Service Oriented Tasking System for WSN

Benjamin Le Corre, Jérémie Leguay, Mario Lopez-Ramos, Vincent Gay, Vania Conan
*Thales Communications, 160 Bd de Valmy, Colombes Cedex, France
{firstname.lastname}@fr.thalesgroup.com

***Abstract*—The increasing ubiquity and diversity in the resources and functionalities of the devices of the *Internet of things* claims the need for (1) a device-independent abstraction layer to expose (and consume) device capabilities as services on the network and the need for (2) mechanisms to customize the behaviour of the devices to fulfill new or additional operations in collaboration with one another. To this end, this paper presents the extension of the WSN-SOA [1] stack to support efficiently the dynamic deployment of service-oriented collaborative tasks in networks of tiny sensor motes such as MicaZ.**

I. Introduction

The increasing ubiquity of network-connected devices in our everyday environments (e.g., at home, at work, in the city) is driving an evolution in interaction patterns, from traditional human-computer to machine-to-machine scenarios. For instance, remote monitoring and control solutions, previously limited to industrial systems such as SCADA, are currently available in the market as home automation solutions for the automatic control of lights, shutters or heating.

The advent of the Internet of Things

In the same way, in the years to come we will witness the growth of consumer electronics based on wireless technologies such as RFID (Radio-Frequency Identification) and WSN (Wireless Sensor Networks). WSN are currently mostly devoted to the measurement of simple environmental parameters such as temperature or humidity. However, nodes are increasingly affordable and miniaturised, and power consumption optimisations are expected to make batteries last several years.

These advances will make such sensors ready to be integrated in everyday objects such as furniture or clothes. Smart spaces will not only involve the system sensing patterns in the environment, but also the user interacting through objects that were previously inanimated, using them as input or output peripherals and linking their behaviors, thus realizing the dream of an *Internet of Things* world.

Service composition and tasking

The increasing diversity in the resources and functionalities of the devices of the *Internet of Things* raises the need for a device-independent abstraction layer exposing the functionality of each sensor and actuator as a service on the network. In a previous work [1], we presented the WSN-SOA stack, a lightweight service-oriented stack for WSN. With such an approach, ubiquitous applications involving from the simplest temperature sensor to highly optimized cloud computing web services can be easily created through service composition and orchestration. In addition, subscription mechanisms in WSN-SOA avoid the need for polling thus saving on batteries, the scarcest resource in wireless sensors.

Although such an approach is sufficient for a wide range of ubiquitous applications, autonomous cooperation between wireless sensor nodes is limited by the inability to modify the behavior of the devices in order for them to fulfill new or additional operations (e.g. event detection, object tracking, actuator command and control, data aggregation). In this paper, we will first review the existing solutions for the dynamic deployment of processing tasks in WSN. Then, we introduce an extension of WSN-SOA for the dynamic deployment of services and collaborative tasks. This approach benefits both from the modularity of WSN-SOA and the flexibility of the Tenet [2] tasking framework to reconfigure a WSN to execute a specific collaborative process.

II. Related work

A number of propositions exist in the literature for the dynamic deployment of processing tasks in wireless sensor networks. This section reviews the existing solutions in bibliography that enable either to reprogram or to send code to a node or group of sensor nodes. We evaluate these solutions against the three following criteria:

- **Global overhead:** Sending code to a sensor or a group of sensors consumes bandwidth for the *request phase* and consumes memory and processing power for the *execution phase*.
- **Required programming effort:** Depending on the programming model, the abstraction level and the amount of supported functions, reprograming a sensor network can be more or less difficult.
- **Task complexity:** The complexity of the tasks that each framework allows to deploy on the sensor nodes might be different.

Fig. 1 shows a qualitative comparison of the different existing approaches with regards to the previous criteria. An ideal solution would have the maximum score for each axis.

Over-the-air flashing

The Deluge [3] software allows to reprogram sensors by flashing their ROM memory over-the-air. As the entire node program must be transmitted to nodes, it consumes a lot of resources (battery, bandwidth). It potentially requires a significant programming effort as it does not provide any programming support but potentially offers the possibility to

978-0-7695-4160-0/10 $26.00 © 2010 IEEE
DOI 10.1109/DeSE.2010.18

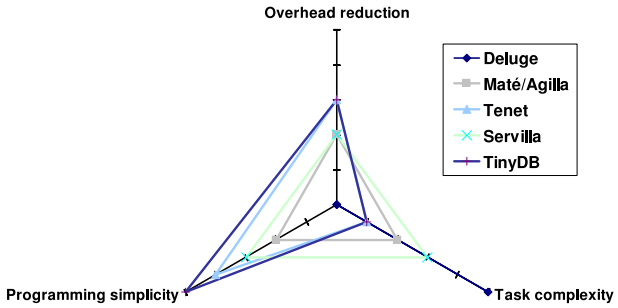

Figure 1. Evaluation of existing solutions.

deploy tasks of a great complexity. Note also that all the tasks running on nodes are being restarted without restoring their state when nodes are reprogrammed.

Virtual machines

Agilla [4] is a virtual machine-based middleware which allows the use of mobile agents. Mobile agents are defined in the form of programs, with a complex syntax similar to the assembly, interpreted on the sensors. Maté [5] takes a similar approach, being a byte code interpreter for TinyOS. In both solutions assembly programs can be deployed remotely by using a mobile agent injector for the first one and a code capsule for the second one. These approaches reduce the global overhead and the programming effort but the task complexity is limited by the available instructions (opcodes) in the nodes and by the fact that no communication support is offered for collaboration between sensor nodes.

Action chains

The Tenet [2] software aims at distributing processing on sensor nodes in the form of tasks. Each task consist in a chain of *tasklets* which are called sequentially. Tenet reduces communication costs with respect to virtual machine-based solution by relying on pre-deployed unitary complex operations. Tenet supports tasking with a very simple programming language but is limited by the number of tasklets available at nodes and does not offer any support for collaboration between sensor nodes.

Service orientation

Servilla [6] is a service-oriented middleware running across heterogeneous wireless sensor nodes (e.g., TelosB and Imote2) on top of which one can execute scripts. Servilla relies on Agilla but provides an easier C-style scripting language and service-oriented features such as dynamic service discovery and invocation. Thanks to its service-oriented approach and its programming language, Servilla enables to deploy more complex tasks in terms of functionality and sensor nodes collaboration while keeping a global overhead similar to virtual machine-based solutions.

Macroprogramming

Whereas previous approaches focused on the modification of the local behavior of nodes, a different class of solution focuses on the global behavior of a distributed sensor network as a whole, also referred as *macroprogramming*. In this class, Cougar [7] and TinyDB [8] abstract a sensor network as a database. They provide easy SQL-like interfaces for extracting the data of interest from sensor nodes. These solutions offer a very easy way of tasking a sensor network while consuming a fairly little global overhead. However, they are limited to data collection operations.

III. Service-oriented tasking

From the survey presented in Sec. II, we found that all the solutions face a general trade-off between programming simplicity, tasks complexity and overhead reduction. However, we argue that without adding overhead or programming complexity, Servilla and Tenet would benefit from the support of a rich SOA middleware to easily support the reconfiguration of complex collaboration processes between sensors.

The *WSN*-SOA stack [1] is a good candidate as it features service discovery and invocation, eventing operations and service hosting. The *WSN*-SOA protocol stack has been introduced in previous work as part of a multi-tier service-oriented architecture for heterogeneous WSN. This architecture seamlessly integrates devices with resources ranging from those of a PDA to extremely constrained sensors such as Crossbow's MICAz.

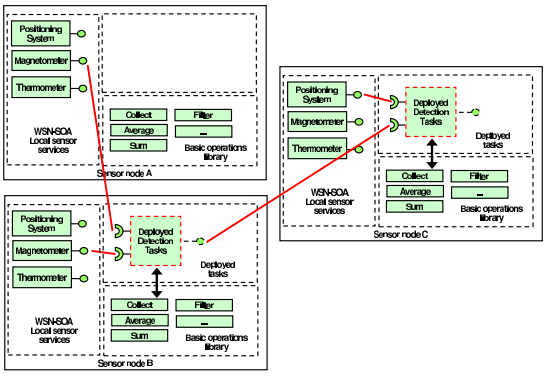

Figure 2. Example of a service-oriented task deployment.

By combining the advantages of WSN-SOA for distributed applications and tasking mechanisms such as Servilla and Tenet, distributed processing tasks can be deployed on nodes to make them cooperate autonomously. We propose that the tasks consist in simple scripts able to invoke local or remote services, subscribe to local or remote events, expose dynamically new services to offer new data or processes, and manipulate data using a library of pre-programmed functions.

Fig. 2 illustrates how service oriented tasks can be deployed. In this example, a task deployed on node C uses a local

positioning service and the information exposed as a service by another task deployed on node B which fuses data from magnetometer services running on nodes A and B.

IV. WSN-SOA: SOA FOR LOW CAPACITY NODES

As targeted devices are not able to process XML messages, we proposed in a previous work WSN-SOA [1], which consists in a simple protocol and software stack that reproduces the architectural concepts and information exchanges of SOA implementations for the plug-and-play networking of devices, such as UPnP [9] or DPWS [10]. The main goal is to make sensor nodes able to host services, announce them in the network, discover others' services, invoke their operations and subscribe to events.

This section presents the WSN-SOA protocol as well as its implementation on the open-source operating system TinyOS 2.1 [11], [12]. We developed the WSN-SOA for the Crossbow MICAz sensors equipped with the MTS310 sensor board attached to their serial port which offers a variety of sensing modalities such as light, pressure, acceleration, temperature and acoustic. This hardware combination provides also two *symbolic* actuators such as a sounder and a set of 3 leds. MICAz nodes have very limited capacity in memory and processing power as they only embed an Atmel ATmega128L microcontroller with 4KB of RAM and have 128KB of programmable flash ROM.

A. Message format

The messages exchanged within the WSN-SOA follow the message format depicted in Fig. 3. WSN-SOA messages are embedded in *multi-hop messages* that we have defined to enable multi-hop communications between sensors. The *src* and *dst* fields indicate respectively the address of source and destination nodes. The *type* field is used to characterize the kind of messages that are embedded in packets (i.e., WSN-SOA messages in our case). WSN-SOA message headers contain the following fields:

- **SrcServID:** This identifier allows to address the service which initiates the information exchange on the source node.
- **DestServID:** This field identifies the service on the destination node.
- **OperationID:** Within a service, several operations can be implemented. They could either correspond to a function which can be invoked or to an event source to which one can subscribe. This field then identifies the operation within the destination service.
- **MEP**: The Message Exchange Pattern field defines the semantics of message exchanges: request, response, subscription, unsubscription, notification, acknowledgment. To exemplify their use, for an invocation procedure, the source service sends a message with the MEP *request* and receives a response message from the destination service with the MEP *response*. The MEPs *subscription*, *unsubscription*, *notification* and *acknowledgment* are used in eventing procedures. Acknowledgments can be sent back to the source service to confirm the reception of subscriptions, unsubscriptions and requests without response.
- **Payload**: This field contains the data exchanged between services. These data can be of any kind but their size needs to be smaller than the WSN-SOA message payload size. Fragmentation can easily be introduced, although we observed that in most cases the exchanged data is very small (e.g., an integer).

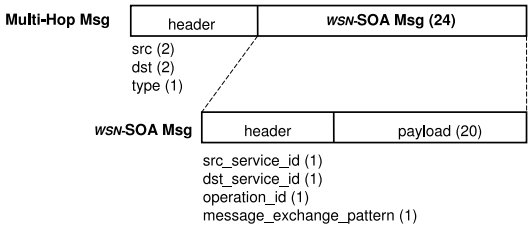

Figure 3. WSN-SOA message format in TinyOS.

B. Software architecture

Fig. 4 presents the WSN-SOA software architecture as implemented in TinyOS. The WSN-SOA protocol and service stack rests upon:

- **Routing module:** This module operates a multi-hop routing protocol. In our context, network topology is completely decentralized and message exchanges can occur between any possible pair of nodes, so the protocol we implemented is unicast and very similar to DSDV [13].
- **Forwarder module:** This module forwards incoming and outgoing messages according to the routes established by the routing module. It processes messages according to a FIFO queuing management mecanism.

WSN-SOA messages enter the WSN-SOA dispatcher which is in charge to receive all the messages (local and remote) and to send them to the right component or service, according to the message type (the MEP) and the serviceID. The core middleware component contains the following modules:

- **Broker engine**: This key module is in charge of discovering services in the network on the behalf of local tasks or services using broadcast requests. Services can express their needs to the broker engine, which then performs the discovery in a transparent fashion. This makes services lighter and easier to implement.
- **Notifier**: This module handles all the subscription and unsubscription requests and updates the *Subscription* database accordingly. This database contains the service ID and the mote address of the subscribers, as well as the local service ID and operation ID of the event they subscribed to. Whenever a notification is issued by a local service, the Notifier notifies all the subscribers of this event by sending dedicated notification messages.

The WSN-SOA stack has been implemented in TinyOS with an extensive use of modules. Indeed, every service is a TinyOS

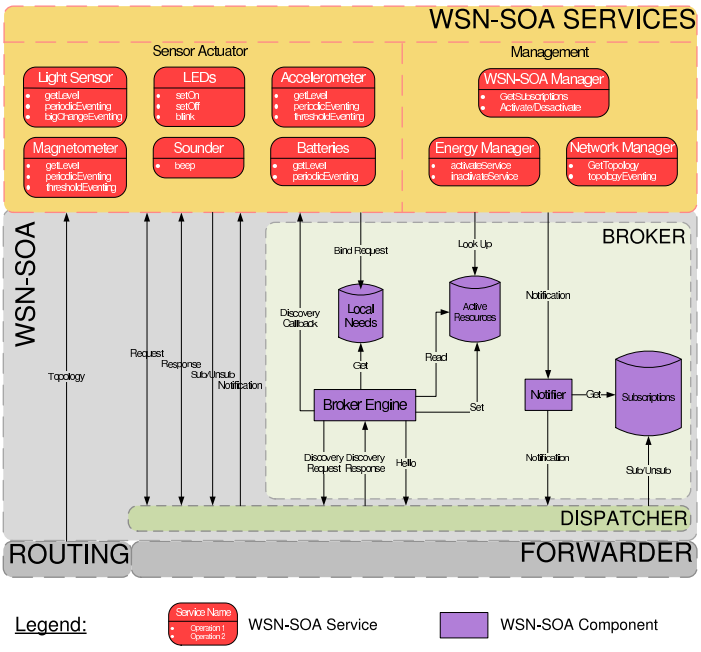

Figure 4. WSN-SOA software architecture in TinyOS.

module which is linked to the WSN-SOA core machinery. Two kinds of built-in services have been defined: *management services* and *sensor/actuator services*.

The *management services* contain vital services such as:

- **WSN-SOA Manager**: This service manages the life cycles of permanent services by allowing to activate or deactivate them, for instance to gain energy if not used. It also performs supervision activities on the core middleware components and offers operations to retrieve or delete subscriptions in the Notifier.
- **Energy Manager**: It enables to tune the operation of energy-consuming protocols and services (e.g., sleep interval of the MAC protocol, duty cycle of sensor/actuator), according to energy profiles such as *Best Performance* or *Best Autonomy*. Such profiles are meant to offer various compromises with regards to the trade-off between the reactivity, functionalities, and the lifetime of the network.
- **Network Manager**: This service performs supervision of network protocols. For now, it offers local topology information (i.e., neighboring nodes) through invocation or periodic eventing operations.

The *sensor/actuator services* offer operational services. As an illustration, we have implemented in TinyOS the following services: Light sensor, Magnetometer, Batteries, LEDs, Sounder and Accelerometer. For instance, the accelerometer sensing service exposes operations (i.e. interfaces) to the rest of the network related to the accelerometer. It offers the possibility: (1) to get the latest values of the accelerations over the x and y axis, (2) to be notified periodically of these values and (3) to be notified of these values whenever a significant change between two consecutive measurements occurs.

V. Service-oriented tasking in *wsn*-SOA

This section presents the original Tenet system and shows how we integrated it in *wsn*-SOA.

A. Tenet basic principles

Tenet allows to easily define a task and to send it to remote sensors. A task consists in a character string which defines an oriented chain of actions, called *tasklets*, that the sensor nodes execute sequentially. About a hundred tasklets have been already defined in Tenet and adding new ones is easy.

The following task can for instance be defined:

```
repeat('500') -> count(1,'0','1') -> set_leds(1)
```

This task orders the sensors to display every 500 milliseconds on the LEDs the last 3 bits of an incremented counter. The *count* tasklet initializes the variable 1 to 0 and increments it by 1 each time the tasklet is executed. The *set_leds* tasklet displays in binary on the LEDs the value of the variable 1.

The Tenet software consists of (1) a code for sensor nodes which aims at receiving, installing and executing the tasks, and (2) a code for the gateway node (e.g., a laptop) which parses the tasks and sends the matching short tasks descriptors to the sensors node. The results of tasks can be transmitted (if required) to the gateway node.

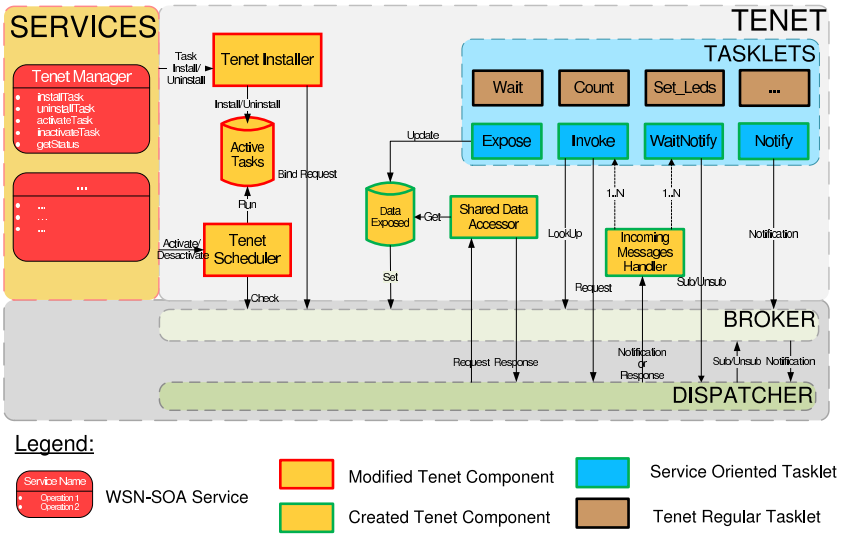

Figure 5. Software integration of Tenet in *WSN*-SOA.

B. *Tenet integration in* WSN-*SOA*

Fig. 5 presents the revised *WSN*-SOA software architecture including Tenet. A dedicated *WSN*-SOA service has been created in order to manage the Tenet layer, called *Tenet Manager*. This service is able to install or uninstall a task on-demand or to get its status (i.e., installed, running, waiting, uninstalled). This service is invoked by the gateway node to send the tasks. Since the task messages can be usually bigger than the length of a *WSN*-SOA message, fragmentation is implemented inside the service using a simple buffer. The gateway node invokes the Tenet Manager of the target node as much as it needs to transmit the entire task. It allows to send a task to a specific mote which was impossible with the original Tenet software.

C. *Service-oriented tasklets*

The specific tasklets have been specified and implemented to enable service-oriented tasking.

- **Invoke(DestAdd, SrcServID, DestServID, OpID, Attr)** This tasklet performs an invocation to a local or remote service by sending a *WSN*-SOA request message to the service DestServID. It then waits for a response message and put the response data (e.g., temperature) in the Tenet variable Attr. It takes the following parameters:
  - **DestAdd:** destination node address.
  - **SrcServID:** ID of the source service.
  - **DestServID:** ID of the invoked service.
  - **OpID:** ID of the invoked operation.
  - **Attr:** data structure with the invocation payload.
- **WaitNotify(DestAdd, SrcServID, DestServID, OpID, Attr)** This tasklet performs the subscription to a particular event and then waits for a notification. This tasklet blocks the whole execution of the task and waits for a *WSN*-SOA notification message.
- **Notify(SrcServID, OpID, Attr)** This tasklet notifies subscribers in the network of a particular event (e.g, temperature is higher than a fixed threshold). This event is identified by the couple SrcServID-OpID and contains in the payload the value of Attr. This tasklet sends a *WSN*-SOA notification Message to the Notifier.
- **Expose(ServID, Attr)** This tasklet is used to expose the result of a complex task realized by a mote (e.g, average temperature from neighboring nodes). This tasklet pushes the data to the *Data Exposed* database. The data contained in Attr is then offered through invocation by other service using the service ID ServID with the help of the Tenet Shared Data Accessor. The latter receives the *WSN*-SOA request messages, and sends data back.

To support the previous service-oriented tasklets, the following modules have been added to Tenet:

- **Shared Data Accessor:** this module exposes the data that tasklets can store in the *Data Exposed* database to the rest of nodes using a simple invocation. Each of these values has a corresponding service ID which can be used by other services to invoke a standard get operations to access this data. These *virtual* services can be discovered dynamically by other services as their availability are registered in the *WSN*-SOA active resources database.
- **Incoming Messages Handler:** Since the system is able to deal with multiple instantiations of the same tasklet, a manager to link a response or notification message to a specific tasklet instantiation is required. Each time a tasklet needs to be joined by such messages, this modules forwards the message to the appropriate handler functions. The dispatcher discriminate messages destined for Tenet using the simple rule that destination service IP is higher than 50.

Furthermore, we have added pairs of **do/while** tasklets to create more complex tasks and a **GotoIf(Attr, Cond, Value, Inc)** tasklet to support condition-based tasks. This GotoIf tasklet verifies a condition on the variable Attr and make the Tenet scheduler jumps if satisfied to a specific tasklet in the overall executed task. It takes the following parameters:

- **Attr:** value to be used to make the decision.
- **Cond:** the operator to be used (e.g.: $\leq$, $\geq$ or $=$).
- **Value:** The value to compare with Attr.
- **Inc:** The relative address in the active task of the next tasklet to execute.

VI. Scenario

To highlight the added -value of the service-oriented tasking extensions to WSN-SOA that we proposed in this paper, we present a simple scenarios in which sensors wait for a seismic vibration in order to activate a siren. Fig. 6 graphically shows the interactions between nodes.

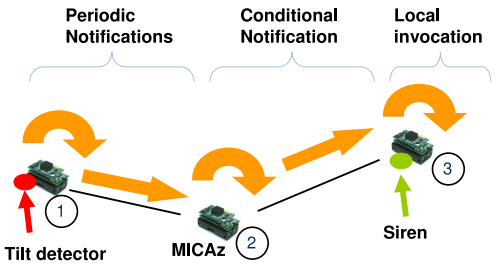

Figure 6. Case study.

To achieve this global task, the following tasks are deployed.

On Sensor 1: This sensor embeds an accelerometer service which is polled by the local task. Periodically, values are sent to mote 2 for remote processing. This solution is of course not optimal but illustrates the possibility of our solution. This translates in the following Tenet task:

```
repeat('1000')->invoke('1','52','3','1',1)
              ->notify('55','1',1)
```

Every 1000 milliseconds, sensor 1 invokes the operation 1 (i.e. getValue) of the local service 3 (i.e. the accelerometer service). It then publishes a notification to operation 1 in service 55 (the notifier will then send it to subscribers)

On Sensor 2: This sensor compares the value against a threshold and sends a notification to sensor 3 if necessary. This translates in the following Tenet task:

```
repeat('0')-> waitNotify('1','51','55','1',1)
           ->gotoIf(1,'1','2','threshold','2')
           ->notify('65','1',1)
```

Whenever sensor 2 receives an event, it checks if the value is under the threshold (e.g., 2) and sends an alert to mote 3.

On Sensor 3: Whenever this sensor receives alerts from sensor 2, it turns on a local siren. This translates in the following Tenet task:

```
repeat('0')->waitNotify('2','61','65','1',1)
           ->invoke('3','52','1','1',1)
```

Sensor 3 subscribes to the first operation of the service 55 on the mote 2 and waits for a notification. As soon as it receives a notification it executes the next tasklet which activates a small siren by invocation.

VII. Conclusion

After having reviewed state-of-the-art solutions for the dynamic deployment of tasks in WSN, this paper has laid the basis for the extension of the WSN-SOA stack to support the dynamic deployment of service-oriented tasks. The combination of these powerful mechanisms will enable the seamless creation of cooperative behaviours in WSN, thus paving the road for a whole new set of applications in the customization of *Internet-of-Things* environments.

Ongoing works focus on applying a Model-Driven Architecture approach to create a complete toolchain, dramatically enhancing the added value of the presented solution through (1) the easy definition of tasks using a graphical modeller, (2) the prior-to-deployment validation of the coherence of collaborative tasks and (3) the runtime enforcement of pre-designed workflows to discovered capabilities.

Acknowledgments

This work is supported by the EU ITEA-2 Project 2008005 "Do-it-Yourself Smart Experiences", funded by the French Ministry of Industry, and by the FP7 VITRO project, funded by the European Commission.

References

[1] J. Leguay, M. Lopez-Ramos, K. Jean-Marie, and V. Conan, "An efficient service oriented architecture for heterogeneous and dynamic sensor networks," in *Proc. IEEE SenseApp*, 2008.

[2] O. Gnawali, B. Greenstein, K. Jang, A. Joki, J. Paek, M. Vieira, D. Estrin, R. Govindan, and E. Kohler, "The tenet architecture for tiered sensor networks," in *Proc. ACM Sensys*, 2006.

[3] F. C. Delicato, P. F. Pires, and L. D. Rust da Costa Carmo, "A flexible web service based architecture for wireless sensor networks," in *Proc. International Conference on Distributed Computing Systems Workshops (ICDCSW)*, 2003.

[4] B. D. Noble, M. Satyanarayanan, D. Narayanan, J. E. Tilton, J. Flinn, and K. R. Walker, "Agile application-aware adaptation for mobility," in *Proc. ACM Symposium on Operating Systems Principles*, 1997.

[5] P. Levis and D. Culler, "Mat: a tiny virtual machine for sensor networks," in *Proc. ASPLOS-X*, 2002.

[6] C.-L. Fok, G.-C. Roman, and C. Lu, "Enhanced coordination in sensor networks through flexible service provisioning," in *Proc. of Coordination*, 2009.

[7] Yong Yao and J. E. Gehrke, "Query processing in sensor networks," in *Proc. of CIDR*, 2003.

[8] S. Madden, M. J. Franklin, J. M. Hellerstein, and Wei Hong, "Tinydb: An acqusitional query processing system for sensor networks," in *Proc. of ACM TODS*, 2005.

[9] "Universal Plug and Play (UPnP) specification," http://www.upnp.org.

[10] "Devices Profile for Web Services (DPWS) specification," http://schemas.xmlsoap.org/ws/2006/02/devprof/.

[11] J. Hill, R. Szewczyk, A. Woo, S. Hollar, D. E. Culler, and Kristofer S. J. Pister, "System architecture directions for networked sensors," in *Architectural Support for Programming Languages and Operating Systems*, 2000, pp. 93–104.

[12] "TinyOS, open-source operating system for wireless embedded sensor networks," http://www.tinyos.net.

[13] Charles Perkins and Pravin Bhagwat, "Highly dynamic destination-sequenced distance-vector routing (DSDV) for mobile computers," in *Proc. ACM SIGCOMM'94 Conference on Communications Architectures, Protocols and Applications*, 1994, pp. 234–244.