Android/OSGi-based Machine-to-Machine Context-Aware System

Matko Kuna, Hrvoje Kolaric, Iva Bojic, Mario Kusek and Gordan Jezic
University of Zagreb
Faculty of Electrical Engineering and Computing
Department of Telecommunications
Unska 3, HR-10000 Zagreb, Croatia
{matko.kuna, hrvoje.kolaric, iva.bojic, mario.kusek, gordan.jezic}@fer.hr

Abstract— **This paper presents a context-aware system that uses Machine-to-Machine communication to retrieve sensor data collected by an Android operating system smartphone. It uses the eXtensible Messaging and Presence Protocol for communication with a remote server where data are stored, while both ends are implemented in the Open Service Gateway initiative framework providing system modularity and portability. Although our system is designed as a generic system that can be easily applicable for various monitoring purposes or on-event reactions, its functionality has been evaluated by implementing an environment-variables monitoring application used in industrial sectors. The case study presented in this paper elaborates a context-aware application in terms of monitoring and alarming functions. In our case study Android devices are used for communication purposes and monitoring environment parameters in order to track context data and react to it.**

I. Introduction

Nowadays devices (e.g. PCs, smartphones, positioning devices, health monitors) in our environment are expected to work on high levels of independence, performing programmed actions that benefit their users in everyday life. In order to meet the set of requirements, these different devices are connected together performing certain tasks. This concept is known as Machine-to-Machine (M2M) communication [1]. M2M is a concept that defines the rules and relations between devices while cooperating. It implies a highly automated usage of a set of devices simultaneously, without much need for human interaction.

Although with the increase of computational power, now it is even possible to run different M2M tasks on various consumer electronics (e.g. television sets, set-top boxes), smartphones are still more frequent used in M2M domain. In 2007 smartphone sales number exceeded laptop sales [2]. Development of those devices and the technology behind them enabled PC-like performance on pocket-size gadgets. Just a few years back the main part of a cell phone was the communication processor, while the application processor was just a low-end microcontroller. Today, with the falling cost of smartphones, application processors have the main role.

Even more, with further development those devices will become capable of running software agents which will further broaden their usage domain. Additionally, smartphones are equipped with various sensors (e.g. temperature, accelerometer, lights) making it easy to collect different information and build context-aware systems (e.g. like in [3]). For instance, by keeping track of information that describes current location of a mobile user, it is possible to develop real-time monitoring or aiding systems. Obviously, there are many possibilities for new services and systems and that is the motivation behind our work. Namely, our main idea is to develop easy-to-use, fast and cost-efficient systems with significant possibilities for appliance.

Therefore, in this paper the focus is on describing a generic context-aware M2M framework that traces changes of a set of environment variables that are monitored using M2M devices. Since there is a high percentage of fragmentation in terms of different hardware and software components on the market, we use Open Service Gateway initiative (OSGi) framework to overcome the existence of different devices. For the security reasons we have chosen to use eXtensible Messaging and Presence Protocol (XMPP) for communication between devices in the M2M environment.

The problem we are addressing is how to collect sensor data from different devices in a heterogeneous M2M environment, send them through different M2M networks and act upon them on a dedicated server. However, for the purpose of the case study, we implemented our system using a smartphone running on the Android operating system as an end device. A framework like our could be easily applied to monitor environmental changes in different areas like various geographic locations or industrial facilities. Additionally, it could be used to track positioning data and changes of other variables (e.g. magnetic field, noise level). And since today's smartphones come with multiple built-in sensors and operating systems such as Android, it becomes easy to use them and connect them with other computing devices.

The rest of this paper is organized as follows. Section II introduces M2M and context-aware concepts, Section III refers to the papers where similar combination of technology is used and gives insight to diversity of appliance. Section IV describes key technologies used in our work, while Section V introduces our system architecture and describes relations between system entities. Section VI elaborates design and implementation of a proof-of-concept described as a case study in Section VII. Section VIII gives evaluation results, while Section IX concludes the paper and proposes directions for future work.

II. Machine-to-Machine and Context-Aware environment

The number of devices connected through a network where they can mutually communicate is rapidly growing, and thus the M2M concept is gaining significance. M2M network is a network of devices with diverse functionalities and capabilities that interact with only limited human intervention. A plethora of different devices communicating by using different technologies (e.g. wire lines, WiFi, 2G/3G, Bluetooth) creates a heterogeneous environment where data can be collected, analyzed and acted upon, depending on system design. Data collection and data assessment are also related to context-awareness. In order to have a context-aware service, data have to be collected and used to enable provision of context-dependant actions. For instance, data that describes a certain environment, can be used for a location-based service.

A. *Machine-to-Machine concept*

Appliance of M2M can be found in many industry branches, such as transportation, manufacturing, security and e-Health [4]. Such a strong presence in different areas creates new large scale business opportunities and makes M2M a key concept, especially in telecommunications.

A typical M2M based application contains four basic stages: data collection, data transmission (through a communication network), data assessment and response to the available information [5]. Data are collected through sensors – different kinds of them are built-in part of smartphones (e.g. microphone, camera, gyro sensor, GPS transceiver), and they can be collected periodically or on request. Sensors can also be stand-alone modules that are controlled by a M2M device (i.e. in this case a smartphone). After collecting data, data are sent through a network to another M2M device that can store that data, analyse them or act upon them (see Figure 1). Those actions belong to data assessment and response phases that come after the receiving data phase.

In a M2M system, communication network presents the infrastructure that enables communication between M2M entities. It can be carried out over a cellular network, Public Switched Telephone Network (PSTN) or via communication satellites. The trend in today's technology is focusing on cellular networks since they have good land coverage and have become affordable to transmit large amounts of data. The advantage of cellular network over PSTN is the provided mobility and advantage over satellite communication is cost-efficiency.

As for the software side, operating systems are of help in M2M application development since they enable running the same software on devices that are different hardware-wise. The downside is that there are a lot of operating systems on the market which makes software development a demanding task. This is the problem we are addressing in this paper. Our contribution is a specific solution for sensor data collection that runs on the Android operating system. We are overcoming the problem of various existing mobile operating systems (e.g. Android, iOS, Symbian, Windows Phone 7, webOS, Bada) by using the OSGi framework. With solutions like the Prosyst mBS [6] it is possible to run the same software on operating systems like Android and Symbian, even on embedded low power hardware hosting Java Virtual Machine (JVM) [7]. By using Android operating system and ProSyst mBS OSGi solution we managed to overcome the hardware and software issues mentioned above.

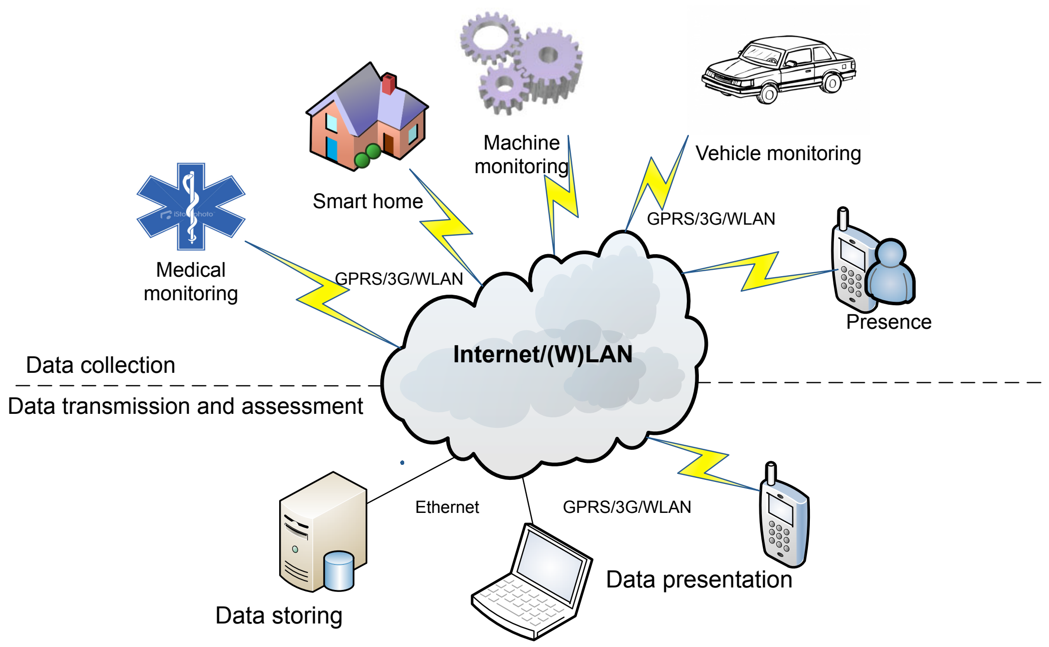


Figure 1. M2M architecture for different domains

B. Context-aware systems

An important part of ubiquitous or pervasive computing principle, meaning integration of devices and general technology into everyday activities of people, are context-aware systems. The term *context* refers to elements of user's environment that are relevant for the application. Such information can have its external dimension (e.g. location, most commonly used [8]) and internal or logical dimension specified or gathered from the user [9]. Therefore, context-aware systems react to the current context and make actions without the need of explicit user intervention. Implementation of such systems can have three basic approaches:

- direct sensor access without processing such information,
- middleware infrastructure that encapsulates sensor retrieval details, enabling reuse of the previous principle and
- context server, used as a distributed approach, where gathering and processing sensor data are moved to a remote unit.

The context server approach can have three models for managing multiple processes:

- process-centric view using widgets, that enables reusability, often controlled by a widget manager and are not robust to component failure,
- service-oriented model that has more complex network based components for discovering services, but provides robustness and
- data-centric view (i.e. blackboard model) where processes post messages to a shared media and process of adding new context sources and configuration are simplified.

Data structures used in representing and exchanging context information define several context models:

- key-value model where key-value pairs are used to describe the capabilities of a service,
- markup scheme models which typical example are *profiles* (e.g. Composite Capabilities/Preference Profile),
- graphical models like Unified Modelling Language,
- object-oriented models that use various objects to represent different context types such as temperature or location and
- logic and ontology based models.

Ontologies represent a description of the concepts and relationships, and ontology models are most expressive ones. System described in this work uses a variation of object-oriented model for exchanging physical sensor information – the context (external dimension) uses context-server approach and data centric view (a version of blackboard model).

III. Related Work

Our work is marked with several technologies such as OSGi, XMPP and context-awareness. To the best of our knowledge, these three technologies have not been previously combined within one M2M environment in order to provide portable and scalable (because of OSGi), secure and highly available (because of XMPP) system of greater quality (because of context-awareness).

OSGi is gaining momentum and is being implemented as a feature technology in many new systems and services. The usage of OSGi framework has advantages over standard operating-system-specific applications since it brings the benefits of high modularity, portability and resource-effectiveness [10]. There is a variety of papers where the OSGi framework is used to achieve the aforementioned software features in different areas of appliance.

One example of appliance is in telematics [11], convergence of telecommunications and information technology in automotive industry. Paper [12] introduces a service oriented infrastructure that aims to provide a flexible and extensible platform for provisioning, managing and developing telematics services. Moreover, this system, which consists of the OSGi backend system and the OSGi service gateway platform, provides architecture for development of context-aware automotive services.

In [13] authors present the use case of Android operated smartphones running on OSGi framework, used for monitoring road-side assistance and driver aids. They describe how to combine OSGi Vehicle Expert Group into an Android platform in order to enable third-party application manufacturing. This is of vital importance since the problem with OSGi in the telematics domain is that existing technologies are in most cases Original Equipment Manufacturer platforms and are only open for in-house development.

Ambient systems [14] make another area of appliance. The focus in [15] is to manage applications that are running on devices with scarce resources. This is achieved by defining a minimal bootstrap environment in order for the application to run. All other application resources are never stored on the device but downloaded directly into the JVM memory when needed. Constant connectivity is required in order for this to work, and thus the whole concept is similar to cloud computing [16].

Papers [12], [13] and [15] demonstrate the strength of OSGi as a technology that provides modularity and efficient software component management. However, OSGi framework does not provide sufficient security mechanisms [17] needed in M2M environments proposed in aforementioned papers. On the other hand, XMPP, used for communication in our work, provides additional security features such as secure communication and authentication of end users. Additionally, XMPP can be used for communication as an all-in-one solution – for messaging, presence information and file transferring. It also benefits from continuously open connection which is an advantage when M2M devices are behind a gateway and cannot be directly reached.

A slight disadvantage is relatively higher battery consumption and network traffic. Possible issues like information overhead of Extensible Markup Language (XML) on which the XMPP is based [18], and which could be of greater importance in low-bandwidth networks, can be solved using existing standards and techniques

[19]. Our work is designed as a generic framework for M2M environments, and thus we choose XMPP over some other alternatives (e.g. publish-subscribe systems) designed only for specific solutions. Moreover, XMPP provides better options for interoperability with other systems that will be used in future expansions of our work.

The paper [20] combines OSGi and XMPP designing system architecture for the detection function for digital refrigerators. OSGi benefits by providing a communication platform for different digital appliances, while XMPP is applied to build a mechanism of message notification. However, they do not use the process of data collection and assessment needed for context-awareness required for a generic M2M system such as ours.

In some other cases, server applications running on remote computers are used for calculations or storing collected data. Such M2M communication between a user's device and a remote computer often serves for publishing information or notifications that trigger certain actions. For instance, [21] describes a system that relates users, context and content providers creating a context-aware augmented reality system that provides real-time train station navigation and information processing. This context-aware system utilizes information that describes current location in order to provide location-dependent services to the user. It enables real-time interaction by requiring certain sensors or specific hardware that can locate users [22]. Although, this study showed that today's smartphones (e.g. Android) are already capable of performing highly complex tasks and combining multiple technologies simultaneously, without using OSGi as a front end framework, it is hard to achieve portability (i.e. usage on different pieces of hardware).

IV. Background

The following section provides information about used technologies and describes their key advantages which made them suitable for the purposes of this work.

A. Open Service Gateway initiative (OSGi)

OSGi [10] is a framework that enables creation of modular Java applications. Classes of object-oriented software are divided into bundles – dynamic components that can be installed, started or updated without having to restart the application or service. It also enables remote management of bundles. All mentioned features facilitate work on web servers where OSGi is largely applied. It provides higher modularity than standard Java programming since applications can be combined from various existing bundles, which can be found in web repositories. On smartphones, OSGi provides better resource management since it allows multiple Java applications to run using a single JVM. Moreover, OSGi can be used in different areas of appliance such as industrial automation, automobiles [13] and for grid computing [23].

B. Android

Android is an operating system and Application Programming Interface (API) for smartphones released by Google in 2007 [24]. Essentially, it represents a Linux distribution that includes a version of JVM (i.e. Dalvik VM), and thus applications development is mainly done using Java programming language with the Android Software Development Kit in combination with development environments such as Eclipse [25]. The main advantages of this platform is its open-source type, simple access to core hardware functions of the smartphone (e.g. cameras and accelerometers) and accessible use of built-in services and applications. In our system we use Android devices since they have these aforementioned advantages and support is available for needed OSGi framework (e.g. Prosyst mBS [6]) and XMPP protocol API (e.g. Smack API for Android [26]).

C. eXtensible Messaging and Presence Protocol (XMPP)

XMPP [27] was developed by the Jabber open-source community and later standardized by Internet Engineering Task Force. This protocol uses an XML format permitting two entities to exchange XML elements over a network. XMPP is basically a client-server model, where clients communicate to each other using public servers or servers for local communication. Communication is realized using three core XML stanzas:

- *presence* stanza used for information about an entity's network availability,
- *message* stanza for sending asynchronous information from one entity to another and
- *iq* stanza for general request-response mechanism.

These stanzas enabled a number of applications, from classical instant messaging [27] and presence services [28], to network monitoring [29] and remote controlling [11]. Monitoring and controlling through sensors [30], especially in the M2M environment, tend to show perspective [31] while using XMPP, and therefore this type of communication is built in the designed architecture.

V. System Architecture

Architecture designed for our system consists of three basic components – smartphones, XMPP server and Context Data Processor (CDP). Components (i.e. nodes) communicate using XMPP protocol, and therefore a dedicated XMPP server is required to enable such communication. The central part of the system is the CDP, that has several roles as a main logic unit (e.g. storing and filtering context data), but essentially it serves as a data collection and processing server. Smartphones represent the client part of the system gathering required data in terms of current sensor readings and forwarding it to the CDP. Both client devices (i.e. smartphones) and the CDP are implemented using OSGi framework.

A detailed view on the communication between network components is presented in Figure 2. Classical client-server model is used here, in such a manner where smartphones represent the client, and CDT the server part of the system. Since XMPP protocol is used for communication, this model additionally needs to have a XMPP server which represents central connection point between the XMPP elements (i.e. smartphones and CDP).

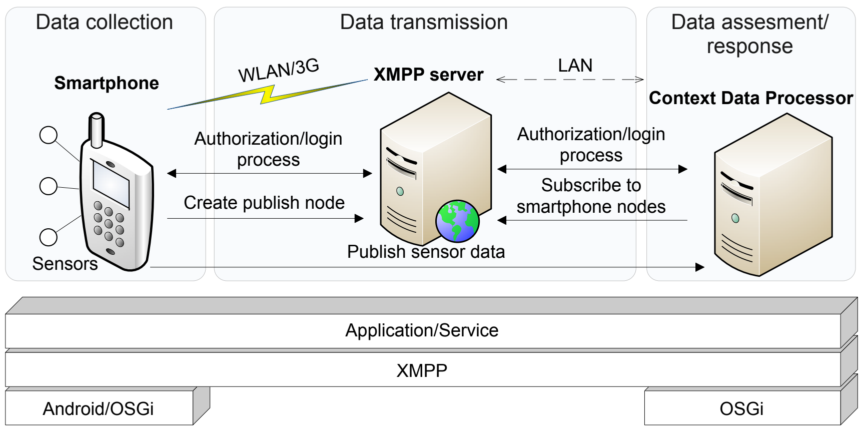


Figure 2. Communication principle between the components

Communication between smartphones and XMPP server is realized either through a 3G network or Wireless Local Area Network for local usage, while XMPP server and CDT communicate on a Local Area Network based network for a more reliable communication. XMPP protocol is in the core of the system, and therefore all of the standardized XMPP messages (i.e. *presence*, *iq* and *message* XML stanzas) are used here, with addition of explicit use of publish-subscribe mechanism [32]. Using these mechanisms, CDT tracks the status of every available smartphone and receives required context data in terms of group sensor readings. Moreover, all of these endpoints connected through XMPP server are built on appropriate versions of OSGi framework (i.e. mBS Mobile for Android application and Equinox OSGi framework for the CDP).

VI. System Design and Implementation

This section contains description and specification in which manner the implementation was done on both client and server side of the system. Here, logic part of the system is described as well as implementation in OSGi framework.

A. Client implementation

Components, which provide context information through sensor data, are implemented on Android based smartphones (e.g. HTC Hero and HTC Legend). An application based on OSGi framework is run on the device (see Figure 2), primarily as a background service with minimal graphical interfaces. The idea is to limit user interaction with the application and enable configuration of basic settings only – types of sensors to track and frequency of sending those. Application then, using XMPP functionalities, authenticates and logs as a XMPP user on the remote (or local) server. A publish-subscribe node is then created on the server and sensor data is deployed to the server over a wireless interface in formatted XMPP messages using a publish-subscribe mechanism. Sensor readings, representing current context, are published depending on the required configuration which can contain frequency and types of sensors. Except configuring initial settings through graphical interface, all other described actions (e.g. registration with XMPP server) are done automatically.

Our application is meant to run on the device as long as it is needed for context data retrieval. Disabling the running service could be done programmatically or manually by the device user. Although this application could easily be run as a classic Android application, realization is done using OSGi framework – dividing application into several bundles, each for a specific purpose (e.g. XMPP libraries, sensor data class packages, services and graphical interface) [33].

B. Server and CDP implementation

XMPP server is run on a dedicated computer, connected to the CDP in a local network. No special requirements are needed here, only standard XMPP support, but possibility to upgrade these functions (e.g. using Ad-Hoc commands [34]) should be taken in concern.

CDP is implemented as an OSGi application divided into several bundles. These bundles are divided in the same manner as the mentioned Android application – each bundle for specific groups of packages divided by functionality (e.g. XMPP libraries, application logic, graphical interface). For better control of these packages in the OSGi command line, graphical interface with minimal required functionalities is implemented (see Figure 5).

When CDT gets started, it goes through the authentication and login process with the XMPP server and then subscribes to all publish-subscribe nodes that devices have created. The benefit of using XMPP is clear in this process; using presence mechanisms, it is relatively simple for CDP to be aware of all devices currently logged into the system, and whose sensor data are being monitored. While being subscribed to these nodes, CDP receives any published sensor data and handles them for the specific purposes; storing them as formatted data and examining current context based on previous data and data received by other nearby devices. Such information in terms of context data can be used for a wide spectrum of objectives such as gathering statistical research, tracking and alarming in specific situations and other.

VII. Case Study

In order to provide general perspective on applying our system for specific purposes, use case is described in this section. The case study gives an industrial-like application of the system in terms of monitoring and alarming functions. It includes smartphones being used by employees in a kind of industrial compound with potentially dangerous environment. The devices are used for general communication purposes and monitoring environment parameters. System includes standard components (i.e. smartphones, XMPP server and CDP) and additionally some sort of control center used in the company. Messages and actions taken in the scenario are shown in the sequence diagram (see Figure 3). CDP examines context data of the devices, and if a potentially dangerous set of readings is detected (e.g. increased temperature in a period of time), CDP alarms the control center, and then actions of prevention or control can be taken regarding security policy of the company.

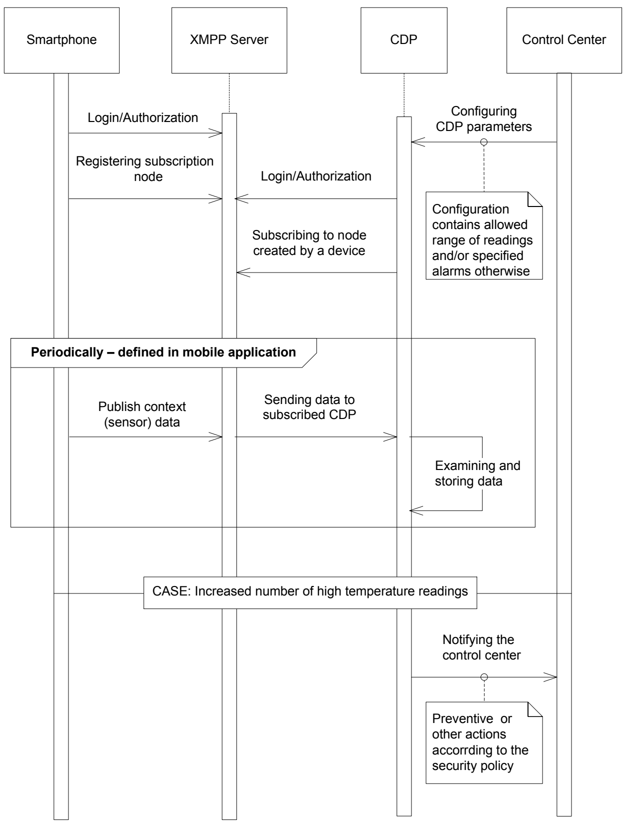
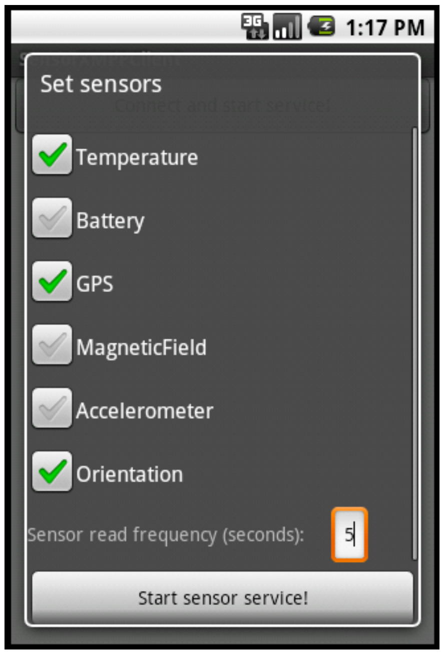

Smartphone
XMPP Server
CDP
Control Center
Login/Authorization
Configuring CDP parameters
Registering subscription node
Login/Authorization
Configuration contains allowed range of readings and/or specified alarms otherwise
Subscribing to node created by a device
Periodically – defined in mobile application
Publish context (sensor) data
Sending data to subscribed CDP
Examining and storing data
CASE: Increased number of high temperature readings
Notifying the control center
Preventive or other actions accrording to the security policy

Figure 3. Sequence diagram of described case study scenario

The process of establishing our system includes configuring and starting the service on a smartphone, running the XMPP server and CDP with a interface needed to provide context data handling configuration. Through the mobile application, users set the sensor listening service choosing sensors and frequency of sending the data. Figure 4 shows an example of graphical interface for configuring such Android application.

The started service then logs in the XMPP server and registers its publish-subscribe node for sensor data. After configuring and starting, CDP goes through the same authorization process with an XMPP server and subscribes to nodes created by devices (Figure 5). Afterwards, for the defined period, sensor data are being published on the registered node, so the data are instantly redirected to the CDP. Scenario shown in Figure 3 includes a case where a number of readings include increased values of temperature. Supposing that these kinds of readings are defined in the CDP configuration as potentially dangerous, the CDP then alarms the control center.

Although the primary concept of implementation is designed for user devices meant for tracking, additionally some other features, such as chart displaying history of sensor values varying, are implemented (see Figure 6). These kinds of features can be of use for quick filed-type analysis of the sensor changing history for the people in charge of such actions, since the primary focus is on tracking context data and reacting to it.

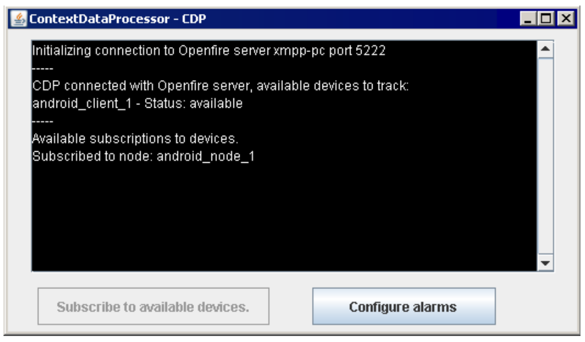


Set sensors

- ✓ Temperature
- Battery
- ✓ GPS
- MagneticField
- Accelerometer
- ✓ Orientation

Sensor read frequency (seconds): 5

Figure 4. Screenshot of the Android application - sensor listening service settings



```
Initializing connection to Openfire server xmpp-pc port 5222
-----
CDP connected with Openfire server, available devices to track:
android_client_1 - Status: available
-----
Available subscriptions to devices.
Subscribed to node: android_node_1
```

Figure 5. Screenshot of CDP - simple example of the implementation

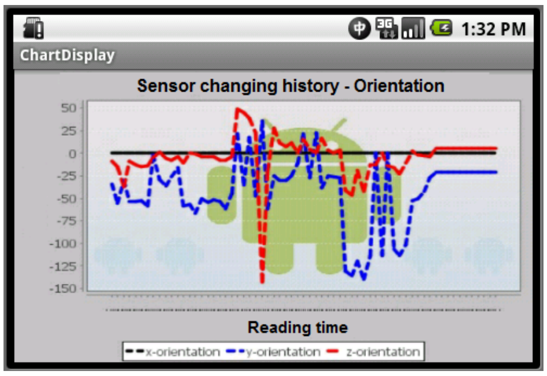


Figure 6. Screenshot of the Android application - simplified representation of sensor variation history

VIII. Evaluation

Technologies used in realization of the described system represent key points in recognition of the added value that the system makes. First of all, it is important to emphasize features of XMPP such as extendibility for supporting additional capabilities and existing support for security and reliability as described in [35]. Even though this work does not go into details about security and reliability issues, these requests could be of great importance in some cases (e.g. life-saving information). Knowing these facts, more demanding implementations of this system should also be considered.

Optimizing system resources like memory usage, application size and energy consumption should be taken in consideration while deploying solutions for smartphones such as Android/OSGi bundles. Regarding this matter, evaluation of these parameters was done, comparing Android/OSGi and native Android realization of the mobile application. Evaluation was done using command-line tools for inspecting process memory usage, the OSGi framework application for Android (its memory information utilities) [6] and Android applications for system monitoring [36]. Generally, an assumption is made that the OSGi framework is started on the device and is running for the needs of different application bundles, so the considered application parameters (e.g. starting time, memory usage and application size) are observed without the influence of the OSGi framework itself.

Test results for our developed mobile application, except energy consumption which is discussed separately, are shown in Table I. Tests were performed on three actual Android devices: HTC Hero (processor: 528 MHz, system memory: 288 MB, Android version 2.1), HTC Legend (processor: 600 MHz, system memory: 384 MB, Android version 2.2) and HTC Desire HD (processor: 1 GHz, system memory: 768 MB, Android version 2.2). Average values of considered parameters were taken for each device together with the OSGi framework parameters. Results show that the start-up time of the bundle is several times faster (9 ms for the slowest device) compared to the native Android application (15 ms for the fastest device). Also, memory consumption is much less (depends on the device, but always less than 1 MB) than for the native Android application, which takes up to 3.2 MB. Despite of OSGi framework as an additional element considering application size and memory consumption, the Android/OSGi bundle shows improved results when compared to the classic Android application. In [13] testing and evaluation of these parameters in the Android/OSGi environment were done, and also showed similar results regarding memory and application start-up time of Android/OSGi based applications, not including the OSGi framework itself.

Regarding battery usage, using the OSGi framework does not show to be energy-consuming through a shorter period of active usage when publishing sensor data every couple of seconds. Although this kind of parameter cannot be measured as precise as memory or application size, testing on mentioned Android devices showed that OSGI framework itself uses roughly around 0.5% of battery power on Android system (after starting and installing bundles) with addition of running bundles that spend around 0.05%, depending on intensity of communication. Compared to native Android application, which uses also around 0.05% of battery power through a similar short period of active use, using Android/OSGi bundles does not affect the devices energy autonomy significantly. Therefore, together with portability and ease of use, using Android/OSGi version is definitely an advantage even from the optimization point of view and points out a contribution of OSGi platform in terms of scalability and more effective use of system resources.

TABLE I
Evaluation results

	HTC Hero (Android/OSGi)	HTC Legend (Android/OSGi)	HTC Desire HD (Android/OSGi)	Native Android
Application size	1.61 MB (1.52 MB external libraries bundle and 8.86 MB OSGi framework)			2.03 MB (1.52 MB external libraries)
Application start-up time	9 ms (12000 ms OSGi framework)	6 ms (8035 ms OSGi framework)	4.5 ms (3600 ms OSGi framework)	38, 30 and 15 ms respectively
Memory usage	600KB (9 MB OSGi framework)	800 KB (9.5 MB OSGi framework)	700 KB (10 MB OSGi framework)	3.2 MB (all devices)

IX. Conclusion

Smartphones can already be used for different purposes, and with the breakthrough of technologies like OSGi combined with M2M communication, they are breaking existing barriers for providing PC-like applications and functionalities on such handheld devices. However, most of the existing systems focus on narrow usage scenarios that are of help only in highly specific domains. We consider that with help of the right technologies and system design it is better to develop multi-purpose generic systems which can be adapted for different appliances with minimal implementation changes.

Therefore, in this paper we presented the concept of such a system that can be used for monitoring purposes or on-event reactions. By keeping the system design simple and by using technologies like OSGi and XMPP we implemented a system that is easy to upgrade and combines modularity, scalability, portability and resource-effectiveness. The evaluation pointed out concrete advantages of OSGi applications in managing resources – 80% less memory usage and shorter start-up time (average values besides OSGi framework itself and given values for native Android, taken from Table I) and negligible energy consumption.

Future work on this subject will be focused on implementing XMPP commands for communication between entities, improving scalability options in terms of better user management in the CDP and porting the system on other operating systems like Symbian and Windows Phone 7. Also, additional functions of OSGi framework regarding remote controlling and software updates [37] will be considered for implementation in our system.

Acknowledgment

The authors acknowledge the support of research project "Content Delivery and Mobility of Users and Services in New Generation Networks" (036-0362027-1639), funded by the Ministry of Science, Education and Sports of the Republic of Croatia. Additionally, the authors would like to acknowledge the assistance of Jurica Kupresak and Davor Jurisic who participated in developing the initial version of the described system.

References

[1] Machine-to-Machine web site, *http://m2m.com/index.jspa*.

[2] R. Want, "When Cell Phones Become Computers," *Pervasive Computing*, vol. 8, pp. 2–5, 2009.

[3] V. Podobnik, K. Trzec, and G. Jezic, "Context-Aware Service Provisioning in Next-Generation Networks: An Agent Approach," *International Journal of Information Technology and Web Engineering*, vol. 2, no. 4, pp. 41–62, 2007.

[4] V. Galetic, I. Bojic, M. Kusek, G. Jezic, S. Desic, and D. Huljenic, "Basic principles of Machine-to-Machine communication and its impact on telecommunications industry," in press.

[5] M2M Communications website, *http://www.m2mcomm.com/about/what-is-m2m/index.html*.

[6] ProSyst, *http://dz.prosyst.com/devzone/Mobile*.

[7] ProSyst, "The worlds smallest OSGi solution," *Technical White Paper*, p. 3, 2010.

[8] B. Schilit, N. Adams, and R. Want, "Context-Aware Computing Applications," *First Workshop on Mobile Computing Systems and Applications*, pp. 85–90, 1994.

[9] M. Baldauf, S. Dustdar, and F. Rosenberg, "A survey on context-aware systems," *International Journal of Ad Hoc and Ubiquitous Computing*, vol. 2, no. 4, pp. 263–277, 2007.

[10] OSGi Alliance, *http://www.osgi.org/Main/HomePage*.

[11] T. Kailey, "In-vehilce telematics - past, present future," in *Developing Commercial In-Vehicle Information Services and Systems*, 2003, pp. 5–26.

[12] D. Zhang, X. H. Wang, and K. Hackbarth, "Osgi based service infrastructure for context aware automotive telematics," in *Vehicular Technology Conference, 2004. VTC 2004-Spring. 2004 IEEE 59th*, vol. 5, 2004, pp. 2957–2961.

[13] M.-C. Chen, J.-L. Chen, and T.-W. Chang, "Android/OSGi-based vehicular network management system," *Computer Communications*, vol. 34, pp. 169–183, 2010.

[14] Z. Pousman and J. Stasko, "A taxonomy of ambient information systems: four patterns of design," in *Proceedings of the working conference on advanced visual interfaces*, 2006, pp. 67–74.

[15] S. Frenot, N. Ibrahim, F. Le Mouel, A. Ben Hamida, J. Ponge, M. Chantrel, and D. Beras, "ROCS: a Remotely Provisioned OSGi Framework for Ambient Systems," *Network Operations and Management Symposium*, pp. 503–510, 2010.

[16] M. Armbrust, A. Fox, R. Griffith, A. D. Joseph, R. Katz, A. Konwinski, G. Lee, D. Patterson, A. Rabkin, I. Stoica, and M. Zaharia, "A view of cloud computing," *Communications of the ACM*, vol. 53, pp. 50–58, 2010.

[17] The OSGi Alliance, *http://www.osgi.org/download/r4v42/*.

[18] R. Lawrence, "The space efficiency of XML," *Information and Software Technology*, vol. 46, pp. 753–759, 2004.

[19] XML Binary Characterization, *http://www.w3.org/TR/xbc-characterization*.

[20] M.-F. Horng, M.-H. Hung, Y.-T. Chen, J.-S. Pan, and W. Huang, "A new approach based on xmpp and osgi technology to home automation on web," in *International Conference on Computer Information Systems and Industrial Management Applications*, 2010, pp. 487–490.

[21] F. Uijtdewilligen, "A framework for context-aware applications using augmented reality: A train station navigation proof-of-concept on Google Android," *Master's Thesis*, p. 102, 2010.

[22] Building a context-aware service architecture, *http://www.ibm.com/developerworks/architecture/library/ar-conawserv/index.html*.

[23] J. Zhao, J. Xu, X. Dong, Z. Zhu, and Z. Wang, "A Scalable and Low-Cost Grid Portal," *Seventh International Conference on Grid and Cooperative Computing*, pp. 570–576, 2008.

[24] Google Inc., *http://code.google.com/android/what-is-android.html*.

[25] Eclipse web site, *http://xmpp.org/extensions/xep-0024.html*.

[26] asmack, *http://code.google.com/p/asmack/*.

[27] R. Jennings, E. Nahum, D. Olshefski, D. Saha, Z.-Y. Shae, and C. Waters, "A study of Internet instant messaging and chat protocols," *IEEE Network*, vol. 20, pp. 16–21, 2006.

[28] XEP-0163: Personal Eventing Protocol, *http://xmpp.org/extensions/xep-0163.html*.

[29] Maa, Antonio and Rudolph, Carsten and Labidi, Wael and Susini, Jean-Ferdy and Paradinas, Pierre and Setton, Michael, "XMPP based Health Care Integrated Ambient Systems Middleware," in *Developing Ambient Intelligence*, 2008, pp. 92–102.

[30] A. Hornsby, P. Belimpasakis, and I. Defee, "XMPP-based wireless sensor network and its integration into the extended home environment," in *13th IEEE International Symposium on Consumer Electronics*, 2009, pp. 794–797.

[31] J. Wagener, O. Spjuth, E. L. Willighagen, and J. E. Wikberg, "XMPP for cloud computing in bioinformatics supporting discovery and invocation of asynchronous web services," *BMC Bioinformatics*, vol. 10, no. 1, p. 279, 2009.

[32] XMPP publish-subscribe mechanism, *http://xmpp.org/extensions/xep-0024.html*.

[33] K. N. Khan, "State-of-the-art Study and Design of a Small Footprint Version of the COOS Plugin Framework," *Master's Thesis*, p. 89, 2010.

[34] XMPP Standards Foundation, *http://www.eclipse.org*.

[35] M. Almeida and A. Matos, "Bridging the Devices with the Web Cloud: A Restful Management Architecture over XMPP," in *6th International Mobile Multimedia Communications Conference*, vol. 10, 2010.

[36] Android Assistant, *http://www.appbrain.com/app/android-assistant2812-features29/com.advancedprocessmanager*.

[37] OSGi Remote Management Tool, *http://wiki.eclipse.org/OSGi-Remote-Management-Tool*.