Users context-aware

The GAMBAS Middleware for Smart City Applications

Marcus Handte [1], Stefan Foell [2] , Gregor Schiele [3], Umer Iqbal [1] , Wolfgang Apolinarski [1], Josiane Xavier Parreira [3], Pedro Marrón [1] , Gerd Kortuem [2]

[1] Unversity of Duisburg Essen, Germany firstname.lastname@uni-due.de

[2] The Open University, UK
firstname.lastname@open.ac.uk

[3] National University of Ireland, Galway, Ireland firstname.lastname@deri.org

ABSTRACT : *The concept of smart cities envisions IoT services that provide distraction-free support for citizens. To realize this vision, the services must adapt to the citizens' situations, behaviors and intents. This requires them to gather and process the context of their users. Mobile devices provide a promising basis for determining context in an automated manner on a large scale. However, despite the wide availability of mobile platforms, there are only few examples of smart city applications. One reason for this is that existing software platforms only provide limited support for common high-level tasks such as efficient data acquisition, secure and privacy- preserving data distribution or interoperable data integration. As shown by the Bus Navigator – a mobile transport application that has been deployed in the city of Madrid – the GAMBAS middleware can flexibly support such tasks and thus, reduce the development effort for a broad spectrum of smart city applications.*
KEYWORDS: *IoT, middleware, smart city, context acquisition, semantic data models*

1. Introduction

With the advent of powerful mobile devices an increasing number of people have constant access to information on the Internet. Nowadays, these devices are causing a drastic paradigm shift in the way people deal with information. Yet, the technical means to access information have only changed marginally. In most cases, information is accessed via the web which requires persons to memorize long URLs, click through web pages or browse through search results. In contrast, the concept of smart cities envisions IoT services providing distraction-free support. To realize this vision, the services themselves must adapt to the user's situation, behavior and intents at runtime. This requires services to gather and process the user's context.

Mobile devices provide a promising basis for determining user context in an auto- mated manner on a large scale. The vision of smart cities, however, extends beyond the boundaries of a single service as many city-wide applications will require the coopera- tion between multiple data providers and the citizens to exploit their full potential. As a consequence, developers that create applications for smart city settings are facing a broad range of challenges that are typically not addressed by widely available mobile platforms such as Android or iOS. Since these platforms primarily focus on low-level resource management, they only provide limited support for high-level tasks such as efficient data acquisition, secure and privacy-preserving data distribution or interop- erable data integration. As a result, application developers must manually tackle the resulting challenges.

The GAMBAS middlware with its associated software development kit (SDK) ad- dresses these challenges by simplifying these high-level tasks by means of a) models and infrastructures that support the interoperable representation and city-wide pro- cessing of (context) information, b) frameworks and methods to enable resource- efficient data acquisition using the mobile devices carried by the citizens, and c) pro- tocols and tools to derive, generalize, and enforce privacy-policies allowing citizens to control the sharing of their information. In the following, we describe the GAMBAS middlware and its SDK in more detail and we describe the Bus Navigator – a mobile transport application for the public bus network in the city of Madrid that has been developed and deployed successfully.

2. Smart Cities

GAMBAS envisions a smart city as a cloud of intelligent digital services that pro- vides adaptive and dynamic information to citizens. Conceptually, these services and their data can be grouped into so-called layers that cover different aspects of people's life in the city. A shopping layer might encompass services that manage store lo- cations and experience reports on different stores. Similarly, a mobility layer might encompass services that manage bus routes, subway stations, or traffic information. A social layer might manage relationships between citizens, events that take place in the city, etc. An environmental layer might manage information related to water or air quality in the city or it might capture the noise levels at different places. Clearly, some of the services found in these layers can apply to multiple layers as some pieces of information and some of the services might be applicable to multiple aspects.

To enable the creation of dynamic mash-ups of services, the services export (parts of) their information. Thereby, the information is represented using an interoperable data representation that allows automatic linking of different pieces of information. This makes the information accessible to other services which can then add additional value by providing, for example, a better experience for a specific group of citizens. In order to simplify the integration of services, a distributed query processing system enables the execution of queries across different information sources.

For providing up-to-date information and adaptive information to users, the layers capture information from different sensors embedded in various Internet-connected objects. Thereby, the objects may belong either to a particular service provider or they may belong to a citizen. The devices in the first category, may, for example, encompass sensors embedded in a taxi or a bus or they may be deployed at specific positions such as a bus stop or a metro station. The devices in the second category may encompass the personal mobile devices of the citizens such as their smart phones but they also may contain traditional systems such as their desktops at home, for example.

To protect the privacy of the citizens, they can control the collection and sharing of data with services in different layers. Towards this end, behavioral data is stored and processed on the devices that belong to the citizen. Optionally, in order to access additional services, they may share their information with specific service providers or other citizens. In order to avoid the expensive task of manually controlling the shar- ing process, automatic proposals for different settings can be computed automatically based on social relationships that are formalized by means of existing policies that the citizens created for similar contexts.

As a consequence, developers of applications in the smart city domain will typ- ically face the following three challenges. First, to provide up-to-date information in an adaptive manner, applications must be able to acquire (parts of) the context of the citizens using the mobile devices of the citizens. As mobile devices are typically battery powered, the acquisition of data must be resource-efficient. Second, to en- able services to leverage the acquired data, it must be shared. Intuitively, this sharing must be performed securely and it must respect the user-specific privacy requirements. Third, to enable devices and services to interact with each other, they must establish a common understanding of their respective data such that it can be linked to each other in an interoperable and extensible manner. Our GAMBAS middleware provides the tools to simplify these tasks, as described next.

3. GAMBAS Middleware

In the following, we describe the different components of the GAMBAS middle- ware and detail how they support data acquisition, data distribution and data integration.

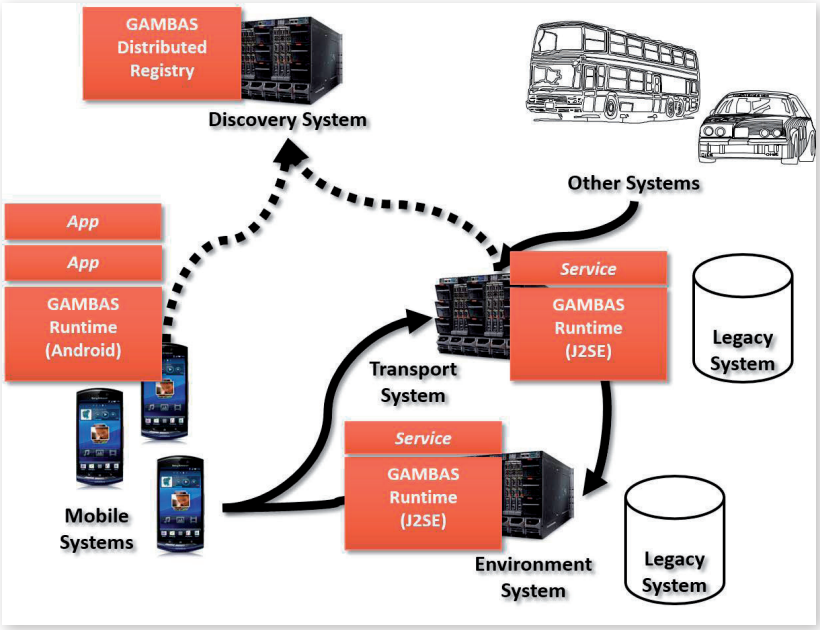

Figure 1. GAMBAS System Architecture

Figure 1 shows the overall system architecture. Conceptually, a GAMBAS deployment consists of different systems. The mobile systems of the citizens are act- ing as both data sources and data sinks. The city systems from different layers are managing data and providing services to the mobile systems. To do this, they may rely on information provided by other services as well as data provided by the mobile systems. Furthermore, they may integrate data from legacy systems and they may be receiving data from other systems through existing means. For example, a transport system might already be collecting information through sensors deployed in a city or in vehicles. Similarly, it may already have access to geographic data describing the different roads in the city.

Intuitively, it is impractical to assume that the existing systems and data sources are likely to be replaced by any middleware. Instead, GAMBAS embraces existing systems and provides ways to simplify the sharing of the information hosted by them. To do this, the GAMBAS middleware introduces three system components, namely the distributed registry, the Android runtime and the J2SE runtime. In the following, we briefly outline their functions. Targeted at mobile devices, the *Android runtime* enables developers to acquire data using the built-in sensors. Furthermore, it allows secure sharing of data with services while enforcing the privacy requirements of the user. Finally, it simplifies the dynamic retrieval of data provided by services. Targeted at systems that provide services, the *J2SE runtime* enables application developers to capture data provided by mobile devices and to offer service-specific data in a way that enables dynamic discovery and simplifies linking. Finally, to support dynamic in- teraction, the *distributed registry* enables mobile devices and services to publish meta data. This enables devices and services to find each other and to establish commu- nication channels. The Android and the J2SE runtime are the core subsystems that application developers interact with. In contrast to this, the distributed registry is pro- viding the glue in order to enable interaction. For the sake of brevity, we focus our following descriptions on the different runtime systems. To avoid repetitions, we first discuss the individual components. Thereafter, we discuss how they are integrated in the Android and the J2SE runtime.

3.1. Runtime Components

Conceptually, the runtime system consists of four main components, namely com- munication, data acquisition, data processing and privacy and security.

3.1.1. Communication

To enable communication between different devices using different technolo- gies, the GAMBAS runtime system relies on the BASE communication middleware (Handte *et al.,* 2010b). BASE enables the spontaneous interaction between different devices, while supporting many communication technologies through a plug-in archi- tecture. Additionally, BASE bridges between different communication technologies, whenever necessary. Using a gateway that is integrated into the distributed registry, BASE is able to establish communication channels, even if devices are behind a cor- porate firewall or a NAT router - which is commonly the case in mobile or home networks. In addition to communication, BASE provides a simple service abstraction which is used by GAMBAS to export system services that are used, for example, to execute distributed queries over the data provided by services or to establish secret keys. From an application developer's perspective, however, the GAMBAS runtime hides all details related to communication and BASE. Consequently, we would like to refer to (Handte *et al.,* 2010b) for more details on BASE.

3.1.2. Data Acquisition

Besides from communication, another capability of the GAMBAS runtime is its ability to capture data on behalf of the user or a service provider. For this it encom- passes a data acquisition component that is capable of running on different devices. Internally, the data acquisition component is using the NARF activity recognition framework (Handte *et al.,* 2010a). NARF is a component-based, extensible activ- ity recognition framework that integrates with different types of physical sensors such as an accelerometer or GPS as well as with virtual sensors such as the user's calendar, for example. It provides an existing toolkit of reusable components such as preprocessors in the time and frequency domain, classifiers, etc. Furthermore, it has been used to build various context recognition applications. When multiple context recognition applications are executed simultaneously on a single device, NARF optimizes the data acquisition with respect to energy consumption by removing redundant sampling and redundant computation (Iqbal *et al.,* 2012). Using a visual editor that is integrated into the Eclipse development environment, an application developer can compose compo- nent configurations via drag-and-drop to recognize a particular piece of context. Once the composition is completed, the developer can export the configuration into Java code that can be interpreted and executed by the data acquisition component.

3.1.3. Data Processing

To store and manage data of services as well as data generated by devices, the GAMBAS runtime encompasses a data processing component. Internally, the data processing component itself is structured in three sub-components that (a) store the data, (b) query for data and (c) discover data. To store the data of the user on a lo- cal device or remotely at a particular service, the runtime uses a semantic data storage (SDS) component. Similar to the data acquisition component, the SDS is primarily tar- geted at resource-constrained devices. The data that is stored in the SDS component follows the linked data principles (Bizer *et al.,* 2009) and makes use of interoperable data representations, storing data using RDF (RDF Working Group, 2004). Further- more, the data storage is able to interface with different types of query processors, depending on the resources available on the device. To make the data stored in the SDSs available to services and applications, two types of query processor components are used. The query processors are capable of executing queries on top of the storages. As query language, SPARQL (RDF Data Access Working Group, 2008) is used. Since SPARQL queries can be heavyweight, on resource-constrained environments such as smart phones, queries are executed using RDF-on-the-go (Le-Phuoc *et al.,* 2010). It is compatible to other, full-fledged semantic web frameworks like Jena (Apache Jena project team, 2013), but runs on Android devices. To enable transparent distributed query processing, the query processors must be able to discover the data sources that are available on the network. To make the data discoverable, a device may announce the data available in the SDS to the data discovery registry which in turn will typi- cally use a semantic data storage component to manage the announcements. In case of personal devices, the announcement may be limited or modified depending on the privacy preferences of a certain end user. To enable this, the semantic data storage and the data discovery registry are interfacing with the privacy and security component.

3.1.4. Privacy and Security

As some data processed within smart city applications, such as the user location, might be sensitive from a privacy perspective, it is necessary to limit the data acqui- sition and the data sharing such that it respects the privacy preferences of different entities. Achieving this is the primary task of the privacy and security component. The component interacts with the SDS as well as the data acquisition component that is deployed on each personal device over a privacy manager interface. In addition, the privacy and security component may also be used to limit the access to information that is provided by a particular service. For this, it is integrated into the device that is offering the service. Using a privacy policy that can be generated automatically by means of plug-ins that access proprietary data sources, the privacy and security component takes care of exporting sensitive data in a way that it can only be accessed by legitimate entities. Furthermore, depending on the user preferences, it may apply obfuscation in order to limit the data precision and it may anonymize the data in order to unlink the data from a particular user. Envisioning the use of mobile devices as primary sources of data, the privacy and security component is supporting not only traditional laptops and PCs, but also resource-constrained devices as its execution platform. In the runtime system, the privacy and security component is divided into a key-exchange mechanism, policy-based access control and encryption protocols.

3.2. Runtime Integration

As depicted in Figure 2, the integration of the components of the runtime sys- tem described previously is realized by: (1) a set of interfaces, support libraries and tools called the software development kit (SDK), and (2) the GAMBAS CoreService which provides the accompanying runtime environment. The SDK in turn consists of two parts. The service programming interface (SPI) is used to develop middleware functions. The application programming interface (API) is used to develop GAMBAS apps. The CoreService sets up the GAMBAS runtime and manages the lifecycle of runtime components. Furthermore, the CoreService realizes the SPI by linking each component to all other components that they use in their own execution via interfaces from the SPI. This effectively provides a tight and efficient integration between the components without inducing dependencies to their actual implementation. Finally, the CoreService implements the GAMBAS API towards GAMBAS applications both on Android and J2SE platforms. It receives calls, forwards them to the right compo- nent and delivers results back to the original caller.

The J2SE version of the runtime system specializes and implements the generic middleware architecture described before for server systems running J2SE. This al- lows service providers to integrate their services into the GAMBAS platform. The J2SE version of the runtime system is implemented as a library that is linked to an ap- plication using it. To start using the middleware, an application has to first import and instantiate the CoreService. To use functionality of the GAMBAS middleware, e.g. to store data, applications can call a number of methods on the CoreService. The Core- Service in turn forwards this request to the corresponding local system component, retrieves results from it and forwards them to the calling application. On Android the runtime system is realized as a stand-alone Android app instead of a linkable library. This design allows us to efficiently share a single instance between all 3rd party apps, reducing the needed resources and thus allowing the OS to keep more apps active in memory for a longer time. As a side effect, direct calls are no longer possible. Instead, we use Android intents which are small messages that a process can publish and that can be received by other processes. The user interface of the middleware application enables the user to configure a multitude of aspects such as the used data discovery registry and communication gateway, the user's pseudonym, known friends, their keysand privacy policies. This allows users to inspect and adapt the current system state in one integrated place and makes it easier for them to understand what data is currently made available to whom.

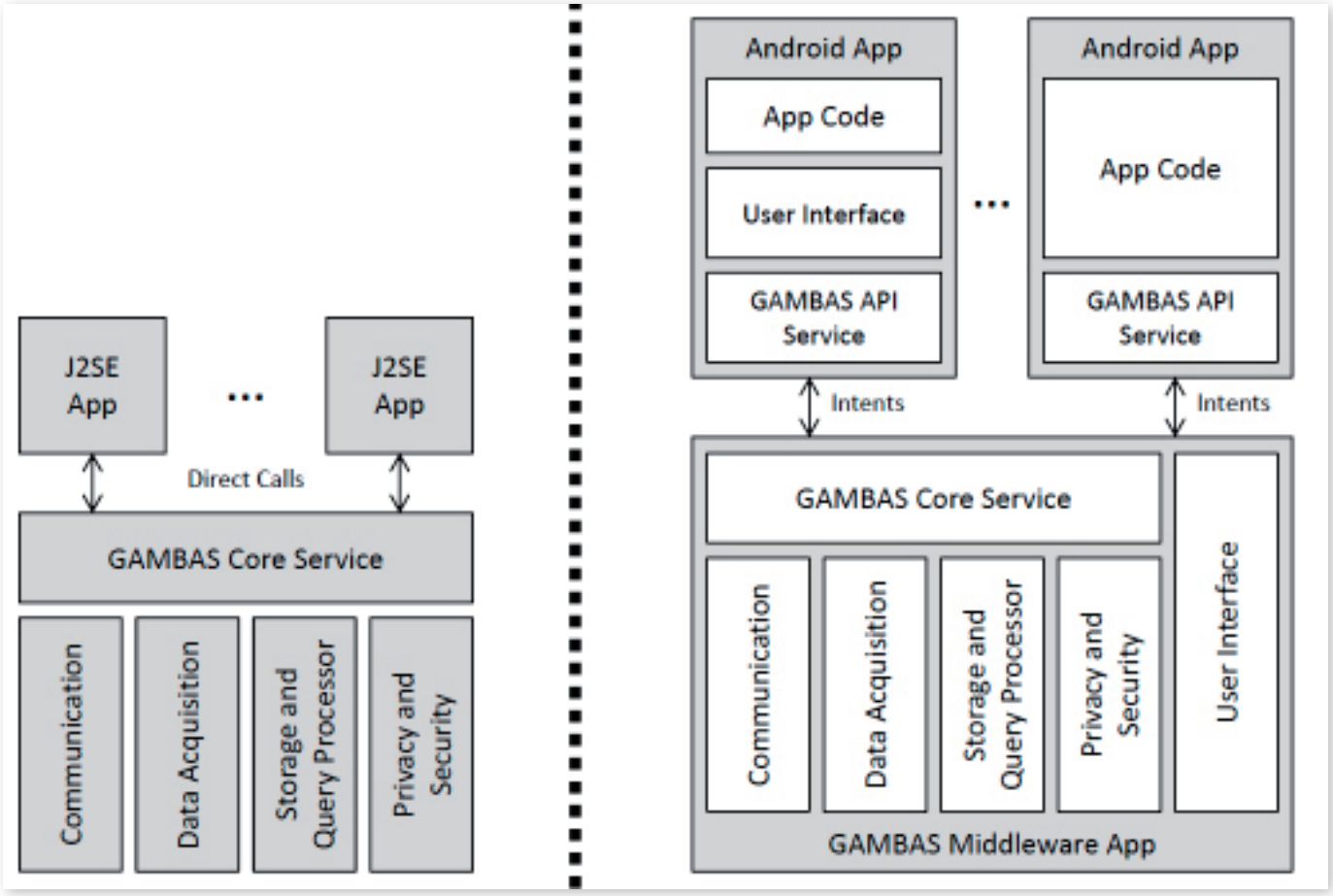

Figure 2. GAMBAS Runtime System for J2SE (left) and Android (right)

4- Mobile Transport Application

The Bus Navigator (c.f. Figure 3) is a mobile transport application prototype for the public bus system in the city of Madrid. It represents one of the two proto- type applications used to validate the GAMBAS middleware and it has been deployed and tested successfully for several months in Madrid. As a common feature of mo- bile transport applications, the Bus Navigator provides up-to-date information on bus schedules and routes. In contrast to existing applications, however, the Bus Naviga- tor provides several innovative and powerful features built on top of the capabilities provided by the GAMBAS middleware to help travelers find their way through large, city-scale transport networks. Most importantly, the Bus Navigator is designed as a continuous navigation service that can take advantage of the sensors built into the smart phones carried by travelers. These sensors capture context and activity informa- tion, which is then interpreted and processed on the smart phone to provide and update real-time transport information relevant to the traveler's current transport behaviour. The key features enabled by the GAMBAS middleware include responsive and pre- cise speech input, advanced trip tracking and information visualization as well as bus ride detection and crowd-level estimation bundled into interoperable behavior-driven services provided by GAMBAS.

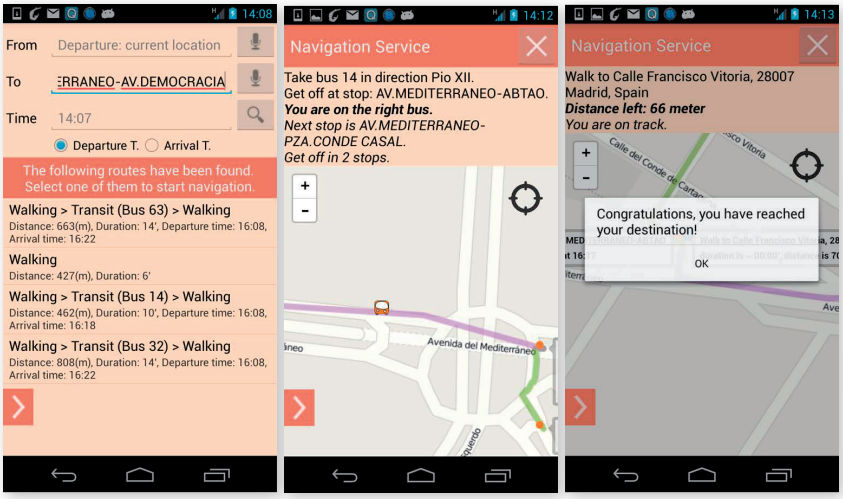

Figure 3. Bus Navigator for the Public Bus Network of Madrid

To support bus detection and trip tracking, the bus navigator ties into the Wifi access points that are already deployed in the buses and provide Internet connectivity to the travelers. Using this deployment, advanced algorithms running on the smart phone determine autonomously when the travelers enter and leave a bus – even if they are not connected to the access points. This enables the application to display relevant information such as the next bus stop, the number of remaining bus stops on the traveler's current route or the fact that the traveler has accidentally missed the stop. When a bus ride has been completed by the traveler, it is stored locally on the smart phone in a SDS. The resulting history of bus rides is used to determine the regular trips made by the traveler, thereby, enabling the automatic prediction of future trip destinations. If a future destination cannot be predicted, travelers can configure voice commands to speed up and optimize the interaction with the Bus Navigator – resulting in a more efficient and more enjoyable user experience. Besides enabling bus detection, the Wifi access points in the buses are also able to determine the number of passengers in the bus. This crowd-level information is collected and anonymized on the access point and transmitted to a remotely located SDS. To the benefit of the travelers, the information is aggregated and displayed as part of the routing data. This enables travelers not only to optimize their trips with respect to timing but also with respect to comfort. In addition, the bus operator may use this information to dispatch additional buses and to optimize the bus network schedules.

5 -Conclusions

Despite the wide availability of versatile programmable mobile platforms, there are only few examples of smart city applications. One reason for this is that existing software platforms primarily focus on low-level resource management which requires application developers to repeatedly tackle high-level tasks. The GAMBAS middleware simplifies the development of smart city applications focusing on three common tasks, namely efficient data acquisition, secure and privacy-preserving data distribution as well as interoperable data integration. As indicated by the Bus Navigator ap- plication, with the current implementation of the GAMBAS middleware, it is possible to develop and successfully deploy a smart city application in the transport domain. As a next step, we are working on extending this application to encompass several additional domains such as environmental monitoring and social activities. Towards this end, we are refining the data models and optimizing the middleware performance with respect to data acquisition and data processing. More details on the GAMBAS middleware, SDK and applications can be found at *http://www.gambas-ict.eu*.

Acknowledgment

This work is supported by UBICITEC e.V. (European Center for Ubiquitous Technologies and Smart Cities) and GAMBAS (Generic Adaptive Middleware for Behavior-driven Autonomous Services) funded by the European Commission under FP7 with contract FP7-2011-7-287661. The authors would like to thank the remain- ing members of the GAMBAS consortium for their work on and support for this paper.

6. References

Apache Jena project team, « Jena, a Java framework for building Semantic Web applications », http://jena.apache.org, 2013.

Bizer C., Heath T., Berners-Lee T., « Linked Data - The Story So Far », International Journal on Semantic Web and Information Systems, vol. 5, n 3, p. 1-22, 2009.

Handte M., Iqbal U., Apolinarski W., Wagner S., Marron P. J., « The NARF Architecture for Generic Personal Context Recognition », Sensor Networks, Ubiquitous, and Trustworthy Computing (SUTC), 2010 IEEE International Conference on, p. 123-130, 2010a.

Handte M., Wagner S., Schiele G., Becker C., Marron P. J., « The BASE Plug-in Architecture Composable Communication Support for Pervasive Systems », 7th ACM International Conference on Pervasive Services, July, 2010b.

Iqbal U., Handte M., Wagner S., Apolinarski W., Marron P. J., « Configuration folding: An energy efficient technique for context recognition », Pervasive Computing and Communi- cations Workshops (PERCOM Workshops), 2012 IEEE International Conference on, 2012.

Le-Phuoc D., Parreira J., Reynolds V., Hauswirth M., « RDF on the Go: An RDF Storage and Query Processor for Mobile Devices », Posters and Demos of the ISWC 2010, 2010.

RDF Data Access Working Group, « SPARQL Query Language for RDF », http://www.w3.org/TR/rdf-sparql-query/, 2008. RDF Working Group, « Resource Description Framework (RDF) », http://www.w3.org/RDF/, 2004.