Jabber-based Cross-Domain Efficient and Privacy-Ensuring Context Management Framework

Zakwan Jaroucheh, Xiaodong Liu, Sally Smith
School of Computing
Edinburgh Napier University
10 Colinton Road, EH10 5DT, Edinburgh, UK
{z.jaroucheh, x.liu, s.smith}@napier.ac.uk

***Abstract*—In pervasive environments, context-aware applications require a global knowledge of the context information distributed in different spatial domains in order to establish context-based interactions. Therefore, the design of distributed storage, retrieval, and dissemination mechanisms of context information across domains becomes vital. In such environments, we envision the necessity of collaboration between different context servers distributed in different domains; thus, the need for generic APIs and protocol allowing context information exchange between different entities: context servers, context providers, and context consumers. As a solution this paper proposes *ubique*, a distributed middleware for context-aware computing that allows applications to maintain domain-based context interests to access context information about users, places, events, and things - all made available by or brokered through the home domain server. This paper proposes also a new cross-domain protocol for context management which ensures the privacy and the efficiency of context information dissemination. It has been robustly built upon the Jabber protocol which is a widely adopted open protocol for instant messaging and is designed for near real-time communication. Simulation and experimentation results show that *ubique* framework well supports robust cross-domain context management and collaboration.**

Keywords—pervasive computing, cross-domain context management, context modelling, Jabber protocol, privacy.

I. INTRODUCTION

In the emerging and challenging pervasive environments, users will wear smart clothes that will monitor their bio signals; they will carry smart cards that will handle automatically their transactions; invisible chips will be embedded everywhere in the smart homes and offices to assist them in their daily life tasks; more sophisticated control navigation and control will be embedded into their vehicles. All these devices will cooperate together to create a context-aware pervasive environment that supports humans in everyday activities, e.g., business, health care, or education. In this respect, the user will enjoy a new experience in a non-obtrusive way as the existing infrastructures will be more proactive and dynamically adaptable to current situations; user preferences; and environmental context in a less intrusive way [1]. Context-awareness is the cornerstone to achieve the vision of such a pervasive environment. It helps to support non-intrusive adaptability of applications to new situations and to turn a static computing environment into a dynamic ecology of smart and proactive applications [2].

In this paper, we base our context management framework on the notion of context domain explained in [3] which organizes the pervasive environment hierarchically and establishes a context management scope. A context domain is defined as an abstraction of a spatial area which has a clear boundary and it is built on top of the traditional notion of network domain. Essentially, context domain establishes (i) the place and responsibility of context instances storage; (ii) the responsibility for managing context providers and consumers inside the domain; and (iii) a set of sub-domains.

Although users are more interested in context information related to their location, other context information from other domains may also be relevant to the current task at hand. For instance, a dynamic recalculation of the quickest routes for a trip involves acquiring the latest contextual information such as traffic congestion from remote sources. In this respect, we can imagine a domain-based context management system where the context information available in each domain is managed by a separate context server. While moving, the user roams across domains. In addition, each domain may maintain its own sensors and mechanisms for inferring context related to this user. Consequently, collaborative context management across domains is needed.

In particular, an efficient cross-domain context management middleware system for such a setting needs to fulfil key requirements that include: (i) domains of context perception, (ii) uniform API interface for accessing context servers, (iii) efficient context information dissemination, (iv) support of cross-domain reasoning, (v) dynamic matching between context providers and consumers, and (vi) support for privacy. In this paper, we propose *ubique*, a new domain-based context management infrastructure for disseminating context information between context providers, context consumers and context servers, and a set of APIs for interfacing between these entities. *ubique* fulfils the above mentioned key requirements and it forms an underlying robust and generic infrastructure for context management, which significantly simplifies the development of context-aware pervasive applications.

This paper is structured as follows. In Section 2, we outline the different requirements that should be fulfilled by a cross-

domain context management system. Then in Section 3, we critically review the advantages and weaknesses of existing solutions with respect to the defined criteria. Section 4 describes the context dissemination problem. In Section 5, we detail our new proposed approach for context management. Finally, we evaluate the proposed approach by mean of real experimentation and simulation and we draw conclusions.

II. Requirements and Challenges

Hereafter, we refer to the computational entity responsible for transparently binding the context consumers (CCs) (i.e. applications) with corresponding context providers (CPs) a context server (CS). The context management in each domain is done by the context server available in that domain. The complexity of developing context-aware applications that require context information available in different CSs makes the use of a cross-domain context management middleware crucial. From our pilot experiments and literature analysis, we identify that a middleware for such a setting must fulfil the key requirements such as:

Domains of context perception: Since the context information is naturally distributed, the context management must be distributed in order to allow efficient and scalable dissemination of context. However, the task of context-aware developers becomes more difficult as it requires a priori knowledge of the computational entities responsible for providing the context information they are interested in. Their task becomes even more complex when context providers dynamically enter and leave the pervasive environment. Thus, there is a need for a dynamic discovery mechanism of context providers.

Furthermore, the middleware scalability could be increased by restricting the access and perception of the context to some domains [3]. Moreover, as we will see later, the notion of home domain server reduces the number of CSs that may be involved in the resolution of context interests. This requirement conforms to the principle of system boundary [4] of pervasive applications.

Uniform API interface and protocol: In order to enable every party to become a context provider and implement its own CS, every CS should: (i) obey a certain protocol with which context information can be federated between different CSs; and (ii) implement a standard API which allows context providers to register and publish context information in it, and context consumers to acquire context information they are interested in. This way, for instance, an organization can operate a CS for its members, and an individual can run a CS as a context provider for a single user or family members. Therefore, similar to the Next Generation Service Interfaces (NGSI) [5], providing a standard API for accessing such information, allows third party application developers to build new services based on the context made available to them.

Efficient context information dissemination: With regard to situations involving mobile users roaming across domains, additional restrictions may arise (e.g. concerning limited connectivity and bandwidth, unknown network conditions, etc.), thus exchanging context information between domains should be fast and only the required information should be transferred when users roam across domains. This requirement calls for a federation protocol between CSs. Furthermore, the middleware should support the "publish on demand" mode of operation. That is, usually context providers publish context constantly and independently of existing consumers. In this case if a context provider publishes at a higher rate the context information is more accurate in terms of freshness. However, this is a costly operation in terms of the network bandwidth usage (i.e. increase of the number of messages sent through network), processing power, and energy consumption (e.g. battery usage of WiFi scanners). Thus, the middleware should enable providers to publish when there is a corresponding consumer.

Cross-domain reasoning: As the context information is originated from different domains, a cross-domain context management system should facilitate the context information reasoning that spans multiple domains. That is, in order to track user's behaviour there is a need to consider the context information available in the different domains the user visits [6]. Hence, understanding the user's current situation may require considering the different states the user experienced in these domains. For example, to identify if the current day was busy for the user there is a need to consider the different activities and states the user has experienced in work, shopping, on the road, etc.

Dynamic matching between context providers and consumers: Typically developers define context interests which should be transparently kept across distributed CSs. The main challenge in such dynamic environment is therefore to accommodate changes on the environment without infringing active context interests. The middleware should allow the context consumers (applications) to register their interests in context information; and the context providers to register their capabilities. Then, for any change in either the context consumers or providers, a matching function should be triggered so that applications asynchronously receive notifications of context information that match their interests. In addition, the application should be able to specify its context interests on the basis of context types and meta-attributes such as precision and accuracy and to indicate additional restrictions based on properties of the provider or the context publication. In this case, the middleware has to be responsible for choosing the most adequate context providers among a dynamic set of available ones.

Support for privacy: The flow of context information between different distributed domains obviously raises user privacy issues. A cross-domain system should protect user's information and guarantee privacy across domains. As we will see later the usage of a home domain server provides an interesting approach for control privacy of context access, since it is a central point of access for a given entity's context. A user can control the context dissemination for some consumers through modifying its privacy policy published in his home domain server.

III. Limitations of Current Approaches

Classical work in context-aware computing has developed centralized and application-specific solutions such as Context Toolkit [7] which provides a set of abstractions that can be

used to implement reusable software components for context sensing and interpretation. The context information is directly acquired from a sensor by means of the context widget component. Widgets can be combined with interpreters, which transform low-level information into higher-level information that is more useful to applications, and aggregators, which group related context information together in a single component. Finally, context-aware applications can invoke actions using actuators, and locate suitable widgets, interpreters, and aggregators using discoverers. Another interesting work is Gaia [8] which adopts the concept of active spaces, which are physical spaces where devices in a heterogeneous network, such as PDAs and printers, can discover each other, auto-configure and dynamically start a context-aware interaction. It provides a framework to develop user-centric, resource-aware, multi-device, context-sensitive and mobile applications. However, these approaches offer solutions for restricted and small-size smart spaces environments, with localized scalability.

GLOSS [9] composes heterogeneous context management systems through hierarchical or peer-to-peer interconnection methods. By introducing the notion of Global Smart Spaces, GLOSS supports interaction amongst people, artifacts and places while taking account of both context and movement on a global scale that facilitates the implementation of location–aware services. It allows users to pick up small notes left for them in the environment. GLOSS uses the idea of home nodes, however, it has been designed to manage location context only.

More recent middleware offers access to context information in distributed repositories. For example, the Context Fabric (Confab) [10] provides architecture for privacy-sensitive systems, as well as a set of privacy mechanisms that can be used by application developers. It maintains context information in distributed tuple-spaces called infospaces. Each infospace is a repository responsible for storing one or more context types. An application interested in a certain context, builds a context query using the address of the responsible infospace. In order to handle queries over distributed infospaces, Confab offers a query processing service, which distributes queries over distributed infospaces and composes the query results. Privacy is supported by adding operators to an infospace to carry out actions when tuples enter or leave the space. However, as Confab focuses so heavily on privacy, it does not adequately address the other middleware requirements such as mobility or context information dissemination across domains.

The scalability issue is considered in PACE [11], which is another distributed middleware focusing on offering a flexible context model called CML (Context Modeling Language) and advanced context-based programming abstractions for distributed context-aware applications. PACE is organized in layers that provide, in addition to context management, an interface to execute distributed context queries, and an adaptation layer, which maintains a reusable repository of adaptation abstractions. Applications use a catalog and meta-attributes to discover which repository satisfies their context requirements. However, when a user roams across domains, this discovery mechanism does not allow developers to identify the repositories existing in the domains visited by the roaming user which contain his context information.

CAMUS [12] is another distributed middleware where context-aware system federation is composed by environments based on CAMUS services, which disseminate context information as tuples, in order to increase dissemination efficiency. Each service of an environment must be registered in a Jini discovery service. A CAMUS context domain is an environment that supports a minimum set of CAMUS services. The set of Jini services responsible for each CAMUS domain composes a federation. In order to access context information or to use a service of a specific domain, a client must query the Jini federation, using parameters such as the name and localization of the domain. CAMUS, however, does not address cross-domain context dissemination and how to ensure user's privacy.

Another interesting approach to allowing distributed context management based on federating context-aware services is Nexus [13]. Nexus supports heterogeneity among context management systems' context models, i.e. each context management system can adopt a particular context model and must implement an abstract interface and register itself at an Area Service Register. Thus, it focuses on the data management aspect of large-scale pervasive computing systems. A client may access context information provided by the federation, by using a query language. However, there is no concept such as domain or environment: each context server is a repository of a specific context type [3].

The Context Management Framework (CMF) proposed in MobiLife project [14][15] is designed for the discovery of, exchange of, and reasoning on context information. It is a set of components, which are connected at run time, that together provide the relevant context information for the service or application, using sensing and interpretation mechanisms. The main tasks for the CMF are to enabling the discovery of context providers, to provide a published agreement or interface contract between context providers and context consumers, and binding context consumers with the matched context providers in order to use their context service functions through the use of context broker. Therefore, in CMF there is no concept such as domain so that the application is able to specify the domain(s) from which the context information is originated. In addition, the infrastructure needed for setting and enforcing privacy of user-controlled data available through context providers is controlled by the Trust Engine. However, we believe that this setting weakens enforcing the privacy since a malicious context provider can skip contacting the trust engine to verify if the context consumer is eligible to access the context information; thus a centralized trusted entity responsible to enforce the privacy is needed.

ICE [16] is a scalable context management middleware for Next Generation Networks. It is based on the concepts of context sessions and context flows. The idea is to separate signaling data from content exchange, as in IP Multimedia Subsystem, to establish context sessions for more scalable and adaptive management of context information. The Context Access Language (CALA) has been designed to support context queries and subscriptions. However, ICE focuses

heavily on efficient context information dissemination between context sources and sinks. Thus, it ignores in its designed protocols ensuring entities privacy. In addition, context sources' descriptions and context sinks' queries/subscriptions must be registered in a centralized entity - the context broker. Thus, as the user roams between domains, this adds complexity to the developers as they must know in advance which context broker they have to contact to get the context information they are interested in.

From the perspective of globally connecting sensors, the Open Geospatial Consortium provided the Sensor Web Enablement (SWE) initiative [17] to building a framework of open standards for exploiting Web-connected sensors and sensor systems of all types such as flood gauges, air pollution monitors, Webcams, etc. SWE provides the opportunity for adding a real-time sensor dimension to the Internet and the Web. It focuses on developing standards to enable the discovery, exchange, and processing of sensor observations, as well as the tasking of sensor systems in order to achieve a "plug-and-play" Web-based sensor networks. Thus, SWE cannot be directly applied to achieve context-awareness because, for example, Sensor Model Language (SensorML) describes sensors systems; provides information needed for discovery of sensors, location of sensor observations, etc. but it does not consider modelling the entities about which the sensor is able to provide information.

Compared to this solution, Chen et al. [18] propose a data-centric infrastructure based on Context Fusion Networks (CFNs) to support context-aware pervasive-computing applications. CFNs are based on an operator graph model, in which context processing is specified by application developers in terms of sources, sinks and channels. In this model, sensors are represented by sources, and applications by sinks. Operators, which are responsible for data processing, act as both sources and sinks. At runtime, the implemented peer-to-peer (P2P) infrastructure instantiates the operator graphs on behalf of context-aware applications. Solar consists of a set of functionally equivalent hosts named Planets. The components messages will be delivered to a Planet with the numerically closest ID; therefore, unlike our proposed approach, Solar services focuses on the data objects instead of on where they live i.e. from which domain they are originated. In addition, Solar does not address privacy enforcement. Another hybrid approach to modeling contextual information that incorporates the advantages of object-oriented and ontology-based modeling techniques is introduced by Lee and Meier [19]. The objective is to support a specific large-scale pervasive domain, namely the transportation domain. Their notion of Primary-Context Model and the Primary-Context Ontology is used to share context between different domains. Although their approach is interesting, it does not address other issues such as mobility and cross-domain context dissemination.

Zebedee et al. [20] introduced ACMF, an adaptable context management system by adopting autonomic computing paradigm. This system is implemented by using the Web services and the Web Services Distributed Management (WSDM) standards. ACMF views each device in terms of the roles it plays with respect to context management which includes client, server, and context proxy. ACMF defines a context model and a set of context exchange protocols between devices. ACMF models the pervasive computing environment as a collection of domains where each domain contains a set of regions and a set of device types. A domain is a logical representation of a physical space, such as a building or campus, containing regions and device-types. In this respect, their domain concept is very similar to the domain concept used in our approach. However, because the focus is on exchanging context information between devices available on a local area (one region) ACMF does not address cross-domain context dissemination, which is a requirement in pervasive environment. Therefore, querying context information available in distributed domains is not possible in their approach.

Closely integrated with an application domain of e-health, Pung and Gu et.al proposed a Context-Aware Middleware for Pervasive Homecare (CAMPH) [21]. The middleware offers several key-enabling system services that consist of P2P-based context query processing, context reasoning for activity recognition and context-aware service management. The key contribution of CAMPH is physical context data collection and reasoning, however, it lacks innovation in the architecture of context management.

Most of the previous work focussed on the software engineering perspective of the distributed context management. From a knowledge management perspective, Castelli and Zambonelli [22] addressed the distributed management of context information from a knowledge management perspective. They propose a self-organized agent-based approach to autonomously organize distributed contextual data items into knowledge networks. These data atoms as well as any higher-level piece of contextual knowledge represents a fact which can be expressed by means of a four-fields tuples (Who, What, Where, When); they call it W4 Data Model. This model is able to represent data coming from heterogeneous sources and to promote ease of management and processing. These knowledge atoms are linked via general-purpose mechanisms and policies to form W4 knowledge networks which can facilitate services in extracting useful information out of a large amount of distributed contextual items. The usage of tuple-space like repositories supports heterogeneity and facilitates building the knowledge network; however, because the focus is on the knowledge management perspective other requirements e.g. mobility between domains has been partially addressed. In addition, despite the efficiency in retrieving tuples during query resolution phase, using the spidering approach to create the knowledge networks may be inefficient when considering the rapidly changing context information such as entities location.

If we look at the aforementioned requirements and at the approaches described above, it reveals that research in the area of context management is well established and many ideas have been developed for addressing most of the above requirements individually. However, none of the examined approaches supports all of our requirements to a sufficient extent. Therefore, there is a need to design a new context management framework that takes into consideration the distribution of context in different domains and the necessity to protecting user's privacy.

IV. CONTEXT DISSEMINATION PROBLEM

Consider a simple context federation scenario: a user is subscribed to a CS located in domain A; namely CSA. This server maintains the profile information of its subscribed users and maintains a sensor infrastructure for domain A. We call this server the *home domain server* (HDS) of its subscribed users. Likewise, the context server CSB maintains users' profiles and physical context information of domain B. Obviously as long as the user is still in the domain A the scenario is rather simple; all the context information needed by the application about this user exists in CSA. However, when the user move from A to B (we call the user a *foreign entity* in domain B), the context information related to the users maintained by CSA and CSB (such as location or environment context information) may become relevant to the applications interested in the user's context. In this case, we call the CSB the *visited domain server* (VDS). Thus, there is a need for a mechanism which allows applications to know which domains are visited by the user at any point of time and the context information gathered about the user in these visited domains.

One possible solution is to use tuple space (e.g., Confab [10]). Confab architecture structures context information into distributed tuple-spaces called *infospaces*, which store tuples about a given entity. An application interested in a certain context, builds a context query using the address of the responsible *infospace*. Although distributed *infospaces* contribute to decrease the context management overhead in a distributed environment, this distribution is not kept transparent to applications, which must know what *infospace* contains the desired context information. Another possible solution is to maintain in the HDSs "links" to the VDSs. In this case, in order to handle the application's queries about the users (or entities) over distributed domains, the HDS may have to distribute queries over the VDSs and compose the query results (e.g. [10][18]). However, this approach requires maintaining the link list of the VDSs, and may degrade the system performance as it requires distributing the application query over different servers and regrouping the result.

On the other hand, the notion of home and visited domains are also used by mobile telephone networks like GSM. The main idea used in these networks is that users have their "home domains" in which their context is gathered but when they roam to another domain this domain becomes a "visited domain". When a mobile device moves into a different domain, the server of the visited domain inter-links the mobile device and its home server. The home server redirects query statements to the server of the visited domain, which finally dispatches it to the mobile device. This is achieved by using the Home Location Register (HLR) and Visitor Location Register (VLR) approach of the GSM user profile database [23]. This approach addresses the location-awareness problem by minimizing the invocation of multiple updates in the home node each time a mobile user changed his/her location. However, the effectiveness of this mechanism is questionable for other types of context information, as it requires the application to submit their queries through a web of pointers from the home node to the visited node of the mobile user [24].

In fact, the main problem of context dissemination across domains originates from the observation is that in a distributed system there is an obvious trade-off between costs of updates and costs of requests; i.e. between the communication cost introduced by the fully replicating context data to the home node and the degree of replication that is eventually necessary. This has a direct impact on the achieved system performance and on the provided context precision. For example, when the volume of context data or the rate of change is high, providing high precision context value tends to degrade the performance; on the contrary, optimal performance can only be achieved by sacrificing the precision of the context copy. In the proposed approach, as we will see, the context consumers play a decisive role in the process of context replication as well as the update rate of the relevant context data.

V. THE PROPOSED APPROACH

Basically, when a CS receives a query referring to an entity's context information stored in the local repository the procedure is straightforward. When the required context information is not stored in the local repository it has to be retrieved from a remote CS. An efficient look-up mechanism for finding this context information is essential for the scalability of the whole system. To achieve this mechanism, we choose to synchronize the context information with the HDS only when there is a consumer for this information. This choice is made for the following reasons:

(i) Efficient cross-domain query handling: having all context information related to an entity in one place (HDS) can be exploited during the query resolution phase in order for the applications to retrieve the context information more efficiently. That is, handling a query submitted to the system requires considering the context information in the entity's HDS replicated from different domains instead of sending sub-queries to all VDSs. Thus, the querying response time decreases significantly.

(ii) Privacy ensuring: the alternative to publish the actual data at the HDS would be to only keep references to the relevant visited context server. However, this weakens the privacy support as the context data is stored by the foreign domain that provides the sensor infrastructure. Thus, we choose, as we will see, to design a protocol between CSs which force the context information to be centralized in the HDS. This way, enforcing user's privacy policy will be feasible.

(iii) Cross-domain reasoning: it becomes possible to reason about the context information across different domains (e.g. tracking and understanding user's tendency) and to identify the contextual situations which span different domains (see [6] for example). Moreover, this enforces the idea that each domain should have its own inference mechanism and in the home domain a cross-domain inference mechanism becomes possible.

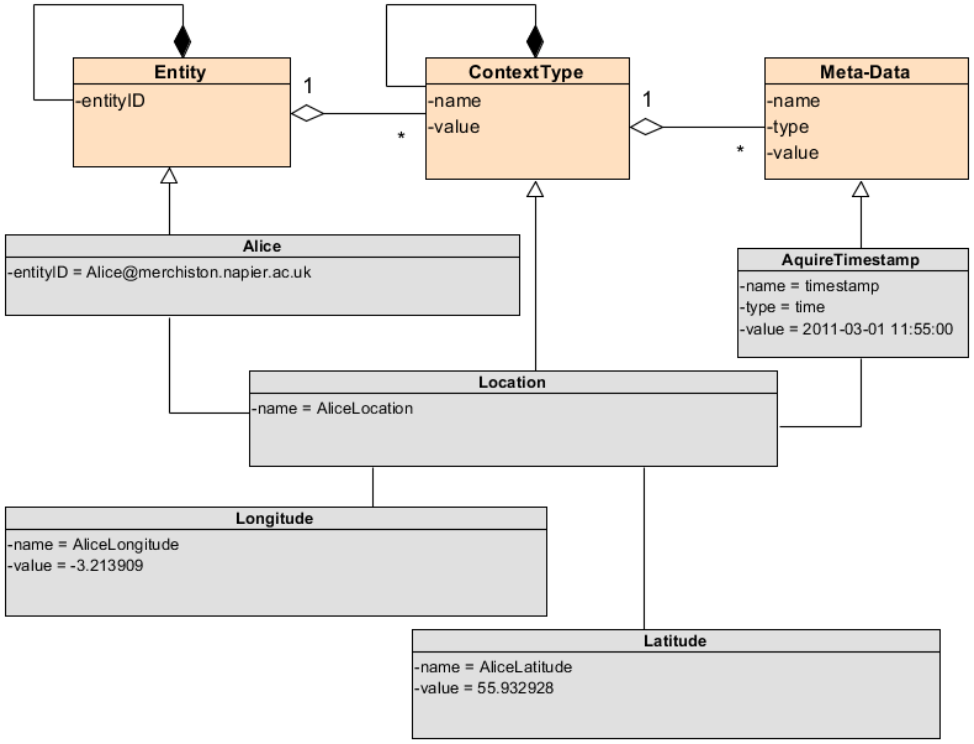

Fig. 1. The proposed context meta-model

(iv) High efficiency: it would be more efficient if we establish context replica on the HDS depending on how often the context change and at the same time on the context consumers needs. In situation of roaming users across domains, additional restrictions may arise (e.g. concerning the limited network connectivity, device power consumption, privacy enforcement, etc.), rendering imperative the need to establish an optimized mechanism in support of optimized context information dissemination among domains taking into account the explicit requirements of consumers.

In the following subsections, we present our designed and implemented framework, *ubique*, which aims at optimizing and controlling the amount of exchanged context information in such a way that context information can efficiently and easily flow from context providers to consumers. *ubique* envisions a highly distributed and loosely coupled solution in order to exchange context information between context providers, CSs, and applications. Therefore, *ubique* context management framework aims at: (i) enable the discovery of context providers, (ii) standardize context exchange between providers and consumers, (iii) federate contexts among CSs, (iv) standardize and enforce privacy, (v) allow context providers to publish on demand where there is a consumer, (vi) relieve CSs from the burden of replicating frequent updates to the HDS, and (vii) prohibit overloading the context consumers with context information that does not interest them for the time being.

A. *ubique Context Meta-Model*

Context information can be represented in many ways. For *ubique* context modeling, we choose an approach based on XML. As illustrated in Fig. 1, the context information is represented in terms of context elements, which provide information about *context entities, context types* and *meta-data*.

The main assumption in the proposed model is the representation of relationships between entity and information: context entities (such as persons, places, events, etc.) are identified and classified by an ID. Each context entity is associated with a set of context types (such as address, location, etc.) which may include other context types. Further, each context type may be characterized by a set of metadata which contain, for example, source of information, timestamps, expiration time, and any Quality-of-Context information such as accuracy and confidence.

B. *Context Management Componentss*

The *ubique* context management framework is designed for the discovery and exchange of context information across domains. It provides the relevant context information for the service or application, using distributed sensing infrastructure and centralized storing mechanisms. We define *ubique* context management framework as a set of components which are loosely coupled to provide relevant context information both by sensing and interpreting mechanisms. These key components or building blocks are depicted in Fig. 2, and described below.

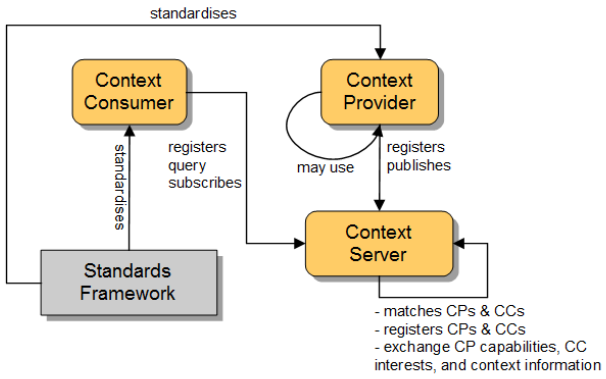

Fig. 2. *ubique* components

Context Consumer: (CC) is a software entity that uses the CS interface to register its context interest or query. The CC receives the requested context information asynchronously by submitting context interest and synchronously by submitting context query to the CS. A CC exposes interfaces to start receiving context information from the corresponding CS when they become available. These interfaces adhere to standards defined in the Standards Framework (SF).

Context Provider (CP): is a software entity that uses the CS interface to register its capability of providing context information. A CP exposes interfaces to publish context information to the corresponding CS on-demand. These interfaces adhere to standards defined in the SF. It is registered in the CS so that context consumers can discover and introspect it. Note that any software agent, reasoner, or storage component can be a CP as long as it adheres to the interfaces defined in SF. Usually, CPs wrap context sources such as GPS receiver or temperature sensor to provide their information.

Context Server (CS): provides a registration service for CPs to register/update/unregister their capabilities that uniquely describe their functionalities and for CCs to register/update/unregister their context interests that can be matched against the available CPs, and enables the discovery of various context providers. Additionally, it provides services to exchange the CCs' context interests and CPs capabilities between CSs as we will see later.

Standards Framework (SF): A set of specifications describing the CP capabilities, the CC interests and queries, the interfaces to exchange commands and context information between different components, a format to exchange an atomic context information element, as well as a format for privacy tags.

In *ubique* we rely on the reasonable assumption that a CS is identified by its Internet domain name and that the CS is responsible for managing the context information available in its domain. Additionally, each entity (sensor, user, application, etc.) has a unique ID that should be registered in one of the CSs. For example Alice ID could be `Alice@merchiston.napier.ac.uk` as she is a registered user in the CS of the domain `merchiston.napier.ac.uk` which is Alice's HDS.

C. Context Interfaces and Operations

ubique provides three different interfaces which allows integrating CSs, CCs, and CPs into the eco- system. In the following we describe the main interfaces and the main corresponding operations.

1)Integrating Context Providers: The provided operations allow registering CPs and their information with the CS as well as providing a discovery function with which participating components can check for available CPs.

registerContextProvider: This operation is used by the CP to advertize its capabilities in terms of the types of context information it can provide and the relevant entities playing a role in this information. Additionally, the registration provides a set of available CP meta-data (which mention information about the provider as well as quality of context information it provides). For example, the user's location can be measured with different quality by location sensors like GPS, CellId, WLAN-in-range, etc. Finally, registration provides further information about the registered entities. The CP capabilities XML scheme is depicted in Fig. 3.

Basically, the CP specifies in its capabilities its ID, the domain its information is originated from, and one or more *capability*. Each *capability* specifies its ID, the entities having the context information, and the supported context types. Optionally, it specifies the meta-data about these context types, its different attributes (features), and collection policies.

discoverContextProviders operation is used by CCs to get the list of available CPs and their capabilities for later query.

sendCPCommand: This operation is used by the CS to command a specific CP to start/stop publishing its information. The command message contains a reference (tuple ID) where the context information should be pushed.

2)Integrating Context Consumers: The provided operations allow registering CCs with the CS, querying (synchronously), as well as subscribing in order to be notified about context information (asynchronous).

queryContextServer: This operation is used by the CC to synchronously request for context information. The CC specify its interest in terms of the needed context types of specific entity(ies), as well as additional constraints on the CPs and context types meta-attributes.

subscribeContextConsumer: This operation enables long-lasting monitoring of the system. Basically, the logic of this operation is similar to the latter operation, but the request context information is returned in the form of an asynchronous "notify" callback operation. Fig. 4 depicts the CC interest XML scheme. The CC can specify one or more *interests*. Each *interest* specifies its ID, the entities the CC is interested to get their context information, and the interested context types. Optionally, it specifies the condition(s) on the context types, the domain(s) this information is originated from, the required feature(s) from the CP, and the ID of a specific CP.

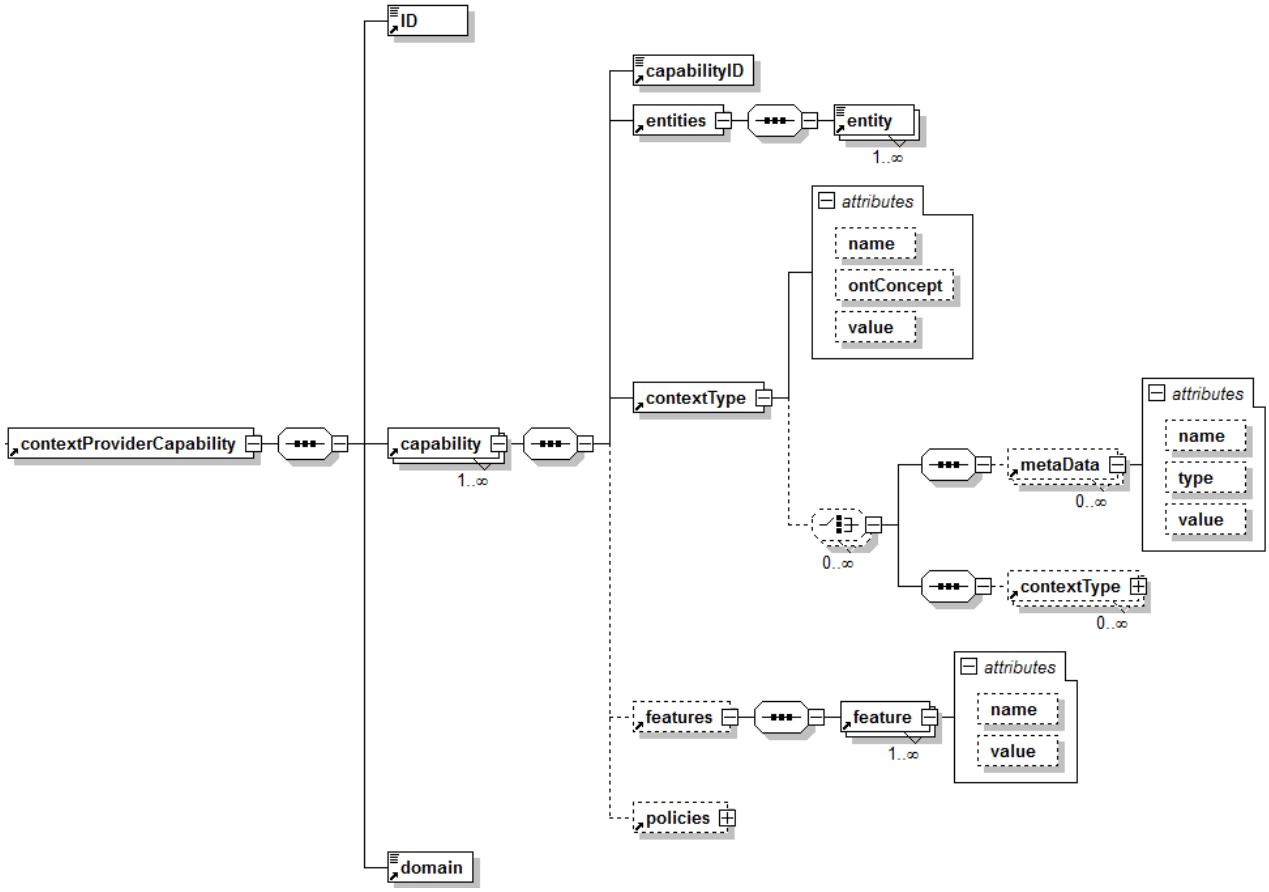

Fig. 3. CP capabilities XML scheme

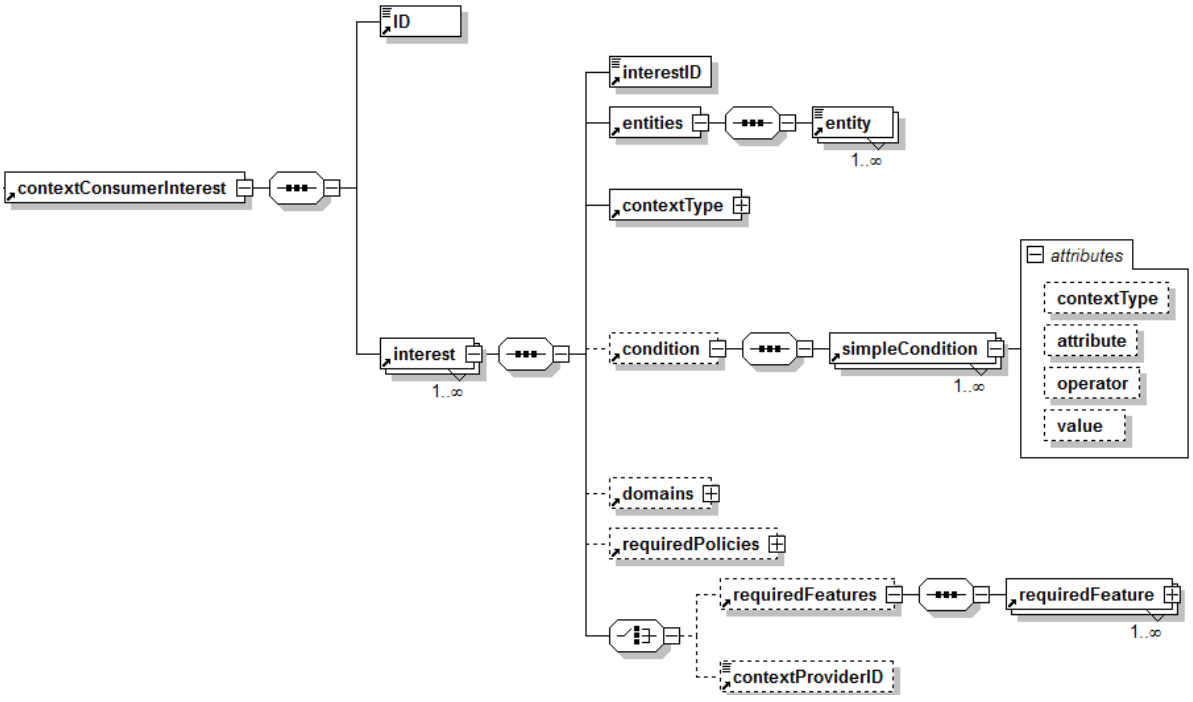

Fig. 4. CC interest XML scheme

sendCCCommand: This operation is used by the CS to command a specific CC to start/stop receiving the information it has subscribed to. The command message contains a reference (tuple ID) where the context information should be popped.

3)Federation between CSs: as already mentioned, every CS is responsible for providing and storing context information related to entities registered in it. Since the sensor infrastructure in each domain may provide context information about roaming entities, a collaboration protocol is needed between CSs in order to federate this information to the entities' HDSs. We can distinguish here between three types of information exchanged between CSs:

- CP Capabilities: CPs may advertise their ability to provide context information about entities not registered in the current domain. For example, a GPS sensor of Alice mobile phone can provide location information about Alice@domain1.com to the CS available in domain1.com (Alice's HDS). However, when Alice move to domain2.com, then this CP advertise its capability to provide Alice location information to CS of the domain2.com. In this case, CS of domain2.com should federate the CP capability to domain1.com (Alice's HDS) which is responsible to handle all queries related to Alice.

- CC Interests: A CS may receive context interest about entities not registered in it. In this case, the CS should federate these interests to the HDS of the corresponding entities.

- Context information: The idea is that each CS has to maintain a repository for all CP capabilities able to provide context information about its registered entities as well as all CC interests related to these entities. Any change in this repository (i.e. addition, updating, or deletion of CP capabilities or CC interests) should trigger a matching function which tries to bind a CP with a CC. When a match is found, a new tuple has to be created; a *startPublishing* command message has to be sent to the CP (via *sendCPCommand* operation) along with the corresponding CC interest and tuple ID; and a *startReceiving* command has to be sent to the CC (via *sendCCCommand* operation) along with the tuple ID. The CP now has all the information necessary to know what kind of context types, for which entities, and when to publish to the tuple (e.g. regularly or for a context changes greater than a specific threshold, etc.). Note here that when, for example, an application is interested in Alice location in `domain2.com`, the CS of `domain1.com` (Alice's HDS) will create a tuple in CS of `domain1.com` and ask the CP of Alice location to start publishing in this tuple. In other words, all the context information related to Alice, even those emerging from foreign domains, will be kept in her HDS. This way, we have more control about ensuring entities privacy. This mechanism is illustrated in the example usage in the Section 6. Fig. 5 depicts the XML scheme of the published context information which we call it *contextlet*. Basically, each *contextlet* specifies the CP ID, the interest ID (so that the CC knows that this information is related to which interest he has submitted), the domain from which this information is originated, the entity in question, a list of the requested context types and their values.

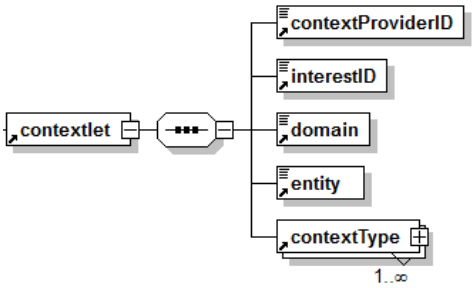

Fig. 5. Contextlet XML scheme

D. Privacy

Privacy is about protecting users' personal information, which may include also context information e.g. location, mood, etc. Obtained context information might be severely misused, e.g., to track users. In context aware environments, the devices belonging to the user communicate with the available CSs all the time, thus revealing privacy sensitive information about the user. In *ubique* approach, to ensure the confidentiality of the privacy-sensitive information, users have the flexibility to define their own privacy policy covering all types of context information that may be distributed in different domains.

Obviously, the sensor infrastructure in each domain may report context information related to entities out of the scope of the current domain which in turn weaken the privacy ensuring mechanism and loosen control over the context originated in different domains. In this case we need a mechanism with which the context information of the foreign entities can be moved to their HDS with the following conditions: (i) there is a corresponding consumer for this information, and (ii) revealing this information does not violate the privacy policy specified by the user. That is, when the CS finds a match between a CP and CC, it retrieves the privacy policy of the entity the CC specifies its interest in getting context information. If this request does not violate the user's privacy then the CP is asked to start publishing the required context information at the entity's HDS; otherwise, an "access denied" response is sent to the CC. Fig. 6 shows the privacy tag schema used in *ubique*. Each user (or each entity in general) has the flexibility to specify its privacy policy for each context type and for each domain. The *privacyTag* specifies for each context type the CCs having the right to get access to the context information and the time intervals during which this context information can be revealed to them.

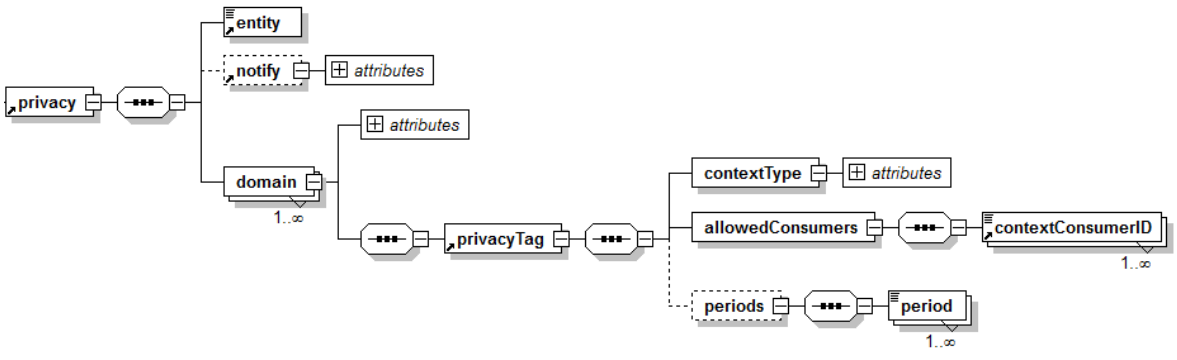

Fig. 6. Privacy XML scheme

Finally, secure storage of context information requires proper authentication and authorization to access it. Therefore, we assume here that each CC is a computational entity registered in one of the CSs which means that it has a unique ID and password, and it must be authenticated by its CS.

E. *ubique Implementation*

Fig. 7 illustrates the proposed domain-based context-aware computing eco-system. In general, the system should integrate distributed hardware and software components and provide naming scheme for those entities. The eco-system starts from a single system with client-server architecture; then multiple systems federate together through server-to-server communication to form the eco-system. A single system usually manages local clients, such as users and devices in a specific domain.

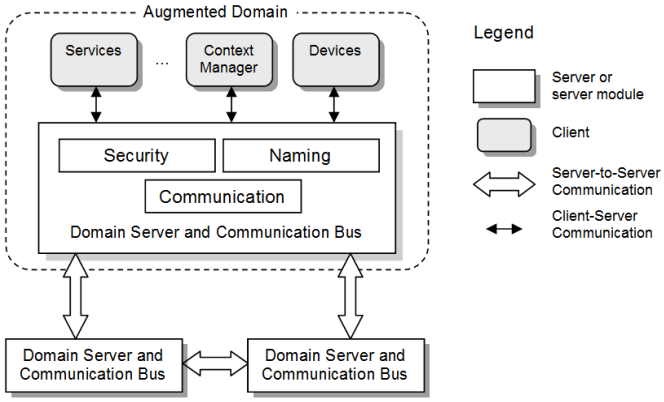

Fig. 7. Domain-based context-aware eco-system

The server is called Domain Server and Communication Bus. As indicated by its name, the server provides core functionalities, such as security and naming, and acts as a communication infrastructure for clients available in its administrative domain. The naming scheme is similar to that of e-mail systems. Each server has a unique domain name; clients have their names concatenated to the server name. Clients from different systems can also communicate with each other with the server-to-server communication. Clients could be devices, such as sensors, and applications that provide services to the user. Clients can be also services that provide functionalities the server does not provide such as the context manager (see Fig. 7). Clients have to be authenticated by the server to use the system.

Notice that the server does not provide context management service itself, leaving that responsibility to a separate client, the context manager. The context manager can be easily replaced or upgraded without affecting the whole system. The client-server and server-to-server communication interfaces are standardized, which facilitates the system extensibility.

In order to robustly implement the *ubique* approach, relying on a standard or already established protocol is obviously a preferred choice. The eXtensible Messaging and Presence Protocol (XMPP) [25] (also known as a Jabber protocol) is widely adopted open protocol for instant messaging and is designed for near real-time communication. In the following section we describe Jabber technologies by which *ubique* is inspired and based on.

1) *Jabber Overview*

Jabber is an extensible instant messaging (IM) system. More precisely, Jabber is a set of streaming XML protocols and technologies that enable any two entities on the Internet to exchange messages, presence, and any other structured information in near real-time.

The Internet Engineering Task Force (IETF) has standardized the core Jabber protocol as the XMPP protocol [26]. The architecture of the Jabber system is distributed. A Jabber server has a number of registered clients. Clients on the same server interact through that server; clients on different servers interact through server-to-server communication. Jabber enables message transfer not only between people, as in traditional IM systems, but also between any two entities. An entity can be a person, a device, or a software service. Each entity has a unique Jabber ID (JID). A JID is similar to an e-mail address. For example, a JID for Alice is `Alice@merchiston.napier.ac.uk`. Each entity is allowed to have multiple resources. For example, Alice may have a laptop and a cell phone which could be identified as `Alice@merchiston.napier.ac.uk/dell` and `Alice@merchiston.napier.ac.uk/nokia` respectively.

Furthermore, Jabber enriches the communication support beyond chat to many other interaction semantics thanks to the

XMPP extensions. The Jabber Software Foundation develops extensions to XMPP through a standards process centered on XMPP Extension Protocols (XEPs) [27]. Examples of these extensions are the Jabber RPC [XEP-0009], ad-hoc commands [XEP-0050], streaming audio and video [XEP-0166], and so on.

In addition, Jabber has an interesting pubsub facility [XEP-0060], in which both publishers and subscribers are Jabber entities. A publisher publishes a message item to a topic, and then all the topic subscribers will be notified to receive the newly published item. In this communication mechanism, since the publisher does not know who will receive the message, and a subscriber does not know who sent it, the time-coupling and reference-decoupling between publishers and subscribers are assured. This pubsub mechanism is ideal for implementing *ubique*, where context providers and consumers can be associated and disassociated dynamically.

2) Jabber and Domain-based Context Management

As aforementioned, the proposed domain-based context management architecture is based on Jabber technologies. Jabber has been chosen because its design, architecture, and features match our requirements: In the pervasive environment the interaction between different entities should be generic and not in a particular format. Jabber provides a rich set of communication mechanisms (see Section 5.5.1). Moreover, the context management infrastructure should support the interaction between different users, devices, and software components in a universal way. In Jabber systems, any entity that implements the XMPP-Core and its extensions protocols can establish a connection with a Jabber server and interact with other entities on any Jabber server. Thus the open architecture and standardization of the Jabber platform ease its adoption to build *ubique*.

Other than these capabilities, Jabber has other advantages such as its increasing popularity and community support; the availability of a set of servers, clients, and software libraries supporting a low-barrier entry for developers; and its adoption of XML to communicate messages between entities make it possible to leverage existing XML tools and libraries.

3) Jabber and Context Manager

Jabber entities can be implemented either as clients or as external server components. Clients use the protocols defined in "XMPP Core" to connect to the Jabber server; external components use the "Jabber Component Protocol" (JCP) [XEP-0114] for the connection. These two types of entities are functionally similar; thus for a given service, we can implement it as either a client or a component. However, unlike client components whose contact lists and subscription are maintained by the Jabber server, external component has to manage its subscriptions and contact lists by itself. The naming convention for external components is different from client components. For example, the context manager JID might be `context@merchiston.napier.ac.uk` if it is implemented as a client, and `context.merchiston.napier.ac.uk`, if it is implemented as an external component.

In *ubique* the context manager has been implemented as an external Jabber component. The choice of considering the context manager as an extension to the Jabber server functions is more of design decision than a functional one. Fig. 8 shows the architectures of the context manager: *Context*. The pubsub server is also a Jabber component. Context component connects to a Jabber server using JCP. The actual context data (contextlets) is stored in the pubsub so that the pubsub server will notify the subscriber of any context changes.

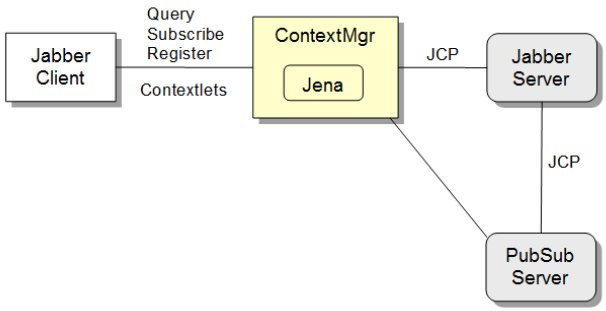

Fig. 8. The context manager external component

In Fig. 9, two Jabber servers are inter-connected; one of them connects to a CP and the other connects to a CC. The context manager, *Context*, connects to the Jabber server as a Jabber external component. The continuous lines represent the transport connections which are the actual routes for transferring data. On the other hand, the dashed lines indicate logical connections which means the communication between two end points does not happen directly, but through physical ones.

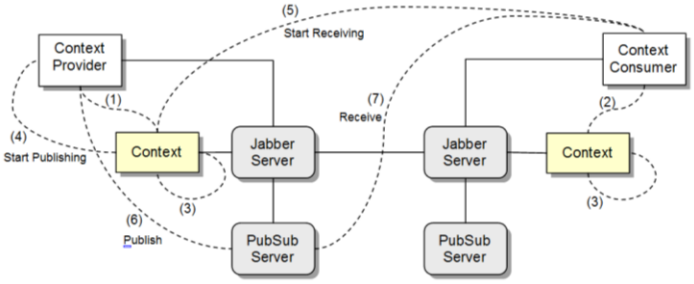

Fig. 9. *ubique* components interactions

When the system starts up, both CP and CC logon to their Jabber servers which may or may not be the same one. Then, the capabilities of each CP and the interests of each CC are registered into the corresponding Jabber server (Step 1 and 2). Thus the context manager can match the published CPs' capabilities with the CCs' interests or queries (Step 3). If the context manager decides that the CC interest matches the CP capability and this does not violate any entity's privacy, then it creates a tuple space in the local PubSub server and sends the *startPublishing* command message to the CP (Step 4) and the *startReceiving* message command to the CC (Step 5) along with the tuple space ID embedded in the message. Once the CP publishes a new contextlet (Step 6), the CC can receive it

asynchronously (Step 7). For the CC query, when the context manager decides which CP can have the requested context information it queries that CP and returns the result to the CC synchronously.

ubique is built on top of a number of technologies, such as Jabber (we use OpenFire [28] as a XMPP server), OWL, Jena [29], and XML. It leverages these enabling technologies to achieve the goal of controlling the context information dissemination between administrative domains in a way that is efficient in terms of saving network bandwidth and devices energy, as well as respecting people privacy in the pervasive environment. The system has a clear architecture and is highly extensible.

VI. CASE STUDY ON *UBIQUE* CONTEXT USAGE

Alice and her husband Bob work as lecturers in Edinburgh Napier University in Merchiston campus. Alice has a daughter, Carol, who studies in the same university in Sighthill campus. Alice would like to keep updated about her husband activities and her daughter location.

Fig. 10 depicts the sequence of exchanging information between different components: CPs, CCs, and CSs.

This is described as follows: The CP `ActivityProvider@merchiston.napier.ac.uk` registers the following capability in its HDS and wait for confirmation (Step 1).

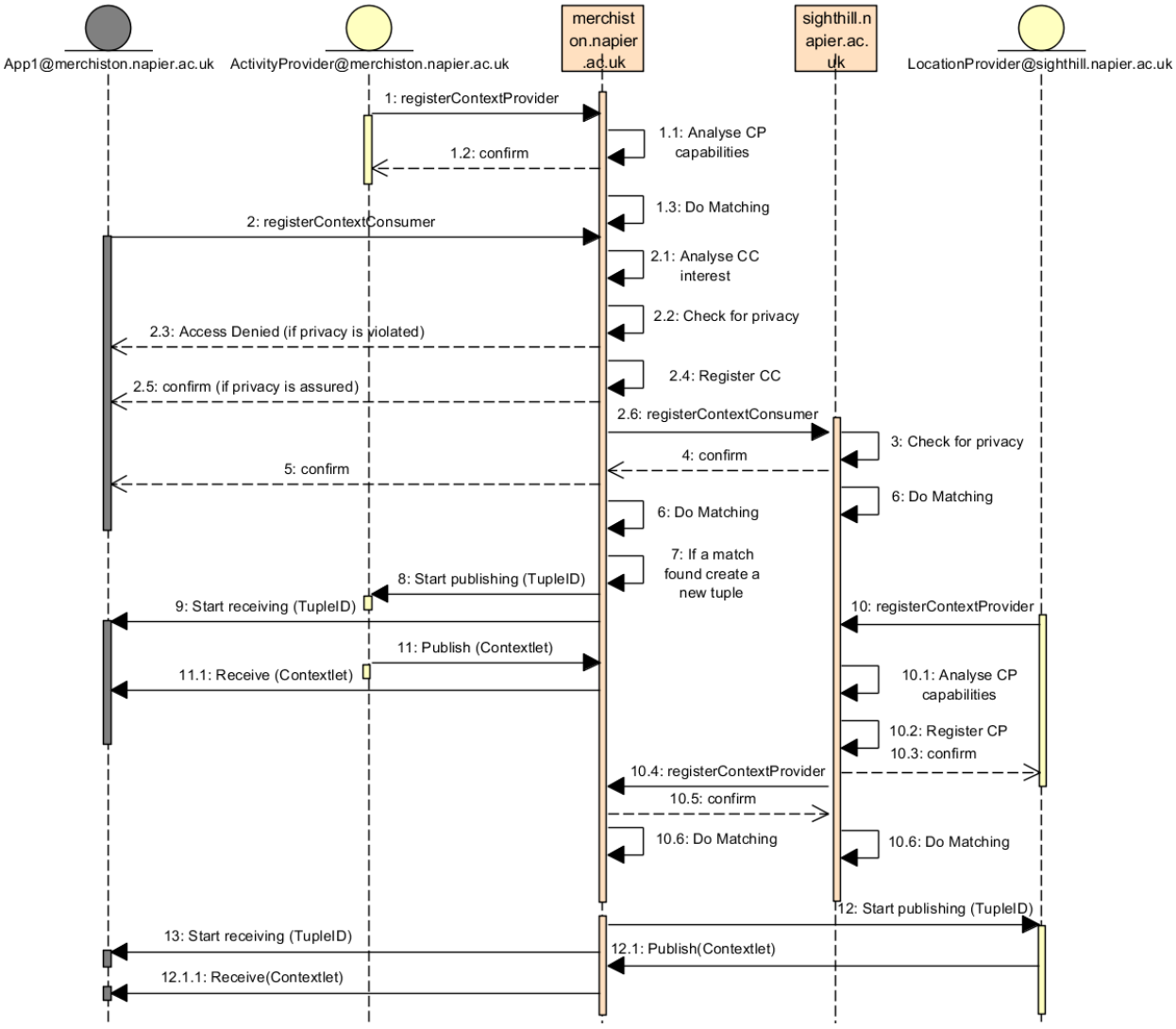

Fig. 10. Interaction between different components

```
<contextProviderCapability>
  <ID>CP1</ID>
  <capability>
    <capabilityID>CPC1</capabilityID>
    <entities>
      <entity>Bob@merchiston.napier.ac.uk</entity>
      <entity>John@merchiston.napier.ac.uk</entity>
    </entities>
    <contextType name="activity" ontConcept="http://www.napier.ac.uk/ontologies/percom.owl#Activity"/>
    <features>
      <feature name="confidence" type="float" value="0.85"/>
    </features>
  </capability>
  <domain>merchiston.napier.ac.uk</domain>
</contextProviderCapability>
```

Fig. 11. Example of the activity provider advertized capabilities

The CS analyzes the received CP capability to see if any of the supported entities is not registered in it. Because this CP does not provide context information about entities not registered in `merchiston.napier.ac.uk` no further interaction with other CSs has to be taken. Obviously, any change in the available CPs or CCs triggers the matching function.

For the sake of simplicity and without loss of generality, the example application `App1@merchiston.napier.ac.uk` is registered in Alice's HDS. It registers the following CC interest (Step 2):

```
<contextConsumerInterest>
  <ID>CC1</ID>
  <interest>
    <interestID>CCI1</interestID>
    <entities>
      <entity>Bob@merchiston.napier.ac.uk</entity>
    </entities>
    <contextType name="activity" ontConcept="http://www.napier.ac.uk/ontologies/percom.owl#Activity"/>
    <condition>
      <simpleCondition contextType="Activity" operator="gt" attribute="timestamp" value="2011-03-01 10:30:00"/>
    </condition>
    <domains>
      <domain>merchiston.napier.ac.uk</domain>
    </domains>
    <requiredFeatures>
      <requiredFeature featureName="confidence" operator="gt" value="0.8"/>
    </requiredFeatures>
  </interest>
  <interest>
    <interestID>CCI2</interestID>
    <entities>
      <entity>Carol@merchiston.napier.ac.uk</entity>
    </entities>
    <contextType name="location" ontConcept="http://www.napier.ac.uk/ontologies/percom.owl#Location"/>
    <condition>
      <simpleCondition contextType="Latitude" attribute="minAccuracy" operator="lt" value="0.000005"/>
      <simpleCondition contextType="Longitude" attribute="minAccuracy" operator="lt" value="0.000005"/>
    </condition>
  </interest>
</contextConsumerInterest>
```

Fig. 12. Example of an application context interest

This CC interest shows that the application is interested to know the location of Carol in any domain and the activity of Bob in the `merchiston.napier.ac.uk` domain. Note here that any CP registered in `merchiston.napier.ac.uk` domain or in any of its sub-domains is eligible to be matched with the interest CCI1. For each context interest, the CS checks for the corresponding entity privacy before registering it. Fig. 13 shows an example of Carol privacy tag.

```
<privacy>
  <entity>Carol@merchiston.napier.ac.uk</entity>
  <notify value="mailto:carol@merchiston.napier.ac.uk"/>
  <domain name="sighthill.napier.ac.uk">
    <privacyTag>
      <contextType name="location" ontConcept="http://www.napier.ac.uk/ontologies/percom.owl#Location"/>
      <allowedConsumers>
        <contextConsumerID>App1@merchiston.napier.ac.uk</contextConsumerID>
        <contextConsumerID>App4@sighthill.napier.ac.uk</contextConsumerID>
      </allowedConsumers>
      <periods>
        <period>...</period>
      </periods>
    </privacyTag>
  </domain>
</privacy>
```

Fig. 13. Example of a privacy policy

If the privacy is violated, an "access denied" message should be sent to the application; otherwise the following context interest will be registered and a confirmation message should be sent to the application.

The CS of `merchiston.napier.ac.uk` finds out that there is a match between the CP capability whose ID is CPC1 (Fig. 11) and the CC interest whose ID is CCI1 (Fig. 12), therefore, it creates a tuple and sends the necessary commands so that `ActivityProvider@merchiston.napier.ac.uk` starts publishing contextlets in the created tuple and `App1@merchiston.napier.ac.uk` starts receiving the published contextlets. Fig. 14 shows an example of the contextlet sent by the activity provider. Alice may like to send Bob a congratulations message when he finishes his presentation.

```
<contextlet>
  <contextProviderID>CP1</contextProviderID>
  <interestID>CCI1</interestID>
  <domain>merchiston.napier.ac.uk</domain>
  <entity>Bob@merchiston.napier.ac.uk</entity>
  <contextType name="activity" value="FinishPresenting" >
    <metaData name="timestamp" type="time" value="2011-03-01 11:55:00"/>
  </contextType>
</contextlet>
```

Fig. 14. Example of contextlet received from activity provider

In `merchiston.napier.ac.uk` there is no provider for Carol location. When Carol roams to `sighthill.napier.ac.uk` the CP `LocationProvider@sighthill.napier.ac.uk` reports its ability (Fig. 15) to provide Carol as well as other entities locations to CS of `sighthill.napier.ac.uk`.

```
<contextProviderCapability>
  <ID>CP2</ID>
  <capability>
    <capabilityID>CPC1</capabilityID>
    <entities>
      <entity>Carol@merchiston.napier.ac.uk</entity>
      <entity>Sally@sighthill.napier.ac.uk</entity>
    </entities>
    <contextType name="location" ontConcept="http://www.napier.ac.uk/ontologies/percom.owl#Location">
      <contextType name="latitude" ontConcept="http://www.napier.ac.uk/ontologies/percom.owl#Latitude">
        <metaData name="minAccuracy" type="float" value="0.000002"/>
      </contextType>
      <contextType name="longitude" ontConcept="http://www.napier.ac.uk/ontologies/percom.owl#Longitude">
        <metaData name="minAccuracy" type="float" value="0.000002"/>
      </contextType>
    </contextType>
  </capability>
  <domain>sighthill.napier.ac.uk</domain>
</contextProviderCapability>
```

Fig. 15. Example of the location provider advertized capabilities

The CS of `sighthill.napier.ac.uk` finds out that the location provider is able to provide Carol location which is not registered in it; thus, it federates the CP capability depicted in Fig. 16 to Carol HDS: `merchiston.napier.ac.uk` (Step 10.4 in Fig. 10). Notice that this capability is the same of Fig. 15 except that the entities not registered in `merchiston.napier.ac.uk` have been removed.

```
<contextProviderCapability>
  <ID>CP2</ID>
  <capability>
    <capabilityID>CPC1</capabilityID>
    <entities>
      <entity>Carol@merchiston.napier.ac.uk</entity>
    </entities>
    <contextType name="location" ontConcept="http://www.napier.ac.uk/ontologies/percom.owl#Location">
      <contextType name="latitude" ontConcept="http://www.napier.ac.uk/ontologies/percom.owl#Latitude">
        <metaData name="accuracy" type="float" value="0.000002"/>
      </contextType>
      <contextType name="longitude" ontConcept="http://www.napier.ac.uk/ontologies/percom.owl#Longitude">
        <metaData name="accuracy" type="float" value="0.000002"/>
      </contextType>
    </contextType>
  </capability>
  <domain>sighthill.napier.ac.uk</domain>
</contextProviderCapability>
```

Fig. 16. The location provider capabilities federated to the Carol HCS

After the re-matching process, the CS of `merchiston.napier.ac.uk` finds out that there is a CP able to provide Carol position. Therefore, as in the previous case, it creates a tuple and sends the necessary commands to the corresponding entities; however, this time the locally published contextlets are pushed by a CP from other domain. Fig. 17 shows an example of a contextlet published by the location provider indicating Carol location.

```
<contextlet>
  <contextProviderID>CP2</contextProviderID>
  <interestID>CCI2</interestID>
  <domain>sighthill.napier.ac.uk</domain>
  <entity>Carol@merchiston.napier.ac.uk</entity>
  <contextType name="location" >
    <metaData name="timestamp" type="time" value="2011-03-01 11:55:00"/>
    <contextType name="latitude" value="55.923215" >
      <metaData name="timestamp" type="time" value="2011-03-01 14:15:00"/>
      <metaData name="acurracy" type="float" value="0.0000002"/>
    </contextType>
    <contextType name="longitude" value="-3.286835">
      <metaData name="timestamp" type="time" value="2011-03-01 14:15:00"/>
      <metaData name="acurracy" type="float" value="0.0000002"/>
    </contextType>
  </contextType>
</contextlet>
```

Fig. 17. Example of Carol location contextlet

Fig. 18 depicts screenshots of the example application. The cyan circles represent roughly the domain border of each CS. Each small dot circle represents a contextlet.

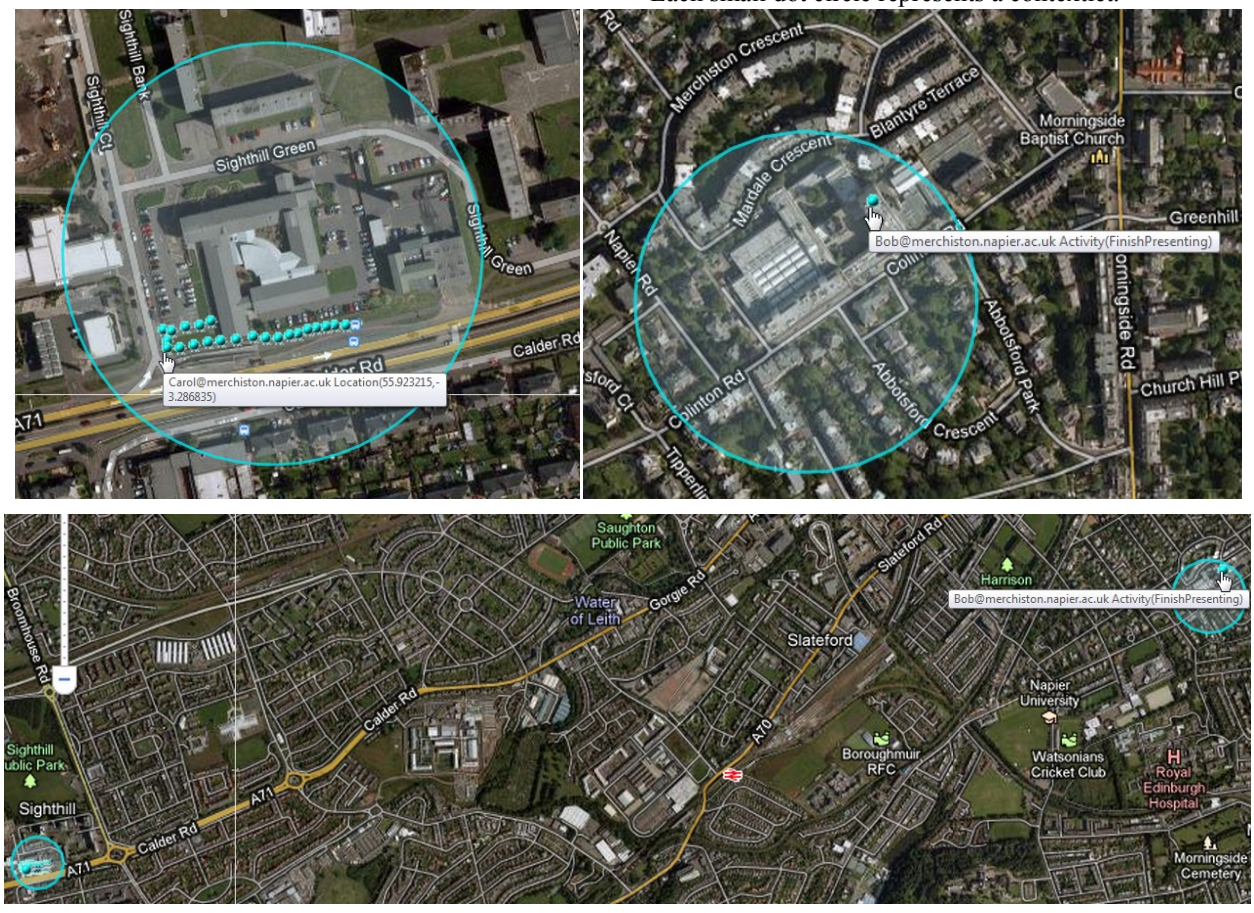

Fig. 18. Screenshots of the example application

VII. EVALUATION

In this section, we first analyze the suitability of the *ubique* approach according to the requirements of context management in pervasive applications as proposed in Section 2. The efficiency of *ubique* is then evaluated via a set of experiments based on the case study (Section 6).

A. Analysis of ubique vs. the requirements

Domains of context perception: This requirement, which is compliant with the principle of system boundary of pervasive applications, is achieved by using CS in each domain and the federation between CSs across different domains. Additionally, the notion of home domain CS simplifies application developments as it is the reference point for any context information related to the entities registered in it.

Uniform API interface and protocol: By providing the *ubique*'s set of open and generic APIs, context is made available to third party application developers to build new services without having to define specific mechanisms for context distribution and management between domains. In addition, these APIs and the proposed protocol between different entities enable external providers and consumers to be integrated into the *ubique* system to provide or consume context information.

Efficient context information dissemination: Since the communication resources are limited, and since most context information gathered by a context server will not be necessarily used by any application, *ubique* considers filtering and replicating only the context information that is explicitly required by an application.

Cross-domain reasoning: *ubique* provides an enabling infrastructure to support reasoning about the context information across different domains and to identify the contextual situations which span different domains. Moreover, this enforces the idea that each domain should have its own inference mechanism whereas in the HDS a cross-domain inference becomes possible.

Dynamic matching between context providers and consumers: In *ubique* the matching function of the context manager ensures efficient context information dissemination. In addition, since the CPs specify their capabilities in providing context information that correspond to different domains, an application can specify in its interests or queries the domain(s) from which it is interested in retrieving the context information.

Support for privacy: Since the context information is centralized in one CS (HDS), enforcing user's privacy policy which spans different domains is feasible. In addition, the dissemination protocol between CPs and CSs on one hand, and the between CSs on the other hand, ensures that the context information will not be stored everywhere and that this information will be disseminated only if the receiver has the privilege to get it.

B. Performance evaluation

The efficiency of *ubique* has been evaluated in terms of update latency. As part of the case study, evaluation experiments were done using four CSs distributed in four university campuses (Merchiston, Craighouse, Sighthill, and Craiglockhart) which store the context information available in their corresponding campuses. All 4 servers have the same hardware capability: Pentium 4, 3.40GHz and 4GB RAM. The aim is to measure the latency average of federating the contextlets from one CS to another. Fig. 19 shows the variation of the latency time (milliseconds) with respect to the number of contextlets simultaneously federated. Obviously the latency increases when the volume of data increases; however, the results show that the increase is not in a linear pace with the amount of contextlets, i.e. the latency is higher when the amount of contextlets is over 150. The latency could reach around 1.5s for sending 200 contextlets simultaneously, which is reasonable and acceptable even for the highly dynamic context information e.g. noise level.

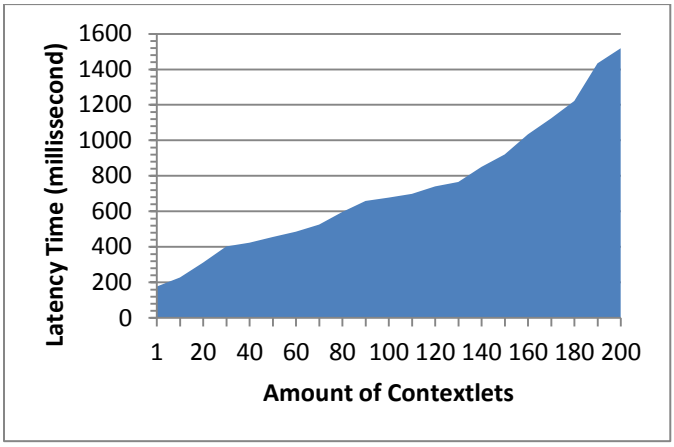

Fig. 19. *ubique* performance evaluation

VIII. CONCLUSION

The essence of context-awareness is to let applications and users take full advantage of the available context information e.g., users' or devices' locations. The requirement for universal context access demands for a middleware solution as an essential requirement for building context-aware systems. In order to address these new challenges, it is essential to establish innovative data storage and dissemination mechanisms. The architecture of *ubique* presented in this paper hides the increasing complexity of context management from applications and incorporates advanced mechanisms that support mobile users. In *ubique*, a Jabber-based context information dissemination protocol has been adopted. The storage and dissemination of the context information is performed by federation between distributed CSs. *ubique* brings several unique features to cross domain context management as discussed in section 7, all of which have been verified by the case studies.

Further research plans involve exploring the use of the middleware in more complex scenarios, extending *ubique* to support the geographic location based access to context information, the extension of the privacy protection scheme to consider not only specified domains but also domain types (e.g. a restaurant or a swimming pool), and ubique extension to

support context queries on the basis of the entities' and domains' types.

ACKNOWLEDGMENT

The work in this paper has been sponsored by the Lawrence Ho Research Fund (LH-Napier2012).

REFERENCES

[1] M. Weiser, "The Computer for the 21st Century," Communications, vol. 3, no. 3, pp. 3-11, 1991.

[2] D. Preuveneers, K. Victor, Y. Vanrompay, P. Rigole, M. K. Pinheiro, and Y. Berbers, "Context-Aware Adaptation in an Ecology of Applications," in Context-Aware Mobile and Ubiquitous Computing for Enhanced Usability: Adaptive Technologies and Applications, 2009, pp. 1-25.

[3] R. C. A. da Rocha, "Context Management for Distributed and Dynamic Context-Aware Computing," PhD Thesis, 2009.

[4] T. Kindberg and A. Fox, "System Software for Ubiquitous Computing," Pervasive Computing, IEEE, vol. 1, pp. 70–81, 2002.

[5] M. Valla et al., "The Context API in the OMA Next Generation Service Interface," in Proceedings of ICIN 2010, 2010.

[6] Z. Jaroucheh, X. Liu, and S. Smith, "Recognize contextual situation in pervasive environments using process mining techniques," Journal of Ambient Intelligence and Humanized Computing, vol. 2, no. 1, pp. 53-69, Dec. 2010.

[7] 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, no. 2, pp. 97-166, 2001.

[8] M. Román, C. Hess, R. Cerqueira, and R. H. Campbell, "A Middleware Infrastructure for Active Spaces," IEEE Pervasive Computing, vol. 1(4), pp. 74-83, 2002.

[9] A. Dearle et al., "Architectural Support for Global Smart Spaces," in Lecture Notes In Computer Science; Vol. 2574. Proceedings of the 4th International Conference on Mobile Data Management, 2003, pp. 153-164.

[10] J. I. Hong and J. A. Landay, "Architecture for privacy-sensitive ubiquitous computing," in 2nd International Conference on Mobile Systems, Applications, and Services, 2004, vol. p, pp. 177-189.

[11] K. Henricksen, J. Indulska, T. McFadden, and S. Balasubramaniam, "Middleware for Distributed Context-Aware Systems," in On the Move to Meaningful Internet Systems 2005: CoopIS, DOA. Proceedings of the OTM Confederated International Conferences: CoopIS, DOA and ODBASE 2005, Part 1., 2005, vol. 3760, pp. 846-863.

[12] S. L. Kiani, M. Riaz, S. Lee, and Y.-K. Lee, "Context Awareness in Large Scale Ubiquitous Environments with a Service Oriented Distributed Middleware Approach," in Fourth Annual ACIS International Conference on Computer and Information Science (ICIS'05), 2005, vol. 5, pp. 513-518.

[13] M. Grossmann, M. Bauer, N. Hönle, U.-P. Käppeler, D. Nicklas, and T. Schwarz, "Efficiently Managing Context Information for Large-Scale Scenarios," in Third IEEE International Conference on Pervasive Computing and Communications, 2005, no. PerCom, pp. 331-340.

[14] P. Floreen et al., "Towards a Context Management Framework for MobiLife," in In IST Mobile & Wireless Communications Summit, 2005.

[15] M. Klemettinen, Enabling Technologies for Mobile Services: The MobiLife Book. 2007.

[16] M. Strohbach, M. Bauer, E. Kovacs, C. Villalonga, and N. Richter, "Context Sessions – A Novel Approach for Scalable Context Management in NGN Networks," in MNCNA '07 Proceedings of the 2007 Workshop on Middleware for next-generation converged networks and applications, 2007, pp. 1-6.

[17] G. Percivall, C. Reed, and J. Davidson, Open Geospatial Consortium Inc . OGC White Paper OGC ® Sensor Web Enablement : Overview And High Level Architecture ., no. December. 2007, pp. 1-14.

[18] G. Chen, M. Li, and D. Kotz, "Data-centric middleware for context-aware pervasive computing," Pervasive and Mobile Computing, vol. 4, no. 2, pp. 216-253, 2008.

[19] D. Lee and R. Meier, "A hybrid approach to context modelling in large-scale pervasive computing environments," Proceedings of the Fourth International ICST Conference on COMmunication System softWAre and middlewaRE - COMSWARE '09, p. 1, 2009.

[20] J. Zebedee, P. Martin, K. Wilson, and W. Powley, "An Adaptable Context Management Framework for Pervasive Computing," in Context-Aware Mobile and Ubiquitous Computing for Enhanced Usability, 2009, pp. 114-146.

[21] H. K. Pung, T. Gu, W Xue, et al., "Context-Aware Middleware for Pervasive Elderly Homecare," IEEE Journal on Selected Areas in Communications (JSAC), Special issue on wireless healthcare, vol. 27, no. 4, pp. 510-524, 2009.

[22] G. Castelli and F. Zambonelli, "Contextual Data Management and Retrieval : a Self-organized Approach," in 2009 IEEE/WIC/ACM International Conference on Web Intelligence and Intelligent Agent Technology, 2009, pp. 535-538.

[23] A. Mehrotra, GSM System Engineering. Mobile Communications Series, Artech House Publishers., 1997.

[24] I. Roussaki, M. Strimpakou, C. Pils, N. Kalatzis, and N. Liampotis, "Distributed Context Management in Support of Multiple Remote Users," in Context-Aware Mobile and Ubiquitous Computing for Enhanced Usability, 2009, pp. 84-113.

[25] XMPP, "XMPP Standards Foundation," http://www.xmpp.org/, 2004. .

[26] P. Saint-Andre, "Extensible Messaging and Presence Protocol (XMPP): Core," http://www.ietf.org/rfc/rfc3920.txt, 2004. .

[27] "XMPP Standards Foundation (XSF). XMPP Extensions," http://xmpp.org/xmpp-protocols/xmpp-extensions/, 2010. .

[28] OpenFire, "OpenFire Server," http://www.igniterealtime.org/projects/openfire/index.jsp, 2010.

[29] "Jena2 Semantic Web Toolkit," http://jena.sourceforge.net, 2010.