Enhancing M2M communication with cloud-based context management

Bachir Chihani, Emmanuel Bertin, Noel Crespi

Enhancing M2M communication with cloud-based context management

Bachir Chihani[#*], Emmanuel Bertin[#*], Noël Crespi[*]

[#]*Orange Labs*
42 rue des Coutures, 14066 Caen, FRANCE
{bachir.chihani, emmanuel.bertin}@orange.com

[*]*Institut Telecom, Telecom SudParis, CNRS 5157*
9, rue Charles Fourier, 91011 Evry, FRANCE
noel.crespi@it-sudparis.eu

***Abstract*— Contextual information are used to describe situations of entities (users and devices) and their interactions. Applications need such information in order to adapt their behaviour in response to changes in the entities context. In certain scenarios and in particular machine to machine, the integration of a large number of components makes the management of the generated contextual information a very complex task. There absence of scalable and easily programmable solution for an efficient context management in complex and distributed environment is a show stopper. In this article, we propose a scalable cloud-based framework for context management able to handle contextual information in large distributed environments. This framework has been implemented and evaluated; it provides applications with a XML-based programming language to allow them customizing subscriptions to context changes and defining how context data.**

***Keywords*— Context, Cloud Computing, XML, M2M.**

I. INTRODUCTION

Context-aware applications share common functionalities (context acquisition, storage, processing, distribution and usage) that can be implemented in different components. In the literature, these functionalities are usually distributed into three types of entities. First, the sensors, also called context providers, are in charge of contextual information acquisition (e.g. location, activity or status of the monitored entities) [1] [2]. Second, context consumers are applications responsible of using context data to adapt their behaviour in response to specific changes in their environment [3]. Third, context management functionalities (storage, processing and distribution) are usually performed by a central context broker that is also in charge of connecting providers to consumers [4].

However, in certain scenarios, like machine to machine use cases, the context broker becomes quickly a bottleneck and the delay time, as well as memory consumption, generated from managing contextual information grows exponentially. Context processing and reasoning consume most of the resources as detailed in [5]. Thus the context broker needs to scale to be able to perform its management functionalities. A cloud implementation of the context broker is a promising solution to guarantee scalability, automatic and on demand resource provisioning. Such a cloud platform will provide Reasoning as a Service for pre-processing published contextual information. It will also provide an efficient subscription/notification paradigm to context exchanges that will limit the need for applications to continuously request context and reason on it locally. Such approach will enable low capacity devices (such as in Internet of Things [6]) to adapt their behaviour responding to changes in situations, and also to be able to modify how the adaptation is performed.

A flexible language is needed for subscribing to specific context changes and for describing application specific context processing operations that need to be implemented by the Context Broker. This language should provide custom filtering definition capabilities so that context consumers will not be overwhelmed with so many context events each time context data are published by sensors.

In this work, we propose a cloud-based context management platform providing Reasoning as a Service (RaaS) for processing context. An XML-based language is used by applications to define how contextual information have to be processed at the cloud platform and to define their callback addresses on which to receive notification resulting the context processing. As a case study, we consider a M2M (machine to machine) scenario from logistic domain to derive real requirements for evaluating our cloud-based context management platform.

The rest of the paper is organized as follows. Section II presents our context management framework: its design considerations, architecture and context processing. Section III presents benefits of the framework. Section IV presents a M2M case study implementation where our framework is used. Section V presents background works. Section VI concludes the paper.

II. CLOUD-BASED CONTEXT MANAGEMENT FRAMEWORK

Cloud computing [7] has emerged as a new computing paradigm allowing the delivery of resources (platforms, infrastructures and softwares) as a service instead of products as in traditional paradigms. It performs an efficient sharing of centralized resources and computation power among users connected through any kind of devices (e.g. computers, smartphones). With cloud computing, web applications can

easily scale gradually and provision needed resources on demand (e.g. after temporary traffic spikes). In this section we present a context management framework for cloud environment.

A. *Design considerations*

We propose a generic framework that relies on Cloud and XML technologies to provide a flexible solution to context management complexity and automated context-based processing and reasoning. This cloud platform hosts context reasoning and enables applications to customize, through a flexible XML-based language, their context-based adaptation rules at design and runtime.

Contextual information are heterogeneous by nature and depending on the application they may not be processed in the same way. Some contextual information like localization of transportation truck or temperatures of transported products need to be monitored in real time to facilitate intervention in case of emergency (e.g. temperature exceeded a threshold). Other contextual information are used as timers that should trigger some actions when expired, for instance when usage time of an engine reach its limit then the corresponding maintenance process should be triggered. Some other contextual information are stored to be checked when needed, for instance they might be used as conditions in a decision tree for executing appropriate actions in a given situation. The language defining context processing should be able to express the different ways how context can be processed.

The component responsible for processing context is the bottleneck of the whole framework as it will receive huge amount of information from different sources, process them and send out notifications. Thus it should be designed in an efficient way to face peaks in information amount.

B. *Framework architecture*

Figure 1 presents our cloud-based framework for managing contextual information at a large scale. The architecture expands the main components of an earlier context management framework, presented in [4], into smaller components implementing basic functionalities (e.g. sending notifications) and elastic enough to scale on demand. Providers (CxP) are the components responsible of wrapping source of context and publishing it to the ecosystem. Consumers (CxC) are implemented by context-aware applications that request context to adapt its behavior to changes in the user situation (set of contextual information). The proxy server (PS) is the cloud gateway, it receives context updates and publications from CxP and route them to reasoning engine. Also, it receives from context-aware applications a description file of their adaptation rule that will be hosted by the platform. Reasoning Engines (RE) are responsible of instantiating these description files in order to implement corresponding context-reasoning process. This reasoning results in a set of events to be sent to the corresponding application in order to adapt its behavior in response. Notification Servers (NS) support asynchronous protocols (e.g. Comet) to callback and notify CxC about new published context or with events generated by a reasoning engine after processing context. The callback happens on the reference defined by the application in its description file.

All communication messages are handled with standardized protocols: HTTP for synchronous communication; Comet for asynchronous communication and event notifications. The cloud components (PS, RE and NS) are implemented as separate RESTful web services that may scale efficiently.

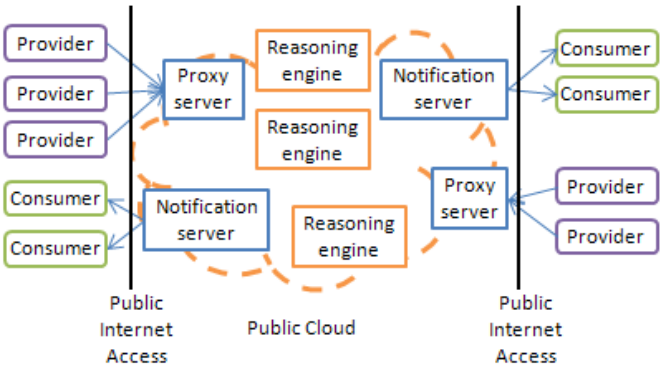

Fig. 1: Architecture of the cloud-based context management platform

Following figure depicts the sequence diagram of the exchanged information between the different components. Reasoning engines have to register with a proxy server in order to be solicited later for implementing reasoning processes. Application developers upload to the proxy the description file of their application context reasoning process. The proxy server then chooses a reasoning engine to host and instantiate the reasoning process. A simple Load balancing mechanism based on Round-Robin is used by gateways to choose a registered reasoning engine that will process the new published context. Context-aware applications subscribe with the notification server. When a context is published by the context provider to the proxy server, the later forward it to the corresponding reasoning engine to process it and may trigger an event to consumers through the notification server. At any time, developers can upload a new version of the description file of their application reasoning process and thus adapting its context-awareness to meet new requirements.

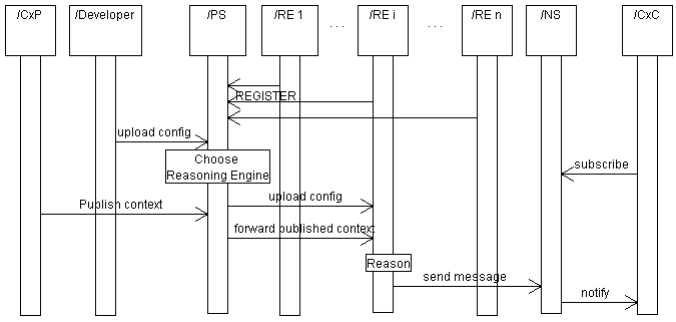

Fig. 2: Sequence diagram of exchanged information between different components of the platform

C. Context processing

XML-based programming languages [8] leverage markup language representations and pipeline transformations for supporting flexible customization, separation of concerns, and process automation. In our framework, context processing is performed at the cloud side and described in a XML language called CPDL (Context Processing Definition Language). Context processing consists of an abstraction step that aims to abstract raw contextual information into higher valued knowledge and an aggregation step that defines a situation based on a collection of values of context data. The abstraction is implemented as a finite state machine where each state is high level information derived from a raw context data. Transitions between states are performed after receiving new published context data. Conditions can be added to transition in order to limit the cases where the finite state machine can transit from a state to another. The aggregation step consists of an IF-THEN rule where the conditions can be on a finite state machine or on raw context data. The corresponding action can be for instance sending a notification back to the application on a given call-back address or triggering another Abstract-Aggregate cycle by sending an internal event to one of corresponding finite state machines.

```
<!DOCTYPE Definition [
<!ELEMENT Definition (FiniteStateMachine | Rule)*>
<!ELEMENT FiniteStateMachine (State*, Start-State, End-State)>
<!ATTLIST FiniteStateMachine Id CDATA #REQUIRED Name CDATA #REQUIRED>
<!ELEMENT State (Transition)*>
<!ATTLIST State Name CDATA #REQUIRED >
<!ELEMENT Transition (Condition | Event)*>
<!ATTLIST Transition Dest CDATA #REQUIRED >
<!ELEMENT Rule (Condition | Event)*>
<!ATTLIST Rule Id CDATA #REQUIRED Name CDATA #REQUIRED >
<!ELEMENT Condition EMPTY >
<!ATTLIST Condition Type CDATA #REQUIRED Key CDATA #REQUIRED Operator CDATA #REQUIRED Value CDATA #REQUIRED >
<!ELEMENT Event EMPTY >
<!ATTLIST Event Type CDATA #REQUIRED To CDATA #REQUIRED Message CDATA #IMPLIED >
]>
```

Fig. 3: Context Processing Definition Language (CPDL)

Figure 3 depicts the DTD [9] (Document Type Definition) file that describes the context composition language. XML description files of a reasoning process have to complain with this DTD in order to be implemented by the reasoning engine.

III. Benefits to User, Developer and Platform Side

From user's perspective, the platform enables smart and lightweight applications to be able to adapt their behaviour to user context changes. Such applications reduce the effort needed by user in the interaction with him as they continuously monitor its situation and are aware of his need at a given moment. An example of such applications is a context-aware call management application that monitors user situation (e.g. his availability on a given communication channel) to adapt the handling of his communications (e.g. change the communication channel from SMS text message to voice).

On the other side, application developers will be able to concentrate their effort on building their application business logic while ignoring the complexity resulting from the management of contextual information as this will be hosted at the platform side. Also, as context management can be specified with XML files, then it becomes possible for developers to modify the initial reasoning approach and extend it to new contextual information at run time and in a transparent way while the user is using the application without having to modify its code.

The platform provides several enhancement regarding existing approaches. First, it enables an easy deployment of innovative context-aware applications and a powerful support to their upgrading. Second, the reasoning engine can be multiplied by deploying the platform in a cloud environment to support scalability of context-aware applications. Also, notifications due to context changes are consumed and filtered locally before being propagated to end applications which reduce efficiently the number of messages exchanged between the platform and context consumers.

The solution reduces but do not eliminate all complexities related to building context-aware applications. As application logic is explicitly separated from context management. Developers still has the task of describing how context should processed by the platform.

IV. Case Study: M2M Communication

M2M [10], also called embedded mobile, communications enable collecting crucial information from many connected objects via wireless network (e.g. short-range radio, 3G/4G network) to a backend server for aggregation and processing. It is widely used in many domains [11], like energy, health, security, transportations, remote maintenance, or HCI [12] (human-computer interaction).

In logistics, exchanging contextual information (e.g. objects geospatial information, agents' health condition) between operational agents, in field equipments with company back office is very crucial. The monitoring of context changes enables real time process execution supervision and exceptions handling. As process execution may not comply with predefined plans, for instance traffic jams may delay a delivery plan. Back office agents have to be alerted in case of emergency in order to be able to respond immediately (e.g. change plan or abort delivery).

Next, we present our implementation for enabling efficient M2M communication in a transportation scenario.

A. *Case study implementation*

As a case study, we implemented a solution based on the previously presented framework for a M2M scenario where a shipment truck has to deliver a product from a location (e.g. Paris) to another (e.g. Caen). During this travel, contextual information of the truck are sent to the cloud platform (servers responsible for managing context in the cloud domain). Also, presence information of the back office agent are sent from the internal presence server (here we used Microsoft Lync Server) to the cloud domain hosting the context management. Context-awareness is implemented in the public domain thanks to the context composition language. It aims to send a voice message to the back office agent on its smartphone (where an Android supervision application is installed) if the temperature of the fridge exceeded a certain threshold and the agent is not available in front of the supervision interface (his presence status is offline).

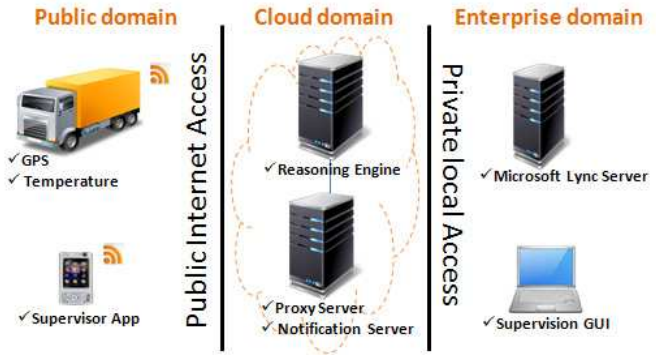

Fig. 4: Test environment configuration

The testbed environment is illustrated in figure 4. The GPS coordinates of the route between Paris and Caen was generated with CloudMade[1] Routing API, and played by a location provider java program. Temperatures of the truck fridge are simulated and published to the platform.

Figure 5 depicts the supervision interface that is Swing-based GUI (Graphical User Interface). It is a context consumer application that listen to changes in location of the supervised truck and temperature of its fridge, then display them in a suitable interface to help back office agent to track in real time delivery situation.

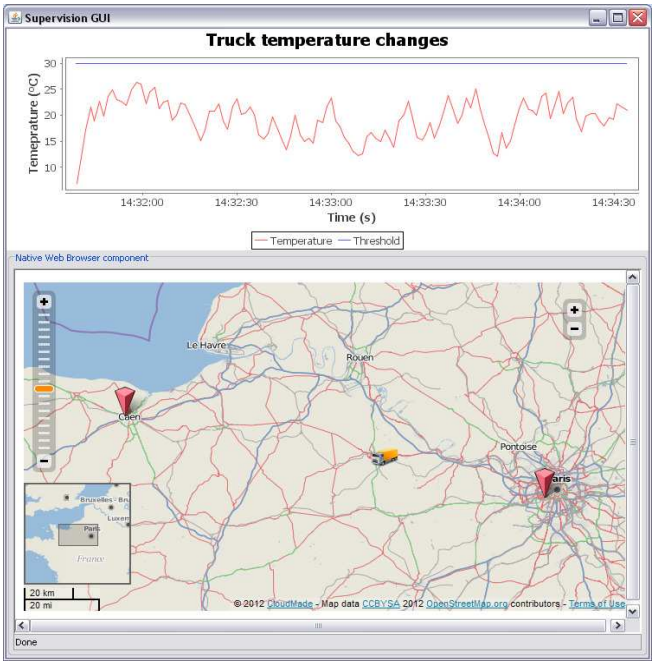

Fig. 5: Supervision GUI

B. *Performance evaluation*

For performance evaluation of the cloud-based context management we used the following environment: a server running a Microsoft Windows Server 2003 SP2 with following hardware characteristics (Vender: Intel, Model: Xeon, CPU: 2.8GHZ, memory: 2 GB). On this server is installed a Jetty web server to run proxy server and reasoning engine as web applications on top of the RESTlet[2] framework. Also, on this server is running CometD[3] server to play the role of the notification server, to send context updates to subscribed clients. For persistence, contextual information is stored on an object database DB4O[4].

A context provider and consumer were developed as simple java program to publish context and receive updates. This program sends context publication messages to the context management platform. It runs on Microsoft Windows XP SP3 installed on Dell D630 laptop with the following characteristics (Vender: Intel, Model: Core 2 Duo, CPU: 2.2GHZ, memory: 2 GB).

Figure 6 depicts the response time of the context management platform correspondingly to the number of the received context updates. The aim is to model response time as a function that takes number of parallel requests as parameter. This model will help measuring the reactivity of our framework and estimating future response time.

[1] http://www.cloudmade.com/

[2] http://www.restlet.org/
[3] http://www.cometd.org/
[4] http://www.db4o.com/

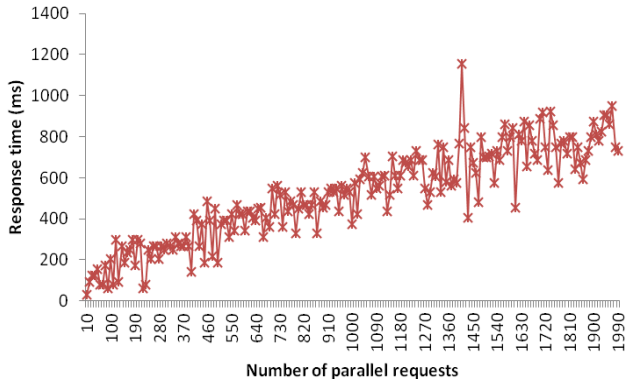

Figure 6: load performance

The variation of response time can be approximate with the following linear:

$ResponseTime(RequestNumber) = \alpha + \beta * RequestNumber$

Where the function parameters α and β can be approximately computed as follow:

β = average ($\Delta ResponseTime_{i+1,i} / \Delta RequestNumber_{i+1,i}$)
α = average ($ResponseTime_i - \beta * RequestNumber_i$)

After calculation we found α= 162.097766 and β= 0.35454545. Response time increases slowly with the number of parallel request as the β ratio is less than 1.

Performance test results show the platform ability to notify context information with a good response time (only 1 second for approximately 2000 parallel context updates).

V. Background and Discussion

A. Related work

In [13], the authors proposed a new approach for M2M communication that extends business processes execution by integrating mobile devices. Through a cloud-based solution, this integration enable in field handling of operational exceptions in industries like transport and logistics. As M2M telecommunication world do not use same technology and protocols as enterprise domain, a proprietary component was used as gateway to provide a flexible end to end communication and for translating network protocols (e.g. SIP to SOAP and vice versa) between these two domains. A back end system was used to implement business logic and exceptions handling. However, the proposed solution is not cloud-based as it is hosted by the client. Also, there were no explicit separation of concerns between handling contextual information and application logic.

In [14], the authors described the redesign of Java EE technologies commonly used in enterprise domain and their implementations to support telecommunication domain requirements like scalability for facing heavy loads and asynchronous nature of telecommunication systems. The result is a scalable, asynchronous, event-driven non-blocking platform for service composition. In this platform, no worker thread is dedicated to a composite application. Instead, the execution of composite applications is divided into small tasks processed by a pool of different workers. Tasks requesting long operations like I/O are put into a waiting list before being processed again, this waiting time is used by the corresponding thread worker to handle another task. The solution is not limited to context management and can be used in many other situations. However, specification of a composite application cannot be modified easily at runtime as this involves the modification to the pool of workers in charge of executing the application steps.

In [15], the authors propose a cache method for contextual information in a cloud environment. The work is motivated by the strong temporal nature of context data due to their limited validity duration. The caching will allow the centralization of context data into a context broker memory and considerably improve the context-provisioning performance. The conducted work evaluated many replacement strategies to be used when the cache system is full while there is new information to be cached, namely: remove least used (LU) first, oldest first (OF) or soonest expiring (SE) first. After evaluating different approach, the work suggest the use of a bipartite replacement strategy based on dividing cache in two segments one for short validity duration context managed with OF and the other of long validity duration managed with SE. The framework aims only to provide context caching for enhancing context management performance and response time to context requests.

In [16], authors propose cloud architecture for provisioning contextual information. The architecture is based on the provider-broker-consumer model, where providers are responsible of getting context data from its source and publishing it to be available to others. Consumers take the published context to adapt the behavior of an application to the current situation described by a set of context data. Brokers are used to loosely couple between providers and consumers: when a consumer looks for context it ask the broker that will look up where it can be found, it contacts then the corresponding provider. The brokers are the components who can be duplicated in a cloud to scale the architecture to a large context-centric ecosystem. However, the framework does not involve context-aware application developers in context processing and their specific requirements cannot be considered.

B. Discussion

Only few works conducted earlier aimed to provide context-management in a cloud environment. Some of them were focusing in providing efficient and customized infrastructure for handling contextual information (mainly location) in a machine to machine communication and their integration with enterprise business processes. Others provided more generic and scalable framework for provisioning context in distributed environments, their aim was mainly the distribution of context among different parties.

Our platform enables ease of integration of new computation components and connected objects at design and runtime and ease of application development and maintenance. It provides the ability to change applications reasoning process by only having to upload new versions of corresponding descriptive file. Also, this file is writing in an expressive language based on XML that make it human readable and easy to interpret. Real-time performance is provided through the implementation of a scalable platform based on web service technologies.

However, the proposed context processing language could be more comprehensive, as it allows only monitoring context in a real-time and building decision trees on context but does not provide a way for building context-based timers that when expire should trigger an action. The flexibility of our cloud platform could be also enhanced, as the provisioning of reasoning engines is not automated, but need to be launched manually.

VI. CONCLUSIONS

In this work we presented a generic framework for context management based on Internet technologies: XML and Cloud computing.

Our contribution is the conception of a generic yet powerful cloud-based framework enabling the segregation of context reasoning and application logic. The reasoning algorithm can be hosted by the framework in order to filter context events and notify end application only when a situation of interest occurs. A flexible XML-based language is used for describing the reasoning and adaptation to context changes; it has to be uploaded (at design and/or runtime) to the framework for instantiating reasoning process for the corresponding application, and it can be modified at any time.

However, inference rules may become complex especially if too many contextual information is involved in the reasoning which makes its specification in XML files a tedious work for application developers. This can be addressed by building a specific graphical IDE (Integrated Development Environment) to support developers during application development and automate some tasks (e.g. generation of XML specification from a graphical representation of the reasoning).

Moreover, security aspects are currently not addressed. In a future work, we plan to implement distributed security mechanisms that match the specificities of cloud-based context management. We plan also to extend the proposed context processing language to provide support for defining context-based timers and investigate further the possibility of adapting the generation of XML description files based on feedback acquired from the application environment. This will enable an automated maintenance of context processing and an advanced separation of concerns of context management from application business logic.

REFERENCES

[1] T. Soikkeli, J. Karikoski, H. Hammainen, "Diversity and End User Context in Smartphone Usage Sessions," 5th International Conference on Next Generation Mobile Applications, Services and Technologies (NGMAST'11), Cardiff, Wales, UK 2011.

[2] B. Chihani, E. Bertin, F. Jeanne, N. Crespi, "Context-Aware Systems: A Case Study," Proceedings of International Conference on Digital Information and Communication Technology and its Applications (DICTAP'11), Dijon, France 2011.

[3] F. Toutain, A. Bouabdallah, R. Zemek, C. Daloz, Interpersonal context-aware communication services, IEEE Communications Magazine, Volume: 49, Issue: 1, January 2011.

[4] B. Chihani, E. Bertin, N. Crespi, "A Comprehensive Framework for Context-Aware Communications Systems," ICIN, Berlin, Germany 2011.

[5] J. Zhu, H. K. Pung, M. Oliya, and W. C. Wong, "A Context Realization Framework for Ubiquitous Applications with Runtime Support," IEEE Communications Magazine, Vol. 49, No. 9, September 2011.

[6] H. Kopetz, "Internet of Things," Real-Time Systems Journal, Springer, DOI: 10.1007/978-1-4419-8237-7_13, 2011.

[7] T. Sridhar, "Cloud Computing - A Primer," The Internet Protocol Journal, Volume 12, No.3, September 2009.

[8] C. Reichel, R. Oberhauser, "XML-based programming language modeling: An approach to software engineering," in IASTED Conf. on Software Engineering and Applications, IASTED/ACTA Press, pp. 424-429, 2004.

[9] Doctypedecl, "Extensible Markup Language (XML) 1.1," W3C Recommendation, 2004.

[10] S. Talwar, K. Johnsson, N. Himayat, K.D. Johnson, "M2M: From mobile to embedded internet," IEEE Communications Magazine, Volume: 49, Issue: 4, April 2011.

[11] L. Atzoria, A. Ierab, G. Morabito, "The Internet of Things: A survey," Computer Networks (ELSEVIER), Volume 54, Issue 15, 28 October 2010.

[12] P. Holleis, A. Schmidt, "Embedded Interaction: Interacting with the Internet of Things," IEEE Internet Computing, Volume: 14, Issue: 2, March-April 2010.

[13] K. Vandikas, N. Liebau, M. Döhring, L. Mokrushin, I. Fikouras, "M2M Service Enablement for the enterprise," In proceedings of 15th International Conference on Intelligence in Next Generation Networks (ICIN), Germany (Berlin), 2011.

[14] K. Vandikas, R. Quinet, R. Levenshteyn, J. Niemoller, Scalable service composition execution by means of an asynchronous paradigm, In proceeding of 15th International Conference on Intelligence in Next Generation Networks (ICIN), Germany (Berlin), 2011.

[15] S. L. Kiani, A. Anjum, N. Antonopoulos, K. Munir, and R. McClatchey, "Towards context caching in the clouds," International Workshop on Intelligent Techniques and Architectures for Autonomic Clouds (ITAAC'11), Australia (Melbourne), 2011.

[16] S.L. Kiani, A. Anjum, N. Bessis, R. Hill, M. Knappmeyer, "Context Parsing, Processing and Distribution in Clouds," In proceedings of 3rd International Conference on Intelligent Networking and Collaborative Systems (INCoS), Japan (Fukuoka), 2011.