EURASIP Journal on Wireless Communications and Networking
a SpringerOpen Journal

RESEARCH Open Access

A framework for service provisioning in virtual sensor networks

Lambros Sarakis[1*], Theodore Zahariadis[1], Helen-Catherine Leligou[1] and Mischa Dohler[2]

Abstract

The majority of research and development efforts in the area of Wireless Sensor Networks (WSNs) focus on WSN systems that are dedicated for a specific application. However, this trend is currently being replaced by resource-rich WSN deployments that are expected to provide capabilities in excess of any application's requirements. In this regard, the concept of virtual sensor networking is an emerging approach that enables the decoupling of the physical sensor deployment from the applications running on top of it, allowing in this way the dynamic collaboration of a subset of sensor nodes and helping the proliferation of new services and applications beyond the scope of the original deployment. In this context, the article presents the architecture of a system for the realization of Virtual Sensor Networks (VSNs). The aim of the proposed architecture is to enable the realization of scalable, flexible, adaptive, energy-efficient, and trust-aware VSN platforms, focusing on the reduction of deployment complexity and management cost, and on advanced interoperability mechanisms. The efforts have been put towards specifying a service provisioning architecture and mechanisms for advanced sensor and middleware design.

Keywords: wireless sensor network, virtualization, system architecture

1. Introduction

Over the last few years, we are witnessing an explosion of interest in research on Wireless Sensor Networks (WSNs) that are accompanied by a constantly increasing interest for physical WSN deployment and commercial exploitation of the services provided by these networks. In this regard, the majority of relevant research and development efforts have focused on WSN systems that are dedicated for a specific application. However, a trend is currently developing towards resource-rich WSN deployments that are expected to provide capabilities in excess of any application's requirements.

In that sense, sensor-based applications/services will utilize sensors for purposes beyond the scope of the original sensor design and deployment, creating in this way an instantiation of a Virtual Sensor Network (VSN) [1,2] to serve user-specific requests. Virtualization is the key enabler for decoupling the physical sensor deployment from the applications running on top, and is thus a significant step towards the decoupling of ownerships in the Internet of Things. In rough analogy to virtualization in more "traditional" network infrastructures (both at the core and access network levels) [3], WSN virtualization aims to accommodate multiple logical network instances over a single physical network infrastructure with the ultimate goal of (a) supporting applications with different requirements both in terms of nodes and communication functionalities and (b) utilizing in an efficient and cost-effective manner the available network resources.

With these goals in mind, one could think of several applications that can benefit from the virtualization of sensor network infrastructures. In the following, we discuss two types of applications that correspond to what we consider to be the most significant use cases for VSNs. The first type involves geographically overlapped applications [1]. In this case, a WSN that is deployed to support an application can utilize communication resources offered by another WSN operating in the same area and being deployed to support a different application. The main benefit from the collaboration of the different WSNs, in this case, is the reduction of the

* Correspondence: sarakis@teihal.gr
[1]Technological Educational Institute of Chalkida, 34400 Psachna, Euboea, Greece
Full list of author information is available at the end of the article

number of sensors of each type without losing accuracy or degrading the required user functionality. With the smart usage of previously deployed networks, this approach contributes to cost savings and helps future evolvement and addition of new types of sensors without the need to deploy a full network from scratch.

The second VSN use case involves applications operating over multi-purpose sensor networks. A common use of VSNs, in this case, could be the separation of nodes with multiple sensing elements in different (virtual) networks, each one being responsible to track a different sensing parameter. Using this approach, a better administration at node level and enhanced usability can be achieved with an increased number of different end-users gaining access and control in sensors' information according to their needs.

Whilst mechanisms are in place to facilitate virtualization in the core network [3], major challenges remain in networks that connect embedded sensor devices to the Internet. On the one hand, these have to do with individual functionalities of a VSN system related, for example, to middleware, routing, security, trust, and energy awareness of protocols and mechanisms at various layers of the communication protocol stack. On the other hand, significant challenges arise as far as the development of a holistic approach to VSN is concerned, which should take into account not only individual enhancements related to the aforementioned functions, but also the need to federate resources across networks belonging to different administrative domains, as well as the need to support promising business models.

The challenges just presented are currently addressed by the VITRO project [4], the main objective of which is to achieve the realization of a scalable, flexible, adaptive, energy-efficient, and trust-aware VSN platform, focusing on the reduction of deployment complexity and management cost, and on advanced interoperability. The VSN architecture that is proposed by VITRO is presented in this article.

The motivation behind the proposed architecture stems from the current trend to overcome the constraints of application-specific WSNs and provide a federated system that is more efficient, flexible, and adaptive in a wide sense. In this context, the goal of the system presented in this article is to support a number of advanced applications/services provided by VSNs. Each of these applications will have their own resource and service requirements, which must be fulfilled in order to ensure specific service level agreement requirements. The abstraction introduced by the resource/service virtualization mechanisms will allow network operators to manage, modify, and utilize WSNs in a highly flexible and dynamic way. The flexibility gained through such an approach can be used to increase sustainability (in terms of cost-effectiveness) of deployed network/node resources.

Given the heterogeneity of the deployed WSNs (in terms of platforms, operating systems–OSs, programming paradigms, addressing schemes, etc.), it is a tremendous challenge to make effective and seamless usage of such resources and services in an integrated VSN service platform. In this context, this study aims at describing an integrated system architecture that extends the notion of a WSN towards the realization of large-scale, cross-organizational VSNs. What is introduced is a new paradigm in the design, deployment, and usage of WSNs, one in which the sensor network is not designed and deployed for one specific application, but for a variety of applications. In this way, WSNs can in principle support many applications and, at the same time, a new application can count on a wide base of sensor networks already deployed.

The remainder of the article is organized as follows. Section 2 presents the related study on sensor networks virtualization, while Section 3 introduces the proposed VSN system architecture and discusses the principles behind its design. The architecture of the VSN gateway and the architecture of the sensor node are described in Section 4. The interfaces between the system components are presented in Section 5, while example interactions for service registration and negotiation, and session establishment are illustrated in Section 6. Finally, Section 7 concludes the article and presents future directions.

2. Related study

Virtualization gains increasingly more attention in computing and networking targeting more efficient resource utilization, lower cost, increased flexibility and manageability, and improved administration and interoperability among different computing devices. Aspects of virtualization have been introduced in WSNs with the aim to give solutions to the constraints mentioned above and lead to a wider use of these networks and a faster realization of the Internet of Things. Research efforts in sensor networks virtualization include (a) OS virtualization, (b) sensor virtualization, (c) network virtualization, (d) virtual machines, and (e) middleware layers, with the last being the predominant area of interest.

Running multiple OSs at node level is at least inefficient due to constraints in sensor hardware. Moreover, running one or more applications that are not supported by the host OS, which is the most reasonable use of full platform virtualization, is a much more infrequent case in sensor nodes than traditional PCs. Up to date most OSs are built under the event-driven framework (e.g., TinyOS [5], Contiki [6]). A traditional thread-based approach is supported by the kernel of the MANTIS OS

[7]. Platform virtualization with a node running distinct host and guest OS threads has been demonstrated by the TinyMOS implementation which can run a typical TinyOS program as a MANTIS thread [8].

Sensor virtualization is supported by inserting an abstraction layer between the application logic and the sensor driver with the aim to address issues like incomplete *a priori* knowledge of the area of operation, recalibration of sensing equipment, changing conditions, etc. A component targeting sensor virtualization, which is programmable at runtime and is able to execute different adaptation schemes that may change during the application lifetime, is presented in [9]. Another area of research on virtual sensors is related to inference when physical sensors of the WSN do not work anymore. In this context, the study of [10] presents a method where virtual sensors infer approximation values using fuzzy logic rules with the assumption that physical quantities sensed by neighbor sensors are related.

Network virtualization in WSNs focuses on algorithms and protocol support for the formation, usage, adaptation, and maintenance of (possibly dynamically varying) subset of sensors collaborating on specific tasks and being organized as a VSN using resources of a shared physical infrastructure [1]. These nodes rely on those outside the subset to achieve connectivity and overcome the deployment and resource constraints. To make VSNs a reality, a number of mechanisms for VSN maintenance (e.g., adding/deleting nodes, entering/leaving a VSN, and merging/splitting of VSNs) and membership maintenance (e.g., dynamic (re)assignment of sensor roles) must be developed. A mechanism for self-organization of VSN members relying on a cluster tree-based scheme is presented in [11].

Virtual machines are widely used in high-end servers and PCs for various purposes such as platform independence and isolation. In sensor networks, however, the focus is on re-programmability, i.e., the capability of injecting new code into each node on site dynamically. Examples of stack-oriented virtual machines include Maté [12] and ASVM [13]. Their goal is to provide an application-specific virtual machine that delivers the needed flexibility to support a safe and efficient programming environment. By providing a limited number of instructions necessary for a specific application, Maté/ASVM can reduce the size of the assembly code to be transmitted to each node. Melete [14] extends Maté and supports multiple concurrent applications. VMStar [15] is another framework for building application-specific virtual machines and allows the dynamic update of the system software, such as the virtual machine itself, as well as the application code.

In the last few years, many varying middleware solutions for WSNs have been presented and their architectures have been influenced by the progressive shift of application paradigms from targeting single sensor networks towards implementing the Internet of Things, where applications will operate over multiple interconnected networks on a global scale. Each middleware solution makes its own architectural design assumptions that are largely based on the application domain(s) it aims to support. Thus, each solution provides different degree of support for features of virtualization such as resource and service discovery, collaboration of heterogeneous nodes, interconnection of sensor sub-networks, and connectivity to external networks, ease of deployment and maintenance, concurrent execution of multiple applications, scalability, adaptability, and reliability. Most of the current solutions try to facilitate high-level querying of sensor data, help to mask the distribution and heterogeneity in the sensor network, and address resource constraints by providing, for example, energy-aware routing and query processing.

In terms of their models for querying and data aggregation, their assumptions about the topology and other characteristics of the network, as well as the degree of support for the aforementioned aspects of VSNs, the proposed middleware frameworks can fit into one or more of the following categories:

- Database-inspired approaches, where the middleware configures the sensor network to behave as a database management system accepting and processing database queries from the application layer. Examples of this type of middleware include Cougar [16], TinyDB [17], and Global Sensor Networks [18].
- Middleware making use of a distributed virtual shared space. Many approaches in this class are based on the tuple space shared memory model, which was notably used in Linda [19], or simply provide a shared repository to enable global coordination and execution of group algorithms. Solutions belonging to this category include TinyLIME [20] and FACTS [21].
- Event-based approaches, which rely on publish/subscribe mechanisms for sensor data delivery (e.g., Mires [22], PSWare [23]). In this case, the middleware architecture allows sensor nodes to advertise the types of sensor data they can provide, client applications to select from the advertised services, and sensor nodes to publish their data to clients in accordance with their subscriptions.
- Middleware targeting adaptability and reliability. In this category, the middleware approaches are designed with special consideration for the issues of adaptability to dynamic environments and self-organization, while some employ fault-tolerance

mechanisms. Example solutions of this type of middleware include Impala [24], RUNES [25], and DARMA [26].

• Middleware with special support for heterogeneity and scalability. The formation of VSNs and the collaboration of their heterogeneous smart nodes require middleware frameworks able to efficiently manage the underlying differentiation in terms of software and hardware resources and subsequently hide it from the applications. Scalability is also required as VSNs can be formed across multiple and heterogeneous WSNs possibly belonging to different administrative domains. Regarding single sensor networks, some representative approaches in this category are TinyDB, Cougar, and Agilla [27]. Multiple sensor network management is a concept supported by jWebdust [28], Hourglass [29], and Global Sensor Networks. This concept proposes the notion of a VSN comprised of a number of discrete sensor networks that can be managed as a single entity. Another project that enables the interconnection of multiple sensor networks is Sensor Web Enablement [30].

• Middleware with explicit support for mobile nodes. These nodes could have the role of a mobile sensor, of a "data mule" node enabling or assisting communication across disconnected parts of the network or of a mobile sink node. TinyLIME is specifically designed to support mobile gateways and Impala is also designed to function in high mobility scenarios.

• Middleware supporting connection to larger networks. Solutions in this category support the interconnection of WSN "islands" that export their resources and services to external networks (e.g., TCP/IP) in order to be shared and exploited by monitoring and controlling entities. The Sensor Web Enablement framework provides such interface to the Internet. A common characteristic in this class of middleware is that it operates on (peer-to-peer) network overlays formed over the participating sensor networks, where each network is represented by an application-level gateway node (e.g., ShareSense [31], Hourglass, Global Sensor Networks).

Frameworks that support the interconnection of multiple sensor networks and provide interfaces between such localized networks and external ones (e.g., the Internet) specify models and services that can be used for service provisioning in WSNs. A notable contribution in this area comes from the SWE [32], which has specified standard models for (a) describing sensor systems and processes associated with sensor observations (Sensor Model Language), and (b) encoding sensor observations and measurements (Observations & Measurements). The project has, in addition, specified standard web service interfaces for collecting observations and system information (Sensor Observations Service), requesting the execution of observations and sensor data acquisitions (Sensor Planning Service–SPS), publishing and subscribing to alerts produced by sensors (Sensor Alert Service–SAS), and delivering in an asynchronous manner messages and alerts from SAS and SPS (Web Notification Services). A service-oriented sensor Web architecture leveraging the SWE technologies is presented in [33].

Taking a more telecom operator-oriented approach to service provisioning in Machine-to-Machine (M2M) networks (which can be regarded as a generalization/superset of sensor networks), ETSI recently completed the specification of the functional architecture for M2M communications [34]. This architecture relies on a service capability layer located at the network and gateway (or device) domains, which is responsible for functions like application enablement, secure transport session establishment, network communication selection, network reachability and addressing, remote entity management, secure service bootstrap, etc. In contrast to SWE, the ETSI M2M architecture builds on the use of resources for the exchange of information between system components. The structure of the resources as well as API primitives has been specified and the information exchange follows a RESTful approach. In compliance with this approach, the communication between the gateway and the resource-constrained devices takes place through the COAP protocol [35], which is currently under standardization in IETF.

The SWE and ETSI M2M service frameworks inherently support developments that are based on different software architectural styles, namely service-oriented and resource-oriented, respectively. By combining individual strengths of the two approaches (i.e., extensibility of service-oriented approach and simplicity of resource-oriented approach), hybrid implementations are possible. The architecture presented in this article will utilize a combination of the two approaches (being more service-oriented at the core part of the provisioning framework and resource-oriented at the WSN part) in an attempt to fully exploit the potential of virtualization to decouple applications from the underlying hardware. A similar approach has also been used in [36]. However, compared to previous studies, the proposed framework takes a more integrated approach to virtualization by combining virtualization enablers at several layers of the sensor protocol stack (middleware, network, and MAC). To the best of the authors' knowledge, this is the first study that presents such a holistic approach to virtual sensor networking.

3. System design principles and architecture

3.1. Design principles

A design choice for the presented VSN system is the selection of the Internet as the physical bearer between sensor platforms and the applications. The motivation for this choice is that VSN-compliant sensor networks can be considered as a huge network, whose core is the Internet, where VSN-compliant client applications run. At the periphery of this global network, there are the single WSNs used by those applications. In this new context, where sensors are deployed by some organizations and then used by other organizations and where virtualization may imply virtualization of services but may also have a more physical implication like virtualization of node resources, two are the main drivers for the architectural design:

- Advanced sensor design which is able to support advanced features in several fields (energy saving, routing capability, middleware support, etc.).
- Middleware design to mediate between applications and sensors. Considering that a sensor does no longer know the application that will use it, and an application does no longer know the sensors that it will use, major issues that the system architecture must address are (a) how a new application finds the sensors that it needs, (b) how it negotiates for the right to use those sensors with the organizations who deployed and administer them, and (c) how an application reacts to changes in the network. In other terms, the mechanisms for matching between the applications (user) and the resources (used) have to be specified.

The proposed system has tackled both these issues, by defining a new reference architecture for the sensors and an architecture for the middleware that binds together the sensors and the applications. On the other hand, with the aim of not leaving out the numerous current WSN deployments, we have considered a third driver in the architecture design by including proprietary and legacy, non-VSN-compliant sensor networks. The VSN design, wherever possible, has avoided imposing specific characteristics or features to be mandatory implemented within the sensor node, and has instead allowed those to be implemented somewhere else on the Internet, provided that the accordance with a specific interface is ensured. The motivation for this choice is to facilitate early adoption of VSN services among existing WSNs that, with very limited investment, may implement adaptation servers for their WSNs, and open them to being used by VSN applications, favoring quick bootstrapping of the global VSN market.

3.2. General architecture

Prior to detailing the architecture of the VSN system, terms used extensively throughout the article are introduced:

- *Virtual Sensor Network (VSN)*: A VSN is the seamless grouping of WSNs also called Wireless Sensor Islands.
- *Wireless Sensor Island (WSI)*: A WSI is any grouping of one or more legacy, proprietary, or VSN-aware sensor networks which are able to communicate, respectively, via a legacy, proprietary, or VSN-aware gateway node. A WSI is an autonomous administrative domain offering at least one service (autonomous sensors, which are uniquely addressable from outside a WSI can be considered as a simplified WSI).
- *VSN service*: Defined as any combination of one or more operational and/or supporting services. As operational service, we define any sensing capability offered individually by a sensor node or collectively by a WSI. As supporting services, we define functions that support the provisioning of the operational services. The supporting services may be offered by one or more WSIs or it may be hosted outside the WSI by external service providers. Many VSN services may be combined to offer new VSN services (as service mash-ups).
- *Resource*: A resource is defined as any physical or logical entity of a sensor node or of a WSI, which can be allocated, utilized, and released in order to realize a service. As such, resources may be considered as service enablers.

The overview of the general architecture is illustrated in Figure 1. With respect to this architecture, the VSN business model we envisage foresees an open market environment, which enables open competition and flexible service provisioning. In this model, three main actors can be identified: WSI Enablers, VSN Service providers, and Users. The WSI Enablers are responsible for offering services collectively provided by WSIs under their administration. (For the sake of discussion, in this business model it is assumed that the WSI Enabler plays the dual role of being both WSI network operator and service provider.) A WSI Enabler may own, lease, manage, and/or administrate more than one WSI and offers at least one operational or supporting service. Based on the discovered capabilities and services provided by the underlying WSIs, it may additionally compose new services and provide for them structured service descriptions. The services may be offered to the VSN Service provider either free of charge or under specific

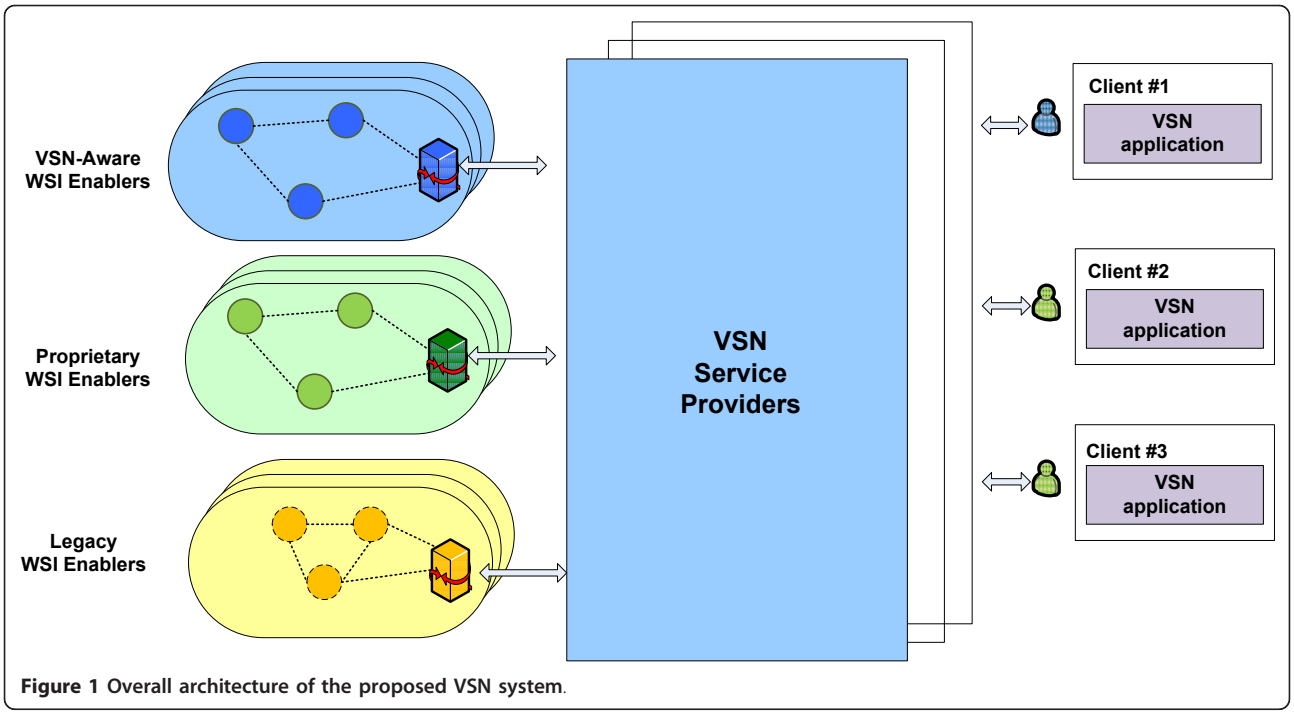

Figure 1 Overall architecture of the proposed VSN system.

contracts. In the latter case, guaranteed contracts and/or service level agreements may also be offered.

The VSN Service providers (VSPs) are the actors offering the VSN services to the Users. The main operations of a VSP are to publish, negotiate, provision, and monitor the execution of services. A VSP may be a WSI Enabler itself, utilize open-access WSIs, or may have established permanent or negotiate on-demand service contracts with one or more WSI Enablers. The VSP may provide to the user the services of the WSI Enablers (acting in this case like a service reseller) but may also provide enhanced (or even customized) services composed from service parts provided by different WSI Enablers. In either case, semantically rich descriptions of the services can be used to facilitate service classification and searching. Depending on the business model, open access, restricted, or premium application services may be offered to different types of users.

The Users are the actors that negotiate and exploit the VSN services, potentially under a Service Level Agreement with the VSP. A user may have a permanent or on-demand service contract with one or more VSPs.

It is noted that the VSN business model is not constrained to that depicted in Figure 1. In fact, the business model can support a fully cascading architecture (Figure 2), where a VSN application may utilize a hierarchy of n tiers of VSN Service Providers along with external third-party Service Providers. Various VSN service providers may also establish permanent contracts, or negotiate them on demand, with WSI Enablers, or other VSN service providers (in a collaborative fashion) or even with third-party service providers.

3.3. Service provisioning

The proposed architecture for service provisioning is shown in Figure 3. Each VSN Service Provider offers a number of applications through a VSN Core framework that consists of one Dynamic Virtual Network Server (DVNS), a number of instanced VSN Managers and may also host some Legacy Gateway Interworking Functions (LGW IF). In brief, the DVNS is responsible for discovering/registering and publishing VSN services to the end users and initiate service provisioning. The VSN Manager is responsible for service negotiation, session establishment, and monitoring, while the LGW IF enables the interaction with legacy WSIs.

The core part of the DVNS is the Services Registry. This is a database where all the known/registered WSI Enablers and the VSN services, which are known to this VSN service provider, are listed and described. In addition to the service name, ID, and description, the Service Registry has knowledge of the WSI Enabler(s) that offer the services, their gateways, etc., and in particular knows which Gateway(s) is/are responsible for each service. The VSN services may be composed from several "more elementary" services (hereafter called "abstract services") according, for example, to some pre-defined composition rule. Abstract services are described through a number of inputs, outputs, and parameters. An example of such a service is one that passively

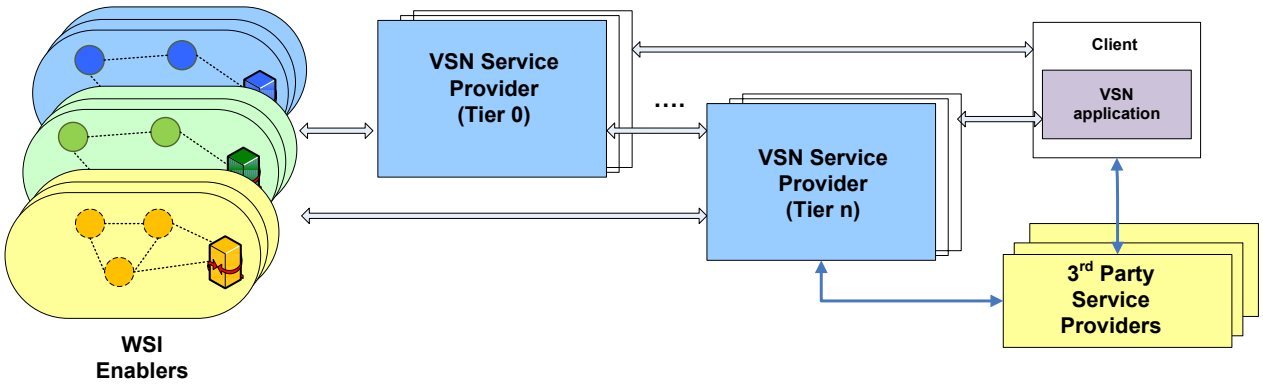

Figure 2 The VSN cascading architecture.

collects measurements of sensing capabilities in a selection of areas. For this service, the parameters may correspond to the set of desired sensing capabilities and areas of interest, the type of the data aggregation function (minimum, maximum, average, etc), the QoS requirements (e.g., tolerated time for a response, accepted percentage of estimated error), the type of reporting pattern (periodic or one-shot) and associated

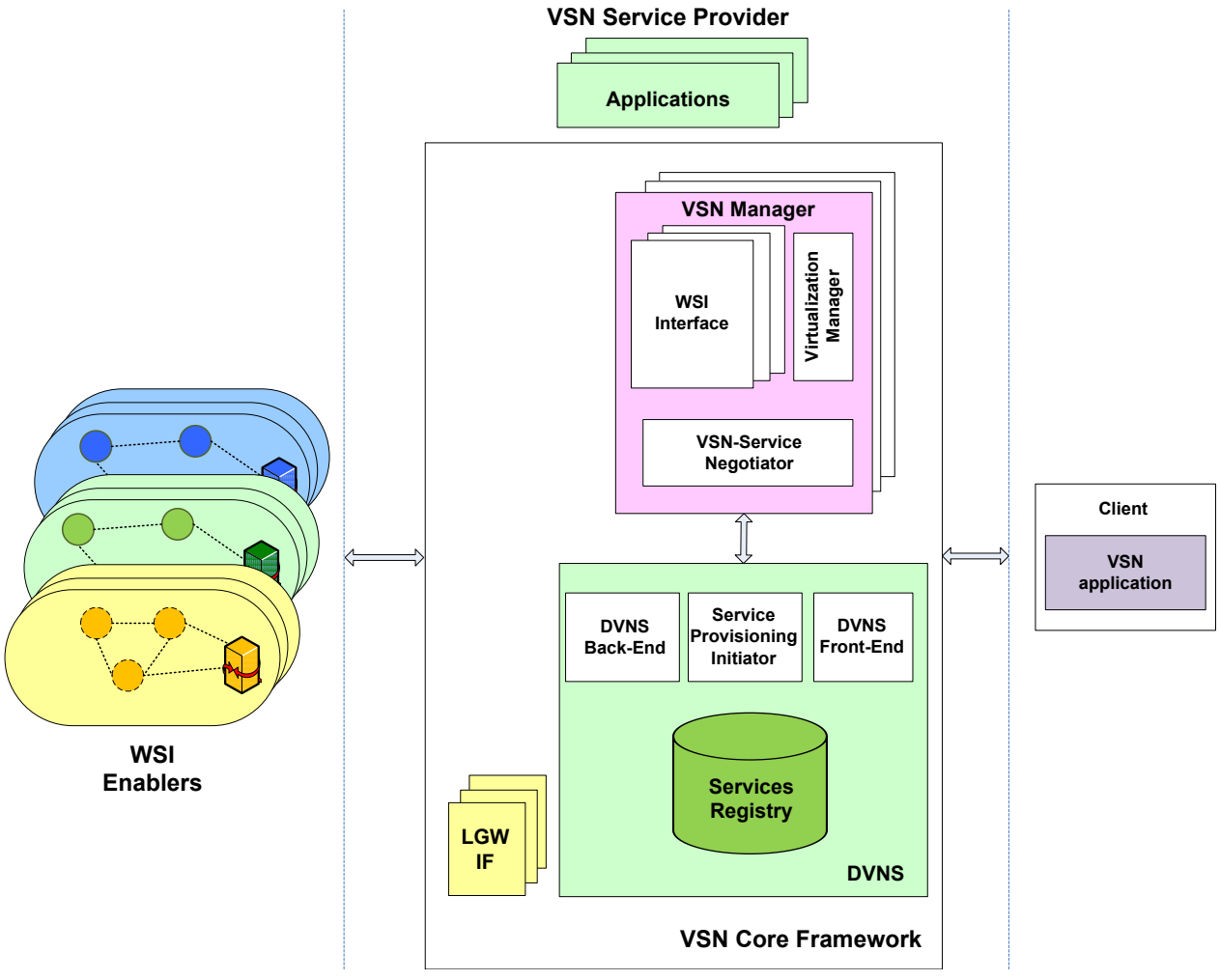

Figure 3 Building blocks of the VSN core framework.

measurement thresholds, and the unit of measurement. The input in this service is the sensor observations and the outputs can be the raw and processed sensor data, as well as the history of the sensor data.

The additional DVNS components support the Services Registry functionality in the following manner. The DVNS Back-End component is responsible for updating the Services Registry on a periodic or on-demand basis. In particular, using crawling techniques the DVNS Back-End queries all the known/registered WSI Enablers in order to discover new services or remove services that are not offered any more. It may also enable collaboration between DVNSs hosted by different service providers in case a contract is established. In addition, the WSI Enablers and/or the offered services may directly be registered to the Services Registry via the DVNS Back-End component. The main functionality of the DVNS Back-End is to register/discover new WSI Enablers, performing crawling of known/registered WSIs for new operational and supporting services and implementing ontology-based representations of services' description. The discovered or registered VSN services are then published by the DVNS Front-End to the VSN end-users/subscribers. The DVNS Front-End is also used for querying and retrieving VSN services based on end-user criteria. The main functionality of the DVNS Front-End includes publishing of VSN services in a transparent way to the end-user applications, processing of user queries and translation to VSN services and interactive ontology navigation features.

As soon as a VSN service has been preliminary qualified by the end-user, the Service Provisioning Initiator component is invoked to authenticate the user and check the user permissions. Then, it initiates a VSN Manager instance. In more detail, the Service Provisioning Initiator will contain a user database with the needed information in order to check user access rights and authentication privileges, get the user's contracts service level agreements, obtain from the services registry all necessary information and initiate a VSN Manager instance.

The VSN Manager, responsible for service negotiation, session establishment, and monitoring, may also perform service renegotiation and accounting/billing of the VSN services. It consists of the following subcomponents: the VSN Service Negotiator, the Virtualization Manager, and the WSI Interface. As it shown in Figure 3, the interface between the VSN Manager and the DVNS is implemented by the VSN Service Negotiator. This component receives from the DVNS (Service Provisioning Initiator) all necessary information in order to negotiate and establish a new VSN service. This information includes a list with the operational service(s) requested by the user, the WSI Enablers that offer such operational service(s), the WSI entry points that are responsible for the relevant operational and supporting services, etc. The VSN Service Negotiator queries all WSI and gets up-to-date knowledge of the relevant operational and supporting services, along with the relevant WSI capacity to support a new instance of the operational service (i.e., the WSI available resources). Based on this information and additional non-functional information (e.g., user profile, access permissions, contracts, prices, etc.), the VSN Negotiator will negotiate with the user application the VSN service provisioning. In this phase, the user may refine his/her requirements and negotiate the VSN service establishment.

If the service negotiation is not successful, the session is closed and the VSN Manager instance is dissolved. Otherwise, the Virtualization Manager is invoked. At contract start-up, the Virtualization Manager communicates with the various known/registered WSIs and requests the allocation of the necessary resources. This module is also responsible for hiding from the end-user application the actual WSI sensor nodes and the real service(s) (which may run in multiple WSIs), and offer the VSN service in a transparent way, appearing as a single operational service of the VSN service provider. Each WSI Interface is a logical process that is responsible for interfacing the WSI and hides any specificities of the WSI.

At run time, and based on the Contract Service Level Agreement, the Virtualization Manager may communicate with the WSIs that offer the requested operational services and check the session status/quality. Additionally, the WSI Interfaces may inform the Virtualization Manager if an error has been reported or trapped. In case of failure to meet the negotiated contract, the Virtualization Manager will communicate with the VSN Service Negotiator. The Negotiator will initially try to adopt countermeasures by negotiating with the WSI Enablers, a new session establishment. If anomalies cannot be compensated and the Service Level Agreement is broken, the Negotiator may try to re-negotiate with the end-user application or close the connection with an error message.

Finally, the LGW IF enables the interconnection between Legacy WSIs and the VSN network architecture. An entity outside the WSI should offer an LGW IF, which should be offered either by the WSI Enabler or by a Service Provider. Alternatively, each VSN Service Provider should implement instantiations of VSN Gateway (VGW) as interconnection functions for LGW. The LGW IF should offer the logical VGW functionality (i.e., status and negotiation of services and resources, ontological representations), without realizing the remaining physical layer functions. In addition to the VSN Core framework, the VSN Service Provider may offer

applications to the end-users, collaborating with VSN service providers or third-party service providers. These applications are customized to each VSN service providers and form the Service Provider's portfolio offering.

4. Wireless sensor islands

A WSI consists of at least one sensor node (normally a group of sensor nodes) and one gateway (acting as the WSI entry point). Collocation of their functionality is not excluded in case of autonomous addressable sensor nodes. If the gateway is mobile, IP radio access should be provided in a transparent way by the network infrastructure (see, e.g., [37]).

Within a VSN, the following three types of WSIs are considered (Figure 4):

- *VSN-aware WSI*: These are WSIs, which consist of VSN-aware Sensor Nodes (VaSN) and a VGW. It is assumed that within a VSN-aware WSI each node (sensor or gateway) is VSN-aware and has knowledge of the VSN service offerings. As it is shown in Figure 4, each VaSN may instantiate one or more virtual sensor nodes in the process of sharing its resources across multiple VSNs.
- *Proprietary WSI*: These are WSIs, which consist of Proprietary Sensor Nodes (PSNs) interconnected via a Proprietary Gateway. Due to the complexity that they might inherit, in order to guarantee interconnection with the VSN network architecture, we assume that the proprietary gateway offers a VSN-compliant interface, so it is considered as VGW.
- *Legacy WSI*: These are WSIs, which consist of Legacy Sensor Nodes (LSNs) interconnected via a Legacy Gateway (LGW). We assume that within a Legacy WSI, in order to offer interconnection with the VSN network architecture, an entity outside the WSI should offer a LGW IF. This interworking function should be offered either by the WSI Enabler or by a service provider. Alternatively, each VSN Service Provider should implement instantiations of VGW as interconnection functions for LGW.

In the following, we focus on the VSN-aware WSI as we assume that the other WSI types may appear as VSN-aware WSI with limited functionality.

4.1. Gateway architecture

The VGW is the device in charge of bridging between one VSN-aware WSI and the VSN Service Provider. It handles the bridging in terms of "lower layer" communication protocols (e.g., Medium Access Protocol, IPv4/IPv6, routing protocol), and protocols for upper layers (e.g., middleware, application). The VGW manages the discovery of operational and supporting services, as well as available resources, within the WSI. If required, the VGW notifies VSN Service Provider(s) about the available services and resources.

A functional view of the VGW is shown in Figure 5, and its main components are described next:

- *VGW API Interface*: this is an interface used by any external component to instantiate a VSN configuration within the WSI, retrieve the list of available resources/services, retrieve a history of data, or execute an action onto a specific node.
- *VSN controller*: This module receives the instructions to configure, within the WSI, all aspects related to the enforcement of VSNs for supporting VSN services. It maintains various registries including a registry of the subscriptions per VSN to a set of operational and supporting services provided by the WSI, a registry of tasks that are deployable on the sensor nodes involved in the operation of a VSN, a data logger that maintains a history of the data published by nodes involved in an enforced VSN (e.g., measurements, alarms, etc.) and a registry of the routing instances configured in the WSI. The VSN controller embeds a component named WSI and Sensor Node Configurator to configure the functioning of individual sensor nodes or networking protocols at the scale of the WSI. When receiving the configuration instructions to instantiate a VSN, this configuration component interacts with: the Routing Engine (e.g., an RPL driver, in order to run an

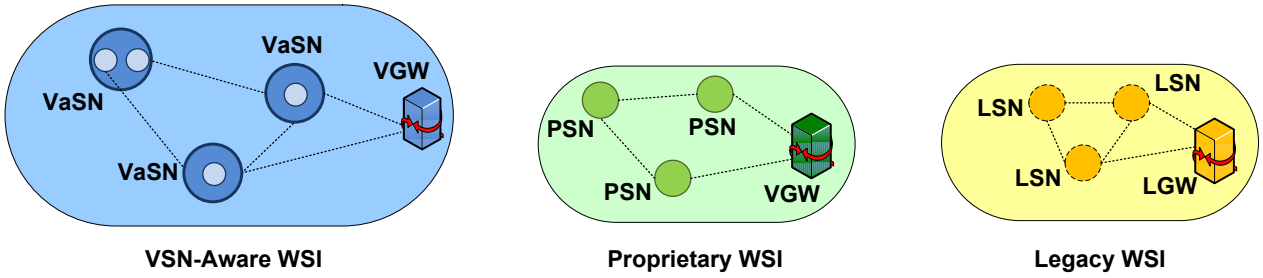

Figure 4 Types of WSIs.

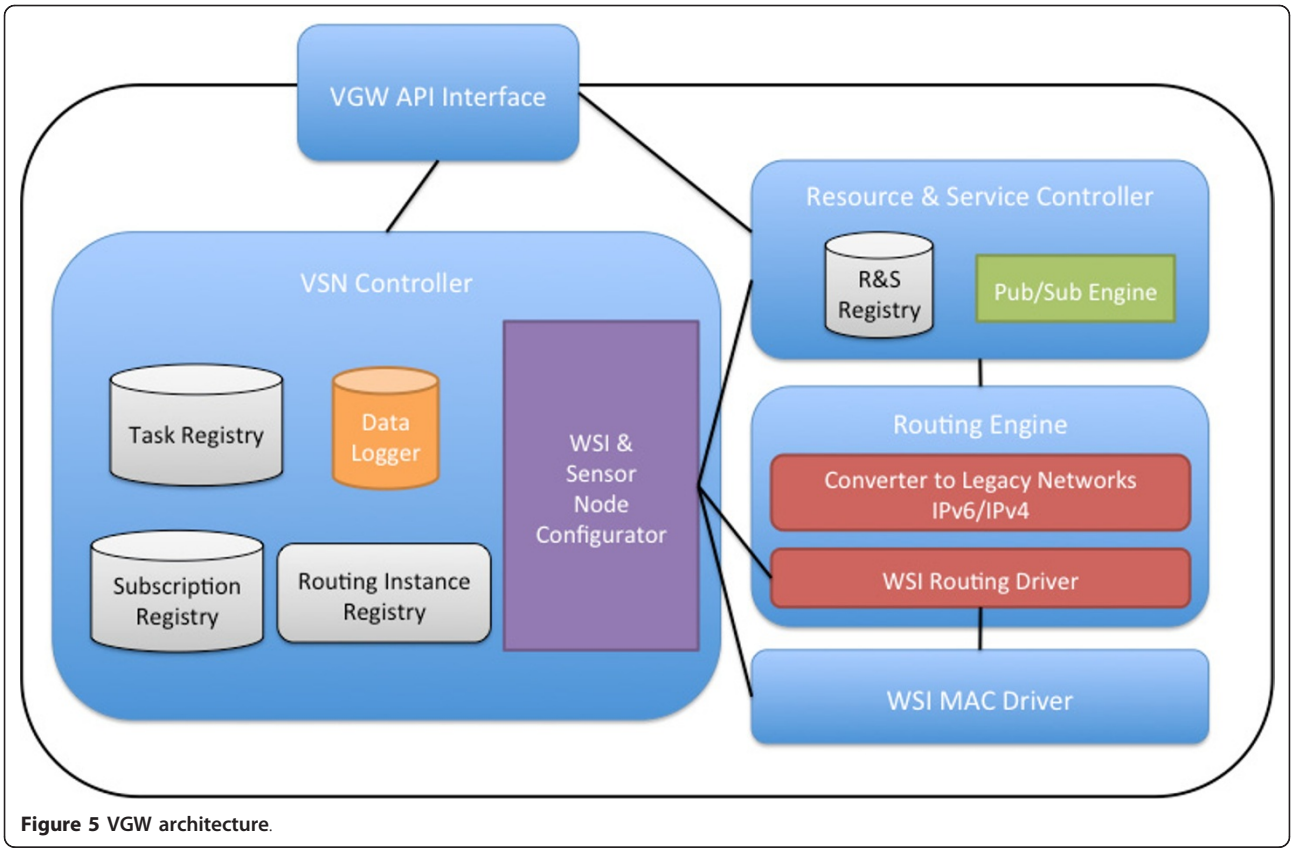

Figure 5 VGW architecture.

additional instance of the RPL protocol [38]), the WSI MAC Driver module, in order to instantiate a specific MAC scheduling and the Resource and Service controller, in order to subscribe to specific publishing services and manage the resources.

• *Resource and Service Controller*: This module negotiates with the VSN Service Provider and keeps a registry of all available operational and supporting services and the status of all WSI available resources. It uses a scalable publish/subscribe engine in order to handle published events and inform the VSN Manager about the subscribed services and the allocated resources status, as well as pushing actions to specific nodes.

• *Routing Engine*: it receives from the VSN controller the instructions to configure one or more routing instances (e.g., multiple RPL Destination-Oriented Directed Acyclic Graph instances), with one or many VSN applications per routing instance. This engine includes a WSI Routing Driver which is in charge of handling the operation of the WSI routing protocol (s), and handling for each protocol, the possibly multiple instances of routing plane. In addition, the engine encompasses a routing converter which is in charge of converting received packets from the WSI into the appropriate format to be sent over the legacy network.

4.2. Sensor node architecture

The functional architecture of a VSN sensor node is described in terms of its protocol stack as shown in Figure 6. In this architecture, virtualization is realized through efficient management of (a) node's services and resources and (b) functions at the network and MAC layers. Furthermore, the virtualization is done in a way that takes into account availability of security features and requirements for energy consumption.

Virtualization at the node level is undertaken by the Node Virtualization Manager (NVM). This component manages the services and resources provided by the sensor node. To discover available resources, the NVM interacts with several components (e.g., Energy Manager, components at network, MAC, and physical layers) in order to retrieve information regarding remaining energy, queue status, channel availability and usage, unoccupied RAM size, etc. This information is made available to the middleware component, which, in turn, presents it to potential resource consumers in a platform-independent resource description format. The

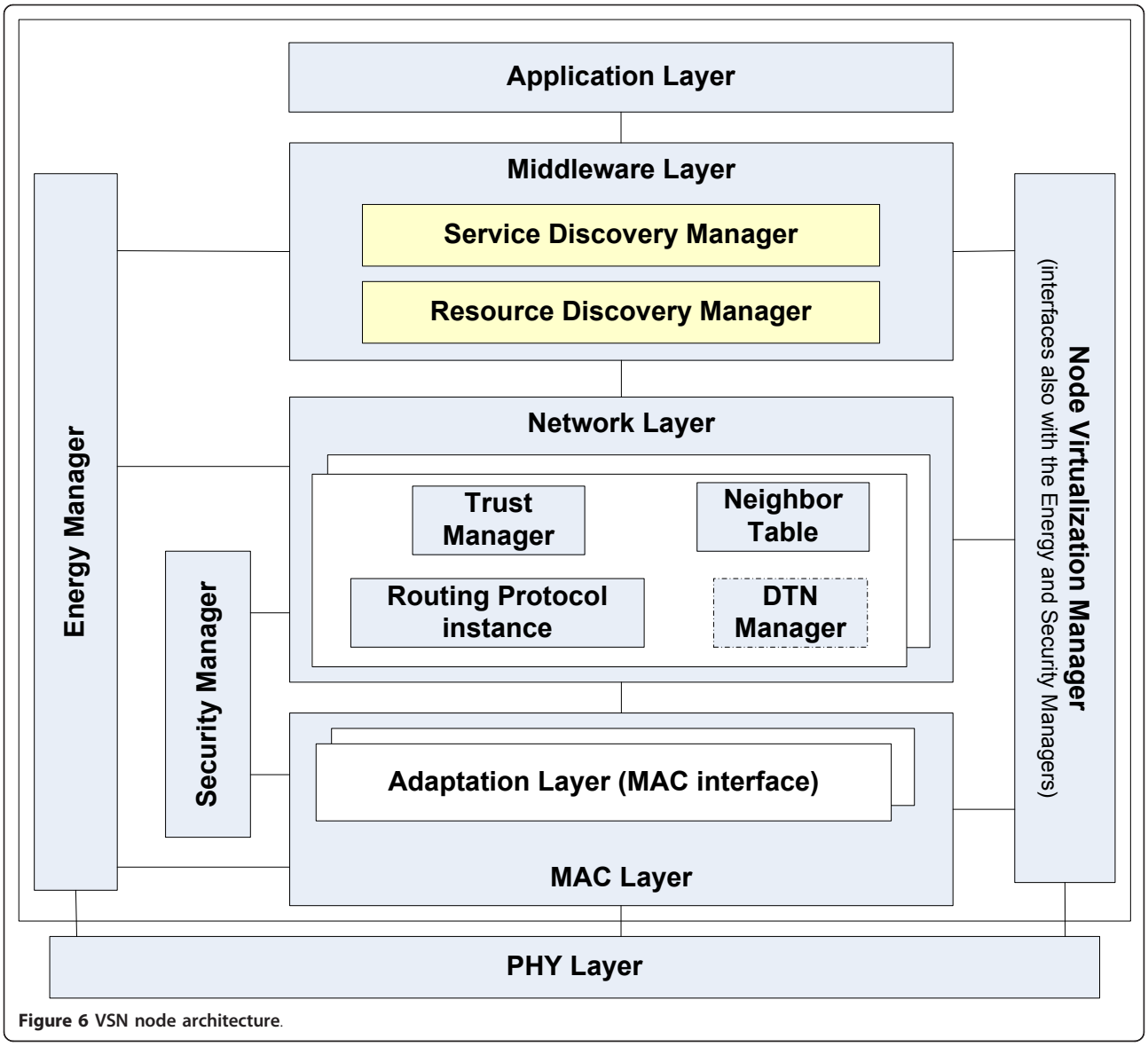

Figure 6 VSN node architecture.

NVM also maintains an association between the QoS requirements imposed on the delivery of the different services (e.g., in terms of reliability, delay, and trust) and maps different service and resource requests to different processes, routing instances, and configurable MAC protocol parameters. Furthermore, it can enable the Delay-Tolerant Networking (DTN) Manager when the application serviced by a VSN is delay-tolerant. In addition, the NVM interacts with components at middleware layer to store/update the services offered by the sensor node in a dynamic way (e.g., after the completion of a reconfiguration query from VGW).

The role of the middleware layer is twofold: first, to provide an abstracted and standard view of supported operational services (in particular, separated from the functioning of sensor hardware), and second to enable service and resource discovery management within a WSI. As such, the VSN middleware hides the hardware and software implementation details from the application layer and allows seamless data management between nodes. Additionally, it encompasses mechanisms to enable management of the discovery of services and resources within the WSI and provides, through specific services, the capability for re-purposing existing sensor network deployments and creating VSNs, according to user needs. Regarding the middleware realization, we are currently elaborating on a solution based on the emerging CoAP application protocol [35] running on top of UDP.

In what regards the functions of the network layer, main focus is placed on the RPL routing protocol [38]

as it represents a promising solution for supporting virtualization through the realization of multiple coexistent routing instances. Each one of these instances may have a Neighbor Table component, which will be maintained in this layer to facilitate the routing/forwarding process. In this respect, the selection of proper routing metrics to accommodate different and sometimes contradicting requirements set by different applications, in such a way that consistency, optimality, and loop-freeness routing requirements are met, is an open research topic that we are currently working on.

Tackling with security-related characteristics, the proposed routing solution will specify trust metrics that will be able to investigate and exclude malicious nodes from the traversed path to the destination node (VGW). The attacks that a trust management system can detect depend on the number and type of node behavior aspects that are monitored [39] and range from network up to application layer attacks (data inconsistency). Trading off hardware resource requirements for the realization of the trust system and mitigation of the most probable attacks, the VSN trust system will be capable of detecting the black-hole, selective forwarding, denial of routing service, and selfish behavior attack. All these attacks lead to data loss which (when systematic) lead to loss of connectivity.

Furthermore, the trust-aware routing protocol will make use of possible security services offered by the Security Manager (e.g., encryption in the MAC or network layers) in order to select the optimal (QoS-based) path, according to the user request. Given that encryption aims to ensure data confidentiality, privacy, data integrity and authenticity, and a variety of encryption schemes can be realized, different security levels can be defined. The encryption scheme implemented on a node and/or the offered security level can be used to decide whether the node can be included in a specific routing instance satisfying the application requirements. In this way, the availability of security features represents constraints taken into account by the routing component during the creation of the routing tables. All information related to trust management will be stored and processed by a certain component, called Trust Manager. Through the discrimination between nodes supporting security services and through the implementation of a trust management system, the proposed VSN defends against network layer attacks and data confidentiality, integrity, and authentication attacks.

In case there is a lack of continuous network connectivity and the running applications can tolerate delays, the presented VSN approach may optionally make use of a DTN mechanism. This mechanism is handled by the DTN Manager, which closely collaborates with a specific routing protocol instance and takes care of communication when a disconnection in the network cannot be repaired by the routing protocol's maintenance mechanisms.

In the VSN case, DTN is used when a formerly connected WSI, becomes disconnected for a long period of time, and therefore partitioned in a number of smaller WSIs. DTN basically relies on some nodes' mobility in order to deliver data among these WSIs and establish end-to-end communication. The DTN mechanism can be activated when the routing protocol (specifically, the component of the protocol that is responsible for neighbor discovery) detects disconnection in the network which cannot be repaired by the protocol's mechanisms. Also, upon detection of the re-establishment of connectivity, the network layer can deactivate the DTN Manager.

According to the specific DTN protocol used, the DTN Manager can include functionalities, such as node addressing, buffering, queue management policies, metric calculations for probabilistic delivery rate, etc. These functionalities and buffers can be assumed as being internal to the DTN Manager, but some structures can be shared with the routing protocol (for instance a neighbor discovery process and the neighbor table).

To support the concept of virtualization, different MAC layers may be instantiated in a wide variety of applications. Apart from the MAC layer responsibility to manage transmission of packets from the upper layers across the physical wireless channel to a neighboring node, an interface to Energy Manager is foreseen in order for the MAC layer to manage issues related to RF energy (e.g., transmission power and sleep times). Multiple instantiations of the MAC Adaptation Layer will act as an interface to upper layers (network layer) and cross-layer entities (Energy Manager and NVM) to tune MAC layer parameters (e.g., to instantiate different time/frequency scheduling patterns). In the proposed node architecture, the challenge at MAC is to support different hardware owners and for each different applications. For example, in subsequent developments related to the emerging IEEE 802.15.4e, this is solved by having different superframes with different duty cycles running in parallel. Other link layer technologies, such as IEEE 802.15.4-2003/2006, would need to handle such virtualization challenge sequentially and thus, not to be depleted quickly, need to be highly energy efficient.

Finally, The Energy Manager component is responsible for the transmission power level of the sensor node as well as for the provision of information regarding the remaining battery energy. Under certain circumstances and depending on the signal strength, this information can be used to decrease transmission power and, thus,

prolong network lifetime, or even achieve load balancing.

5. Interfaces between system components

The interactions between the system components, namely the VSN application, the DVNS, the VSN Manager, the VGW, and the sensor node, are realized through the interfaces depicted in Figure 7. Prior to describing the main operations that are executed over these interfaces we briefly discuss the phases of the application's lifetime in order to depict in a more clear way the interactions involved between components.

The lifetime of VSN applications is divided in three phases:

- *Creation*: During this phase, the user and the VSP negotiate and agree a "service contract". In this phase, the user describes to the VSP a desired service, and the VSP proposes its possible implementation based on the availability of VSN resources. This interactive cycle is repeated through successive adjustments of the description of the desired services, proceeding towards a more and more detailed qualification of the desired services and how it will be provided. At the end of this interactive negotiation, the user and the VSP agree on a detailed description of the application and of the conditions under which it is to be provided. When this agreement is reached, the creation phase is over and the service can begin.
- *Execution*: During this second phase, the VSN provides the service requested, and the application consumes it. The execution of the service is taken care of by the network, automatically and transparently to the user. This phase can be very short (execution of a query and immediate return of the query result) or can be long lasting; in any case, while the service is being executed the VSN Manager takes care of its execution.
- *Termination*: The end criteria for a service can be implicit in the service requested (this is the case, for instance, when the service is the execution of a single, immediate query), or the service can explicitly be terminated by the application (this typically happens for long-lasting services). Termination conditions may also be agreed at service negotiation (e.g., "for one day", or "until this predicate becomes true").

5.1. Interface between application and DVNS

The DVNS is the first contact point between the applications and the VSN system. The applications may run either at the end-user's machine or be hosted by the VSP. In either case, functions provided by this interface are used during the application initiation/creation and preliminary service negotiation phase. The application uses the following functions which are provided by the DVNS:

- *Authenticate_User*: This is used at the beginning of the application creation phase in order for the user to request authentication.
- *Browse_Services*: the application uses this function repeatedly, to qualify in growing detail the desired service.
- *Start_Negotiation*: this is a request sent by the application in order to declare that it wishes to proceed with negotiation for a specified service offered by the system.

5.2. Interface between DVNS and VSN manager

As soon as the nature of the service has been clarified and the user application has declared to the DVNS its intention to proceed with service negotiation, the DVNS

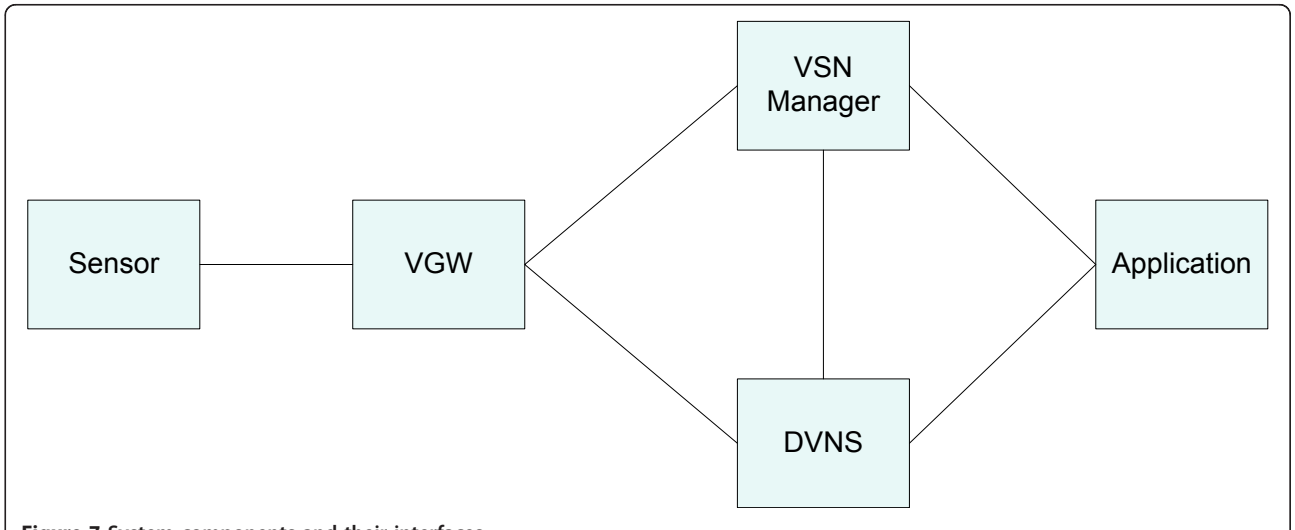

Figure 7 System components and their interfaces.

spawns a new instance of the VSN Manager to handle the service negotiation and provisioning. This task is supported by the *Create_Instance* function of the DVNS-VSN Manager interface.

5.3. Interface between application and VSN manager

This interface is used for service negotiation and initiation between the application and the VSN Manager and for data delivery from the VSN Manager to the application. When the negotiation phase is complete, the user is aware of the current status of the requested services described as up-to-date operational services. The following functions are supported over this interface:

- *Negotiate_Service*: It is used by user/application to negotiate service provisioning with the VSP. This function is also used for service renegotiation in case changes to the service parameters are requested by the application (e.g., extension of the planned duration, modification of the geographical scope of the service, change of the data gathering rate, etc.). It is also used to terminate a service explicitly.
- *Initiate_Service*: This function is used by the Application after successful negotiation with the VSN Manager to denote intention for service execution.
- *Retrieve_Service_Data*: It is used by the application to ask for data in a synchronous manner (i.e., on a request/response basis) according to the specified service.
- *Publish_Data*: It is used by the VSN Manager to feed the application with the data that it has requested. This is particularly important for applications whose primary purpose is data gathering. However, this function is also used for other types of applications because it is through the *Publish_Data* function that the VSN Manager is able to communicate to the application relevant events like, for example, the inability to accomplish the required service.

5.4. Interface between DVNS and VGW

Using this interface, the DVNS retrieves the list of services offered by the WSI under the VGW's control. In addition, this interface is used by the VGW to notify the DVNS of services updates and to allow for registration of a WSI with the DVNS. The following functions are provided by this interface:

- *Publish_Services*: When invoking it, the DVNS queries all known/registered VGWs for discovering new services, removing services not offered anymore or updating changes in the services provided by WSIs.
- *Update_Services*: The DVNS provides this function in order to allow a VGW to asynchronously update the list of WSI supplied services. The function can be invoked on a periodic or on-demand basis.
- *Update_WSI*: This provides the capability for new WSIs accessible by the service provider to be registered in the Services Registry (part of the DVNS) so that applications can access them.

5.5. Interface between VSN manager and VGW

This interface allows the VSN Manager to retrieve the current status of a WSI regarding its available services and resources and to request service initiation. It is also used by the VGW to update its status and provide requested data to the VSN Manager. The functions of this interface include the following:

- *Update_Status*: The VGW provides this function to allow the VSN Manager to get an up-to-date view of the available operational services and resources at the involved WSI. Based on the VGW response, the VSN Manager can provide to the application further feedback to be used in the negotiation phase (e.g., for service parameter reconfiguration).
- *Initiate_WSI_Service*: This function is used by the VSN Manager to communicate to the VGW requests for service initiation. Execution of this function involves the enforcement of a VSN at the WSI level.
- *Publish_WSI_Data*: This function is used by the VGW to export data published by sensor nodes to the appropriate VSN Manager.
- *Retrieve_Service_Data*: It is used to support data retrieval from the VGW on a request/response fashion.
- *Update_WSI_Status*: This is used by the VGW to communicate to the VSN Manager changes in the status of the corresponding WSI supplied services and resources.

5.6. Interface between VGW and sensor node

Using this interface, the VGW becomes aware of the services and resources that are available at the sensor node and the sensor node receives requests for reconfiguration of supporting services and resources. Furthermore, functions of this interfaces are used for transferring data from the sensor node to the VGW according to the publish/subscribe or request/response

communication models. The functions of this interface include the following:

- *Get_Supported_Services*. This function is used by the VGW to query the services supported by the sensor node.
- *Get_Available_Resources*: The function is used by the VGW to retrieve the list of resources provided by a sensor node.
- *Retrieve_Service_Data*: This is used by the VGW to trigger data collection from sensor nodes in a synchronous manner.
- *Subscribe*: The VGW uses this function to send requests for subscription to particular services of a sensor node. The requests can be sent in a unicast or broadcast fashion depending on the user query. The subscription request is expressed through a conjunction of constraints over attribute values, in a form of a tuple (attribute, operator, value).
- *Rcv_reconfiguration*: Reconfiguration of basic functionalities such as scheduling schemes or routing algorithms is handled through this function. Besides the basic reconfiguration of certain parameters dealing with the communication protocols (e.g., MAC duty cycle), this function also deals with the reconfiguration of supporting services and resources (e.g., reconfiguration of the sampling rate for the sensed attribute).
- *Get_Status*: This is used by the VGW to get current information on usage associated to a service or to the consumption of resources. Resource information may include, among others, the number of instances that a sensor node participates in (either as a router or as a sensing device) or memory/processor allocation.
- *Rcv_notification*. When the constraints of the subscription to a service are satisfied, the VGW receives, through this function, a notification from the sensor node. The notification is of the form (attribute, value).
- *Notify_supported_services*: Using this function a sensor node can proactively notify the VGW about its supported services.
- *Notify_available_resources*. This is used by a sensor node to communicate to the VGW, in a proactively manner, its available resources.

6. Service registration, service negotiation, and session establishment

The registration of the services in the VSP involves two steps (Figure 8). In the first step, the services of all WSIs that have a service agreement with the VSP are discovered by their respective VGW. To accomplish this, the VGW collects information related to services provided by all sensors comprising the WSI, as well as their available resources.

Prior to any service and resource discovery, a path between the VGW and every sensor node has to be established. To do this, the VGW configures the operation of the MAC layer as well as the operation of the routing protocol within the WSI, which deals with the creation of multiple instances of routing planes (e.g., several RPL Destination Oriented Directed Acyclic Graphs, with different metrics per RPL instance). By using the *Get_supported_services* and *Get_available_resources* functions, the VGW can collect (and periodically update) all services/resources available to the sensor nodes in order to support (new or existing) user queries. Another option consists in sensor nodes announcing proactively their provided services/resources to the VGW, using the *Notify_supported_services* and *Notify_available_resources* functions.

In the second step, the collected services and resources are sent to the DVNS by calling the *Update_WSI* (in case of new WSIs) and *Update_services* functions, or can be browsed (possibly in a periodic fashion) by the DVNS via the *Publish_services* function. All discovered services are stored in the Services Registry module of the DVNS.

At the beginning of the application creation phase, and prior to service consumption, the user interacts with the DVNS to negotiate the provided service (Figure 9). The first operation is to qualify himself/herself, by using the *Authenticate_User* function. After doing that, the user becomes aware of the available services by calling (possibly in a repeatable fashion) the *Browse_Services* function. As soon as the desired service has been identified, the user's application notifies the DVNS of the intention to negotiate the implementation details of the service (QoS, guaranteed level of services, billing, fallback policies, etc.) by sending the *Start_Negotiation* message. The service negotiation and the subsequent service delivery processes are handled by an instance of the VSN Manager, which is created for this purpose by the DVNS through the execution of the *Create_Instance* function. To serve negotiation requests, the VSN Manager interacts with the involved VGWs, by calling the *Update_Status* function, to retrieve the list of available WSI services and resources. The VGW will, in turn, communicate with the sensor nodes, using the *Get_Status* function, to obtain an updated view of the available services and resources at the node level.

After the service negotiation phase is completed successfully, the application asks for service initiation by sending an *Initiate_Service* request to the VSN Manager (Figure 10). Based on the requested service, the VSN Manager identifies the WSI(s) that must be engaged for

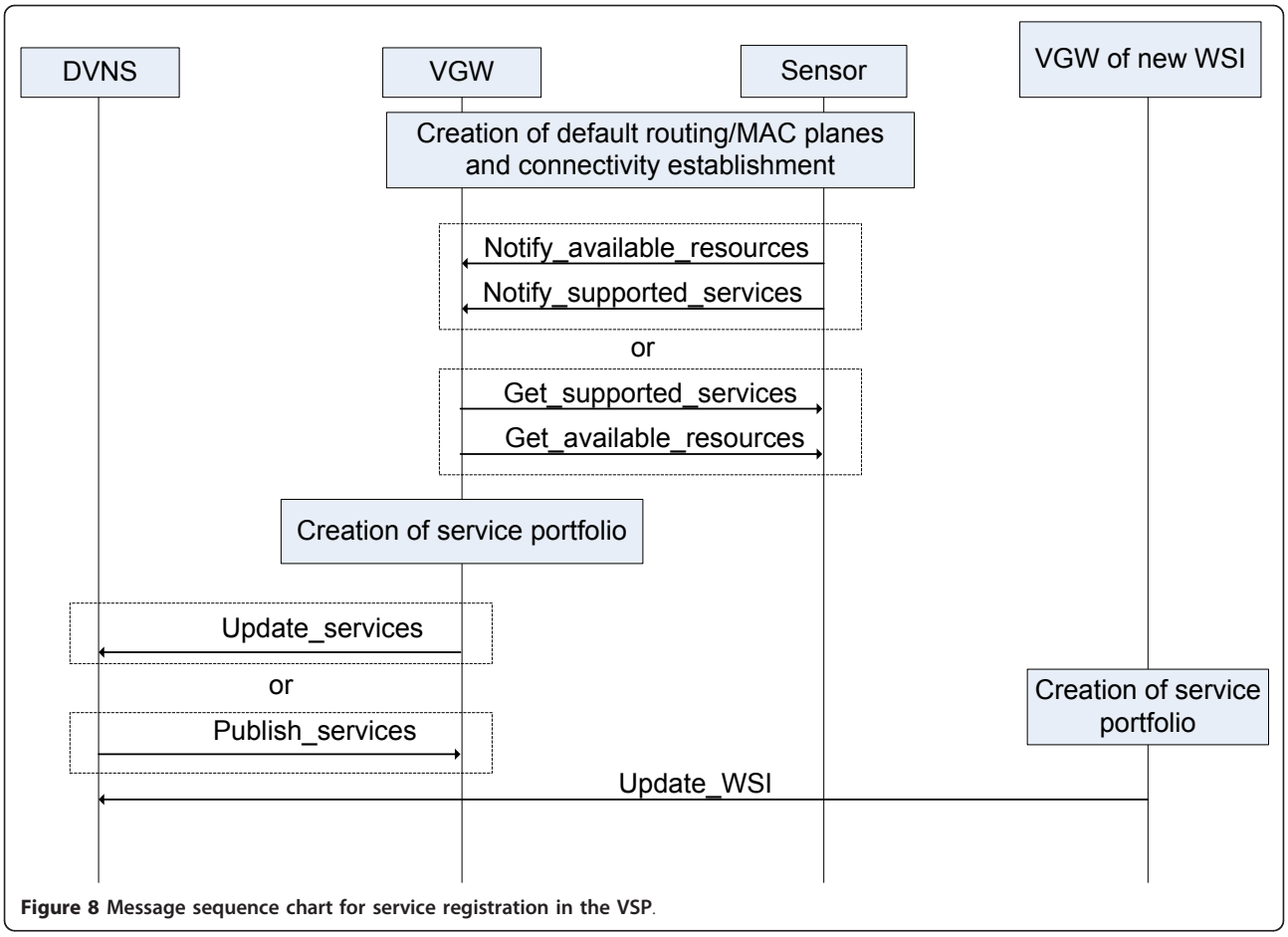

Figure 8 Message sequence chart for service registration in the VSP.

the provisioning of the service and contacts its(their) VGW(s) through the *Initiate_WSI_Service* function call. This function is used to:

- Specify the number of nodes to be used for the provisioning of the service (e.g., exact, maximum or minimum number of nodes).
- Specify the operational services to be subscribed on the selected nodes.
- Specify the parameters that must be communicated to the WSI Routing Driver of the VGW.

The VGW uses the *Rcv_Reconfiguration* function to enforce/configure a VSN on the WSI and the *Subcribe*

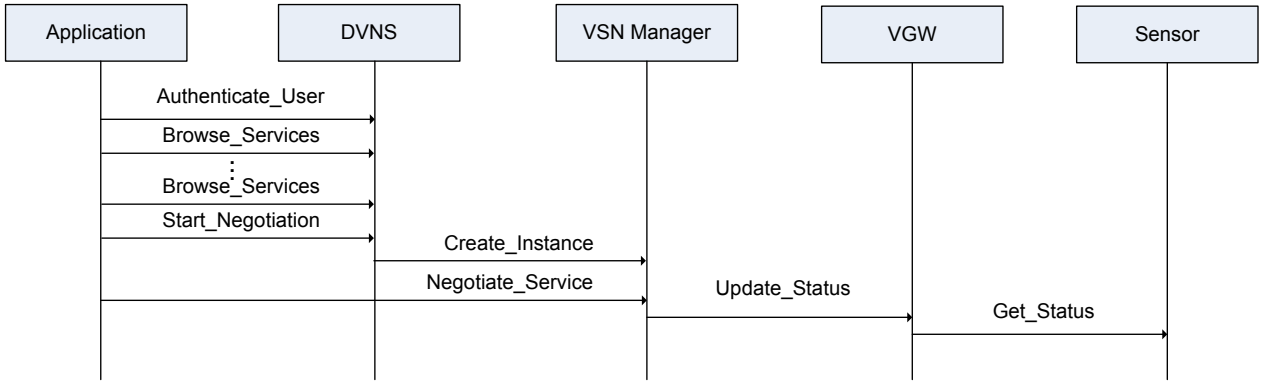

Figure 9 Message sequence chart for service negotiation.

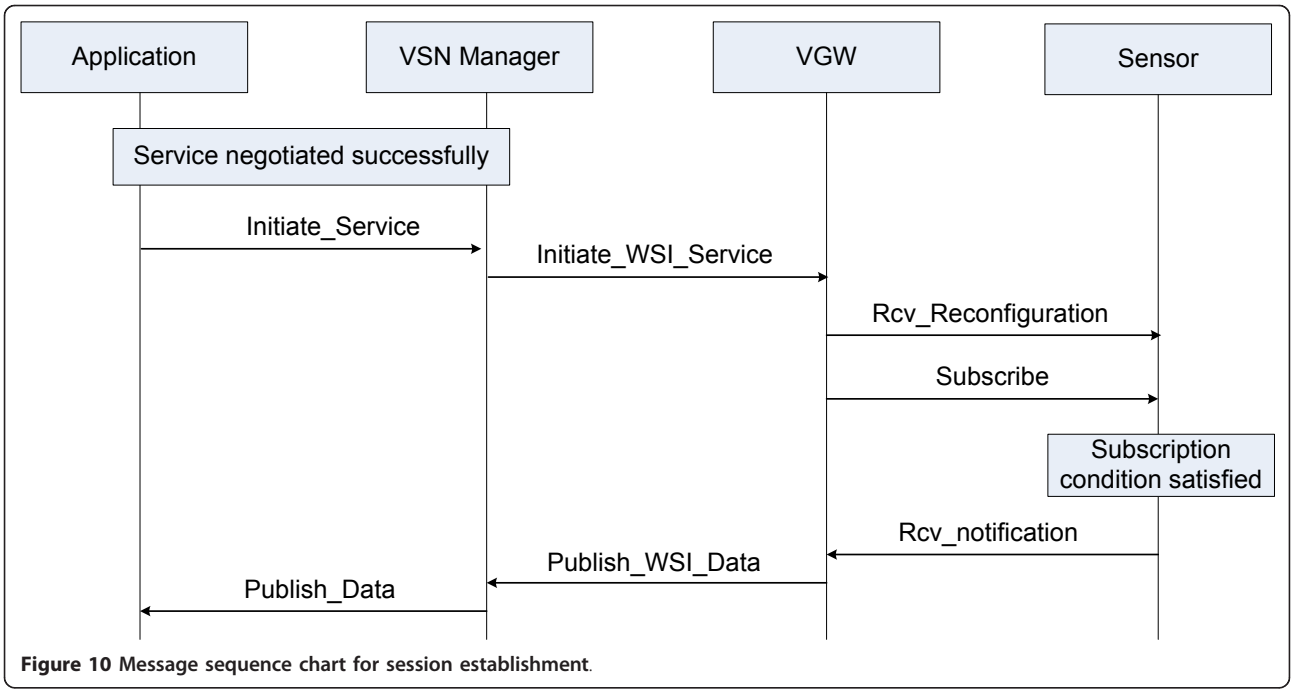

Figure 10 Message sequence chart for session establishment.

function to register itself as a recipient for data produced by the sensor when the condition of the subscription is satisfied. These data are communicated back to the VGW through the *Rcv_notification* function. In order to export data published by sensor nodes towards the appropriate VSN Manager instance, the VGW uses the function *Publish_WSI_data*.

Figure 10 illustrates the sequence of messages involved in sensor data delivery based on the publish/subscribe communication paradigm. For service execution on a request/response basis the Application sends a *Retrieve_Service_Data* request to the VSN Manager. In case cached data are available, the VSN Manager may return the requested data immediately; otherwise, the *Retrieve_Service_Data* is sent all the way to the sensor node.

7. Conclusions and future study

We have presented the system architecture of a VSN system that extends the concept of virtualization down to the sensor nodes while ensuring advanced performance and service exploitation. The global virtualization architecture is built around (a) software components that allow user applications to negotiate, execute, and monitor a service obtained by composing basic services provided by sensors, and (b) the architectural design of the physical VSN-aware sensor node and the VSN-aware gateway. The proposed architecture accommodates the flexibility and adaptability required to achieve an energy-efficient realization of VSNs mainly through the definition of both horizontal and vertical functional layers in the sensor node that allow for different instantiations of the MAC and routing protocols to support the concept of virtualization and provide efficient support to different VSN applications that may run on the node.

Currently, we have developed simulation models for the MAC and trusted routing components and complete software designs for all parts of the proposed VSN provisioning system. The simulation study has produced promising results regarding the capability of the proposed link and routing layer solutions to (a) support the virtualization concepts described in this article, and (b) accommodate applications with diverse communication requirements in terms of delay and trust. Future study will address the implementation of all system components, the development of testbeds that will federate resources from heterogeneous sensor network platforms, and the experimental evaluation of the proposed solutions.

Abbreviations

DTN: Delay-Tolerant Networking; DVNS: Dynamic Virtual Network Server; LGW IF: Legacy Gateway Interworking Functions; LGW: Legacy Gateway; LSN: Legacy Sensor Node; M2M: Machine-to-Machine; MAC: Medium Access Control; NVM: Node Virtualization Manager; OS: Operating System; PGW: Proprietary Gateway; PSN: Proprietary Sensor Node; QoS: Quality of Service; VaSN: VSN-aware Sensor Node; VGW: VSN Gateway; VSN: Virtual Sensor Network; VSP: VSN Service Provider; WSI: Wireless Sensor Island; WSN: Wireless Sensor Network.

Acknowledgements

This publication is based on work performed in the framework of the Project VITRO-257245, which is partially funded by the European Community. The authors would like to acknowledge the contributions to the VITRO Project of colleagues from Hellenic Aerospace Industry, Thales

Communications SA, Telefonica Investigation Y Desarrollo SA, Centre Technologic de Telecomunicacions de Catalunya, Research Academic Computer Technology Institute, Technological Educational Institute of Chalkida, Zodianet SAS, Wlab SRL, and SELEX Sistemi Integrati S.P.A.

Author details
[1]Technological Educational Institute of Chalkida, 34400 Psachna, Euboea, Greece [2]Centre Tecnològic de Telecomunicacions de Catalunya (CTTC), Av. Carl Friedrich Gauss 7, 08860 Castelldefels, Barcelona, Spain

Competing interests
The authors declare that they have no competing interests.

Received: 1 October 2011 Accepted: 6 April 2012 Published: 6 April 2012

References

1. AP Jayasumana, Q Han, TH Illangasekare, Virtual sensor networks - a resource efficient approach for concurrent applications, in *Proceedings of the International Conference on Information Technology (ITNG '07)*, Washington, DC, USA, pp. 111–115 (2007)
2. R Tynan, GMP O'Hare, MJ O'Grady, C Muldoon, Virtual sensor networks: an embedded agent approach, in *Proceedings of the International Symposium on Parallel and Distributed Processing with Applications (ISPA-08)*, Sydney, Australia, pp. 926–932 (Dec 2008)
3. NMMK Chowdhury, R Boutaba, A survey of network virtualization. Comput Netw. **54**(5), 862–876 (2010). doi:10.1016/j.comnet.2009.10.017
4. The VITRO project http://www.vitro-fp7.eu. Accessed 1 Feb 2012
5. P Levis, S Madden, J Polastre, R Szewczyk, K Whitehouse, A Woo, D Gay, J Hill, M Welsh, E Brewer, D Culler, TinyOS: an operating system for sensor networks, in *Ambient Intelligence*, ed. by Weber W, Rabaey J, Aarts E (Springer, 2005), pp. 115–148
6. A Dunkels, B Gronvall, T Voigt, Contiki–a lightweight and flexible operating system for tiny networked sensors, in *Proceedings of the 29th Annual IEEE International Conference on Local Computer Networks (LCN '04)*, Washington, DC, USA, pp. 455–462 (2004)
7. S Bhatti, J Carlson, H Dai, J Deng, J Rose, A Sheth, B Shucker, C Gruenwald, A Torgerson, R Han, MANTIS OS: an embedded multithreaded operating system for wireless micro sensor platforms. Mob Netw Appl. **10**(4), 563–579 (2005). doi:10.1007/s11036-005-1567-8
8. E Trumpler, R Han, A systematic framework for evolving TinyOS, in *Proceedings of the 3rd Workshop on Embedded Networked Sensors (EmNets 2006)*, Cambridge, MA, USA, pp. 61–65 (May 2006)
9. P Corsini, P Masci, A Vecchio, Configuration and tuning of sensor network applications through virtual sensors, in *Proceedings of the 4th Annual IEEE International Conference on Pervasive Computing and Communications Workshops (PERCOMW'06)*, Pisa, Italy, pp. 316–320 (13-17 March 2006)
10. L Caroprese, C Comito, D Talia, E Zumpano, A logic approach to virtual sensor networks, in *Proceedings of the Fourteenth International Database Engineering & Applications Symposium (IDEAS '10)*, Montreal, Quebec, Canada, pp. 149–156 (2010)
11. HMND Bandara, AP Jayasumana, TH Illangasekare, Cluster tree based self organization of virtual sensor networks, in *Proceedings of the IEEE Globecom Workshops*, New Orleans, USA, pp. 1–6 (November 2008)
12. P Levis, D Culler, Maté: a tiny virtual machine for sensor networks, in *Proceedings of the 10th International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS-X)*, San Jose, CA, USA, pp. 85–95 (October 2002)
13. P Levis, D Gay, D Culler, Active sensor networks, in *Proceedings of the 2nd USENIX/ACM Symposium on Networked Systems Design and Implementation (NSDI'05)*, Boston, MA, USA, pp. 343–356 (May 2005)
14. Y Yu, LJ Rittle, V Bhandari, JB Lebrun, Supporting concurrent applications in wireless sensor networks, in *Proceedings of the 4th International Conference on Embedded Networked Sensor Systems (SenSys'06)*, Boulder, Colorado, USA, pp. 139–152 (2006)
15. J Koshy, R Pandey, VMSTAR: synthesizing scalable runtime environments for sensor networks, in *Proceedings of the 3rd International Conference on Embedded Networked Sensor Systems (SenSys'05)*, San Diego, California, USA, pp. 243–254 (2005)
16. P Bonnet, JE Gehrke, P Seshadri, Towards sensor database systems, in *2nd International Conference on Mobile Data Management (MDM) LNCS*, vol. 1987. Springer, pp. 3–14 (2001)
17. S Madden, MJ Franklin, JM Hellerstein, W Hong, TinyDB: an acquisitional query processing system for sensor networks. ACM Trans Database Syst. **30**(1), 122–173 (2005). doi:10.1145/1061318.1061322
18. K Aberer, M Hauswirth, A Salehi, The global sensor networks middleware for efficient and flexible deployment and interconnection of sensor networks. Technical Report, Ecole Polytechnique Federale de Lausanne (EPFL) (2006)
19. D Gelernter, Generative communication in Linda. ACM Comput Surv. **7**(1), 80–112 (1985)
20. C Curino, M Giani, M Giorgetta, A Giusti, AL Murphy, GP Picco, TinyLIME: bridging mobile and sensor networks through middleware, in *Proceedings of the 3rd IEEE International Conference on Pervasive Computing and Communications (PerCom 2005)*, Kauai, Hawaii, pp. 61–72 (March 2005)
21. K Terfloth, G Wittenburg, J Schiller, FACTS–a rule-based middleware architecture for wireless sensor networks, in *Proceedings of the First International Conference on Communication System Software and Middleware (Comsware 2006)*, New Delhi, India, pp. 1–8 (January 2006)
22. E Souto, G Guimarães, G Vasconcelos, M Vieira, N Rosa, C Ferraz, J Kelner, Mires: a publish/subscribe middleware for sensor networks. Personal Ubiquit Comput. **10**(1), 37–44 (2006). doi:10.1007/s00779-005-0038-3
23. S Lai, J Cao, Y Zheng, PSWare: a publish/subscribe middleware supporting composite event in wireless sensor network, in *Proceedings of the IEEE International Conference on Pervasive Computing and Communications (PerCom 2009)*, Galveston, TX, USA, pp. 1–6 (March 2009)
24. T Liu, M Martonosi, Impala: a middleware system for managing autonomic, parallel sensor systems, in *Proceedings of the 9th ACM SIGPLAN symposium on Principles and Practice of Parallel Programming (PPoPP'03)*, San Diego, California, USA, pp. 107–118 (2003)
25. F Oldewurtel, J Riihijarvi, K Rerkrai, P Mahonen, The RUNES architecture for reconfigurable embedded and sensor networks, in *Proceedings of the Third International Conference on Sensor Technologies and Applications (SENSORCOMM'09)*, Athens, Greece, pp. 109–116 (June 2009)
26. PJD Cid, D Hughes, J Ueyama, S Michiels, W Joosen, DARMA: adaptable service and resource management for wireless sensor networks, in *Proceedings of the 4th International Workshop on Middleware Tools, Services and Run-Time Support for Sensor Networks (MidSens '09)*, Urbana Champaign, Illinois, USA, pp. 1–6 (2009)
27. CL Fok, GC Roman, C Lu, Rapid development and flexible deployment of adaptive wireless sensor network applications, in *Proceedings of the 25th IEEE International Conference on Distributed Computing Systems (ICDCS'05)*, Columbus, OH, USA, pp. 653–662 (June 2005)
28. I Chatzigiannakis, G Mylonas, S Nikoletseas, jWebDust: a java-based generic application environment for wireless sensor networks, in *Proceedings of the 1st IEEE International Conference on Distributed Computing in Sensor Systems (DCOSS '05)*, Marina del Rey, CA, USA, pp. 376–386 (2005)
29. J Shneidman, P Pietzuch, J Ledlie, M Roussopoulos, M Seltzer, M Welsh, Hourglass: an infrastructure for connecting sensor networks and applications. Technical Report, Harvard TR-21-04 (2004)
30. Botts M, Percivall G, Reed C, Davidson J (eds.), OGC sensor web enablement: overview and high level architecture. White Paper Version 3 (Open Geospatial Consortium Inc., 2007)
31. A Antoniou, I Chatzigiannakis, A Kinalis, G Mylonas, S Nikoletseas, A Papageorgiou, A peer-to-peer environment for monitoring multiple wireless sensor networks, in *Proceedings of the 2nd ACM workshop on Performance monitoring and measurement of heterogeneous wireless and wired networks (PM2HW2N'07)*, Chania, Greece, pp. 132–135 (2007)
32. Botts M, Robin A, Davidson J, Simonis I (eds.), OGC sensor web enablement: architecture document. Discussion Paper Version 1 (Open Geospatial Consortium Inc., 2006)
33. X Chu, R Buyya, Service Oriented Sensor Web, in *Sensor Networks and Configuration: Fundamentals, Standards, Platforms, and Applications*, ed. by Mahalik NP Springer-Verlag, Berlin Heidelberg, pp. 51–74 (2007)
34. ETSI TS 102 690 V1.1.1, Machine-to-Machine communications (M2M); Functional architecture. (2011)
35. Z Shelby, K Hartke, C Bormann, B Frank, Constrained application protocol (CoAP). IETF Internet-Draft draft-ietf-core-coap-08, work in progress http://tools.ietf.org/html/draft-ietf-core-coap-08 (2011)
36. V Tsiatsis, A Gluhak, T Bauge, F Montagut, J Bernat, M Bauer, C Villalonga, P Barnaghi, S Krco, The SENSEI Real World Internet Architecture, in *Towards the Future Internet - Emerging Trends from European Research*, ed. by

Tselentis G, Galis A, Gavras A, Krco S, Lotz V, Simperl E, Stiller B, Zahariadis T (IOS Press, 2010), pp. 247–256

37. D Wisely, H Aghvami, S Gwyn, Th Zahariadis, J Manner, V Gazis, N Houssos, N Alonistioti, Transparent IP radio access for next generation mobile networks. IEEE Wirel Commun Mag. **10**(4), 26–35 (2003). doi:10.1109/MWC.2003.1224976
38. Winter T, Thubert P (eds.), RPL: IPv6 Routing Protocol for Low power and Lossy Networks (IETF Internet-Draft draft-ietf-roll-rpl-19, work in progress, 2011) http://tools.ietf.org/html/draft-ietf-roll-rpl-19
39. Th Zahariadis, H Leligou, P Trakadas, S Voliotis, Trust management in wireless sensor networks. Eur Trans Telecommun. **21**(4), 386–395 (2010)

doi:10.1186/1687-1499-2012-135
Cite this article as: Sarakis *et al.*: A framework for service provisioning in virtual sensor networks. *EURASIP Journal on Wireless Communications and Networking* 2012 **2012**:135.