# Mobile Architecture for Dynamic Generation and Scalable Distribution of Sensor-Based Applications

Marco Picone[1], Marco Muro[1], Vincenzo Micelli[2], Michele Amoretti[1], and Francesco Zanichelli[1]

[1] Distributed Systems Group - Università degli Studi di Parma, Parma, Italy
{picone,muro}@ce.unipr.it,
{michele.amoretti,francesco.zanichelli}@unipr.it
http://www.dsg.ce.unipr.it
[2] RimLab - Università degli Studi di Parma, Parma, Italy
micelli@ce.unipr.it

**Abstract.** The widespread and ubiquitous nature of mobile devices makes them attractive as providers of information collected from their rich equipment of sensors (camera, microphone, GPS, etc.), and also from external sensors (placed on persons, or in the environment). Thus, we envision large-scale sensor networks that use mobile devices as raw data sources, but also aggregated information producers - merging basic data coming from sensors distributed in the environment.

In this paper we propose an architectural framework for agile development and deployment of mobile cloud applications, harvesting heterogeneous sensor data. The novelty of the architecture is the possibility to dynamically manage multiple internal and external sensors, and generate graphical user interfaces to collect user inputs and present semantically integrated information, in a cloud-based personalized fashion.

**Keywords:** context-aware networking, cross platform development, mobile cloud.

## 1 Introduction

Recent years have seen the relentless market success of mobile devices (PDAs, smart-phones, MIDs, PMPs, net-books, etc.), whose ever increasing capabilities make them attractive to a growing number of network applications in business and infotainment domains, that now can be fully experienced in mobility.

At the same time, the widespread and ubiquitous nature of mobile devices makes them attractive as providers of information collected from their rich equipment of sensors (camera, microphone, accelerometers, compass, GPS), and also from external sensors (placed on persons, or in the environment). Thus, we envision large-scale sensor networks that use mobile devices as raw data sources, but also aggregated information producers - merging raw data coming from sensors distributed in the environment. There are several important challenges in

N. Venkatasubramanian et al. (Eds.): Mobilware 2011, LNICST 93, pp. 219–232, 2012.

realizing such types of distributed applications, *e.g.* providing efficient methods for sensor nodes to make their data available to the network, allowing data access from potentially disconnected and highly mobile devices, ensuring that privacy constraints are met, and allowing application developers to build modular, service-oriented applications.

With the growing of the mobile application market, new solutions for software distribution have been recently introduced, with the dominance of online application stores (such as AppStore and Android Market). These virtual marketplaces have solved many problems to developers and common users, allowing for easy dissemination and installation of apps with specific security and update management policies. If on one hand this approach allows for widespread distribution of applications, on the other hand it appears to quite unsuitable for highly dynamic scenarios where application needs may change very frequently within a week or even the same day, according to user credentials, location or purposes [5].

In this context, we propose an architectural framework for agile development and deployment of mobile cloud applications, harvesting heterogeneous sensor data. The novelty of the architecture is the possibility to dynamically manage multiple internal and external sensors, and generate graphical user interfaces to collect user inputs and present semantically integrated information, in a cloud-based personalized fashion.

The proposed architecture is characterized by two main functional modules, namely the *service platform* and the *mobile platform*. The former is characterized by a semantic service engine, that allows for dynamic composition of sensed data, to be presented to mobile users in a personalized fashion. The latter is a lightweight mobile application that is able to manage sensors and interact with the service platform for storing/retrieving data. The mobile platform is designed for running over different mobile operating systems (such as iOS, Symbian/RIMM, Windows Mobile and Android) and tolerating the rapid obsolescence of technologies and, consequently, of devices that are frequently substituted by models with different features and functionalities. Its flexibility support the "install once, run forever" paradigm, thus solving the previously discussed software distribution issue.

The paper is structured as follows. Section 2 presents the state of the art in the field of sensor-based mobile architecture. Section 3 specifies the proposed architecture. Section 4 introduces two example scenarios and describes the first prototype of a demo application we are developing. Finally, section 5 concludes the paper summarizing achieved results and proposing some future developments.

## 2 State of The Art

Integration of mobile devices and sensor networks for context awareness is a hot topic in the field of distributed systems. Context-aware computing is a mobile computing paradigm in which applications can discover and take advantage

of contextual information (such as user location, time of day, nearby people and devices, and user activity). In their recent work [12], Soylu *et al.* integrate and extend fundamental and promising theoretical and technical approaches for the development of adaptive, context-aware software systems. Moreover, they present an interesting view point for context-aware pervasive application development, particularly based on higher abstraction where ontologies and semantic web activities, also web itself, are of crucial importance. The long and short of it is that perception, adaptivity, interoperability and standard compliance are key enablers of pervasive computing.

Perception means (especially) sensing the environment. In [4], Bednarz *et al.* present a multi-sensor XML-based communication protocol, called Human System Integration Protocol (HSIP). Hard-wired or wireless sensors are assumed to be connected to a data server to which, after registration, they to transmit information.

Sensor networks consist of tiny low-powered computing devices with extremely restricted computational, communication and battery capabilities. Each device may be equipped with a physical sensor for reading temperature, sound, pressure or other physical phenomena and can operate both as a sensor and a wireless router. One of the major tasks of sensor networks is the distributed collection and processing of sensor readings over extended periods of time. Scalability, self-configuration, ease of deployment and low cost have made sensor networks a very attractive solution for a wide range of environmental monitoring, distributed surveillance, healthcare and control applications. In many situations, collecting data at a certain fixed location is neither possible nor practical. Having mobile collectors (that collect data and transfer messages between individual sensors [8]) can be the only way to solve the problem - a thesis that we support in this paper.

Another approach, suggested by Kansal *et al.* [9], is to implement a sensor network of mobile phones, to be provided as a shared system, as opposed to a system where a single application owns and uses a dedicated set of mobile devices carried by users or vehicles. Using mobile devices as sensors has a significant advantage over unattended wireless sensor networks in that deploying the sensing hardware and providing it with network and power is already taken care of. Secondly, mobile phones can provide coverage where static sensors are hard to deploy and maintain. Thirdly, each mobile device is associated with a human user, whose assistance can sometimes be used to enhance application functionality. For instance, a human user may help by pointing the camera appropriately at the target object to be sensed.

On the problem of creating device-independent interfaces, in literature we found some interesting works. Among others, in [10] Nichols *et al.* present the *Personal Universal Controller (PUC)*, an approach for improving the interface to complex appliances by introducing an intermediary graphical or speech interface. A PUC engages in two-way communication with everyday appliances, first downloading a specification of each appliance's function, and then automatically creating an interface for controlling that appliance.

## 3 Proposed Architecture

We propose a mobile device -centric approach for sensor-based distributed applications. The main idea (sketched in figure 1) is that mobile nodes collect data from sensors (their own ones, or those placed in the surrounding environment), and share such data within the network, by means of cloud services [7]. Authorized users can visualize data thanks to mobile web applications (that are independent from the mobile platform) and dynamically compose the user interface according to currently selected data visualization services, to context and to user profiles.


**Fig. 1.** Mobile device -centric approach

As we know, mobile terminals such as smart-phones and tablets allow to collect a huge amount of data from local and external inputs and sensors connected by multiple interfaces such as Bluetooth, WiFi or USB. Concurrently, user inputs may contribute to enhance the quality and effectiveness of the application, by enriching collected data before they are shared within the network.

Such a mobile device -centric approach has a number of advantages over the centralized approach for sensor data collecting, namely:

- *Energy efficiency* In a mobile scenario, sensors will store data locally and provide them to mobile collectors. Data can be sent directly when the appropriate connection is available, or in a subsequent time, according to user preferences and always trying to preserve sensor battery lifetime.
- *Improved availability of sensor networks* Mobile devices may act as gateways for sensor networks that, for some reasons, are disconnected from the Internet.
- *Ubiquity* Users can query the network and collect data from any location.

Figure 2 illustrates the global architecture, where the following elements are highlighted:

- Mobile Device (MD) - Is the principal entity of the system, interacting with external sensors to collect data, exchanging them with the cloud, and allowing user to enter extra information.

– The Cloud - Provides personalized environments with services for storing information generated by MDs, as well as mobile Web user interfaces (UIs).
– External Modules (EMs) - Represent external software entities that, according with their authorization level, can interact with the Cloud to consume its services, or to provide new ones, for example to integrate multi-source data and provide new information.
– MD Communication Layer - Allows the communication between the Cloud and the MD in terms of login procedures, data exchange, UI specification transmission and notifications pushing to the device.
– EM Communication Layer - Allows the communication between the Cloud and EMs to transmit and receive data.

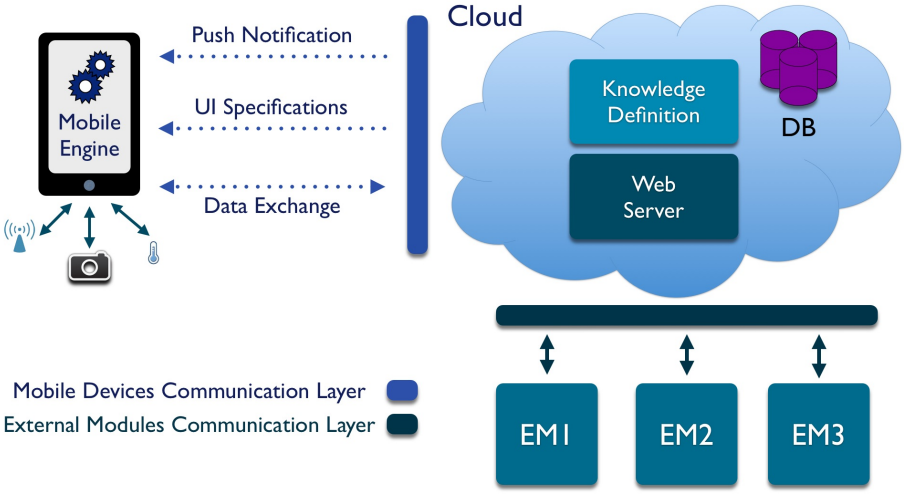



**Fig. 2.** The proposed architecture

In the following we describe the mobile platform that runs over the MD, and the service platform used by the Cloud and EMs.

### 3.1 Mobile Platform

On the mobile side, there is the necessity to communicate with different types of external sensors and to generate a rich user interface to collect user input. To these purposes, a cross-platform solution (*e.g.* HTML-based, such as PhoneGap [11]), for the development of the mobile engine, may not be efficient (not providing thread management and synchronization mechanisms). Similarly, a mobile code solution, where software pieces are transferred between systems (*i.e.* transferred across a network or via a USB flash drive) and executed by the recipient on a local system, has several issues, related to runtime generation of the user interfaces on multiple platforms, and also to the management of different sensors.

For these reasons, in our opinion the mobile platform need to be developed using native programming languages - such as Java, C++ or Objective-C - for

each platform, and provided with the ability to automatically generate, starting from a standardized UI Specification Language, all user interfaces for interacting with available sensors in the environment, and accessing remote services. This approach allows for reduced complexity of the mobile platform dissemination and update process, too (the mobile platform needs to be downloaded once, for example from an app store). This approach is also highly scalable, allowing for real-time, context-driven adaptation of the application, *e.g.* taking into account user location, environment settings, etc.

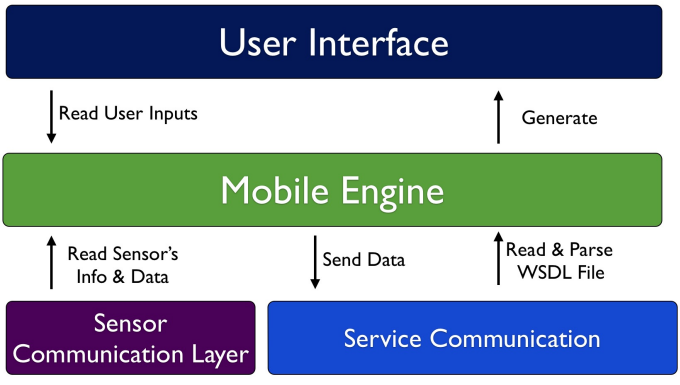



**Fig. 3.** The Mobile Platform

Figure 3 describes the mobile platform, whose main components are:

- Mobile Engine - The intelligent core of the mobile platform, interacting with other mobile modules to send and receive data from local and remote sources, and dynamically generating the GUI that allows the user to enrich sensor data with other useful knowledge.
- Sensor Communication Layer - Middleware enabling the communication with internal and external sensors/inputs through different channels, according to the device profile.
- Service Communication Layer - Middleware allowing the interaction with the Cloud to discover services, test their QoS, send/retrieve data, etc.

Aggregated sensor data are presented to the user on his mobile device by the service platform in a personalized fashion, by means of a Web-based application.

### 3.2 Service Platform

Mobile devices exchange data with the Cloud, that is enabled by a service platform (illustrated in figure 4) provided with the following components:

- Semantic Service Engine - The core of the service platform, managing all available modules in order to provide different types of services to final users.

– Database Communication Layer - A middleware for storing and retrieving data and information from system databases.
– Service Ontology - A Web 3.0 knowledge base encompassing all the characteristics of each service, user, sensor and inputs.
– Client Communication Layer - A middleware that enables mobile devices to connect transmit and receive data.
– Security Layer - A middleware that provides protocols for securing the communication between service platform and mobile clients.
– Web Interface Builder - A component that builds dynamic Web pages for showing real-time or cached data in personalized fashion, according to user needs, profile and credentials.

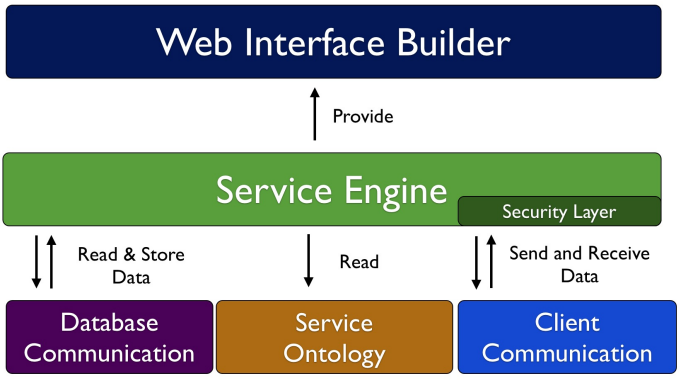



**Fig. 4.** The Service Platform

Services exposed by the Cloud have specific input parameters (representing sensor data). The Mobile Engine discovers such information at runtime. A service can be used only if the Mobile Engine is able to provide all needed sensor data. The service description should be something like:

– **Service.** Defines a specific functionality with different UI sections and a list of required sensors with the following parameters.
  • *Name:* Unique identifier of the service.
  • *Storing Type:* Defines the storing procedure of input and sensor data. The service may enable immediate or delayed data transmission (the latter would require that the Mobile Engine temporarily stores data in a local file) - possible parameters being `REMOTE` and `LOCAL`. This dual opportunity allows the user, once the service description has been retrieved, to execute the service offline, by caching data and uploading the locally stored file later, when connectivity is available.
  • *Storing Location:* In case of a remote storage procedure, this tag contains the service endpoint; otherwise, in case a local storage on the mobile device, it contains the filename.

- **UI-Section:** Defines a single section of service's UI allowing the designer to divide data entry in different slices, that may be either mandatory or not, and may contain labels and figures to show useful information, or input fields.
  - *Name:* Unique identifier of the section.
  - *Description:* Short text containing general information about presented data, or the instructions about required data.
  - *Mandatory:* Boolean field to specify whether the section can be skipped or not.
- **Input**: Represents a generic input for the data entry.
  - *Name:* Unique identifier of the field.
  - *Type:* Defines the type of requested data, *e.g.* string, numeric, list, boolean etc. According to the input type and to the specific platform, the Mobile Engine will properly generate the UI to simplify data collection.
  - *Mandatory:* Boolean field to specify if the section must be filled or not.
- **Label**: Represents a text label with an associated image.
  - *Name:* Unique identifier of the label.
  - *Text:* Label's text.
  - *Image:* Associated image.
- **Sensor List :** Contains the list of sensors required by the service.
- **Sensor**
  - *Type:* Defines the sensor type according to the Service Ontology. By means of this field, the Mobile Engine can search among already discovered and bound sensors to verify if service needs can be satisfied.
  - *Mandatory:* Boolean field that specifies whether the sensor must be available or not.
  - *Working Frequency:* Specifies the sampling rate of the sensor's data.
  - *Reading Limit Type:* Defines if the sampling of sensor data will stop after a specific time, or upon reaching a specific number of collected samples (possible parameters are `TIME` and `SAMPLES`).
  - *Reading Limit Value:* Represents the amount of seconds or the number of samples after which the sampling of sensor data will stop.

The Mobile Engine reads and shows to the user the list of available services. Once a service has been selected, the operator can choose among available and bound sensors which he/she wants to use for the service. If more than one sensor of the same type are available, the user can select which of them must be used. Similarly, if one or more mandatory sensors are no more available, the engine shows an error message and prevents next interaction. After the sensor selection phase, the Web Interface Builder dynamically generates the required UI according to service description allowing the user to add manual inputs associated with the data that could be retrieved from the sensors.

Dynamic UI generation for data visualization is provided by the Web Interface Builder of the service platform, and published to the mobile client as a personalized Web interface. Using platform-specific CSS sheets, such a Web interface can be customized. Finally, the Security Layer allows to visualize information only to users that are provided with the right credentials.

### 3.3 QoS and QoI Management

In the proposed mobile device -centric architecture, quality of service (QoS) management is very important. Each service may specify a minimum guaranteed quality of service, defined in terms of transmission rate, packets error, computational power, connection type, etc. During the connection establishment phase, the server could ask to the client to check if it is able to send data with the requested QoS. This kind of test could be also repeated periodically during the data delivery phase, to identify possible lowering of performance and QoS.

Another important aspect we took into account is the quality of information (QoI), that refers to the ability to figure out if available information coming from sensors is fit-for-use for a particular service. Let us consider a scenario in which several PDAs want to use services that require sensors provided by a Wireless Sensor Network (WSN). Each service has to execute tasks that need resources from the network. QoI management provides mechanisms for investigating new task admission and resource utilization, for controlling the individual QoI provided to new and existing tasks. This can be done using real-time feedback-based monitoring systems. The QoI can be characterized by a set of quality attributes, such as accuracy, latency, and spatio-temporal relevancy. To this purpose, as suggested in [13], we can consider three key design elements: (a) the QoI satisfaction index of a task, which quantifies the degree to which the required QoI is satisfied by the WSN; (b) the QoI network capacity, which expresses the ability of the WSN to host a new task with specific QoI requirements without sacrificing the attained QoI levels of other existing tasks, and (c) an adaptive, negotiation-based admission control mechanism that reconfigures and optimizes the usage of network resources in order to optimally accommodate the QoI requirements of all tasks.

## 4 Analyzed Scenarios and Prototype

In this section we analyze two appealing scenarios that perfectly match with our dynamic architecture. The first scenario is related to an industrial environment where one or more operators make an inspection of different production line, whereas the second one is associated to an e-Health system that collects information about patients through sensor interaction and user feedbacks. The important aspect is that both scenarios can be realized with the same mobile engine implementation, without specific development or efforts on the mobile side, thus reducing costs, focusing on service characteristics and increasing scalability. We have started the development of applications for both scenarios. In the last subsection we describe those being implemented for the industrial one.

### 4.1 Industrial Scenario

Mobile applications have become an important means for improving industrial service processes [2] [3]. Since their increasing complexity should not reduce usability, the personalized UI approach we propose appears to be highly suitable.

Let us consider a small/medium company whose core business is to bottle any type of drink, from water to wine. A large amount of this business is based on the perfect functioning of the production line, that is costly in terms of safety and security. Periodic controls and ordinary maintenance are very important to prevent accidents and to assure the correct working of the system.

Each industrial machine endows one or more sensors that collect status information, allowing to detect potential problems all along the production line. Traditional plants have pipelined machines, each one being cabled with a computer that stores data sent by the machine, and operates on the machine's actuators. Usually the software running on master machines is not standard-based, for which it is difficult to collect and integrate data from several machines, unless they are made by the same vendor.

Our approach would require a unique server running the Service Platform, and a set of mobile devices running the Mobile Engine. Company operators would perform machine maintenance according to the following procedure:

- collect sensor data using a smartphone;
- check bottlers' status by evaluating received data;
- eventually discover any change in the composition and structure of the machines;
- make interviews with workers of the warehouse, to investigate any questioning or difficulty concerning machines, and to enrich gathered data;
- immediately update all collected information.

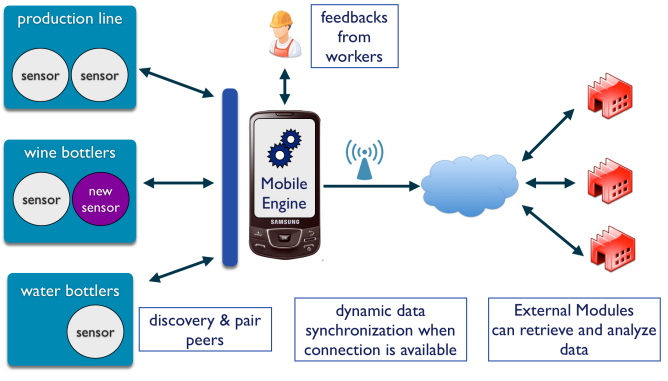



**Fig. 5.** Industrial scenario

Suppose that the company has started a new, more efficient bottling line. The chief operator notifies this change to the operations centre, by adding the sensors of the new line to the list of monitored ones, using his smartphone. The chief operator may also collect feedbacks from workers in order to enrich sensed information provided to the operations centre. The mobile device can also work in offline mode, by locally storing important information and uploading them as soon as a connection is available.

### 4.2 e-Health Scenario

The proposed architecture can be used in a number of healthcare scenarios. In particular, some diseases require to monitor the status of patients that follow specific treatments, assigned by their doctors. Patient monitoring may be very costly, when manually performed by a specialist. Our architecture can help simplifying this process, by automating patient monitoring. Figure 6 shows a scenario we plan to address, related to individuals with walking disabilities, provided with Ankle-foot orthoses (AFOs) to aid in their walking (we prepared and published online an AFO demonstrative video [1]).

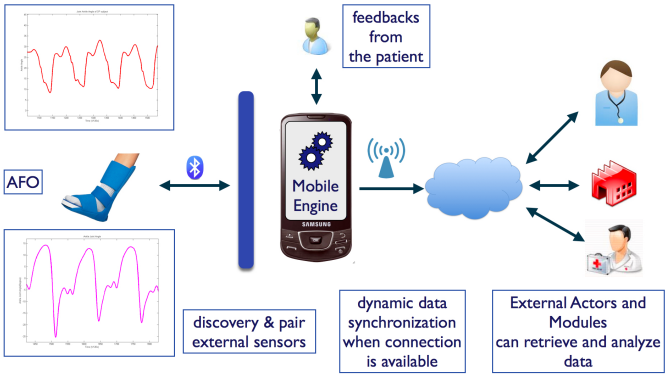



**Fig. 6.** e-Health scenario

Despite the widespread use of AFOs, their performance is not well evaluated, because the quantitative assessment is currently limited to short-term in-clinic observation. To better understand how AFOs perform in aiding individuals with walking disabilities and further enhance the AFO efficacy, a continuous, non-invasive measurement method is necessary. After a careful investigation by Gait analysis experts, ankle angle is selected as a primary Gait parameter for assessing the efficacy of AFO.

Our RimLab is working on a sensor-provided AFO, able to collect data about the ankle angle. Thus, a patient could use her/his PDA to send AFO data and insert additional qualitative information (*e.g.* how much the AFO is comfortable). The Doctor may access such collected data to monitor the patient and to evaluate the effectiveness of the proposed treatment.

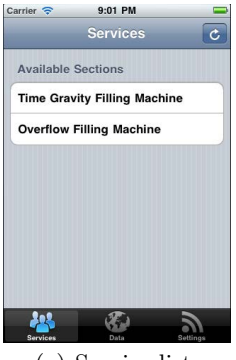
The AFO case study can be generalized, considering any set of sensors, targeting different diseases. The PDA is used to access services that allow to send sensed data, to manually insert additional information, and to get treatment updates, doctor's feedbacks, etc. Similarly, the doctor uses services to read information coming from sensors or user inputs, to define new inputs that the patient has to insert, and to modify the treatment.

### 4.3 Prototype

In order to evaluate our architecture and our approach we have designed and developed a first prototype of the system and a demo application (a video is available at [6]). Main developed components are:

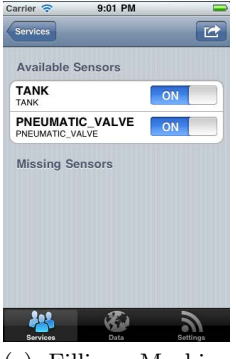
- Semantic Service Engine based on PhP technology to provide the list of available services and their description.
- Database and associated Communication Layer to parse, store and retrieve data.
- Mobile Engine prototype on the iOS platform (namely, iOS-ME) in order to test its scalability and usability.

Up to now we have been mainly focusing on dynamic user-friendly interface generation, data exchange, and onboard sensor interaction, while leaving external sensor interaction as future work. iOS-ME, that has been implemented in the Objective-C programming language, presents to the user three main "views" (shown by screenshots in figure 7). We recall that, in the iOS development environment, a view is a screen presented to the user.

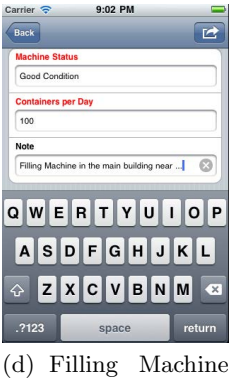



(a) Service list.

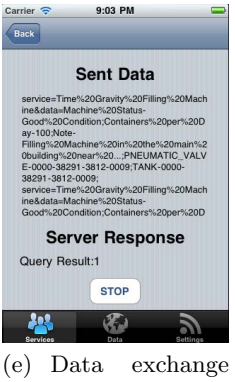

(b) Sensor discovery and pairing




(c) Filling Machine needed sensors




(d) Filling Machine input fields




(e) Data exchange interface


(f) Remote data visualization

**Fig. 7.** iOS Mobile Engine Prototype

The "Setting" tab emulates external sensor management and is used by the Sensor Communication Layer to show discovered external devices and already paired sensors. The "Services" section allows to discover Cloud services, with associated description, and to visualize them in a table structure. When clicking on an available service, iOS-ME shows a summary of required sensors, and the list of those that are already available or missing. The user is allowed to select sensors that she/he wants to use for the specific service. After that, the user can move to the next "view" - the one devoted to data entry. As described before, each UI-section contains both mandatory and not mandatory fields, differently highlighted (red and black). Until all mandatory fields are correctly filled, the engine prevents the user from moving to next view. After that, when all user data are collected and sensors are correctly configured, iOS-ME starts the data exchange with the server and shows a report about each operation in order to show possible communication errors to the user. This task continues until the user clicks the stop button, that ends the uploading procedure. The last application tab is related to data visualization, that (as described in section 3) is based on a mobile Web technologies, with an appropriate style sheet for each device type.

## 5 Conclusion

In this paper we have illustrated a novel architecture for agile development and deployment of mobile, sensor-based applications. The distinctive feature of such an architecture is the mobile device -centric approach, for which mobile devices are providers of information collected from their rich equipment of sensors (camera, microphone, GPS, etc.), and also from external sensors (placed on persons, or in the environment). We have illustrated two significant example scenarios, and a prototype we have developed and tested.

As future work, we are going to complete the development of the Service Platform, improving the semantic service engine. Moreover, we are going to develop versions of the Mobile Engine for other platforms than iOS - *e.g.* Android and BlackBerry.

## References

1. AFO demontrative video, `http://rimlab.ce.unipr.it/Research.html`
2. Aleksy, M., Stieger, B., Vollmar, G.: Case Study on Utilizing Mobile Applications in Industrial Field Service. In: IEEE Conf. on Commerce and Enterprise Computing, Vienna, Austria (2009)
3. Aleksy, M., Stieger, B.: Supporting Service Processes with Semantic Mobile Applications. In: 8th ACM-SIGMM International Conference on Advances in Mobile Computing & Multimedia (MoMM 2010), Paris, France (November 2010)
4. Bednarz, T.P., Caris, C., Thompson, J., Wesner, C., Dunn, M.: Human-Computer Interaction Experiments. In: 24th IEEE International Conference on Advanced Information Networking and Applications (2010)

5. Das, T., Mohan, P., Padmanabhan, V.N., Ramjee, R., Sharma, A.: PRISM: Platform for Remote Sensing using Smartphones. In: MobiSys 2010, San Francisco, California, USA (June 2010)
6. Video of the demo application, http://dsg.ce.unipr.it/?q=node/51
7. Dikaiakos, M.D., Katsaros, D., Mehra, P., Pallis, G., Vakali, A.: Cloud Computing: Distributed Internet Computing for IT and Scientific Research. IEEE Internet Computing 13(5) (September-October 2009)
8. Dyo, V.: Middleware Design for Integration of Sensor Network and Mobile Devices. In: 6th ACM International Middleware Conference, Grenoble, France, pp. 49–54 (November 2005)
9. Kansal, A., Goraczko, M., Zhao, F.: Building a Sensor Network of Mobile Phones. In: 6th IEEE International Symposium on Information Processing in Sensor Networks, Cambridge, Massachusets (2007)
10. Nichols, J., Myers, B.A., Higgins, M., Hughes, J., Harris, T.K., Rosenfeld, R., Pignol, M.: Generating Remote Control Interfaces for Complex Appliances. In: UIST 2002, Paris, France (October 2002)
11. PhoneGap homepage, http://www.phonegap.com
12. Soylu, A., De Causmaecker, P., Desmet, P.: Context and Adaptivity in Pervasive Computing Environments: Links with Software Engineering and Ontological Engineering. Journal of Software 4(9), 992–1013 (2009)
13. Liu, C.H., Bisdikian, C., Branch, J.W., Leung, K.K.: QoI-Aware Wireless Sensor Network Management for Dynamic Multi-Task Operations. In: 7th IEEE Conference on Sensor Mesh and Ad Hoc Communications and Networks, SECON (2010)