Gateway architectures for service oriented application-level gateways

Citation for published version (APA):

Bosman, R. P., Lukkien, J. J., & Verhoeven, R. (2011). Gateway architectures for service oriented application-level gateways. *IEEE Transactions on Consumer Electronics*, 57(2), 453-461.
<https://doi.org/10.1109/TCE.2011.5955179>

DOI:

[10.1109/TCE.2011.5955179](https://doi.org/10.1109/TCE.2011.5955179)

Document status and date:

Published: 01/01/2011

Document Version:

Publisher's PDF, also known as Version of Record (includes final page, issue and volume numbers)

Please check the document version of this publication:

- A submitted manuscript is the version of the article upon submission and before peer-review. There can be important differences between the submitted version and the official published version of record. People interested in the research are advised to contact the author for the final version of the publication, or visit the DOI to the publisher's website.
- The final author version and the galley proof are versions of the publication after peer review.
- The final published version features the final layout of the paper including the volume, issue and page numbers.

[Link to publication](#)

General rights

Copyright and moral rights for the publications made accessible in the public portal are retained by the authors and/or other copyright owners and it is a condition of accessing publications that users recognise and abide by the legal requirements associated with these rights.

- Users may download and print one copy of any publication from the public portal for the purpose of private study or research.
- You may not further distribute the material or use it for any profit-making activity or commercial gain
- You may freely distribute the URL identifying the publication in the public portal.

If the publication is distributed under the terms of Article 25fa of the Dutch Copyright Act, indicated by the "Taverne" license above, please follow below link for the End User Agreement:

www.tue.nl/taverne

Take down policy

If you believe that this document breaches copyright please contact us at:

openaccess@tue.nl

providing details and we will investigate your claim.

Gateway Architectures for Service Oriented Application-Level Gateways

Remi Bosman, Johan Lukkien, *Member IEEE*, and Richard Verhoeven, *Member IEEE*

Abstract — *In this paper we present lightweight and generic application level gateways for connecting service oriented sensor networks to service oriented standards in the consumer electronics domain. We study the trade-offs between interface richness, deployment granularity and overhead in discovery protocols and we present how the varying interaction patterns on both sides of the gateway can be efficiently mapped onto each other¹. As a result we obtain three different application-level gateways architectures. The analysis is kept concrete using the Open Service Architecture for Sensors and Universal Plug and Play (UPnP).*

Index Terms — **Wireless sensor networks, application-level gateway, service oriented computing, event-based.**

I. INTRODUCTION

Wireless sensor networks (WSNs) consist of small electronic devices capable of sensing, computing, communicating and sometimes actuating. Lately, these devices are entering the consumer and health care domains [1]. Examples can be found in scenarios for health monitoring, home surveillance or lighting. As usage of sensor networks by end-users increases, so does the need for integrating sensor networks with consumer electronics (CE).

While nowadays sensor nodes are commercially available, programming them, if possible at all, can be a costly and time-consuming matter and requires specialist knowledge such as dealing with distributed applications, low resource constraints, energy efficient programming and MAC and routing protocols. In addition, the applications which retrieve information from the sensor network store the data in a database and/or present it to an end-user through some graphical user interface. Third party access is then through the database API or requires a significant developer effort to create a programmatic API. This limits the interaction between CE applications and sensor networks, preventing them from manipulating or configuring the sensor network or forcing them to deal with the low-level details of sensor protocols and message formats.

A. Problem description

Service oriented architectures (SOAs) are a popular paradigm for programming distributed applications. Standards

such as UPnP [2], OSGi [3] and Web Services [4] are successful SOAs in the CE domain. To abstract away from the complexities of programming sensor networks, the SOA style is gaining traction in the sensor domain as well. Examples are OASiS [5], SYLPH [6] and the Open Service Architecture for Sensors (OSAS) [7]. The goal of this paper is to expose the functionality of such service oriented sensor networks to the CE domain, making the sensor domain accessible through common (SOA) CE standards such as UPnP.

On the one hand, the solution should simplify the task for application programmers in the CE domain to connect to sensor networks. On the other hand, the development of sensor network applications should not become more complex. To safeguard this, the following requirements should be met. First, applications written for the sensor network do not need to provide explicit support (REQ₁). I.e. the solution needs to work for any sensor application we write or have already written. Second, the solution needs to be efficient, especially with regard to overhead imposed on the sensor network (REQ₂), where overhead refers to computational overhead as well as memory and storage overhead. Third, the solution needs to be automatic (REQ₃). Given a description of the program (components) in the sensor network, they can be automatically exposed to the CE domain without manual intervention by a programmer. Finally, when possible, the solution should be a runtime addition (REQ₄). I.e. CE connectivity can be added or removed during runtime (of the WSN application) even after sensor network deployment and without any (significant) interruption of the running application.

B. Contributions

We analyze and present an approach for connecting two service oriented architectures utilizing an efficient application level gateway. We focus on exposing services within a sensor domain as services in the CE domain, and highlight tradeoffs between interface richness, descriptiveness, service granularity and discovery overhead. From that we obtain a classification of service oriented application gateways.

Our analysis is brought into practice by connecting two SOA platforms as an extended case study. The Open Service Architecture for Sensors (OSAS) in the sensor domain and Universal Plug and Play (UPnP) in the CE domain. A detailed mapping of the interaction styles in both domains is presented.

To evaluate our approach we have implemented a lightweight, generic and automatic application level gateway for the OSAS system which can expose OSAS services without explicit support of WSN applications.

¹ This work was partially funded by the WASP project of the European Information Society Technologies (IST) program, IST.034963.

Authors are with the Eindhoven University of Technology, department of computer architecture and networking, Den Dolech 2, 5612 AZ Eindhoven, The Netherlands. (e-mail: R.P.Bosman@tue.nl, J.J.Lukkien@tue.nl, P.H.F.M.Verhoeven@tue.nl).

C. Outline

This paper is structured as follows. Section II describes the two systems we use as the vehicle for explanation. First the OSAS platform is described and subsequently properties of Service Oriented Architectures (in particular, UPnP) are highlighted. Section III-A describes the various mappings which can be made at the protocol level. Section III-B outlines our approach. In section IV the details of the mapping of various interaction patterns are shown along with an example scenario. Next we present an evaluation of our approach in section V. Section VI describes related work. Finally, section VII concludes the paper.

II. SERVICE ORIENTED ARCHITECTURES

The Service Oriented Architectural style has as its main goal a separation between interface and implementation by exposing this interface on the network. The functionality behind this interface (the service) is accessed via messages. The service does not contain logic that determines the context it operates in thus leading to loose coupling. Applications can be composed by an external entity by connecting this distributed functionality, a process called orchestration: an orchestrator can discover services at runtime through some service discovery procedure.

A. The Open Service Architecture for Sensors (OSAS)

The OSAS system addresses the programmability of sensor networks. It consists of a programming language, virtual machine and message format and provides a toolchain (a compiler, loader and runtime simulator) for developing sensor applications.

The programming notation allows programmers to specify functional units in the form of services. Services periodically produce events and/or handle events from other services. Typical activities of services include sampling, filtering, interpreting and aggregation of sensor data, but they can also perform lower level tasks such as e.g. routing.

Services are initially passive entities. They become active only when one service is connected to another by means of a subscription. Subscriptions specify a many-to-many relationship between providers and subscribers as well as eventing conditions and extra-functional properties. The conditions include the period at which the provider needs to execute its event generation loops as well as boundary conditions such as e.g. a threshold for a sensor which indicates when an alarm needs to be sent. Any party, including parties external to provider and subscriber, can direct nodes to subscribe to other nodes. This third party binding mechanism facilitates external orchestration of the services.

The compiler translates the program to a compact byte code. The runtime environment provides a lightweight virtual machine specifically created for sensor nodes and closely tied to the operation of our service-oriented language. This runtime environment can be compiled for sensor nodes as well as for more regular platforms thus providing a straightforward means of simulating a sensor node on a PC platform. A loader

brings the compiled code onto the network where each node picks up the parts it needs, identified by a so called content-based address. A simple discovery service periodically reports the nodes ID along with a list of services and/or subscriptions.

Nodes communicate by transmitting messages akin to active messages [15]. These messages contain a series of *handlers* and *arguments* and can be fully processed by any node which understands the *handlers* (i.e. has services containing the handlers). The standard handlers provided by the runtime deal with byte code installation and include a very simple flooding algorithm for code distribution.

Fig. 1. OSAS network consisting of two sensor nodes and a simulated sensor node. The simulated node runs on a powerful gateway (such as a pc or laptop) and provides rich functionality such as a display and storage. The bridge node is attached over USB to the gateway and bridges between the 802.15.4 and the UDP domain. This example displays how events from a mode detection service are routed to its subscriber, a storage service, through (application level) routing services.

Finally, a link with neighbors (either a regular link layer like IEEE 802.15.4 [8], or a UDP overlay) acts as a carrier for the message format. Any transport facilities are built on top using the programming language. The UDP and IEEE 802.15.4 domains are linked using a bridge which creates a transparent broadcast domain between real and simulated nodes. I.e. nodes are not aware of differences between real and simulated sensor nodes. Figure 1 demonstrates the runtime organization.

B. Universal Plug and Play (UPnP)

Within the CE domain we have the example of UPnP [2] where services are exposed by (logical) devices and orchestrators are called control points. A special service discovery mechanism allows control points to find devices and services.

Fig. 2. A small UPnP network. The control point connects a MediaServer and MediaRenderer through UPnP messages. The actual video stream uses an out of band protocol. The goal of this paper is to allow a control point to take control of a sensor node, service or network as well.

Interaction within UPnP can take several forms. The most common one is the (synchronous) Remote Procedure Call (RPC) which represents calling a function on the service

interface. A call-back facility is given by an eventing mechanism: a control point may subscribe to an event of a service and register a call-back function. Other types of communication, e.g. streaming, are possible as well but not included in the standard though UPnP can be used to establish the connection. Figure 2 gives an overview.

In this study we want to make the sensor network (and/or individual nodes) available as UPnP devices, and we study different means to do so. Since OSAS is itself built around services one of the options is to bring these services into the UPnP domain. This is not as straightforward as it seems because of the different nature of the OSAS interaction styles (time triggered eventing and asynchronous RPCs) so we further study how to map the different interaction styles. We also want to support the discovery mechanism of UPnP.

III. INTEGRATION

In this section we study the options for bridging OSAS and UPnP protocols. Section III-A analyzes the possible approaches. Section III-B presents our approach and analyzes the tradeoffs that lie therein. The next section (Section IV) describes the details of how the protocol mapping has been realized.

A. Protocol mapping between OSAS and a SOA

In [16]-[18] a number of approaches on interconnecting WSNs with an IP network are presented. These approaches are summarized in Figure 3.

Fig. 3. The application level gateway communicates on both networks and adapts the application messages from WSN format to SOA format and vice versa in such a way that both networks are able to communicate via this gateway. In the sensor network overlay IP approach, IP hosts implement the sensor network stack on top of IP such that they become members of the sensor network communicating using the sensor protocols. The opposite, IP overlay sensor networks, implements IP on top of the sensor network stack making nodes directly accessible over the IP network. This requires node to speak the SOA protocols (e.g. SOAP [20] in case of UPnP). A new abstraction on middleware level can be created to hide conversions completely from applications. The white dots indicate where conversion happens.

An *application-level gateway* translates one application protocol to another. *Sensor networks overlay IP* deploys the sensor network protocol stack over IP by binding to a

transport protocol over IP. *IP overlay sensor networks* implements the IP protocol stack (a lightweight version, e.g. 6LoWPAN [19], uIP [9]) on sensor nodes. A *new abstraction on middleware level* introduces a new abstraction towards the application layer and implements its middleware on both the IP network and the non-IP network protocol stacks to forward packets.

Implementing a new middleware allows one to hide most of the specifics of the underlying protocols by creating a new abstraction which deals equally well with protocols on either side of the bridge. However, all interoperability with existing applications is either lost or the problem is shifted, since current applications do not support nor use the middleware (REQ₁).

The main advantage of the *IP overlay sensor networks* approach is the widespread use of IP protocols. Making IP protocols available to sensor networks seems like an excellent way to integrate with existing networks and is indeed being developed [19]. However, applications need to be adapted to support the constraints on the payload of messages in the sensor network. Application level payloads used in the IP domain (e.g. SOAP messages) are often very resource consuming. Furthermore, for sensor networks employing non-address-based communication (e.g. data driven networks) such an approach is hard to realize efficiently (REQ₂). *Sensor networks overlay IP* solves these restrictions since these protocols are already optimized for compactness and are running efficiently on sensor networks. However, the main advantage of interoperability with arbitrary IP networks is lost.

The main gain from these two overlays comes in the simplification they can bring in implementing *application level gateways*. Building on top of either approach allows *application level gateways* to focus mainly on conversion of the application level payload.

The OSAS system provides a *sensor networks overlay IP* approach to connect simulated nodes with real nodes. In this paper we utilize such a simulated node to realize an *application level-gateway* which bridges SOA networks and wireless sensor networks. To provide additional functionality to the sensor network (such as e.g. SOA services), OSAS provides a framework for defining simulated nodes, therefore the subject of this paper is integration of sensor networks into service oriented architectures and not the other way around.

B. Creation of gateway services

In our approach a simulated sensor node is part of the sensor network and exposes services in the SOA domain which bridges between the SOA and the sensor domain. We call these exposed services *gateway services*. All interaction with the sensor network flows through these gateway services which have the responsibility of converting SOA messages into sensor network messages and vice versa. The first important choice is what a gateway service will represent: we can create a gateway service per node, per service or per sensor network. The second important choice concerns the

functionality which is exposed: an interface to the node (i.e. a low level message passing API), an interface to a service (invoking service actions and subscribing to their events) or an interface to a sensor network as an entity by itself (i.e., running specific applications).

gateway service	interface level		
	node	service	network
per node	(node/node)	(serv/node)	(nwkw/node)
target	implicit	implicit	
interfaces	fixed, one per node	specific, many per node	n/a
functionality	message passing	calling & subscribing	
per service	(node/serv)	(serv/serv)	(nwkw/serv)
target		explicit	
interfaces	n/a	specific, one per service	n/a
functionality		calling & subscribing (1)	
per network	(node/nwkw)	(serv/nwkw)	(nwkw/nwkw)
target	explicit	explicit	implicit
interfaces	fixed, single	specific, for each service	fixed & specific
functionality	message passing (1)	calling & subscribing (2)	loader & appl.

1) Additional interface for device (node) discovery

2) Additional interfaces for device (node) discovery and node-service mapping

Fig. 4. Characteristics of application level gateways based on gateway service and interface granularity. For example, the node/nwkw approach exposes a single gateway service providing access to individual nodes. This means the interface must expose available nodes through e.g. a numbering scheme. Communication is directed towards individual nodes and is of a raw message-passing nature that takes the endpoint (these node numbers) as argument. Alternatively, the serv/node approach implies that for each node we have a gateway service which exposes callable actions through service specific interfaces. Because these gateway services are bound to a node, discovering nodes is done by discovering their gateway services.

We can create up to nine combinations along these axes, which are depicted in Fig. 4. Note that some of these combinations (nwkw/node, node/serv and nwkw/serv) are not meaningful. They lead to the creation of identical and redundant gateway services. We evaluate these combinations in the light of (service) discovery, specifically, SOA service discovery overhead, monitoring of node presence, and (self-) descriptiveness of the discovered gateway services.

The node/node approach exposes a service for each node, to reach this node through simple message passing. This has the advantage that node availability and/or reachability can be mapped directly onto the native discovery protocols of the SOA. The disadvantage is that it can spawn a lot of services and that node presence might fluctuate much more than the native discovery mechanism of the SOA can handle. This approach exposes an interface of a message-passing nature which although static and simple requires intricate knowledge about message semantics. This interface (*Msg API*) allows any handler to be invoked directly on a specified node with zero or more integer parameters. Responses are delivered to a *notification* interface which needs to be implemented by SOA clients.

The serv/node approach is similar in nature, but it exposes service specific interfaces. These service interfaces are much closer to SOA interfaces, expose callable (service specific) actions and hide messaging details. When the SOA does not support the concept of devices or of multiple interfaces per service then a gateway service must be exposed per (node, service) pair which can cause a greater burden on the discovery protocol of the SOA.

Fig. 5. Examples of exposed UPnP devices according to the approaches which expose service specific interfaces. Note that in the serv/node approach UPnP devices scale with the number of nodes, while the serv/serv approach creates a (UPnP) device per service. The serv/nwkw approach always exposes only a single device. The device discovery and node/service mapping interfaces are required to query which nodes exist and which services they provide.

By exposing a gateway service per WSN service (serv/serv), the burden on the SOA discovery is decreased. Fig. 5 shows how services are exposed in the UPnP domain. In general there are fewer services than nodes and the available services only change when the network is reprogrammed. This yields the same descriptiveness as the serv/node approach, but the correspondence between node presence and gateway service presence is lost. Therefore the destination node (i.e. a sensor node which implements the service interface) must be explicitly specified in each action invocation. Available nodes must be learned by querying the service through an additional device discovery API.

Node/nwkw and serv/nwkw place the least burden on the SOA discovery protocol and expose only a single gateway service each. Aside from providing a node discovery interface like the serv/serv approach, these approaches also need to provide an interface to query the mapping between devices and services.

The nwkw/nwkw approach is a special case in which the entire network is exposed within a single gateway service. This yields an API for manipulating and accessing the network as a whole. The first part of this API provides a (*Loader*) interface to add, remove, load and unload programs and subscriptions into the network.

The second part of the API is application specific. The application interface exposes only those parts of services which are required to interact with the application as a whole while hiding all internal interfaces of the sensor network. Section IV-B describes how this interface can be constructed.

Fig. 6. Example of typical service deployment over UPnP. Per node a UPnP device is created which exposes each of the services (serv/node) installed on the node as well as the Msg API (node/node) for direct communication. In addition, a single device is created for the entire network which provides the loader and application service (nwkw/nwk) and the Msg API (node/nwk) for all of the nodes. With the exception of the Msg API (node/nwk) all services are implicitly bound to nodes. For explicit binding of the node/nwk approach a Node Disc(covery) service is exposed.

Note that these approaches of creating gateway services are not mutually exclusive. In Fig. 6 an example is presented on how these approaches can be used in a UPnP domain. Since UPnP has support for device discovery, services can be exposed naturally on a per node basis. This example combines the node/node, serv/node, node/nwk and nwkw/nwk approaches.

IV. MAPPING INTERACTIONS

An application gateway must map interactions in one domain onto interactions in another. This mapping depends on the type of gateway that is selected. In this section we discuss how the various interactions and interaction styles of exposed SOA services are mapped onto communication in the sensor network and vice versa. Section (IV-A) looks at interactions between a SOA client and the gateway service and maps these onto meaningful operations in the sensor network domain. This way the SOA domain can use all of its interaction styles without restriction. This approach is dominantly used to realize the serv/node, serv/serv and serv/nwk approaches and in a slightly modified manner node/node and node/nwk.

Section IV-B looks at interactions between the sensor network and the gateway service and maps these onto operations in the SOA domain. This approach is used to implement the application-specific API of the nwkw/nwk approach. The nwkw/nwk gateway stands apart from all other approaches and requires some additional work.

The goal of both sections is to provide the functionality of the sensor network towards the SOA domain, not the other way around.

We consider the following interaction styles. *Publish and subscribe*, in which a subscriber subscribes to receive information from a provider which responds with zero or more notifications (supported by both domains). These *notifications* – which are asynchronous events – also occur without subscriptions. The synchronous variant of this is the *remote procedure call (RPC)*, a (blocking) call which returns one or more values (supported only by SOA).

service ModeDetect(\$Handler, \$threshold)

"@ \$Handler: ID of callback action (event handler)

@ \$threshold: nr of measurements before mode change is considered stable"

event report

"Sends current mode (0: standing, 1: lying, 2: in motion)

Message format: (\$Handler, NodeID, mode)

Determines mode by comparing a measured vector to a horizontal (lying) and vertical (standing) base vector. The outcome needs to be stable for \$threshold consecutive measurements, otherwise we are in motion."

action SetVector(x, y, z, orientation)

"sets the horizontal (orientation=1) or vertical (orientation=0) base vector"

Fig. 7. Interface of a simplified OSAS mode detection service. It detects the activity of a subject (standing, lying or in motion). Subscription parameters consist of a callback and threshold (to determine mode stability). It sends one type of notification containing the detected mode. The SetVector action is used to configure the orientation of the sensor node.

The simplified mode detection service in Fig. 7 is used as a running example throughout the remainder of the section.

A. Mapping interactions in the SOA domain to the OSAS domain

We look at services providing typical operations found in the SOA domain and map these onto operations within OSAS. The following table gives an overview of how these operations are mapped.

MAPPING OF SOA OPERATIONS ONTO OSAS OPERATIONS	
SOA operation (sent by SOA client)	OSAS operation (received by node)
a) SOA Notification	OSAS Notification (invoke action)
b) SOA Publish & Subscribe ¹	OSAS Publish & Subscribe ¹
c) SOA RPC	OSAS Subscription ¹ (1x; cancel)

1) With sending Publish & Subscribe we mean sending of the subscription, not the notification. The sender of the subscription receives the notifications that the receiver of the subscription publishes.

Fig. 8 presents a sequence diagram of the three interaction styles.

a) SOA sends a Notification

All event handlers (i.e. actions) of a sensor network service are exposed as functions without return values on the gateway service. A SOA client can call such a function which is then converted into the appropriate message format and forwarded to the sensor node. Since both systems support notifications, these can be mapped directly.

b) SOA Publish & Subscribe

The gateway service exposes a subscription method whose signature matches the subscription parameters of the sensor network service it represents. Upon subscribing, it generates a unique eventType (or callback) which is used to associate the notifications from the sensor network service with the SOA client that performed the subscription: the client is looked up and the notification is forwarded to the related client.

Fig. 8. Mapping of SOA interactions onto the sensor network. This is a serv/node approach, upon discovery a gateway service is generated which is implicitly bound to the discovered node (3) and service (ModeDetect). Notification messages and subscriptions are mapped directly onto the same concepts for nodes. Callback parameters (\$Handler in the example) are exposed as RPC. Whenever they are invoked, a one-time subscription is performed and the first result is returned. In order to associate a result to its originating subscription, the gateway generates unique callback IDs.

c) SOA performs an RPC

There is no straightforward RPC in the sensor network. All interaction is based on asynchronous message passing. Such messages are almost always sent as the result of a subscription.

Although an RPC can be implemented using two asynchronous messages, it would require a non-transparent change to the programs running within the sensor network which violates our requirements (REQ₁). We solved this by mapping RPCs onto subscriptions which can take parameters. Since each subscription can generate different types of notifications, a callable function is exposed on the gateway service for each type of notification. These functions take the same parameters as a subscription (with the exception of the subscription period). Upon invocation, a subscription is made and the first notification of the corresponding type is returned. Afterwards, the subscription is cancelled. OSAS supports such one-time subscription by a special parameter setting.

B. Mapping interactions in the OSAS domain to the SOA domain

The nwk/nwk approach as described in Section III-B provides two interfaces on the gateway service, the *loader* interface and the *application specific* interface. In this section we examine the *application specific* interface. Such an interface regards the entire sensor network as a single (composite) service and hides all internal interfaces, services, events and connections. Hence, the user of such an interface needs to know how to interact with the application, but does not require any knowledge of its internal components.

This section describes how to construct such an interface from a collection of individual service interfaces. In order to

facilitate this construction, the special *GW Publish Service* is added on the gateway which utilizes the third party binding mechanism of OSAS to declare which event handlers (through subscriptions) and which events (through subscription requests) are to be exposed publicly. These subscriptions and subscription requests need to be specified by a programmer and loaded into the network through the loader. We get back to this in section IV-C as this violates REQ₃.

MAPPING OF OSAS OPERATIONS ONTO SOA OPERATIONS

OSAS operation (sent by node)	SOA operation (sent(!) by SOA client)
d) OSAS Publish & Subscribe	SOA Notification
e) OSAS Notification (invoke action)	SOA Publish & Subscribe

The table above shows how the interactions in the OSAS domain are mapped onto interaction within the SOA domain. Fig. 9 shows the details of this mapping.

Fig. 9. Mapping of sensor network interactions onto the SOA. Publish and subscribe interaction from the sensor network is mapped onto notifications from the SOA, whilst notification from the sensor network are mapped onto publish subscribe from the SOA. Subscription requests (third party binding, a directive to subscribe) are used in the sensor network to indicate what information needs to be published or is required from the outside world. Asking for information is done by subscribing to a virtual (non-existent) service, the effect of which is publishing of the specified callback. Publishing information is done by passing the Forward callback in a subscription request.

d) OSAS Publish and subscribe

In this interaction style, one or more sensor nodes subscribe to information from SOA services by sending an OSAS subscription towards the gateway. The correspondent notifications are generated by SOA clients as RPCs or as notifications (in this case, unsolicited events). In our example (see Fig. 9), a sensor node subscribes to SetVector events. An invocation of SetVector is forwarded to all nodes which have subscribed to it (one-to-many). The SOA client itself has no knowledge of receivers of this message; it only talks to the gateway of the sensor network.

Contrast this with approach *a) SOA Notification* in Figure 8, where the notification is forwarded to only one node (one-to-one) determined by the SOA client.

In order to facilitate notifications from the SOA domain, the event handlers (such as SetVector) need to be exposed on the gateway service as callable functions. Only those event handlers for which a subscriber exists – i.e. for which a subscription has been sent to *GW_Publish_Service* – are exposed on the gateway. These event handlers are specified as callback in the OSAS subscription. This way the subscription specifies both the endpoint for a specific SOA notification as well as the event handlers which compose the application-specific interface.

e) OSAS Notification

A notification from the sensor network is not unlike a subscription from the SOA domain (compare with SOA P&S in Fig. 8). The main difference lies in the fact that here the sensor network decides the terms of communication (what information to send, how often and under which conditions). In case of a subscription from a SOA (*approach b*), the SOA decides all this.

The subscription from the SOA client is required to inform the gateway about the endpoint of the communication. In this case, the SOA client subscribes to all events forwarded by the sensor network, viewing the network as a single entity (nwkw/nwkw). In fact, it does not even learn from where the communication originates, it looks as if the gateway was the sender.

Just like in the previous case, the application specific interface benefits by only forwarding those OSAS Notifications which are relevant for SOA clients (e.g. by filtering out internal events). To specify which events are relevant, a subscription request with the forward handler as callback is sent to the gateway and only events which are generated by the subscriptions generated out of these subscription requests are forward to the SOA client (see Fig. 9).

C. Comparison of the approaches

In sections IV-A and IV-B two different approaches to bridging sensor networks and service oriented architectures have been examined. Where the former section focused on point-to-point interaction between the two domains, the latter presents a more abstract interface (encapsulating internal components and decisions) towards applications in the sensor network.

Fig. 10 presents an overview of the differences between the two approaches.

In the general case, an OSAS network does not provide an application level interface for the activities of the sensor network, but rather exposes individual services instead. As a result, the node/x and serv/x approaches are more direct and straightforward than the nwkw/nwkw approach.

APPROACH COMPARISON		
	node/x, serv/x	nwkw/nwkw
Service visibility	All individual services are visible, all actions are exposed	Individual services are hidden, actions require explicit publishing
Communication endpoint	Determined by caller	Statically set by sensor network
Action invocation	One-to-one, targeted at a specific node	One-to-many, targeted at multiple nodes
Publish & Subscribe	Many subscriptions from SOA to OSAS, 1 service/subscription	Single subscription from SOA to OSAS, connects to multiple OSAS services
Deployment	Fully automatic	Manual connections between services and GW_Publish_Service need to be made
Parameters, control	Decided externally	Decided internally

Fig. 10. Differences between node/x and serv/x approaches on the one hand and the nwkw/nwkw approach on the other. The nwkw/nwkw approach provides a subset of the other approaches and restricts outside control to yield a simpler interface. The nwkw/nwkw approach internalizes control, composition and parameterization decisions.

In section IV-B we demonstrated how an application level API can be constructed by exposing a subset of all known handlers (*approach d*) and all possible events (*approach e*) through the creation of connections between OSAS services and a special service built into the application gateway (the *GW_Publish_Service*). This approach can be applied to an existing application at runtime and as such satisfies REQ₁. However, although specification of new connections between services is quick and easy to do, it does violate REQ₃.

If one is willing to drop REQ₁ and REQ₃ a viable alternative is to write a service with precisely the goal of providing an application specific interface for the sensor network and subsequently expose this service using the techniques described in section IV-A. Especially in architectures such as OSAS, where the ability to modify services over the air is built-in, dropping these requirements is not a major issue. This does not hold when nodes need to be recovered and reprogrammed to change their applications and in such cases the techniques used in section IV-B can be used instead.

V. EVALUATION

In this section we evaluate the generic UPnP bridge that we have implemented, based on several criteria. First we look at the overhead it places on the sensor network. This criterion is the most important one as the sensor network has strong resource constraints. Second we look at the transparency of our solution. As indicated in the introduction, our goal was to make this an automated solution for any application written for the OSAS platform. Finally we comment on the quality of the interface mapping.

A. Overhead

In order to track node presence and discover deployed services, we have added a simple discovery service to OSAS nodes. This discovery service was already used to monitor the

installation process, and it relies on built-in functionality of the OSAS platform. This service installs a total of 14 bytes on the sensor nodes. In order to pass this data to the UPnP gateway, the UPnP gateway needs to install a subscription, which specifies the reporting frequency that can range from as often as once every 10ms to just once per hour (depending on the requirements). The payload of the response message contains 2 bytes plus 1 byte for every service installed on that node (typically less than 10 bytes total). In case of a static service deployment, a 1 to 2 byte heartbeat message will suffice as services need not be reported constantly.

All communication is mapped onto native OSAS messages without addition of additional data fields, this means notifications, subscriptions and reporting events all use the OSAS message format which is optimized for sensor networks. All remaining overhead of service creation, advertisement, processing and mapping is performed on a virtual sensor node which is deployed on a powerful device which does not have the same resource restrictions as a sensor node.

B. Transparency

The implemented solution is completely transparent with regard to the OSAS services. This means that none of the OSAS services has to be written with UPnP in mind, nor does it have to do anything to support the gateway. On the other hand, specifically designing a service API targeted at UPnP can definitely result in a richer and more meaningful interface. As mentioned before, neither the UPnP gateway nor any of its supporting services (such as e.g. the discovery service) need to be present at the time of deployment of the sensor network, since OSAS supports dynamic addition and removal of services and subscriptions. At any given point in time, the gateway can be connected to the sensor network to automatically expose its services.

C. Interface mapping

Whenever a mapping is made between two protocols or standards there tend to be expressivity mismatches and an OSAS - UPnP mapping is no exception to this. Although a direct one-to-one mapping between actions of OSAS and UPnP can be made, the publish/subscribe mechanism of OSAS has a considerably higher expressivity than the one of UPnP. Subscriptions in OSAS are parameterized and each subscriber can specify conditions under which it wants to receive an event in order to save energy. Since there are no such energy considerations in the UPnP domain, the subscription mechanism lacks this feature. This has been resolved by implementing a dual-layer subscription mechanism. One performs an UPnP subscription to announce its willingness to receive events. Subsequently a subscription action is invoked to specify the exact conditions of the event.

The second expressivity mismatch concerns return values. OSAS callbacks can receive a dynamic runtime-determined set of parameters whereas UPnP declares both input and output statically. As it is in general not possible to

automatically determine the event signature from an OSAS program, the OSAS return values are mapped onto a comma separated list rather than the more descriptive key/value pairs. To change this, the gateway would need to be extended manually to support specific OSAS services.

VI. RELATED WORK

Almost all sensor network discussions start out with the concept of a gateway which acts as a sink for information gathered from the network. Sometimes these gateways merely store data in a database, other times they are attached to a GUI application for displaying real-time gathered data.

The need to integrate in a more general way with the IP domain was quickly identified and yielded various approaches, most of which can be classified by the following four categories: application level gateways, IP overlays, sensor overlays and middleware approaches (see also section III-A).

There are numerous implementations of small TCP/IP stacks specialized for sensor networks, e.g. [9], [10]. While these approaches succeed in transparently providing access to nodes as TCP/IP endpoints, they fail to deal with the resource constraints of nodes at the application layer. Application gateways, such as the one presented in this paper, have the benefit that clients can communicate with sensor networks in exactly the same way as they communicate to other devices.

In [11], the authors describe a sensor overlay over IP consisting of a network of virtual nodes which provide a way to bridge sensor network protocols with the IP domain, e.g. by exposing a database API. This use of virtual nodes is very similar to the approach used in the OSAS system. In this paper we extended such a sensor overlay with an application gateway to provide a natural and direct way to access the sensor network from a SOA standard.

These days a lot of programming systems for sensor networks exist. Some are based around a specific virtual machine or programming language like Maté [12], SensorScheme [13] or Kairos [14]. Recently, more systems are embracing a service oriented approach like OASiS [5] and SYLPH [6]. Although the details of the mapping will differ, these service oriented systems can be mapped using the approach we presented in this paper.

VII. CONCLUSION

In order to increase the accessibility of sensor networks for consumers, it is important to provide access to the sensor network from devices such as PDAs, laptops and smartphones. Technologies which overlay IP protocols over the sensor network (such as e.g. 6LowPan) require end-user programs to take the limits and constraints (e.g. RAM limitations, need for energy efficiency) of sensor networks into account. By utilizing application level gateways, a translation can be made between the two domains which simplifies development of the consumer applications without complicating development of the sensor network.

In this paper we classify various ways to represent wireless sensor networks at the gateway-level. We present a tradeoff between the richness of exposed service interfaces and the deployment granularity of the gateway services in light of the overhead to the SOAs discovery protocols and complexity of client applications.

We describe how interactions in a SOA domain can be mapped onto interactions in the sensor domain and vice versa such that SOAs can transparently interoperate with sensor networks without having any adverse effect on the protocols and operation of the sensor network and therefore maintaining the strengths of the programming models on both sides of the gateway.

To validate our classification an application level gateway has been implemented as a virtual sensor node which utilizes an overlay of the sensor network protocols over IP to communicate with the sensor network. The application level gateway creates gateway services, based on our classification, which automatically translate incoming requests to messages native to the sensor network thus preserving the compactness and energy efficiency of the sensor network protocols.

The implementation shows a typical deployment of OSAS services within the UPnP standard.

ACKNOWLEDGMENT

The authors would like to thank our former/graduated master student, Migiel de Vos, for the initial research on which this work is based.

REFERENCES

- [1] D. Albu, J. J. Lukkien, and P. H. F. M. Verhoeven, "On-node Processing of ECG Signals", *Consumer Communications and Networking Conference (CCNC)*, Jan. 2010.
- [2] B.A. Miller, T. Nixon, C. Tai, and M.D. Wood, "Home networking with Universal Plug and Play", *IEEE Communications Magazine*, vol. 39, no. 12, pp.104-109, Dec. 2001.
- [3] M. Condry, U. Gall, P. Delisle, "Open Service Gateway architecture overview", *The 25th Annual Conference of the IEEE Industrial Electronics Society (IECON)*, vol. 2, pp.735-742, Dec. 1999.
- [4] M. P. Papazoglou, "Service-oriented computing: concepts, characteristics and directions", *Proceedings of the Fourth International Conference on Web Information Systems Engineering (WISE)*, pp.3-12, Dec. 2003.
- [5] M. Kushwaha, I. Amundson, X. Koutsoukos, E. Neema, and J. Sztipanovits, "OASIS: A programming framework for service-oriented sensor networks", *Proceedings of the 2nd International Conference on Communication Systems Software and Middleware*, 2007.
- [6] Dante I. Tapia, Ricardo S. Alonso, Juan F. De Paz, and Juan M. Corchado, "Introducing a Distributed Architecture for Heterogeneous Wireless Sensor Networks", *Proceedings of the 10th International Work-Conference on Artificial Neural Networks: Part II: Distributed Computing, Artificial Intelligence, Bioinformatics, Soft Computing, and Ambient Assisted Living*, 2009.
- [7] R. P. Bosman, J. J. Lukkien, and P. H. F. M. Verhoeven, "An integral approach to programming sensor networks", *Consumer Communications and Networking Conference (CCNC)*, Jan. 2009.
- [8] IEEE Computer Society, IEEE Std 802.15.4-2006: Wireless Medium Access Control (MAC) and Physical Layer (PHY) specifications for low-rate Wireless Personal Area Networks (WPANs), 323 pages, September 2006.

- [9] A. Dunkels, "Full TCP/IP for 8-bit architectures", *Proceedings of the 1st international conference on Mobile systems, applications and services (MobiSys)*, 2003.
- [10] G. Montenegro, N. Kushalnagar, J. Hui, and D. Culler, "Transmission of IPv6 packets over IEEE 802.15.4 Networks", RFC4944, 29 pages, September 2007.
- [11] H. Dai, and R. Han, "Unifying micro sensor networks with the internet via overlay networking", *29th Annual IEEE International Conference on Local Computer Networks (LCN'04)*, pp.571-572, 2004.
- [12] P. Levis, and D. Culler, "Maté: a tiny virtual machine for sensor networks", *Proceedings of the 10th international conference on Architectural support for programming languages and operating systems*, pp.85-95, ACM Press 2002.
- [13] L. Evers, P. J. M. Havinga, J. Kuper, M. E. M. Lijding, and N. Meratnia, "SensorScheme: Supply chain management automation using wireless sensor networks", *IEEE Conference on Emerging Technologies and Factory Automation (ETFA)*, 2007.
- [14] R. Gummadi, O. Gnawali, and R. Govindan, "Macro-programming wireless sensor networks using Kairos", *Proceedings of the International Conference on Distributed Computing in Sensor Systems (DCOSS)*, June 2005.
- [15] T. von Eicken, D. Culler, S. Goldstein, and K. Schauer, "Active messages: a mechanism for integrated communication and computation", *Proceedings of the 19th annual international symposium on Computer architecture*, pp.256-266, May 1992.
- [16] Q. Le-Trung, F. Eliassen, P. E. Engelstad and T. Skeie, Internetworking Wireless Sensor Networks with the Internet.
- [17] L. Shu, X. Wu, H. Xu, J. Yang, J. Cho and S. Lee, "Connecting heterogeneous sensor networks with IP based wire/wireless networks", *The Fourth IEEE Workshop on Software Technologies for Future Embedded and Ubiquitous Systems, and International Workshop on Collaborative Computing, Integration, and Assurance*, May 2006.
- [18] L. Shu, J. Wang, H. Xu, J. Cho, and S. Lee, Connecting Sensor Networks with TCP/IP Network.
- [19] J. W. Hui and D. E. Culler, "Extending IP to Low-Power, Wireless Personal Area Networks", *IEEE Internet Computing*, vol.12, no 4, July 2008.
- [20] T. Jepsen, "SOAP cleans up interoperability problems on the Web", *IT Professional*, vol.3, no.1, pp.52-55, Jan/Feb.2001

BIOGRAPHIES

Remi Bosman (M'09) was born on the 15th of June in Kapelle Biezelinge, the Netherlands. He received his Master degree (cum laude) in Technical Computer Science at the Eindhoven University of Technology (TU/e) in 2005 within the Software Construction group. Afterwards he switched to System Architecture and Networking to conduct his PhD research on the topics of coordination within service oriented architectures and wireless sensor networks. Other research interests include domain specific languages and code generation.

Johan J. Lukkien (M) is head of the System Architecture and Networking Research group at Eindhoven University of Technology since 2002. He received M.Sc. and Ph.D. from Groningen University in the Netherlands. In 1991 he joined Eindhoven University after a two years leave at the California Institute of Technology. His research interests include the design and performance analysis of parallel and distributed systems. Until 2000 he was involved in large-scale simulations in physics and chemistry. Since 2000, his research focus has shifted to the application domain of networked resource-constrained embedded systems. Contributions of the SAN group are in the area of component-based middleware for resource-constrained devices, distributed coordination, Quality of Service in networked systems and schedulability analysis in real-time systems.

Richard Verhoeven (M'07) was born in Veghel, the Netherlands, in 1969. He received the M.Sc. degree in Mathematics and Computer Science from the Eindhoven University of Technology in 1992, and a PhD degree in Computer Science in 2000 from the same institute. In 2002, he joined the System Architecture and Networking group as researcher and systems engineer, where he is involved in multiple national and international research projects. His research interests include computer networks, service oriented architectures and wireless sensor networks.