An Efficient Service Oriented Architecture for Heterogeneous and Dynamic Wireless Sensor Networks

Jérémie Leguay, Mario Lopez-Ramos, Kathlyn Jean-Marie, Vania Conan

Thales Communications

ABSTRACT

The purpose of this work is to bridge the gap between high-end networked devices and wireless networks of ubiquituous and resource-constrained sensors and actuators by extensively applying Service-Oriented Architecture (SOA) patterns. We present a multi-level approach that implements existing SOA standards on higher tiers, and propose a novel protocol stack, *WSN*-SOA, which brings the benefits of SOA to low capacity nodes without the overhead of XML-based technologies. This solution fully supports network dynamicity, auto-configuration, service discovery, device heterogeneity and interoperability with legacy architectures. As a proof-of-concept, we have studied a surveillance scenario in which the detection of an intruder, conducted within the range of a network of wireless sensors (e.g., MICAz from Crossbow), leads to the automatic triggering of tracking activities by a Linux-powered network camera and of alerts and video streams toward a control room.

Categories and Subject Descriptors

C.2.1 [**Computer Communication Networks**]: Network Architecture and Design; C.2.2 [**Computer Communication Networks**]: Network Protocols

General Terms

Design, Experimentation, Management

Keywords

Wireless Sensor Networks, Service Oriented Architectures

1. INTRODUCTION

At the hands of the increasing complexity and heterogeneity of nowadays information systems, Service-Oriented Architecture (SOA) is becoming increasingly essential. This set of architectural concepts separate functions into distinct units (called *services*), which can be distributed over a network and can be combined and reused to create business applications. Major benefits of such approach are modularity, flexibility, loose-coupling and interoperability.

These information systems are increasingly connected to various kinds of sensors and actuators networks having characteristics in processing power, battery, communication capability and availability that span over very wide ranges. Sensor networks have shown a growing interest and are still maturing as stated by Akyildiz et al. [1] and Culler et al. [5]. Sensor networks are typically used for monitoring, tracking, and controlling. Most of the sensors constituting these networks are inherently resource constrained because of their limited processing speed, storage capacity, batteries and communication bandwidth. In this work, we specifically consider *Wireless Sensor Networks* (WSN) where sensors have wireless communication capabilities.

Currently, sensor network architectures are tailored to specific applications with the intention of optimizing the sparse available resources, especially in terms of memory and battery. However, these approaches prevent the adaptation to time-specific operation requirements, the reuse of software components as well as the interoperability between different networks whose sensing range overlaps, thus boosting development costs. Such issues have already been solved in the last years in the field of enterprise information systems by SOA, which has been proven to support more effectively the requirements of business processes and users.

The main contribution of this paper is the proposition and implementation of a multi-level SOA-based architecture for heterogeneous and dynamic wireless sensor networks. This architecture has been designed following two main objectives:

- Extending SOA capabilities to devices with limited processing power, battery capacity, communication bandwidth and availability. We propose to accomplish this extension up to tiny wireless sensors such as the MICAz from Crossbow.
- Facilitating deployment of network entities at all levels by providing auto-configuration facilities both at the network and the service levels. In the surveillance scenario this work considers, sensors and actuators are rapidly deployed at some locations to guard a given area. As a consequence, auto-configuration and service discovery features from all the networked entities

are required so that the system can stay in operation without any human intervention.

In the architecture this work introduces (see Sec. 2), we use for large-capacity deployable wireless sensors (e.g., IP network camera, gateway to networks of tiny wireless sensors such as the MICAz from Crossbow) the Devices Profile for Web Services (DPWS) [7] standard which consists in a Web Services-based device communication framework. While for the tiny wireless sensors (e.g., MICAz from Crossbow), we define (see Sec. 3) the *WSN*-SOA stack which enables the use of the SOA paradigm. To make the *WSN*-SOA stack interoperable with DPWS, we have specified (see. Sec. 4) a software stack running on the Crossbow Stargate sensor network gateway which implements a publish/subscribe communication channel. Finally, we present the demonstration of our implementation (see Sec. 5) and position our work with regards to related research efforts (see Sec. 6).

2. MULTI-LEVEL SERVICE ORIENTED ARCHITECTURE

This section first provides background knowledge on SOA technologies. It subsequently introduces our multi-level architecture and presents the DPWS standard that we propose to use for some of the deployable wireless sensors that we have considered. Finally, it discusses the limitations of the available SOA technologies when applied to highly constrained nodes.

2.1 SOA technologies

SOA is a distributed computing paradigm in which business functionality is provided by autonomous systems called *services*, which are exposed in a network infrastructure through well-defined interfaces. This allows building complex yet flexible systems as well as reusing application logic through the composition of services.

SOA is a concept which is not tied to a particular technology. However, Web Services are currently the preferred framework to deliver interoperable SOA. In Web Services-based SOAs, the contract is defined by a WSDL (Web Services Description Language) document, which stipulates how service consumers can bind to a service producer by exchanging messages using a defined XML (Extensible Markup Language) grammar. The use of XML opens a new world of opportunities thanks to the available tools (XSLT, XPath, XML Security...).

2.2 Architecture overview

The idea behind this multi-level architecture is to turn all the kinds of sensors into reusable resources and to enable distributed cooperation between them via auto-configuration features. In this sensor-actuator architecture, nodes are classified in three different classes depending mainly on their resources and network connectivity.

- **Full capacity nodes**: These entities have high availability and do not have processing power or battery issues. They could be always-online servers or client applications. One could refer to them as regular *internet nodes*.
- **Limited capacity nodes**: These devices can be limited in terms of storage, battery, processing power or communicating capabilities but can still perform complex tasks and host operating systems such as Windows CE or Linux. To be more concrete, their capacity is in the order of that of a PDA (Personal Digital Assistant). However, in some cases they will only be connected when required, as opposed to *full capacity nodes*.
- **Low capacity nodes**: Such devices have extremely low capacity. They have few kilobytes of RAM and are equipped of a microcontroller. As they suffer from battery issues, they often use low power wireless interfaces such as IEEE 802.15.4. The Crossbow MICAz *motes* is the reference hardware that we have considered.

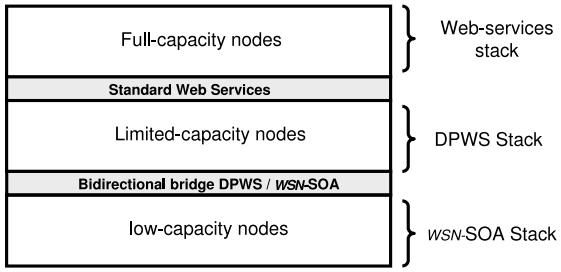

Figure 1: Device capacity Vs. SOA protocols.

Fig. 1 shows the different SOA-based protocol stacks that we propose to use with regards to device capacity. On *full capacity nodes*, usual Web Services stacks (e.g., Axis [2]) and Enterprise Service Buses (ESB) are used. For *limited capacity nodes*, we propose to use DPWS, which is perfectly adapted to dynamic and constrained devices as well as compliant to Web Services standards. A the lowest level, since to our knowledge no SOA-compliant service stacks are available, we propose the *WSN*-SOA further detailed in Sec. 3. Interoperability between Web Services stacks and the *WSN*-SOA is ensured by a bidirectional gateway presented in Sec. 4.

2.3 DPWS: SOA for limited-capacity nodes

Thanks to their affordability and small form factor, the networked devices previously defined as *limited capacity nodes* meet an increasingly important need in machine-to-machine communications where evolving scenarios require dynamic deployment and reconfiguration. As such, their usage profile is often closer to plug-and-play peripherals than to critical business application servers. That is why, although they are ready to implement Web Services technologies efficiently, some additional requirements arise when trying to integrate such kind of devices in a SOA infrastructure.

To overcome this issue, some major industrial and software actors defined the Devices Profile for Web Services (DPWS) specification [7]. DPWS, first published in May 2004 and revised in October 2005, defines a minimal set of Web Services (WS-*) standards and implementation constrains to enable dynamic discovery and event capabilities for Web Services.

This technology has gained considerable relevance, and as of today several mature and open-source implementations are available for C and Java [8]. It is also part of the latest .NET framework as well as being natively built-in in Microsoft Vista.

2.4 Web Services limitations and efforts

One of the main criticisms of Web Services in the world of embedded computing is performance. Indeed, XML format is extremely verbose and its processing consumes significant amounts of time and memory, not to mention bandwidth usage when it is used for network protocols. In the last years, major optimization efforts have been made in order to deliver Web Services to real-time environments and constrained devices:

- **XML parsing**: different techniques such as StAX (Streaming API for XML) [21] or XML Schema specific parsing [3] have significantly reduced the resources required by XML deserialization (generally more time-consuming than serialization).
- **XML binarization**: several specifications have been proposed to define a backwards-compatible compact representation of the XML Infoset [26] in a binary optimized format that simultaneously optimizes performance while reducing bandwidth. The most promising ones are the Fast Infoset format [11] proposed by Sun or the Efficient XML Interchange format (EXI) [10] retained by the World Wide Web Consortium (W3C) for its currently working draft specification.

In both cases, XML Schema information is being used – when available– as knowledge shared between peers to gain performance and compactness. Today, existing stacks like gSOAP [13] deliver soft real-time Web-Services with little memory footprint (around 100 KB of code) to various environments such as embedded GNU/Linux, VxWorks or Windows Mobile. However, devices such as *low capacity nodes* in the previously described architecture will never meet the Web Services requirements.

3. SOA FOR LOW CAPACITY NODES

As targeted devices in the category of *low capacity node* are not able to process XML messages, we propose the *WSN*-SOA which consists in a simple protocol and software architecture that reproduces the architectural concepts and information exchanges of regular SOA implementation. The main goal is to make sensors very limited in capacity able to host services, discover the services of the others, announce their services, invoke services and subscribe to events.

This section presents the *WSN*-SOA as well as its implementation on the open-source operating system TinyOS [15, 22]. We developed the *WSN*-SOA for the Crossbow MICAz [16] sensors equipped with the MTS310[17] 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.

3.1 Message format

The messages exchanged within the *WSN*-SOA follow the message format depicted in Fig. 2. *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). In our TinyOS implementation, these *multi-hop messages* are themselves embedded in standard TinyOS [15, 22] messages, called *TOSMsg*. These messages are used to enable communication between any two devices at the link level. More information about header fields can be found in the TinyOS documentation. One should note that the envelops in which *WSN*-SOA messages are embedded can be easily changed depending on the operating system or routing scheme used.

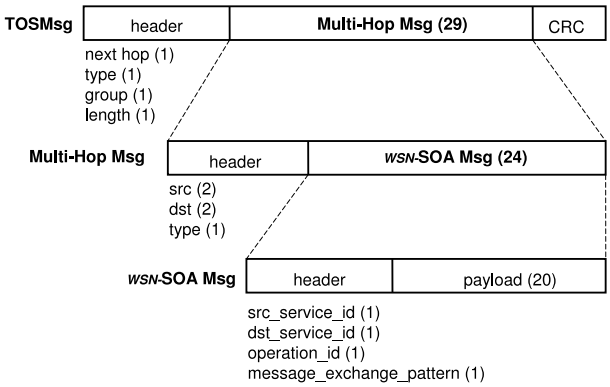

Figure 2: *WSN*-SOA message format in TinyOS.

WSN-SOA messages contain the following fields:

- **src_service_id**: This identifier allows to address the service which initiates the information exchange on the source node.
- **dst_service_id**: This field identifies the service on the destination node. The fact that we distinguish source and destination service identifiers enables services of several kinds to communicate with each other.
- **operation_id**: 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.
- **message_exchange_pattern**: The Message Exchange Pattern (MEP) field defines the semantics of message exchanges. 6 different types have been defined: 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 an answer

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 be sure that a subscription request have been processed.

- **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 could have been envisioned, but we observed that in most of the cases the exchanged data is very small (e.g., an integer). In our TinyOS implementation, the payload structures are clearly defined by nesC structures.

3.2 Software architecture

Fig. 3 presents the *WSN*-SOA software architecture as been implemented in TinyOS. We can see that the *WSN*-SOA protocol and service stack rest upon the IEEE 802.15.4 interface and a queuing management module which has been implemented to handle incoming and outgoing packets. *WSN*-SOA packets enter in the *WSN*-SOA dispatcher which forwards them to the appropriate services.

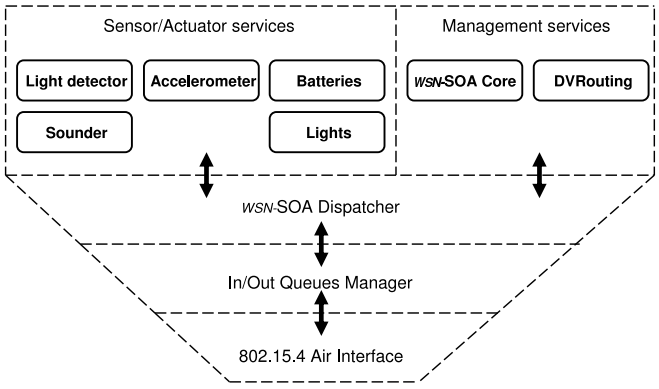

Figure 3: *WSN*-SOA software architecture in TinyOS.

The *WSN*-SOA has been implemented in TinyOS with an extensive use of modules. Indeed, every service is a TinyOS module which is linked to the *WSN*-SOA core machinery. Two kinds of services have been defined: *management services* and *sensor/actuator services*.

The *management services* contain vital services such as:

- ***WSN*-SOA Core**: This key service is mainly responsible for announcing the services hosted by the node using HELLO messages. These messages are sent in response to discovery requests or sent in a voluntary and periodic fashion when the node has just appeared in the network for auto-configuration purpose.
- **DVRouting**: This service corresponds to the multihop routing protocol that we have defined and implemented in TinyOS. As we wanted to demonstrate cooperation between any possible pair of nodes, this protocol is unicast and very similar to DSDV [19]. This kind of protocol does not scale and is not very efficient in wireless sensor networks but as it has been implemented as a service, it can be easily replaced.

The *sensor/actuator services* offer operational services. As an illustration, we have implemented in TinyOS the following services:

- **Accelerometer**: This 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. All the durations and the thresholds involved in these operations can be modified.
- **Light sensor and battery gauge**: These two sensing services expose the same kind of interfaces as the accelerometer, but for the light sensor and the battery gauge.
- **LEDs**: This actuator service provides control over the 3 LEDs present on the MICAz boards. They can be switched off/on or blink.
- **Sounder**: This actuator service controls the sounder of the MTS310 boards and allows to make the mote beep.

4. BRIDGING SOA WORLDS

This section presents the software stack we propose to run on nodes that bridge between *full-capacity nodes* and *low capacity nodes* in our service oriented architecture. The gateway, or *bridge*, has a key mission in our architecture: to connect the *WSN*-SOA and DPWS worlds. Its main functions are to:

- **Enable *WSN*-SOA to DPWS service translation**. This is done through hosted service-specific proxies which can be automatically generated from specific TinyOS module headers. This fine-grained operations enable, for instance, to trigger an actuator in a specific node. This provides a feedback mechanism to the network which is able to react to events and thus enhance sensing capabilities.
- **Provide a high-level interface** which hides the complexity of the underlying network. In a context in which hundreds or thousands of these nodes can be deployed, a one-to-one SOA message translation offers poor control possibilities. By providing meaningful interfaces, the network can be seen as a single *macrosensor*. The chosen data dissemination mechanism is publish/subscribe, which will be explained in Sec. 4.2.
- **Create an extension framework** supporting the deployment of data processing mechanisms such as data fusion algorithms or scheduled tasks, as well as new Web Services interfaces. These software components can be remotely installed or updated without requiring a reboot.

The bridge alone is sufficient for the autonomous functioning of the WSN-SOA network. Of course, more than one bridge may be present in a single network to ensure redundancy and provide load balancing.

4.1 Software architecture

The flexibility of this critical part of the solution is enabled by a precise software architecture, depicted in fig. 4. The main modules that have been implemented are the following:

- **Mote connector**: implements the platform-specific driver required to use the ZigBee radio. It provides two low-level services allowing packet transmission and reception.
- **Mote routing**: manages network layer functionality. It implements the routing protocol used in the rest of the wireless sensor network. Routing table information is exposed to the upper layers as well as node discovery.
- ***WSN*-SOA manager**: provides service layer functionality. Its role is to discover new services, keep an updated service registry as well as offering an API for service consumption.
- ***WSN*-SOA/DPWS bridge**: uses the WSN-SOA manager API to expose discovered WSN-SOA services and provide consumption mechanisms through DPWS. Any other similar layer can be plugged besides this one to bridge the WSN-SOA world to another high level infrastructure.
- **DPWS core**: is the stack runtime that allows DPWS discovery, messaging and event notification.

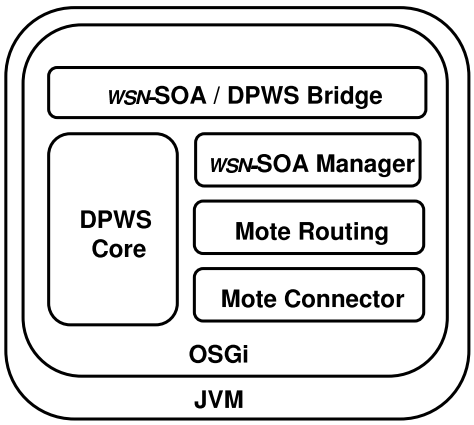

Figure 4: Gateway software architecture.

All these modules have been implemented as *bundles* within an OSGi [18] framework in Java. Each bundle is a software component that can be deployed, loaded, started or stopped dynamically, which means that no restart is required to update the software in the bridge.

This software has been implemented to run on Crossbow Stargate sensor network gateways. The Stargate is an embedded device equipped with an Intel PXA255 Processor running at 400MHz and has 64MB of SDRAM. It offers a large number of interfaces such as a PCMCIA slot where we plugged a Wi-Fi card, a Compact Flash memory card, an Ethernet port and a serial port where a MICAz is hosted to provide Zigbee connectivity.

The MICAz node plugged in the Stargate is only used to provide a ZigBee network interface. The TOSBase TinyOS application makes it work as a simple forwarder from the radio link to the serial interface.

4.2 Publish/Subscribe communication

Despite the fact that the bridge can offer one-to-one service translation, we propose to use a more efficient and relevant data dissemination mechanism based on the publish/subscribe pattern on top of our multi-level SOA.

Publish/subscribe is an asynchronous messaging paradigm in which data sources (publishers) are not programmed to send their messages to specific data sinks (subscribers). Instead, they announce the availability of a certain class of data, which is only delivered to the subscribers that express their interest in it, without the need for publishers and subscribers to be aware of the existence of each other. This loose-coupled scheme offers a greater scalability and adaptation to dynamic network topologies, both of which are important challenges often faced in wireless sensor networks.

Our publish/subscribe solution uses a topic-based message filtering system, in which producers publish messages to named logical channels called *topics*. Subscribers receive all messages published to the topics to which they subscribe. Topics can be created ad-hoc when they become available, and they are organised hierarchically, as proposed by WS-Topics [25]. This means that subscribing to a topic also implies a subscription to all child topics, even if they don't exist yet. Some wildcard functions are also available in topic subscription. For instance, topic */motes/*/acceleration/tilt* corresponds to the tilt event of the acceleration service of any known mote.

The complex brokering and filtering functions are handled by the bridge, which exposes available topics through Web Services asynchronous event sources using WS-Eventing [24], as defined in DPWS. This mechanism enables transparent cross-network data dissemination: any node, either at the WSN-SOA level or at the DPWS level, can be a publisher or a subscriber.

This powerful framework has many benefits:

- **Providing energy-efficient data dissemination**: information is only transmitted when it is required by at least one consumer. This avoids polling, sending periodic updates or receiving unused event notifications, thus saving energy.
- **Enabling flexible in-network intelligence**. For instance, when a new topic provides information about a sensor's neighbouring area, it can be automatically subscribed to obtain more accurate data through fusion mechanisms.

- **Fostering standards-based multi-level Event-Driven Architectures (EDA)**. EDA has emerged as a complementary approach to address the traditional lack of asynchronous support of SOA. Our solution is the first to provide standard Web Services-based asynchronous mechanisms to access sensor/actuator networks. For example, this gives the means to seamlessly trigger alarms in any Enterprise Service Bus (ESB) from a tiny sensor node.

Fig. 5 depicts the different phases in event subscription. When a new sensor node joins the network, (1) it sends information about it and its hosted services to the rest of the nodes. The bridge processes the message and (2) creates the corresponding topics on the publish/subscribe channel. In this particular case, the node 5 hosts 2 services (acceleration and lightsensor) with 2 event sources each (one for significant changes and one for periodic sampling). When a DPWS client (3) discovers and subscribes to one of the topics, the bridge reacts by (4) requesting the subscription to the appropriate node(s). From that moment and until unsubscription is requested, the client will be notified by a WS-Eventing asynchronous message of any significant change in the accelerometer (*tilt* event) of node 5.

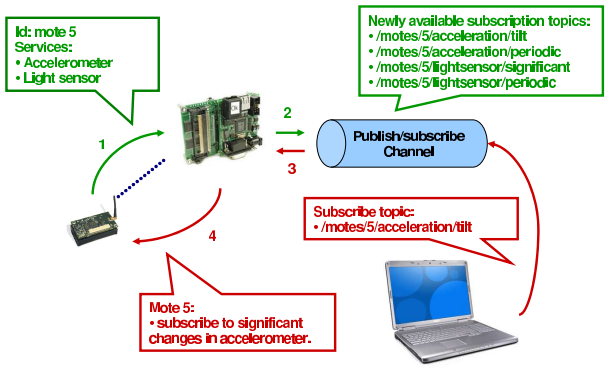

Figure 5: Publish/Subscribe topics on *WSN*-SOA

5. PROOF-OF-CONCEPT

This section presents the demonstration we set up to highlight the benefits of our multi-level service-oriented architecture.

5.1 Demonstration setup

We have considered a surveillance scenario in which sensors forming an ad-hoc network have been deployed to detect intrusions via seismic vibrations. These sensors communicate through the Crossbow Stargate gateway with the entities that have bigger capacities and that are interconnected by a Wi-Fi ad-hoc network, such as an Axis 213 PTZ camera and a laptop running a Command and Control (C2) application.

Fig. 6 presents graphically the scenario and the sequence of information that occurs when an intrusion is detected. In more detail, once the routing process has converged and nodes are able to exchange information, the SOA-based service stacks (i.e., DPWS on the Stargate and the laptop, *WSN*-SOA on the MICAz motes) discover respectively their services and the available topics from the sensor network are advertised in the publish/subscribe channel. At this moment, the control unit requests the subscription to some default topics such as the tilt events of the accelerometer service of the motes. After this self-initialisation phase, whenever a mote detects a significant change in acceleration, a notification is sent to the bridge which then publishes an event within the topic /motes/i/acceleration/tilt, i being the identifier of the source node. The event is received both by the control unit and by the camera which automatically points in the direction of mote i. The video streaming to the control unit allows an operator to see what is happening and potentially react by turning on an actuator such as a projector or a siren.

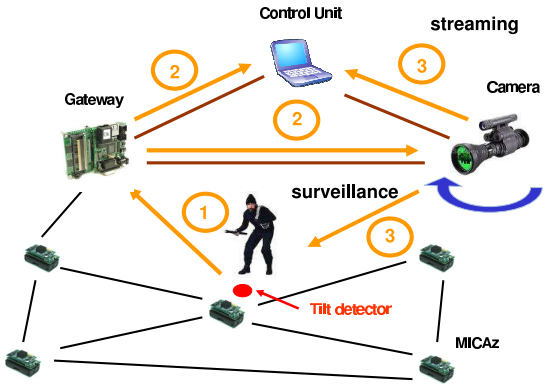

Figure 6: Surveillance scenario.

5.2 Command and Control unit

More specifically, the Command and Control (C2) unit that we have developed allows to manage all the sensors and actuators within the scenario. Fig. 7 shows 4 screenshots from the application. The C2 unit offers an always-visible tree view showing all the devices discovered and their services. The C2 unit allows (1) to locate the different entities on a map and to see the network connectivity between them (top-left); (2) to plot graphs showing the evolution of measurements (acceleration, light, etc.) over time and to send events to actuators through the publish/subscribe channel (top-right); (3) to watch the video streamed from the camera, move it and set preset viewing positions for every monitored zone (bottom-left); and (4) to subscribe to topics available in the publish/subscribe channel (bottom-right).

5.3 Performance measurements

To provide insights on the performance of our implementation work, we have first measured the time needed by the *WSN*-SOA network to be in operation. This includes the convergence time of the distance-vector ad-hoc routing protocol and the time required by the gateway to be aware of all the services that are available on the motes. Then, we measured the time required for a subscription operation to all nodes such as the one corresponding to the topic */motes/*/acceleration/tilt* to be completed. Fig. 8 shows these two measurements for a number of MICAz motes varying between 2 and 10.

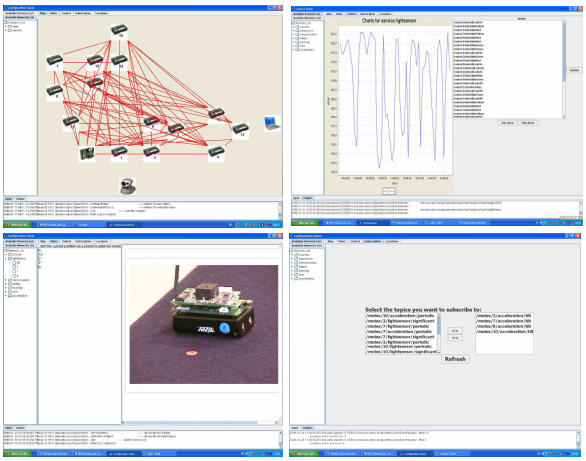

Figure 7: Command and Control (C2) unit GUI.

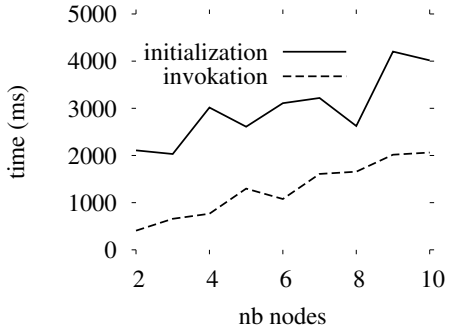

Figure 8: Convergence times.

As one can see, the time required for each of the operations increases with the number of nodes deployed. The performance level depends on a large set of parameters that would need to be engineered for a real deployment such as the beaconing periods in the ad-hoc network protocol or the frequency at which HELLO messages involved in the service discovery are sent. The tuning of all these parameters and the potential adaptive mechanisms that would be needed to optimize performance are left for future work.

6. RELATED WORK

Even in the WSN world, the idea of encapsulating node functionality in *services* following the SOA approach is not new. Delicato et al. [6] identified the potential of Web Services-based solutions for wireless sensor networks.

More recently, the ZigBee [27] specification defined an application layer introducing the concept of hosted services (called *application objects*) which are bound to *endpoints* on each node. One of them is the ZigBee Device Object (ZDO), which manages node and service discovery amongst other things. This is somewhat similar to our approach, which could definitely be built on top of the ZigBee application stack if there were mature open-source implementations. The main difference is the introduction of the above-mentioned publish/subscribe mechanism, that allows applications to abstract from the underlying burden of nodes and services, and focuses on the content of the data itself, thus better adapting to network dynamicity.

The missing element to ease the access to the ZigBee world was created by Archronix, which offers a gateway [12] that allows DPWS clients (such as Windows Vista) to communicate directly with ZigBee-compliant devices, as long as they are recognized by the gateway.

Souto et al.[20] defined the MIRES middleware for TinyOS, allowing applications running on sensor nodes to communicate in a publish/subscribe way. This is complementary work to ours as it is not interoperable with regular Web Services and that our architecture implements publish/subscribe communications also on the sensor network gateway.

Wolff et al.[23] proposed the μSOA which enables to access services running on sensor nodes through a proxy. This approach simply focuses on one-to-one service translations between regular Web Services and embedded services.

The Sensor Web Enablement (SWE) [4] standard achieved by the Open Geospatial Consortium (OGC) defines the *Open Sensor Web Architecture* (OSWA) to provide SOA-based access to and management of sensors. In addition, a set of specifications have been proposed such as *SensorML*, which consists in an XML grammar for the exchange of sensor information, or *Observation & Measurement* (O&M), which consists in another XML schema for the encoding of observations reported by sensors. The objective of these standards is mainly to connect always-on sensors disseminated all around the world and accessible through the internet with computer grids where applications such as weather forecast or earthquake monitoring systems are running.

The architecture that we propose in this paper is more suitable for rapid deployments or changing scenarios, as it supports the dynamicity in the connectivity between networked sensors as well as their heterogeneity. Furthermore, our approach promotes and distributes collaboration and aggregation treatments at the very edge of the network.

The IEEE 1451 standard was created to make easier for transducer manufacturers to interface those sensing devices to networks and systems by incorporating existing networking technologies. Emil et al. [9] proposed a framework to enable Web Service interfaces on IEEE 1451 sensors.

7. CONCLUSION AND FUTURE WORK

This paper has proposed a multi-level service-oriented architecture for sensor networks. This architecture bridges the gap between devices having very different capacities and fully handles network dynamicity by providing auto-configuration features at both network and service level. We have highlighted the use of the DPWS standard for *limited-capacity* devices (e.g., IP camera, PDA, sensor network gateways, large deployable sensors) and we introduced the *WSN*-SOA suite which enables SOA-based communications in networks of *low-capacity* sensors (e.g., MICAz motes). Instead of one-to-one service translation between the DPWS and *WSN*-SOA worlds, we propose to use a more efficient and relevant way of communication based on high-level interfaces and publish/subscribe event notifications using *topics*. The

WSN-SOA service stack has been implemented in TinyOS and the *WSN*-SOA gateway, which also runs a DPWS stack, has been implemented in Java as a set of OSGi bundles. Albeit the code has not been released in open-source, we are willing to make it available upon request.

Future work along these lines include studies on deployable services to potentially offer on-the-fly data aggregation services that would run on the motes or on the sensor network gateway. To do so, we envision to consider operating systems like SOS [14] that support loadable modules and dynamic memory management and to deeper use OSGi features on the gateway. Furthermore, we plan to integrate in the *WSN*-SOA stack new kinds of energy-saving mechanisms to enable low-power operations and networking features to improve scalability. Finally, work remains to be done around QoS to handle for instance priorities among *WSN*-SOA services.

Acknowledgments

This work has been supported by Thales Communications, the European ITEA SODA and IST MORE projects, and the French RNRT SVP project. We especially thank Damien Lavaux, Stefan Michaelis and Jens Schmutzler for all the useful discussions that we had previous to this work within the IST MORE project.

8. REFERENCES

[1] I. Akyildiz, W. Su, Y. Sankarasubramaniam, and E. Cayirci. A survey on sensor networks. *IEEE Communications Magazine*, 40(8):102–114, 2002.

[2] Apache Axis2. http://ws.apache.org/axis/.

[3] K. Chiu and W. Lu. A Compiler-Based Approach to Schema-Specific XML Parsing. In *Proc. First International Workshop on High Performance XML Processing*, 2004.

[4] X. Chu and R. Buyya. Service oriented sensor web. *Sensor Network and Configuration: Fundamentals, Techniques, Platforms, and Experiments. N. P. Mahalik (ed).*, pages 51–74, 2007.

[5] D. Culler, D. Estrin, and M. Srivastava. Guest editors' introduction: Overview of sensor networks. *IEEE Computer*, 37(8):41–49, 2004.

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

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

[8] Open-source DPWS implementation. https://forge.soa4d.org/.

[9] F. S. Emil and L. Ramiro. A Web-Services Framework for 1451 Sensor Networks. In *Proc. of IEEE Instrumentation and Measurement Technology Conference (IMTC)*, 2005.

[10] Efficient XML Interchange (EXI) Format 1.0. http://www.w3.org/TR/exi/.

[11] ITU X.891 standard - Generic applications of ASN.1: Fast infoset. http://www.itu.int/rec/T-REC-X.891-200505-I/en.

[12] D. R. Flickinger. Bridging the Automation/IP Gap. White paper. Archronix. http://www.cepro.com/asset/6037.pdf, 2006.

[13] gSOAP: SOAP C++ Web Services. http://gsoap2.sourceforge.net.

[14] C. Han, R. K. Rengaswamy, R. Shea, E. Kohler, and M. Srivastava. Sos: A dynamic operating system for sensor networks. In *Proc. ACM MobiSys*, 2005.

[15] J. Hill, R. Szewczyk, A. Woo, S. Hollar, D. E. Culler, and K. S. J. Pister. System architecture directions for networked sensors. In *Architectural Support for Programming Languages and Operating Systems*, pages 93–104, 2000.

[16] xBow MICAz: Wireless measurement system. http://www.xbow.com/Products/Product_pdf_files /Wireless_pdf/MICAz_Datasheet.pdf.

[17] xBow MTS310 sensor board. http://www.xbow.com/Products/Product_pdf_files /Wireless_pdf/6020-0047-01_B_MTS.pdf.

[18] OSGi Alliance. About the OSGi Service Platform - Technical Whitepaper Revision 4.0. http://www.osgi.org/documents/.

[19] C. Perkins and P. Bhagwat. Highly dynamic destination-sequenced distance-vector routing (DSDV) for mobile computers. In *Proc. ACM SIGCOMM'94 Conference on Communications Architectures, Protocols and Applications*, pages 234–244, 1994.

[20] E. Souto, G. G. aes, G. Vasconcelos, M. Vieira, N. Rosa, and C. Ferraz. A message-oriented middleware for sensor networks. In *Proc. 2nd Intl Workshop Middleware for Pervasive and Ad-Hoc Computing (MPAC 04)*, 2004.

[21] JSR173 (Streaming API for XML - StAX). http://jcp.org/en/jsr/detail?id=173.

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

[23] A. Wolff, J. Schmutzler, S. Michaelis, and C. Wietfeld. Network-centric middleware for service oriented architectures across heterogeneous embedded systems. In *Proc. IEEE International EDOC conference, Workshop on Middleware for Web-Services*, 2007.

[24] WS-Eventing Specification. http://www.w3.org/Submission/WS-Eventing/.

[25] WS-Topics 1.3 Specification. http://docs.oasis-open.org/wsn/wsn-ws_topics-1.3-spec-os.pdf.

[26] XML Information Set (Second Edition). http://www.w3.org/TR/xml-infoset/.

[27] The zigbee specification, revision q4/2007. http://www.zigbee.org.