Service Oriented Architecture for Heterogeneous and Dynamic Sensor Networks

Jérémie Leguay, Mario Lopez-Ramos, Kathlyn Jean-Marie, Vania Conan
Thales Communications
{firstname.name}@fr.thalesgroup.com

ABSTRACT
The purpose of this demo is to foster a multi-level service oriented architecture for sensor networks that fully supports network dynamicity, auto-configuration, service discovery, and interoperability with legacy infrastructures. We consider a surveillance scenario in which the detection of an intruder, conducted within the range of a network of sensors, automatically triggers tracking activities.

Categories and Subject Descriptors
C.2.1 [**Computer Communication Networks**]: Network Architecture and Design

General Terms
Design, Experimentation

Keywords
Wireless Sensor Networks, Service Oriented Architectures

1. INTRODUCTION
The increasing complexity and heterogeneity of nowadays information systems have called for Service Oriented Architecture (SOA). 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.

The main contribution of this demo is then the proposition and implementation of a multi-level SOA-based architecture for heterogeneous sensor networks. This architecture has two main goals. The first one is to extend SOA capabilities to devices with low capacities in processing power, battery, communication capabilities and availability. The second is to ease the deployment of network entities at all levels by providing auto-configuration mechanisms both at the network and service levels.

2. SERVICE ORIENTED ARCHITECTURE
The service-oriented architecture we propose classifies nodes in three different classes depending mainly on their available resources and network connectivity:

- **Full capacity nodes**: These entities have high availability and do not have processing power or battery issues. They could be critical always-online servers or client applications.
- **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.
- **Low capacity nodes**: Such devices have extremely low capacity. They have few kilobytes of RAM, are equipped of a MicroControler and often use low power wireless interfaces such as IEEE 802.11.4.

On the full capacity nodes, information is exchanged using common Web Services stacks (e.g., Axis). On the limited capacity nodes, we propose to use DPWS [1], which is perfectly adapted to dynamic and constrained devices and compliant to Web Services standards. A the lowest level, since to our knowledge no SOA-compliant service stacks were available, we propose to use a SOA-based solution (see Sec. 3). The interoperability between Web Services stacks and the sensor network is ensured by a specific gateway (see Sec. 4).

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 [3]. 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.

3.1 Message format

The messages exchanged within the *WSN*-SOA follow the message format depicted in Fig. 1. *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 [3] 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.

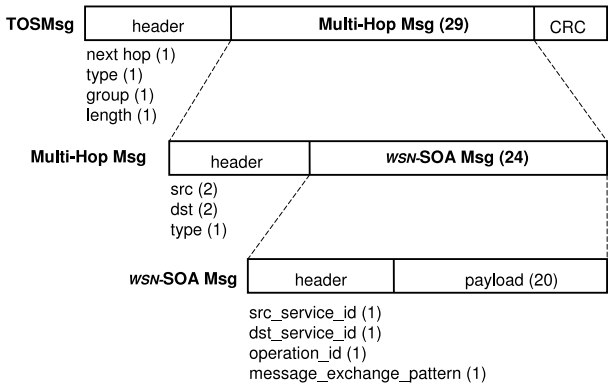

Figure 1: *WSN*-SOA message format in TinyOS.

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. 2 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.

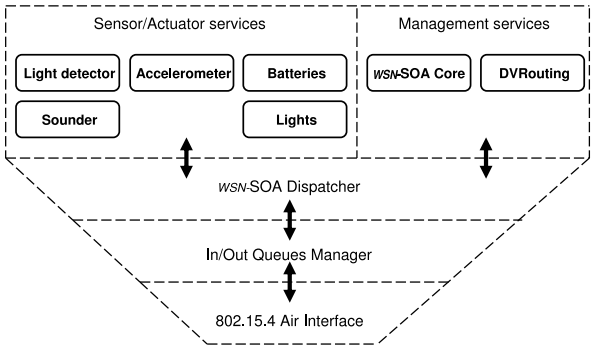

Figure 2: *WSN*-SOA software architecture in TinyOS.

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 multi-hop routing protocol that we have defined and implemented in TinyOS.

The *sensor/actuator services* offer operational services. As an illustration, on of the services that we have implemented

in TinyOS concerns the accelerometer. This sensing service 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.

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 *macro-sensor*. 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.

4.1 Software architecture

The flexibility of this critical part of the solution is enabled by a precise software architecture, depicted in Fig. 3. 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.

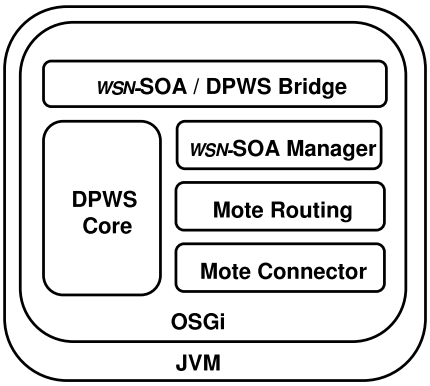

Figure 3: Gateway software architecture.

All these modules have been implemented as *bundles* within an OSGi [2] framework in Java. Each bundle is a software component that can be deployed, loaded, started or stopped dynamically.

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.

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. 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 gateway, which exposes available topics through Web services asynchronous event sources using WS-Eventing. 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 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.

5. PROOF-OF-CONCEPT

5.1 Demonstration Setup

We considered a surveillance scenario in which sensors forming an ad hoc network have been deployed to detect intrusions via seismic vibrations. These sensors communicate using an Wi-Fi ad hoc network through the Crossbow Stargate gateway with an Axis 213 PTZ camera and a laptop running a Command and Control application.

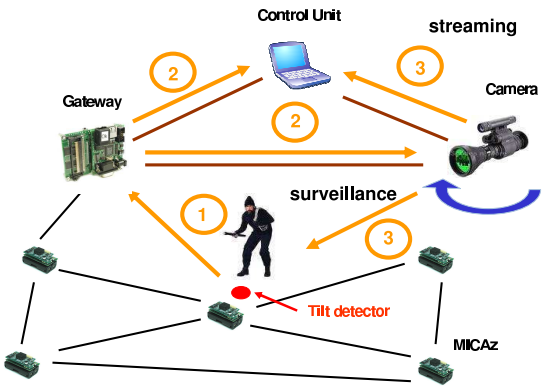

Figure 4: Surveillance scenario.

Fig. 4 presents the scenario that we considered and the sequence of events that take place when an intrusion is detected. Once routing has converged and nodes are able to exchange information, the SOA-based service stacks (both at the DPWS level and at the mote level) advertise their hosted services and topics which are then known to the rest of the network. The control unit then requests subscriptions to the topic providing tilt events from the accelerometer service of the motes. Whenever a mote detects a significant change in acceleration, a notification is sent through the bridge to the control unit and the camera, which react accordingly.

5.2 Command and Control Unit

More specifically, the Command and Control unit that we have developed allows to manage all the sensors and actuators within the scenario. Fig. 5 shows 4 screenshots from the application. The application 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 push an available event into the publish/subscribe pipe (top-right), (3) to watch the video streamed from the camera and set the preset positions corresponding to every monitored zone (bottom-left), and (4) to subscribe to topics available in the publish/subscribe pipe (bottom-right).

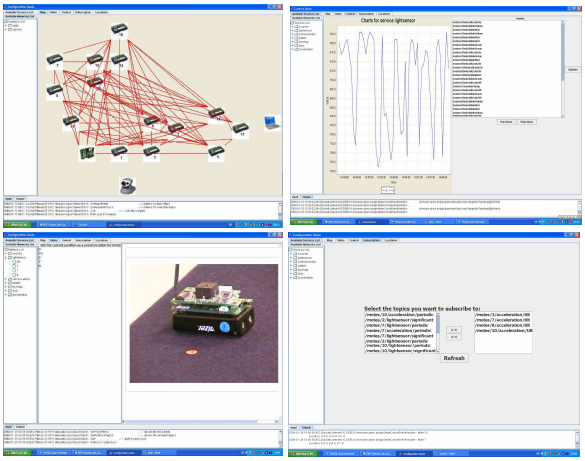

Figure 5: Command and Control unit GUI.

6. CONCLUSION

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 handle network dynamicity by providing auto-configuration features at both network and service level.

Acknowledgments

This work has been supported by Thales Communications, the European ITEA SODA and IST MORE projects, and the French RNRT SVP project.

7. REFERENCES

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

[2] OSGi Alliance. About the osgi service platform - technical whitepaper revision 4.0. Available at: `http://www.osgi.org/documents/`, 2005.

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