the bridges' context discovery and forwarding tasks.

3.3 Context Discovery

Figure 2 shows the architecture of the bridge that enables CDF applications (home/office environment, see Section 2.2) to obtain context information from context producers in an ad-hoc environment that uses Ahoy for discovery (ad-hoc environment, see Section 2.3). The bridge is semi-transparent. The context broker and the context producers of the bridge are part of the CDF (the native CMS) because the nodes in the ad-hoc network are resource constrained mobile devices. Each CDF domain hosts one Ahoy-CDF bridge for all the Ahoy-based ad-hoc networks it interfaces with.

The context broker of the bridge interacts with an Ahoy network through gateway ABF nodes. A gateway node is part of both the CDF network and the ABF network and is the foreign context broker (CB_F) from the perspective of the bridge (cf. Figure 1). The context broker on a gateway node interacts with the context brokers on other nodes in the ad-hoc network to discover the context producers they host (using the Ahoy protocol). The context broker of the ad-hoc network is thus fully distributed.

The bridge's context broker maintains a list of ABFs that reflect the current state of the ad-hoc network. The broker receives these ABFs from the gateway node and uses them to decide if it needs to forward a discovery query from a CDF application (see below) into the Ahoy network. The advantage of this approach is that it avoids traffic floods in the ad-hoc network, which improves the network's operation. Storing the list on the CDF-side (infrastructure) also reduces processing and state on the gateway node and enables the bridge's broker to serve multiple gateway nodes at the same time.

The broker receives a new set of ABFs from the gateway node when the ad-hoc network changes, for instance when a new producer appears or an existing one disappears.

When a CDF application is looking for a context producer, it sends a query message to the CDS (see Section 2.2). If the CDS cannot find the requested producer in the CDF network (by querying CB_N), it directs the CDF application to query the broker of the bridge, which translates it into Bloom codes and matches it against its list of ABFs. If the broker finds a match, it forwards the query (in the form of Bloom codes) to the target Ahoy network through the network's gateway node. Context producers that can serve the query send their binding information back to bridge's broker, following the same path as the query.

When the context broker of the bridge receives a response from an Ahoy producer, it creates a context producer for it, configures the new producer with the binding information provided by the Ahoy context producer, and sends a reference to the new context producer to the CDF application. In our current design, both the CDF and the Ahoy network support XML-RPC for context exchange, which means that the context producers instantiated by the bridge (CP_3 and CP_4) are merely empty stubs.

3.4 Context Forwarding

Figure 3 shows the organization of the bridge that enables CMF applications (home/office environment, see Section 2.2) to obtain context information from CCS context producers (mobile environment, see Section 2.1). CCS-CMF bridges are bound to individual users, which is unlike the Ahoy-CDF bridges that exist at the level of entire CMS domains. Also unlike Ahoy-CDF bridges, CCS-CMF bridges are fully transparent.

CCS-CMF bridges are part of CMF domains (native CMS) because there will typically be many more CMF instances (homes and offices) than there will be instances of the CCS (mobile operators). Hosting the CCS-CMF bridges in the CCS would thus yield a significant increase of state on the CCS-side, which might negatively affect the CCS' scalability.

When a CMF application requests context information from a personal broker ($CB_{N,U}$ in Figure 3), the broker dynamically

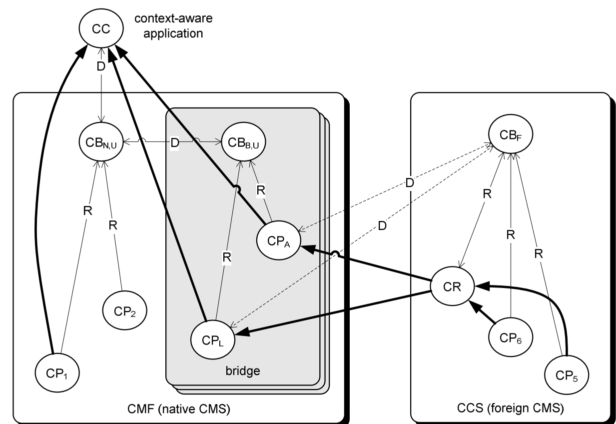

Figure 3. CMF applications using CCS context producers (mobile to home/office environment).

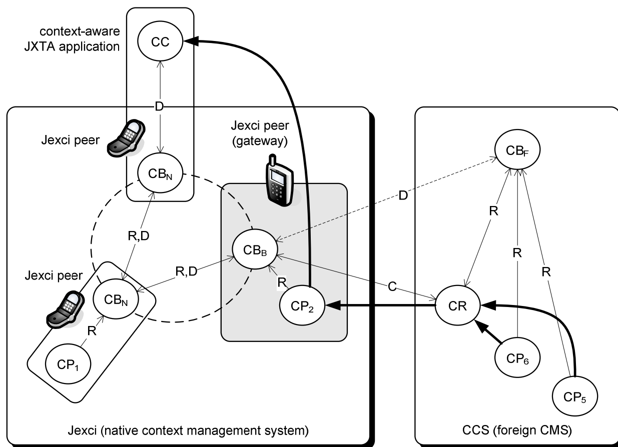

Figure 4. Jexci applications using CCS context producers (mobile to ad-hoc environment).

creates a CCS-CMF bridge if it does not have a link to an appropriate native context producer. The advantage of this approach is that CCS-CMF bridges can be instantiated when and where they are required, thus distributing load and increasing scalability on the CMF-side. This is particularly important in larger CMF domains (e.g., in offices). Because the CCS-CMF bridges are bound to individual users, this approach also delegates identity and authentication mapping between CMF and CCS to each user: each user configures the required bridges with the appropriate CCS identities and credentials.

After creating a CCS-CMF bridge, the personal context broker forwards the discovery request it received from the CMF application to the bridge's broker ($CB_{B,U}$ in Figure 3). This broker creates a context producer for the type of context information that the CMF application requests and returns a reference to the new producer back to the application (via the native personal broker). The context producers of a CCS-CMF bridge are long-lived, which means that they can be reused for CMF applications that request the same type of context information about the same user. As an example, Figure 3 shows two context producers in the bridge: one for location information (CP_L) and one for activity information (CP_A).

A context producer of a CCS-CMF bridge uses the context broker in the foreign CMS (CB_F) to locate the large context reasoner (CR) in the CCS. It then uses the reasoner to subscribe to events that signal changes in a particular type of context information (e.g., location information) for the user that the personal context broker represents. CMF applications can subscribe to these events through the bridge's context producers, which transparently forward the events from the CCS to the CMF application. Similarly, CMF applications can query the context producers in the bridge for context values (of a particular type), which they will then transparently obtain from the CCS and forward back to the requesting CMF application.

The CMF uses a different query language than the CCS: CMF producers support SPARQL queries, whereas CCS producers accept queries in SQL. When a context producer of the bridge receives a SPARQL query, it first gets the corresponding type of context information from the CCS context reasoner by submitting an SQL query to it. Next, it applies the SPARQL query on the context information it receives from the CCS reasoner and sends the result back to the CMF application.

In our current implementation, we statically configured the address of the CCS context reasoner in the CCS-CMF bridges (hence the dashed arrows in Figure 3) and combined the context broker of the bridge with the personal context broker.

Figure 4 shows the organization of the bridge that enables Jexci applications (ad-hoc environment, see Section 2.3) to obtain context information from CCS domains (mobile environment, see Section 2.1). Like an Ahoy-CDF bridge, a CCS-Jexci bridge operates at the level of domains, which means that it makes the large reasoner of a CCS domain available in a Jexci network. CCS-Jexci bridges are fully transparent.

An instance of the CCS-Jexci bridge runs on a gateway node that is part of the Jexci network, but can also establish connections with CCS reasoners, typically via a wide area wireless link. The context broker of a CCS-Jexci bridge is configured with references to foreign context brokers (CB_F in Figure 4). Different nodes in the same Jexci network may host a CCS-Jexci bridge for the same CCS domain to distribute the load of interacting with that domain across multiple Jexci nodes.

During start-up, the context broker of a CCS-Jexci bridge queries its foreign context brokers to obtain a reference to any CCS reasoners. Next, the bridge's broker queries these reasoners to ask them which domains they serve and advertises this information in the Jexci network. The bridge uses one JXTA peer group per CCS domain instead of one peer group per CCS user. The latter would yield way too many peer groups for JXTA to handle and would bring down the Jexci network.

A Jexci application submits discovery requests to its local broker (CB_N) and specifies which types of context it needs for which user. If this is a CCS user, the request ends up at a bridge that serves the CCS domain where the user is registered. The bridge's context broker (CB_B) handles the request and queries the reasoner to ask it which types of context information it can provide and returns this information back to the Jexci application. The return message contains the address of the gateway node, which implicitly acts as a context producer.

The gateway node creates actual context producers only when the Jexci application needs context information, which saves resources on the gateway node. For publish-subscribe interactions, the gateway node creates a context producer (e.g., CP_2 in Figure 4) when it receives a subscribe message from the Jexci application. This producer calls the CCS reasoner to subscribe to the same types of context information and forwards any context updates that occur on the CCS-side to the Jexci application. When the gateway node receives a query for context information from the Jexci application, it creates a temporary context producer that queries the CCS reasoner, passes the result back to the Jexci application, and then terminates.

Despite the above approach, a CCS-Jexci bridge must limit the number of concurrent queries and subscriptions because also the JXTA connections, which are used for context exchange in the Jexci infrastructure, require a lot of resources.

In our current implementation, we statically configured the broker of a CCS-Jexci bridge with the address of a CCS reasoner, thus bypassing the foreign context brokers.

4. IMPLEMENTATION

Table 1 provides an overview of the bridges we developed for the CMSs, including the three bridges discussed in Section 3. The

from\to	CCS	CMF	CDF	Ahoy	Jexci
CCS	Postgres SQL database, Java extensions enabled by Perl/Java, PIDF format.	*Location and presence information from CCS to CMF. Request-response interactions in PIDF.*			*Application that passes emergency-related context info from CCS to Jexci.*
CMF	Location, activity and device information from CMF to CCS.	Core in Java, web services for discovering context producers, SIP for discovering personal context brokers, OWL format.			
CDF	Jini client delivering health-related emergency information from CDF to CCS.		Core in Java, based on Jini technology, XML format.	Gateway node + Jini-based implementation of bridge, which support context discovery from CDF to Ahoy	
Ahoy			*Gateway node + Jini-based implementation of bridge, which support context discovery from Ahoy to CDF*	Core in Ruby, Ahoy service discovery, key-value format.	Jexci/ABF peer, Ahoy module in Java, key-value format.
Jexci	Location and buddy information from Jexci to CCS.			Jexci/ABF peer, Ahoy module in Java, key-value format.	Core in Java, JXTA based service discovery, key-value format.

Table 1. Implemented bridges. Bridges discussed in Section 3 are in *italics*.

bridge that delivers context information from CMS A to CMS B is in row A, column B. Information on the implementation of individual CMSs is in the shaded cells.

The CCS is largely written in Java. The bridges use SQL statements (through JDBC or ODBC) to write data to and read data from the Postgres SQL database. The CMF core and its elements like the personal context broker are developed in Java. The CDF uses Jini technology and lets mobile context producers participate in the Jini network through the Mobile Service Platform (MSP) [19]. In Jexci, the context producers are addressed using XML-RPC (via JXTA pipes or directly). New context en/decoders can be added to a context producer/consumer simply by adding a JAR file to the class path. Ahoy is implemented in the programming language Ruby and uses UDP over IPv6 [20]. Ahoy has been integrated into Jexci to improve the peer group discovery.

5. RELATED WORK

The work that comes closest to ours is that of Lehmann et al. [4], who integrate a CMS for home environments (the Aware Home Spatial Service) with a CMS for mobile operators (Nexus [21]). The difference with our work is that they focus on integrating the data models of the two CMS, whereas our bridges focus on interoperating functionality, in particular context discovery and exchange. Integrating CMS-specific data models is however a crucial component for interoperating CMSs, which is why we consider the work of Lehmann et al. and our work complementary to each other. Another difference is that we also take ad-hoc CMSs into account.

In convergence approaches, various original context information from applications is uniformed by a common context model and ontology in a convergence layer before they are distributed to diverse underlying systems. The ITransIT system has been proposed in [22] to federate advanced pervasive transportation systems. This system uses a common spatial data layer with primary context model (PCM) to model all information uniformly and ontology (PCOnt) to specify the relationships between those information. In [7], a core common model is designed for ubiquitous computing (ubicomp) environments and a convergence layer called Ubicomp Integration Framework (UIF) is implemented to adapt existing ubicomp systems to this common model for wide area access. UIF uses semantic web technology to interact and support dynamic reconfiguration of the exposed models. [9, 23] follow a similar approach to converging different types of CMSs and interacting with one unified API.

Other works on interoperating CMSs deal with federating multiple instances of the same type of CMS, for instance CMSs of mobile operators [5, 24], CMSs for small to mid-sized environments [13], or a combination thereof [25].

6. CONCLUSIONS AND FUTURE WORK

The landscape of pervasive computing will involve different types of Context Management Systems (CMSs) that are specifically designed for different types of environments (e.g., homes and mobile operator environments). To provide context information about (mobile) users to any context-aware application, these CMSs will somehow need to interoperate.

In this paper, we discussed the AWARENESS architecture for interoperating different types of CMSs. The main component of the architecture is a unidirectional bridge, which enables applications that use one CMS (the native CMS) to obtain context information stored in another CMS (the foreign CMS). Our bridges consist of a context broker and one or more context producers that specialize in bridging differences between CMSs for different environments. Our unidirectional bridges may be paired to form bidirectional bridges.

We outlined the functions that these bridges need to provide to, focusing on context discovery and context exchange. We implemented a total of nine bridges (some combined in bidirectional bridges) and illustrated which technologies we used for this purpose.

The novelty of our work is that we concentrate on resolving functional differences between CMSs, in particular for context discovery and exchange. Another innovation is that we also include ad-hoc networks in the interoperability equation.

Our future work includes the enforcement of privacy policies in an environment that consists of multiple interoperating CMSs. This includes mapping the identifiers used in one type of CMS to those used by another and how to manage trust in such an environment. Another item of future work is an analysis of the performance of a few of the bridges we developed.

7. ACKNOWLEDGMENTS

This work has been conducted within the project Freeband AWARENESS. Freeband is co-sponsored by the Dutch government under contract BSIK 03025. Henk Eertink reviewed the draft version of this paper.

8. REFERENCES

[1] M. Weiser, "The Computer of the 21st Century", Scientific American, vol. 265, no. 3, September 1991

[2] A. Dey, D. Salber, and G. Abowd, "A Conceptual Framework and a Toolkit for Supporting the Rapid Prototyping of Context-Aware Applications", Special issue on context-aware computing; Human-Computer Interaction (HCI) Journal, Volume 16 (2-4), 2001, pp. 97-166

[3] H. van Kranenburg, M. S. Bargh, S. Iacob, and A. Peddemors, "A Context Management Framework for Supporting Context-Aware Distributed Applications", IEEE Communications Magazine, August 2006, pp. 67-74

[4] O. Lehmann, M. Bauer, C. Becker, and D. Nicklas, "From home to world - supporting context-aware applications through world models", 2nd IEEE Annual Conference on Pervasive Computing and Communications (PERCOM'04), Orlando, Florida, USA, 2004

[5] I. Roussaki, M. Strimpakou, C. Pils, N. Kalatzis, M. Neubauer, C. Hauser, and M. Anagnostou, "Privacy-Aware Modelling and Distribution of Context Information in Pervasive Service Provision", IEEE International Conference on Pervasive Services (ICPS 2006), Lyon, France 2006

[6] F. Perich, S. Avancha, D. Chakraborty, A. Joshi, and Y. Yesha, "Profile Driven Data Management for Pervasive Environments", 13th International Conference on Database and Expert Systems Applications (DEXA 2002), Aix-en-Provence, France, September 2002

[7] M. Blackstock, R. Lea, and C. Krasic, "Toward Wide Area Interaction with Ubiquitous Computing Environments", 1st European Conference on Smart Sensing and Context, the Netherlands, October 2006

[8] M. van Sinderen, A. van Halteren, M. Wegdam, H. Meeuwissen, and E. Eertink, "Supporting Context-aware Mobile Applications: an Infrastructure Approach", IEEE Communications Magazine, 44 (9). Sept 2006, pp. 96-104

[9] T. Broens, R. Poortinga, J. Aarts, "Interoperating Context Discovery Mechanisms", 1st International Workshop on Architectures, Concepts and Technologies for Service Oriented Computing (ACT4SOC'07), Barcelona, 2007

[10] J. Brok, "CUMULAR Context Solutions", Freeband AWARENESS deliverable Dn2.5. http://awareness.freeband.nl, December 2006

[11] 3rd Generation Partnership Project (3GPP), http://www.3gpp.org/

[12] IETF Applications Area Working Group, SIMPLE charter, http://www.ietf.org/html.charters/simplecharter.html

[13] C. Hesselman, H. Eertink, and M. Wibbels, "Privacy-aware Context Discovery for Next Generation Mobile Services", 3rd SAINT2007 Workshop on Next Generation Service Platforms for Future Mobile Systems (SPMS 2007), Hiroshima, Japan, January 2007

[14] P. Pawar, A. van Halteren, and K. Sheikh, "Enabling Context-Aware Computing for the Nomadic Mobile User: A Service Oriented and Quality Driven Approach", IEEE Wireless Communications & Networking Conference (WCNC 2007), Hong Kong, March 2007

[15] JXTA Peer-to-peer Framework, www.jxta.org

[16] F. Liu, G. Heijenk, "Context Discovery Using Attenuated Bloom Filters in Ad-hoc Networks", Journal of Internet Engineering, Vol 1, No 1, 2007, pp.49-58

[17] Y. Bromberg and V. Issarny, "Service discovery protocol interoperability in the mobile environment", Software Engineering and Middleware (SEM'04), Sept 2004, Linz, Austria

[18] K. Sheikh, M. Wegdam, and M. van Sinderen, "Middleware Support for Quality of Context in Pervasive Context-Aware Systems," 5th Annual IEEE International Conference on Pervasive Computing and Communications Workshops (PerComW'07), New York, USA, March 2007

[19] A. van Halteren and P. Pawar, "Mobile Service Platform: A Middleware for Nomadic Mobile Service Provisioning", 2nd IEEE International Conference On Wireless and Mobile Computing, Networking and Communications (WiMob 2006), Montreal ,Canada, June 2006

[20] R. Haarman, "Ahoy: A Proximity-Based Discovery Protocol", Master's thesis, January 2007

[21] D. Nicklas, M. Grossmann, T. Schwarz, S. Volz, and B. Mitschang, "A model-based, open architecture for mobile, spatially aware applications", 7th International Symposium on Spatial and Temporal Databases, SSTD 2001

[22] D. Lee and R. Meier, "Primary-Context Model and Ontology: A Combined Approach for Pervasive Transportation Services", Fifth Annual IEEE International Conference on Pervasive Computing and Communications Workshops (PerComW'07), New York, USA, 2007, pp. 419-424

[23] R. Meier, A. Harrington, T. Termin, and V. Cahill, "A Spatial Programming Model for Real Global Smart Space Applications," 6th IFIP International Conference on Distributed Applications and Interoperable Systems (DAIS 06), Bologna, Italy, 2006, pp. 16-31

[24] M. Strimpakou, I. Roussaki, C. Pils, M. Angermann, P. Robertson, and M. Anagnostou, "Context Modelling and Management in Ambient-aware Pervasive Environments", International Workshop on Location- and Context-Awareness (LoCA 2005), Munich, Germany, 2005

[25] R. José, F. Meneses, and A. Moreira, "Integrated Context Management for Multi-domain Pervasive Environments", First International Workshop on Managing Context Information in Mobile and Pervasive Environments (MCMP-05), Ayia Napa, Cyprus, May 2005