High-Level Internet of Things Applications Development Using Wireless Sensor Networks

Zhenyu Song, Mihai T. Lazarescu, Riccardo Tomasi, Luciano Lavagno and Maurizio A. Spirito

Abstract Wireless Sensor Networks (WSN)-based data gathering solutions were envisioned from the beginning of the Internet of Things (IoT) paradigm because of their important characteristics such as low-cost, long-term versatile sensing and actuation capabilities, and distributed resilient bidirectional communications. However, many technologies converge into WSN platforms from several disciplines. Their complexity and rapid pace of change may prevent most WSN and IoT players to keep up, since they need to focus on low cost and short development time targets. Also important is the quality assurance to prevent the degradation of the overall reliability perception needed for wide WSN adoption. Consequently, a versatile and reusable WSN platform may benefit both industry and academic research by speeding up and facilitating the distributed WSN application programming. The framework presented provides the application developers with a set of development tools and software component libraries that allow high-level application architecture description, graphical definition and simulation of the component behaviour, and automatic generation of both network simulation models and code for target node programming.

Z. Song · R. Tomasi · M. A. Spirito
Pervasive Technologies Area, Istituto Superiore Mario Boella, Turin, Italy
e-mail: song@ismb.it

R. Tomasi
e-mail: riccardo.tomasi@polito.it; tomasi@ismb.it

M. A. Spirito
e-mail: spirito@ismb.it

M. T. Lazarescu (✉) · R. Tomasi · L. Lavagno
Dipartimento di Elettronica e Telecomunicazioni, Politecnico di Torino, Turin, Italy
e-mail: mihai.lazarescu@polito.it

L. Lavagno
e-mail: luciano.lavagno@polito.it

S. C. Mukhopadhyay (ed.), *Internet of Things*, Smart Sensors, Measurement and Instrumentation 9, DOI: 10.1007/978-3-319-04223-7_4,

1 Internet of Things and Wireless Sensor Networks

More than a decade ago, the Internet of Things (IoT) paradigm was coined [1], in which the computers would be able to access data about objects and the environment without human interaction. Complementing human-entered data was important in order to increase the accuracy and pervasiveness of physical world information in ICT systems at a sustainable cost.

Initially adopted and popularized by the Auto-ID Center at MIT and market analysts [2, 3], the IoT concept received growing attention until recently it became one of the most hyped terms. The vision encompass the ability to uniquely identify, represent and access objects at any time and anywhere, in an Internet-like virtual structure.

Traditionally, radio-frequency identification (RFID) was one technology considered key enabler for the IoT paradigm. Alongside, other low-cost identification and communication technologies are also used: device-centric technologies, such as near field communication (NFC), Bluetooth, barcodes, Quick Response (QR) codes, ZigBee, digital watermarking, and networking technologies, like 3GPP Long Term Evolution/Long Term Evolution-advance (3GPP LTE/LTE-A), Vehicular Ad-hoc Networks, Wireless Sensor Networks, and Wireless Local Area Networks. These blend the IoT concept into the Internet of Everything (IoE) vision, where nearly *everything* is connected to the Internet using machine-originated or machine-to-machine (M2M) communications. Almost all smartphones, tablets and PCs are always connected, and so are becoming other objects (like home appliances, cars, clothes, door locks, light bulbs, toys, thermostats, vending machines, industrial instruments, data collecting sensors) to provide a variety of services without human intervention.

The sheer versatility of the IoT paradigm makes it suitable for an ever increasing variety of applications. In turn, these applications make use and rely on the many diverse technologies and devices that concur to its realization. And all this widening diversity makes increasingly difficult to define "typical" requirements for the devices and their programming [4].

Along the RFID, Wireless Sensor Networks (WSNs) were considered a key technology for IoT paradigm capillary pervasiveness able to bring richer, versatile low-cost sensing and actuation to applications. Intrinsically versatile, WSNs are suitable for a wide variety of applications and systems, with very diverse requirements and characteristics. However, this makes difficult to define specific application requirements and research directions, WSNs remaining a challenging multidisciplinary area. An efficient implementation very often requires a good collaboration between users, application domain experts, hardware designers, and software and firmware developers.

As a long-term value, the IoT technologies allow going beyond application-specific "vertical" pervasive systems. A single "horizontal" ecosystem of interoperable objects and services can seamlessly co-operate to provide new services and optimization opportunities. In this vision, a key role is attributed to the tools that support their modeling, design and development, mainly because these can unify

computing paradigms and development workflows among vendors, architects, integrators and application developers.

2 Developing IoT Systems: Context, Scenarios and Challenges

We will analyze several representative scenarios where the IoT technology is used attempting to extract a subset of abstract use-cases and their common key features. Based on this analysis, we will then extract the key challenges that impact on the IoT development tools and workflows.

2.1 A Reference Workflow for IoT Developments

Figure 1 shows a possible reference development workflow for IoT applications that allows to introduce the IoT development context.

Traditionally, requirement definition starts by considering scenarios of independent vertical domains or markets, also named "silos," e.g., Intelligent Transportation System, Smart Grids, Logistics.

Multiple use cases can be defined within any of these scenarios. This phase normally entails a thorough analysis of domain-specific aspects, including the analysis of the expected uses and needs of key stakeholders and users. Even though a significant part of the IoT vision value lies in its "horizontality," a main usage scenario is usually needed to secure the initial deployment funding.

The initial analysis can produce requirements of different styles and granularities. However, for adequate specificity, they must be mapped or split among a pre-defined set of architecture components. Standard Architecture Reference Models (ARMs) are defined to this purpose, such as that of the IoT-A project[1]. For instance, the mapping defines the information view of the system including all data structures, i.e., the models and protocols employed to share information across different components.

High-level architectures are often split into more controllable components within detailed component design phases. These are then implemented by teams with specific expertise (e.g., communication, interoperable services, low-power embedded systems) and skills for specific hardware/software platforms and tools. At this stage, new components are often integrated with COTS (Commercial Off-The-Shelf) devices at various levels. Where available, these provide specific functions within the system under development, including proper interplay with pre-existing legacy systems where needed.

After the integration and the optional validation phase, the system is ready for commissioning. This consists in its deployment and configuration in the final use

[1] The IoT-A consortium. Internet of things architecture. http://www.iot-a.eu/

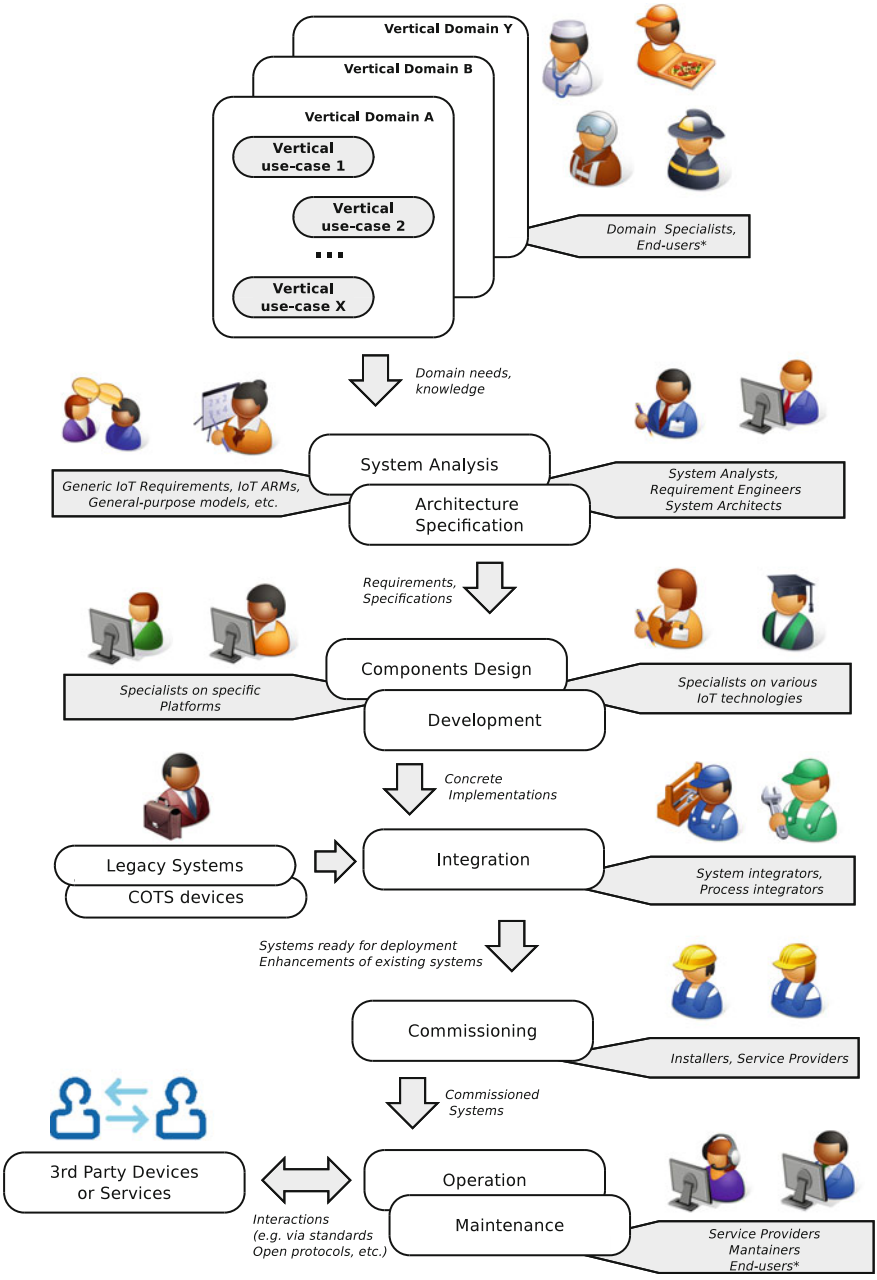

Fig. 1 IoT system design and development work-flow

scenario, and is often associated with the physical delivery to the end-users and their training. Finally, the operational system and receives local or remote maintenance.

The tools can support the development in different phases of the workflow. At high-level architecture definition they help describing and modeling all key aspects of the domain in a consistent information view. They should allow different levels of typing, specification, and semantic annotation for the project elements, such as protocols, key variables and parameters. Such models and parameters, possibly standardized, should be associated to all components and are converted in data structures for project algorithms.

The development-support tools cover a prominent role in detail design and development of specific components. At these levels, they typically link the design high-level abstractions (objects, variables) defined to their specific implementations. They help especially the joint developments by several different experts and to unit test the components.

In this phase, tool interoperability is key. Experts in specific IoT fields should use their specific tools, typically focusing on a well-defined set of technologies. At the same time, these tools should allow sharing models, interfaces and parts of the development process.

During the integration, commissioning and operation stages (including maintenance), the tools should support the development of IoT components able to interoperate with a broad set of COTS systems, legacy devices, and installation and maintenance tools. This may entail supporting the broadest set of available open protocols and standards, where applicable.

IoT Application Scenarios

A wide collection of studies on IoT markets, applications and use cases is available in the scientific, technical and business literature. However, due to high diversity and heterogeneity of the IoT definitions and deployments, a widely accepted taxonomy of IoT applications is still elusive.

Referencing a common IoT application taxonomy is nevertheless important for the IoT community. It simplifies the information exchange and improves the targeting of investments in IoT technical developments. Moreover, it is also a primary tool helping to keep into account the various requirements. Common requirements across different domains shall influence general-purpose IoT features, while domain-specific requirements shall be considered for optional features to be supported by IoT development in specific markets.

This section provides an overview of the main IoT application domains currently identified by the research community. For each domain is suggested a set of key challenges that may potentially impact the IoT development-support tools. A restricted set of possible use cases is analyzed in each domain to identify specific challenges.

The analysis follows the categorization suggested in the IERC (European Research Cluster on the Internet of Things) Clusterbook [5], modified based on other international road-maps, white papers and review articles [4, 6–17].

Smart Cities

Smart Cities vision involves a broad set of applications aiming to make cities more sustainable, safe and enjoyable thanks to new ICT-supported models of cooperation among citizens, institutions and companies (e.g., utilities, service providers). Most applications require a large number of networked devices and systems deployed on city-wide scale, making of Smart Cities a primal playground for IoT technologies [18].

The applications of interest range from systems supporting urban mobility and its safety (e.g., Smart Parking, Traffic Congestion, Intelligent Transportation Systems), to applications monitoring or optimizing assets and critical infrastructures in cities (Structural Health, Smart Lightning, Smart Roads), to systems monitoring and protecting the citizens' quality of life (Noise Urban Maps, Waste Management).

Structural health focuses on monitoring the status of buildings, bridges or other infrastructure, e.g., by sensing their vibrations [19]. The challenges include large scale deployment, unmanned operation, self-healing, processing and detecting events from raw data via (potentially complex) algorithms running on-board the nodes, embedding data transmission and management logic on tiny devices, low-power constraints, inter-node coordination (coherent data time-stamping).

Traffic congestion monitors car and pedestrian traffic leveraging fixed [20] or mobile [21] sensors. The challenges include large scale deployments, integration of data from heterogeneous devices and data sources, interoperability of different vendor sensors, interoperability with other systems (e.g., mobile terminals, legacy traffic monitoring platforms), input and output location-awareness, distribution of intelligence (e.g., due to time constraints), minimizing the commissioning and maintenance effort.

Participatory sensing collects, analyses and shares data through participation of local communities of volunteers [22]. The challenges include working with heterogeneous data and sensors, the need for data annotation and semantic interoperability, integration of crowd-sourced knowledge, interoperability with community-provided devices, trust and privacy.

Smart Environment

The Smart Environment domain include any system monitoring or preventing critical events in wide, unpopulated areas where significant environmental risks have been identified. Such areas require constant monitoring and alerting about critical events, such as fire in forests, landslides and avalanches in mountain areas, earthquakes in seismic areas, exceptional air or water pollution events in heavy industrial or safety-critical areas.

Forest fire detection in remote forest areas to alert public authorities [23]. The challenges include large-scale deployments without fixed communication infrastructure, strict low-power constraints, effective use of energy harvesting, unmanned operation, integration of multi-hop communication solutions, need for efficient data processing on board constrained nodes, real-time reporting of critical events, self-configuration, self-healing.

Remote seismography collects seismic data from remote areas [24], e.g., by means of microphones or seismic-acoustic sensors. The challenges include installations in harsh conditions, need for data-processing techniques to trigger event detection, synchronization and time-stamping of sensed data, reliable data retrieval with constrained bandwidth.

Pollution monitoring of potentially dangerous gases in both urban and remote areas using fixed or mobile sensors [25]. The challenges include data collection from fixed and mobile installations and geotagging, large sets heterogeneous sensor calibration or self-calibration, power supply management.

Smart Water

Smart Water applications include all systems for water monitoring in both civil and natural environments, such as water quality monitoring (e.g., presence of chemicals) in rivers or in water distribution infrastructure, water leakage detection in pipes or buffer tanks, monitoring of levels of rivers, dams and reservoir, e.g., for flood or drought early warning.

Water leakages detection monitors pipes or distribution networks, e.g., by means of pressure, flow or other types of sensors [26]. The challenges include installation and operation in harsh conditions (e.g., underground), time-correlation of spatial data, unmanned operation, combination and fusion of data from heterogeneous sensors possibly provided by different vendors, unmanned management of sensors in remote locations.

Level-monitoring and flood detection for basins, dams, rivers, lakes checks levels and detects floods [27]. The challenges include continuous monitoring and processing of sensed data, lack of communication infrastructure, strict real-time constraints when critical events occurs, resilient communication, distributed processing and monitoring.

Smart Metering

The Smart Metering domain is a wide area of application. It involves remote monitoring (and control) of a large population of networked meters that provide data for accounting and consumption billing of various commodities, such as electricity (both consumed and generated in the so-called "Smart Grid"), water, gas and oil in storage tanks, cisterns or transportation systems (e.g., water pipes, oil pipelines), level of silos stock. The most mature Smart Metering solutions are currently in the electric power distribution market.

Metering in the smart electric grid are applications for monitoring and accounting consumption via remote, automated meters [28]. The challenges include security, privacy, reliability and transparency, interoperable standardized and certified solutions, low-latency and robust communication for more advanced control applications.

Metering of heating systems for monitoring consumption and efficiency of heating systems, e.g., to optimize or to partition heating costs across different units [29]. The challenges include low cost, ease of installation and maintenance, interoperable standardized and certified solutions, interoperability with legacy systems.

Security and Emergency

The Security and Emergency domain, among the first explored in the WSN for IoT field [30], includes protection of areas sensitive to people intrusion (e.g., perimeter access control) or to environmental factors that increase the risks for people and goods (e.g., monitoring liquids in areas with dense electric outlets, detecting presence of gases and leakages in chemical factories, detecting radiations close to nuclear power stations).

Perimeter control and tracking detects and localizes trespassers (e.g., thieves, enemy troops) using distributed sensors and alarms [31]. The challenges include tight security requirements, resilience to interference and disruption, complex data management and processing for intruder detection.

Disaster recovery solutions assist rescuers after major disasters like radiation leaks or floods [32]. The challenges include resilience to interference and disruption (e.g., many sensors disabled or moved), real-time requirements during emergency, interoperability with mobile networks (e.g., aerial vehicles or land robots), geolocation, self-healing and reconfiguration.

Retail

The Retail domain includes systems to simplify stocking (Supply Chain Control), storage (Smart Product Management), marketing (Intelligent Shopping Applications) and selling (NFC Payment) in shops and malls.

Smart shopping employs marketing-oriented applications in retail environments, e.g., to track customers' behaviour or feed targeted advertisements in malls [33]. The challenges include context awareness (e.g., location-dependent behaviour, time awareness), interoperability with existing business systems, location and time-awareness, fusion of heterogeneous data sources to detect user behaviour.

Stock control for shelves and stocks real-time monitoring in shops to simplify the supply chain management, e.g., alerting for stocks that run out [34]. It includes M2M monitoring of automated vending machines. The challenges include real-time operation, ease of maintenance, integration with legacy enterprise systems, deployment-specific configurations.

E-Payments enable payments via short-range communication technologies like NFC [35]. The challenges include strong security requirements, interoperability with payment systems available on COTS devices.

Logistics

The Logistics domain includes systems to support scenarios involving storage and shipping of goods. It includes good monitoring during transportation (e.g., monitoring of vibrations, strokes, box opening, break of cold chain), detection of improper storage conditions (e.g., flammable materials close to heat sources), location of items (e.g., in storages, harbours) or vehicles (e.g., for navigation support or theft prevention).

Smart transport supports good monitoring during transport, e.g., in trucks or cargo ships [36]. The challenges include monitoring system reliability and trust (e.g.,

anti-tampering), automatic association of sensor readings to goods, interoperability across different logistics platforms.

Smart logistics support good detection and tracking in warehouses [37] and combined monitoring [38]. The challenges include standard identification and addressing schemes, tight integration with Enterprise systems, large scale operations, reliable readings, standard solutions for tagging.

Industrial Control

Industrial control is one of the first application domains for IoT technologies (e.g., for remote monitoring of manufacturing lines through SCADA systems). These systems are typically deployed in manufacturing or process industries in order to remotely check machineries (e.g., M2M applications), diagnose the status and position of moving vehicles or robots or to monitor the conditions of the manufacturing environment (e.g., air quality monitoring in food processing industries). These systems help the transition of industrial automations from custom, closed developments towards more flexible internet-oriented schemes, directly integrated with enterprise and management systems. They simplify data-intensive applications like real-time tracking or predictive maintenance.

Manufacturing applications leverage autonomous M2M interactions to monitor and optimize production lines [39]. The challenges include interoperability across heterogeneous system, context-awareness, fusion of information from various data sources, reliability in harsh industrial environments, large scale operations, strong safety constraints.

Mobile robotics applications in industry in which mobile robots interact with fixed IoT infrastructures, e.g., to support internal logistics in manufacturing plants. The challenges include support for mobile operations, context-aware behaviour, interoperability of robots with fixed sensors.

Smart Agriculture

Smart Agriculture focuses on monitoring the soil used for growing agricultural products, plants, greenhouses, or the environmental conditions (e.g., weather).

Crop monitoring for moisture, salinity, acidity, etc. using sets of chemical sensors, e.g., to ensure product quality [40]. The challenges include operation in harsh conditions, lack of central infrastructure, self-configuration, self-powered operations, cost constraints.

Smart green houses are controlled and optimized via wireless sensors and actuators [41]. The challenges include reliable operation (especially for control), integration of heterogeneous sensors, control logic support using distributed sensors and actuators, simple maintenance, low-cost.

Smart Animal Farming

Closely related to Smart Agriculture, Smart Animal Farming enhances the productivity of animals for meat and related products (e.g., milk, eggs) by monitoring the animal health conditions at different stages, animal tracking and identification (also used for product traceability), and living environment.

Animal monitoring looks for animal behavior, e.g., to detect states of sickness or stress [42]. The challenges include data integration from heterogeneous sensors, processing of large sensor datasets, data processing algorithms and techniques, operation in harsh conditions.

Meat traceability from farm to fork for animal-derived products, including monitoring of physical parameters during various operation phases [43]. The challenges include intermittent connectivity, standard identification and addressing schemes, tight integration with enterprise systems, large scale operations.

Domotics and Home Automation

The Domotics and Home Automation are centered on homes and commercial buildings. They include applications to improve home safety, ease of use and sustainability, e.g., through resource consumption monitoring (energy, heating, water), remote control of appliances, optimization of air and ventilation or prevention of theft and intrusion.

Building automation monitors and optimizes building subsystems, such as lighting or heating, ventilation and air conditioning (HVAC) [44]. The challenges include interoperability up to the semantic level, integration with legacy solutions, need for standard protocols and interfaces, secure and safe operation.

Appliances control, automatic and remote, for, e.g., energy usage optimization in Smart Grid Environments [45]. The challenges include interoperability up to the Semantic level, standard protocols and interfaces, ease of use, context-awareness features, real-time operations, secure and safe operations, privacy.

e-Health

e-Health systems focus on monitoring of patients or assets needed to assist the health of people in any specific condition, including, e.g., disabled or elderly or under special training or dietary situation. Specific applications include, e.g., detection of fall of elderly people living alone, or monitoring of stocks of medicines in hospitals.

Patient surveillance in Ambient-Assisted Living (AAL) applications [46] and other systems to monitor patients inside or outside hospitals [47], including monitoring of living parameters and position tracking. The challenges include ensuring the privacy of patients' data, data protection, management of sensed data, support for patient mobility, wearable systems – low-power operation.

Fall detection, prevention and reaction for elderly or disabled people [48]. The challenges include location-awareness, data processing complexity for fall detection, mobile and nomadic operations, reliability requirements.

IoT Application Challenges

The analysis of the main IoT application scenarios in Sect. 2.1 leads to a set of challenges that are currently faced by the field, summarized in Table 1.

Table 1 IoT application challenges by use case domains

Challenges	Domains												
	Smart Agriculture	Home Automation	e-Health	Logistics	Retail	Security and emergency	Smart animal farming	Smart cities	Smart environment	Smart grid	Smart Metering	Smart water	Industrial control
Cloud-based large data processing						○	○	○		○	○		
Collaborative (untrusted) management								○					
Deployment-specific configuration		○	○	○	○								○
Distributed data and logic processing		○						○					
Extraction of context and environment		○		○	○			○					○
Harsh environment	○					○	○		○			○	○
Heterogeneous devices	○	○	○		○		○	○		○	○	○	○
Interoperability up to semantic level		○						○					
Interoperability with processes and systems		○		○	○	○	○	○			○	○	○
Large-scale and/or decentralized				○			○	○	○	○	○	○	○
Limited bandwidth	○						○	○	○				
Location awareness			○		○	○		○	○				
Low cost	○			○									
Low power operation	○		○					○	○				
Low-latency and quality of service	○	○			○	○			○	○	○		○

(continued)

Table 1 (continued)

Challenges	Domains												
	Smart Agriculture	Home Automation	e-Health	Logistics	Retail	Security and emergency	Smart animal farming	Smart cities	Smart environment	Smart grid	Smart Metering	Smart water	Industrial control
Mobility			○				○	○	○				○
No communication or power infrastructure	○								○			○	
On-board processing			○			○		○	○			○	
Privacy		○	○		○			○		○	○		
Reliability	○		○	○		○				○	○		
Resilience to disruption and interference						○				○	○	○	
Safety		○	○			○				○	○		○
Security		○	○		○	○			○	○	○		○
Self-calibration, self-configuring, self-healing	○	○	○	○		○		○	○	○	○	○	○
Simple and low-cost maintenance	○	○	○	○	○	○	○	○	○	○	○	○	○
Standardization and certification		○		○	○		○	○		○	○		○
Time correlation	○	○			○			○	○			○	

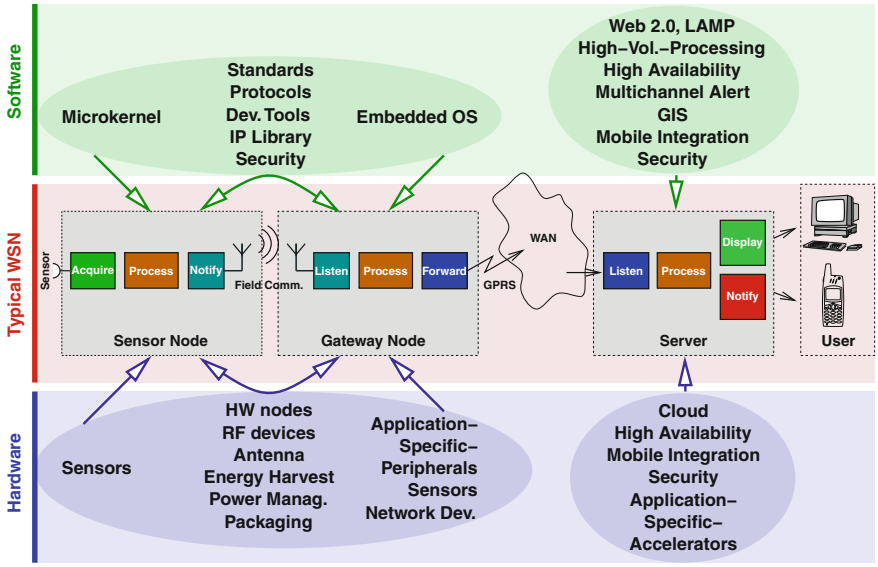

Fig. 2 A wide variety of technologies converge into a typical tiered WSN platform

Beyond the technical/scientific factors mentioned in the table, it is important to remember that the IoT domain is also impacted by non-technical challenges of different nature, including social, process reliability, training, awareness, collaboration and economic challenges, which should be addressed as the IoT domains and technologies evolve and mature.

3 WSN Application Development Overview

The IoT paradigm valued since the beginning the low-cost long-term WSN-based data gathering solutions able to provide versatile sensing and actuation through distributed resilient bidirectional communications.

Early WSN projects made use of large-scale ad hoc, multi-hop, unpartitioned networks of randomly deployed and mostly fixed and homogeneous tiny, resource-constrained devices. This traditional definition is still applicable to a significant class of applications (e.g., for the military and outdoor environmental monitoring domains).

For instance, Fig. 2 shows the main components of a typical tiered WSN platform for monitoring applications and some of the hardware and software technologies it is made of. The block diagram in the middle section shows one or more sensors (transducers) that are attached to the sensor nodes. These are typically small (in size, resources and cost) and are suitably deployed within the application field to

perform the measurements needed by the application. The sensors typically have limited data storage, processing, energy reserves, and short-range communication capabilities that can be used to process locally and forward the field measurements required. Often they are expected to operate reliably and unattended for very long periods (years).

The data sent by the sensor nodes ultimately reach a gateway (or sink) node, either directly or after a few forwarding hops through the network. The gateway nodes usually have larger memory, data processing, energy reserves, and long-range out-of-field communication capabilities. Among other functions, the gateway nodes can, e.g., buffer and further process the field data, and communicate with the application server to transfer field data and configuration instructions.

The server is typically located outside (and can be far away from) the application field and has processing, storage, and supply power to reliably store field data for long periods of time, and allow to retrieve them in formats suitable for visualization or further processing. Also, the server may be able to issue alerts on specific field conditions or events.

However, many emerging WSN applications cannot be adequately defined in this way and the structure of this generic platform and its components often need to be adapted to best suit the application requirements and environmental conditions [49]. For instance, the sensor nodes, the gateways, or the server can be omitted for specific applications. Also, the communication channels between the sensors (if any) and with the gateways can be uni- or bidirectional, and form peer-to-peer, star, tree or mesh topologies.

The top and bottom sections of Fig. 2 shows some of the many and multidisciplinary technologies, both hardware and software that concur to the realization of the WSN platform for most applications. Mastering all these technologies is difficult for most WSN and IoT players, and keeping up with their rapid pace of change adds to the challenges. Very often an efficient collaboration between users, experts of the application domain, hardware designers, and software and firmware developers is necessary to achieve efficient WSN implementations [50].

Considering all these, a definition of the WSN design space can be used as a framework for discussion and coordination of both research efforts and the development of flexible WSN platforms, which can be easily adapted to effectively support the many application needs. Both industry and academic research can benefit from a versatile and reusable WSN platform, easy to tailor to support a broad range of applications, and including development tools, support for several hardware devices, a library of software components for both the embedded and server parts of the application, and tools to support the field planning and deployment [51]. However, the development and maintenance of such a platform can be very costly and also involve professionals with very diverse expertise, from advanced web and UI design to low level firmware, and from IDE design to code generation techniques for high level application descriptions. Quality assurance is also very important, since overall unreliability perception is still a limiting factor to WSN wider adoption.

Hardware vendors usually provide WSN development platforms tailored around the devices they produce. These usually consider most typical uses of the vendor's

hardware, being often hardware-centric rather than application-centric. As such, they can require significant extension or adaptation to cover a broader range of applications and may significantly lag the state of the art in the WSN field, since they follow the progress of the producer.

Most of the open projects for embedded (TinyOS, freeRTOS, Contiki) or server (GSN) components of the WSN platform are largely vendor- and hardware-independent. But they also fail to address all development needs of a complete WSN application, often requiring significant adaptation and integration for an effective use by system integrators.

3.1 WSN Development Abstractions

Wireless sensor networks often operate under tight resource and budget constraints, their range of application continuously diversifies, and, traditionally, WSNs are developed and deployed by single organizations. These circumstances often facilitate the optimization across the typical layers, which are valuable for component interchangeability and reuse for better reliability and lower costs [52]. The early WSN advances and applications, some of which are shown in Table 2, clearly display the tendency to blur the boundaries between layers even after the efficiency benefits of the layers was generally recognized.

Along with the perceived lack of reliability, application development is still an issue that hinders WSN wider adoption. Real-world applications continue to rely mostly on node programming very close to the embedded operating system, which increase the development time, costs, skill requirements, and limit the component reuse and the overall application reliability.

Consequently, domain experts rarely develop efficient WSN applications [50]. The characteristics and requirements of the WSN devices on the one hand, and the approaches for application development on the other provide a rich set of functions which can meet many diverse applications requirements. However, a good understanding of these is needed in order to choose the best platform and methodology for a given application.

Various programming models were proposed to overcome WSN programming difficulties. Considering the level of abstraction from the operating system (OS) and the operation of the underlying hardware, these can fall under the categories of:

1. OS-level programming;
2. virtual machine (VM) or middleware programming;
3. network macroprogramming.

The first two address the node-level programming [53] while the latter hinges on network-level WSN programming (macroprogramming) abstractions [54].

Table 2 WSN layers are usually unevenly covered by development frameworks and applications. Often layer separation is blurred for better implementation efficiency

	A	T	N	L
Aloha [preamble sampling-based protocol]				●
B-MAC [asynchronous adaptive preamble sampling protocol]				●
BVR [Beacon Vector Routing, greedy point-to-point routing]			●	○
CODA [dynamic Congestion Detection and Avoidance]		○	●	○
CTP [Collection Tree Protocol with hop-based metric]			●	○
Deluge [protocol for network-scale node reprogramming]		●	○	
DIP [probabilistic spatial node Density Inference Protocol]		●	○	
Drain [sink-originated distributed best route discovery]		●	●	○
Drip [registration-based unnamed reliable message dissemination]		●	○	
Flush [adaptive reliable bulk multihop transport protocol]		●	●	○
FPS [Flexible Power Scheduling for distributed power management]		●	●	●
Fusion [of multiple distributed field data over unreliable channels]		○	●	○
GAF [Geographic Adaptive Fidelity for location-based routing]	●	●	●	●
Great Duck Island [habitat monitoring application]	●	●	●	●
Golden Gate Bridge [structural health monitoring application]	●	●	●	●
MintRoute [many-to-one cost-based routing for data collection]			●	●
MultihopLQI [routing protocol based on Link Quality Indicator]			●	○
North Sea [industrial sensing application]	●	●	●	●
PEDAMACS [topology-aware multihop TDMA protocol]				●
PSFQ [Pump Slowly, Fetch Quickly transport protocol]		●	●	●
Redwoods [environmental monitoring application]	●	●	●	●
RMST [Reliable Multi-Segment Transport for guaranteed delivery]		●	●	●
SCP-MAC [Scheduled Channel Polling protocol]				●
S-MAC [fixed-duty sleep-synchronized preamble-sampling protocol]				●
SPIN [Sensor Protocol for Information via Negotiation]		●	●	
T-MAC [synchronized contention-based adaptive-duty protocol]				●
Volcano [application to monitor an active volcano]	●	●	●	●
WiseMAC [synchronized preamble sampling infrastructure protocol]				●
X-MAC [short-preamble low power listening protocol]				●

A Application, *T* Transport, *N* Network, *L* Link

OS-Level Programming

OS-level programming models consist in developing the application logic using directly the resources made available by the embedded OS. Of the latter, one of the first and widely used is TinyOS [55], a component-based OS that allows modular programming using the C-based nesC language [56]. The modules have interfaces that are connected using configurations, the latter being used to describe also the application. The encapsulation provided by the modules hides the implementation details, but the low level of abstraction, the event-based style, and the blocking nature of the operations may complicate the implementation even of simple applications.

Asynchronous messages is one technique to reduce the programming complexity, e.g., TinyGALS [57], which allows a globally asynchronous (message passing

through asynchronous FIFO queue), locally synchronous (within sequential modules) programming model. SOS [58] implements message passing between modules using a priority queue, and CoMOS [59] uses preemptive inter-module message handling.

More OS layers or the extension of the event model can also reduce the programming complexity. SNACK [60] facilitates the design and reuse of parameterizable service libraries and applications can be defined by combining services. T2 [61] provides a *telescopic abstraction* hybrid made of a horizontal decomposition (to support different hardware devices at low level) and a vertical decomposition (to support platform-independent functionality at high level). OSM [62] addresses the limitations of the static association of states to actions and the implicit change of the program state in a pure event-driven programming model. OSM separates the state and transitions, the latter and the associated actions being a function of state and events, adds parallel composition for concurrency and hierarchical composition for program refinement.

Thread abstraction is also proposed to address the limitations and complexity of an event-driven programming model. While the event-driven model tends to optimize the microcontroller sleep time, the static per-thread stack allocation may be too expensive for the typically limited RAM of the sensor nodes. However, the blocking execution context of threads can significantly simplify the programs and the programming.

An early lightweight concurrency model is proposed by Fiber [63] on top of TinyOS, while MANTIS OS [64] provides fully preemptive, time-sliced multithreading. TinyThread [65] implements cooperating multithread as a library for TinyOS with explicit and implicit execution context yielding, and per-thread stack estimation. Protothreads [66] implement a similar multithreading approach for another event-driven OS, Contiki, but without a per-thread stack, to improve efficiency. Y-Threads [67] concept is similar to Fiber, but implements preemptive multithreading with a shared stack for non-blocking computation behaviours and separate stacks for blocking control behaviours. The latter typically need a small stack which leads to better overall hardware resource utilization.

Virtual Machine/Middleware Abstraction

Virtual Machines/Middleware (VM/MW) simplify the remote node reprogramming at the expense of some processing overhead and total code size. The latter can be minimized by limiting the generality of the VMs to subsets relevant to application domain(s). Also, VMs provide platform-independent execution models facilitating code reuse.

Interpreter-based VM as Maté [68] and ASVM [69] are implemented on top of TinyOS and provide increased safety and reduced application code size, while Melete [70] extends Maté for concurrent applications. VMStar [71] allows the remote update of both VM and application code. Remote programming is also included at OS- or MW-level, for complete application or at finer grains (module or thread),

as the necessity to update the application in the field rises. t-Kernel [72] addresses reliability by providing OS protection and virtual memory through application code modifications at load time.

Macroprogramming

Other WSN programming abstractions address the development of applications as distributed processing at the *network* level, providing models and semantics to define the communication and coordination among nodes. Unlike node-centric abstractions, where the network behavior emerges from the processing and interactions defined for single nodes, macroprogramming abstractions allow high-level behavioral descriptions at network-level.

The *programming paradigm* that defines how the program elements (functions, variables, computations) and their interactions can be expressed. It has a significant impact on the learning curve, especially for domain experts that may have limited programming experience.

Application description using *imperative* statements, which indicate explicitly the way to change program state, are by far the most used, be these event-driven (e.g., nesC) or sequential (e.g., Pleiades [73]). *Declarative* programming allows the description of the application goal without explicitly defining how to accomplish it. The programming can be functional (e.g., Regiment [74, 75]), rule-based (e.g., Snlog [76]), SQL-like (e.g., TinyDB [77]), and special-purpose (e.g., Logical Neighborhoods [78]). *Hybrids* mix declarative and imperative programming approaches, such as imperative for node computation and declarative for inter-node communications (e.g., ATaG [79]).

Pleiades extensions to C language allow to address the network nodes and their internal state. Its network-level instructions are executed by only one network node at a time with constructs designed to iterate through nodes, which make it particularly suited to guarantee the distributed execution of concurrent applications. *Regiment* constructs allow to apply functions and store the output on one or more nodes in a region and *Snlog* uses logical programming constructs as predicates, tuples, facts, and rules. The rules express the processing, the facts are the initial tuples (initial state), and the tuples (the data) is structured according to predicates, similar to relational database tables. *TinyDB*, as the earlier *TAG* [80], implements an energy-optimized WSN-wide SQL query system from the base station. A sensor table implements the data model as one line per sensor node and one column for each sensing capability. It can be filled with field data following SQL queries or proactively, simplifying the expression of data collection applications. *Logical Neighborhoods* allows the definition of node connections based on their logical affinity in the application, regardless of their physical location. It is mostly suited for heterogeneous and decentralized sense-and-react applications. *ATaG* is built around the abstract task and data item, with copies of tasks that can run on different nodes and abstract channels that connect data items to the tasks that consume them. It is mostly suited for sense-and-react applications that require complex operations to decide the actions.

4 Model-Based Design, Simulation and Debugging Framework for WSNs

WSN application developing and testing is often labor-intensive and error-prone due to the wide diversity of requirements and application domains. A rich set of tools to support WSN application development, testing and deployment is available addressing different aspects of the process: high-level modeling, architectural design, abstract and detailed code development, code generation, application- and network-simulation, co-simulation, deployment, validation, debugging, performance monitoring and evaluation [55–81].

The rich choice of tools, devices, and techniques can provide adequate functionality to meet most application requirements. However, a single comprehensive approach has not yet emerged (see Sect. 3.1). The widening technology and tool diversity is still difficult to master to obtain satisfactory implementations and the application-domain experts often need to collaborate with the users and developers to achieve efficient solutions.

In response to this need, the WSN development tools evolve to improve the support for functional composition and complex distributed scenarios, with service-driven models playing a key role.

In the following we present a complete and flexible framework that can speed up and facilitate the programming of distributed WSN applications. It is based on the graphical design tools Simulink®[2] and Stateflow®[3], both part of the widely used Matlab[4] environment. It provides the application developers a set of development tools and software component libraries that allow a high-level description of the application architecture, a graphical environment for the definition and simulation of component behaviour, and automatic generation of network simulation models and of code for target nodes.

The application architecture is described using the standard Web Service Description Language (WSDL). The functional design of the architecture components can be described as either white, gray or black boxes. The *white* boxes can be graphically constructed using high-level abstract concurrent models, as Stateflow® diagrams. The *black* boxes are either binary components or code written in the C language. The *gray* boxes are a suitable combination between diagrams and external code.

Figure 3 shows the main phases of the development flow. After the high level application design and simulation phase, the high-level abstract application model can be used to automatically generate several representations, using a Model-Based Design methodology. It can be used to generate network simulation models for the widely used OMNeT++ and MiXiM simulation environments, and for some of the most popular WSN operating systems, TinyOS and Contiki OS for deployment on

[2] Mathworks Simulink http://www.mathworks.com/help/simulink/index.html

[3] Mathworks Stateflow – Finite State Machine Concepts http://www.mathworks.com/help/toolbox/stateflow/index.html

[4] http://www.mathworks.com/products/matlab/

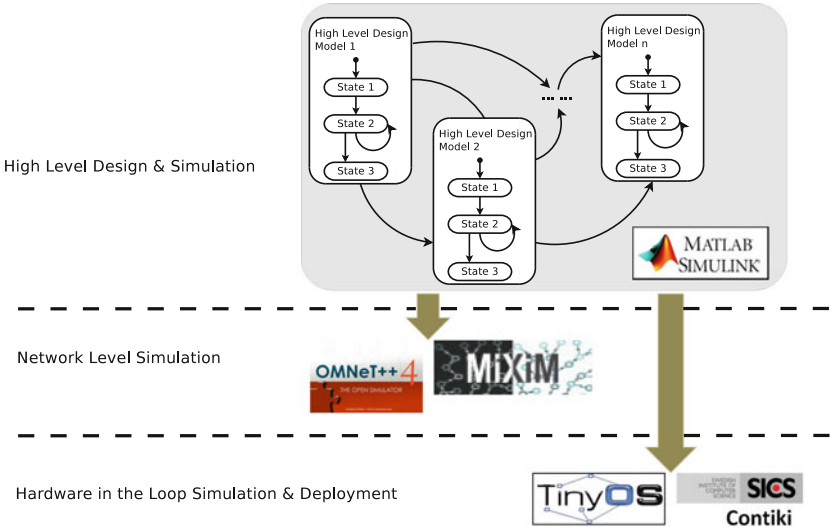

Fig. 3 Overview of the framework development flow

target nodes. Moreover, hardware-in-the-loop simulation can be used to increase the simulation accuracy with real-time physical data from a few node running in sync and communicating with the software simulation.

The integration of the WSN into IoT applications can be simplified when targeting the Contiki OS. In this case, the framework can generate support for web services which can make the node application accessible through typical web-oriented methods (e.g., a normal web browser).

In the following we will illustrate the operation of the platform and how the framework can facilitate WSN application design from high-level conceptual application description, platform-independent graphical design, rapid prototyping, and automatic target code generation for multiple targets.

4.1 Abstract Design Model

The proposed framework [82] makes use of a high-level abstract functional unit, called Abstract Design Model (ADM), to define the structure and behaviour of the target WSN application.

As shown in Fig. 4, each ADM is a programming abstraction, a self-contained unit. It is perceived as a black box by other units and is externally characterized by the tunable attributes and services it uses and provides. The internal details of the ADMs do not depend on external entities, and they can be connected together in a loosely-coupled fashion.

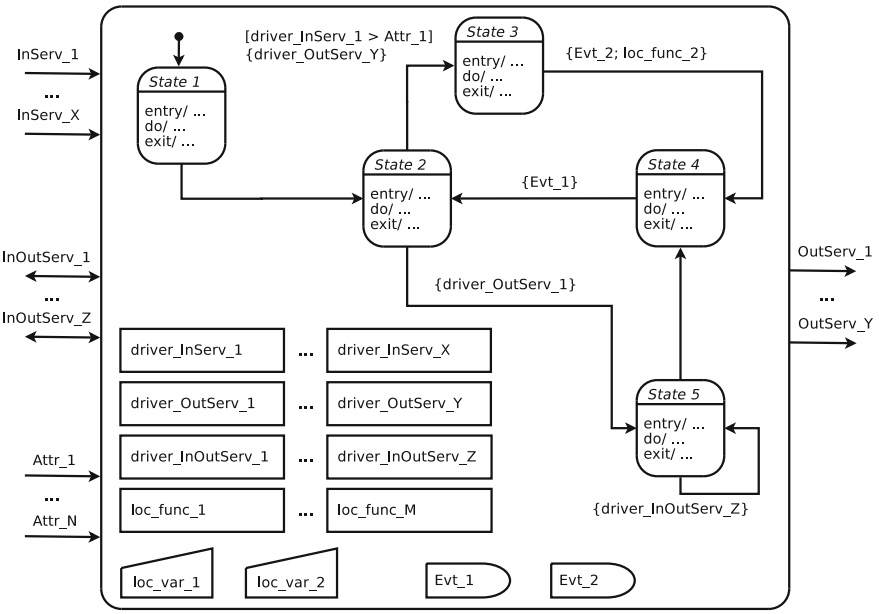

Fig. 4 Components of a typical Abstract Design Model of the framework

Therefore, the WSN applications running on the nodes can be modeled as a set of interconnected ADMs that exchange service messages through their service ports. An inbound service port (e.g., `InServ_1`, ..., `InServ_X` in Fig. 4) imports the incoming messages for an associated service used by the host ADM, while an outbound service port (e.g., `OutServ_1`, ..., `OutServ_Y` in Fig. 4) exports the outgoing messages for a service it provides. Inbound and outbound service ports can also be combined into a bidirectional port (e.g., `InOutServ_1`, ..., `InOutServ_Z` in Fig. 4), useful when a request-response communication pattern is needed. Moreover, the tunable attributes that are exposed by an ADM (e.g., `Attr_1`, ..., `Attr_N` in Fig. 4) allow the developer to modify how it performs without changing its internal logic.

An outbound service port of an ADM can be wired to an inbound service port as long as they share the same service type. These are similar to function calls and can be used to transmit service-specific messages between ADMs, without exposing their implementation. The service ports can also be left disconnected, meaning no incoming messages for the floating input ports and outgoing messages discarded on the floating output ports.

ADM behavior is represented using an event-driven hierarchical Finite State Machine (FSM) described using state charts, as shown in Fig. 4. The logic flow (i.e., the transition from the current state to the next) can be controlled either by its internal default transitions or by service messages imported from other ADMs. These messages are processed by the FSM according to the values of its tunable attributes.

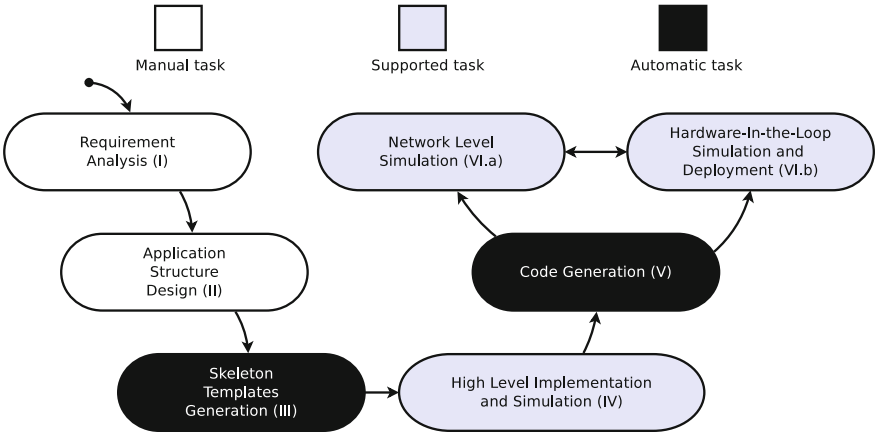

Fig. 5 Framework development flow

Computation results are attached to outgoing service messages sent through output service ports. User-defined operations can be implemented inside each state or between state transitions. They will be executed upon entry, permanence, or exit phases of each state.

Besides tunable attributes, each ADM can have local variables (e.g., `loc_var_1` and `loc_var_2` in Fig. 4) and local events (e.g. `Evt_1` and `Evt_2`).

FSMs can be nested using sub-charts. These can be integrated into higher level FSMs in sequential or parallel execution order, sharing the incoming and outgoing messages, attributes, local variables and local events.

ADMs modularity and loosely-coupled port binding allows to construct an application independent of ADM location within its architecture. ADM port binding can be both intra-node and inter-node. When all application ADMs are within a node, the ports are locally connected and the service messages are implemented by platform-dependent messages or data sharing mechanisms. When not all ADMs are located on the same node, neutral service messages are exchanged by the nodes (e.g., nodes in the radio transmission range) to implement the logic binding of ADM service ports. Moreover, when the ADM service ports are published on the Internet, the developer can raise the target application abstraction level from a single node or local group to the whole Internet.

4.2 Development Flow

Figure 5 shows the V-shape diagram of the development flow of the framework.

There are three types of development activities in the development flow: manual tasks, supported tasks and automatic tasks.

Manual tasks are development activities that are not directly supported by the framework and are performed using other development tools. The *supported tasks* imply some activities performed manually by the developer, but they are also supported by specific tools of the framework. *Automatic tasks* are fully performed by the framework.

The development flow allows the application developer to perform different tasks to design, build, implement or transform the application constituent ADMs. Each ADM can be instantiated in different forms in the framework, for different purposes. For instance, it can be represented by a Simulink®/Stateflow® block in the high-level implementation and simulation phase, for high-level description and debugging. Or it can be an OMNeT++/MiXiM module for large network simulation. It can also be a TinyOS component or Contiki OS process for node deployment, which will be further detailed in the following.

Task I: Requirement Analysis

The first step in the design flow is a *manual task*. It consists in the analysis of the requirements of the target WSN application by listing the desired functionality and supported attributes, e.g., measurands monitored by the application, operations expected from the nodes, state variables exposed as tunable attributes, and criteria employed to validate the application and evaluate its performance.

We will assume a use case where the nodes collect and perform a distributed processing of the data from a temperature sensor. Each WSN node wakes up periodically to carry out the following tasks:

1. sample the temperature values with the desired sampling frequency;
2. collaboratively average these values with those from its one-hop neighbors within a sliding time window;
3. broadcast its calculated average temperature value to its neighbors.

Task II: Application Structure Design

The analysis result can be used to drive the application structure design phase, where the developer conceptually decomposes the application into a set of interconnected ADMs, each carrying out a part of the entire functionality. Since the ADMs are characterized by services and attributes exposed on their boundary, their internal behaviour may not be detailed in this step. The developer just assigns the requirements listed earlier to the constituent ADMs by defining their boundaries. This can be done describing the ADM services and attributes either manually or by importing them from a library (e.g., created in previous designs).

The framework employs Representational State Transfer-based (REST) web service design approaches for their flexible WSN integration. REST is a software architecture for distributed systems [83], such as the World Wide Web that has smoothed

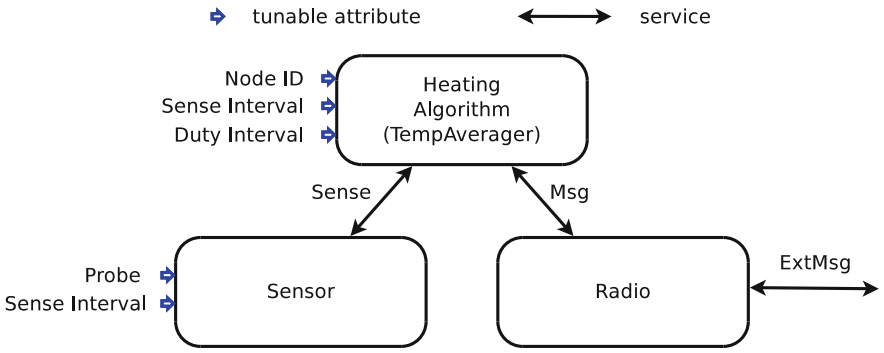

Fig. 6 The model structure of the node application

the way of developing, sharing and reusing IoT applications. It relies on stateless, client-server, cacheable communication protocols, virtually always HTTP.

These help the application developer to construct the ADMs, while WSDL 2.0 is adopted as the ADM description language. The definition of conceptual structures of the application models and related ADMs are *manual task* that can be accelerated using a WSDL graphical editor, such as [84] and [85].

In the proposed use case, the following parameters have been identified as potential tunable attributes for each node:

1. its own node identifier (NodeId);
2. sample refresh interval inside the averaging algorithm (SenseInterval);
3. the sampling period of the temperature sensor (SamplingInterval);
4. the size of the time window to compute averages, which is equal to the node duty interval (DutyInterval).

Based on the requirement analysis (first step in Fig. 5), the design is split on three interconnected ADMs: a `Sensor`, `Radio` model, and `Algorithm` (`TempAverager`) models, as shown in Fig. 6.

The Sensor model samples and pre-processes temperature data. The Radio model interfaces with the protocol stack for short range communication with neighbor nodes. The TempAverager model handles all on-board data processing. Both the Sensor and the Radio models are connected to the TempAverager model to exchange service messages (e.g., `Sense` and `Msg` in Fig. 6).

The developer manually defines a boundary description file for each ADM. These are imported in the framework for the next step, template generation.

Task III: Skeleton Template Generation

This step starts by supplying the ADM description files to the framework to be automatically mapped to skeleton templates defined as a Stateflow® blocks. All

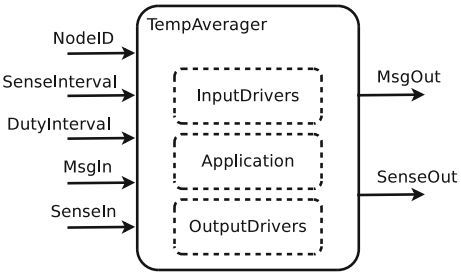

Fig. 7 Overview of the generated skeleton template

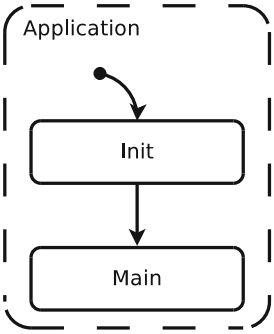

Fig. 8 Application subchart

ADM services and attributes defined in the ADM templates are interpreted as ports or pair of ports (for a request-response service).

For each port, a driver function is automatically created inside the skeleton template, which hides the low-level Simulink® handling of signals and service messages. These functions handle the service messages exchanges through ports. As the input services, each tunable attribute has an input port and a state port variable allowing to set the attribute value from outside the ADM.

Figure 7 shows the skeleton template generated automatically for Stateflow®. Each skeleton template is created with three FSMs that run in parallel: `InputDrivers`, `Application` and `OutputDrivers`.

The `Application` FSM (shown in Fig. 8) contains the ADM main logic.

The `InputDrivers` and `OuputDrivers` FSMs handle the generated input detectors and output actuators for the input and output ports (shown in Figs. 9, 10). These do not need to be changed by the developer.

The generated skeleton templates can be used as the starting point for the implementations in the next step.

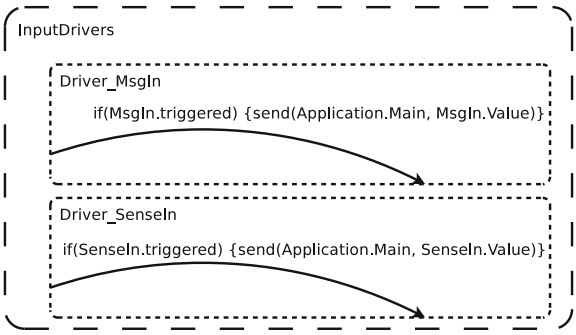

Fig. 9 InputDrivers subchart

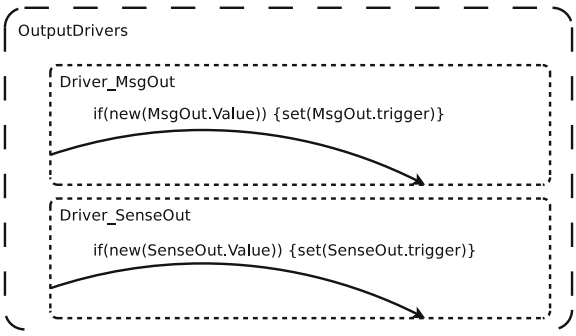

Fig. 10 OutputDrivers subchart

Task IV: High-Level Design and Simulation

In this phase, the developer performs a *supported task* involving the high-level application implementation, simulation and debugging. The implementation consists mainly in skeleton template completion combination:

Skeleton template completion consists in adding the application-specific functions to the skeleton template. These mainly process the imported service messages based on the values of the exposed tunable attributes, and generate the outgoing service messages.
The ADM internal logic is defined at high level, using state charts and flow graphs, independent on the specifications of the target platform. As shown in Fig. 4, the operations defined inside the states will be executed on state entry, permanence or exit phases. The operations defined between two connected states will be executed when the state changes through that state connection. The developer-defined operations can execute any local functions (e.g., `loc_func_1`, ..., `loc_func_2` in Fig. 4) to perform computational tasks or generate outbound service messages.

Skeleton template combination allows to compose the application structure (e.g., as in Fig. 6). This is done by wiring the ADMs outgoing ports to incoming ports and assigning suitable values to the their attribute ports.

Once the implementation of the skeleton templates is completed, the high-level application models can be simulated and debugged at node-level. Afterwards, they can be used to generate the implementations for different simulation or target platforms.

Task V: Code Generation

The skeleton template filled with functional interconnected ADMs, simulated and debugged, can be used for automatic code generation. A framework tool converts the high-level and platform-independent design into target code that runs in different network simulation environments or on target operating systems (OSs) and platforms. These can be:

Simulink®/Stateflow® platform that can be used for node-level and small-scale network simulation. No application translation is necessary;

OMNeT++/MiXiM that can be used for large-scale network simulation. Each ADM that composes the target WSN application is mapped to a *simple component*, the programming unit used by the simulators;

TinyOS which is a popular event-driven embedded OS for WSN nodes. This can be used for code deployment on target nodes. Each functional ADM of the application is automatically converted to a TinyOS *module* component containing the ADM internal logic;

Contiki OS is another popular event-driven embedded OS for WSN nodes. Each ADM is instantiated as a protothread process and the code generated can also be run in the COOJA simulator.
Moreover, Contiki OS allows to generate a RESTful web service support if needed. This allows publishing the application as a standard web service on the Internet without code modifications.

Task VI.a: Network-Level Simulation

Before network deployment, node behaviour and performance should be checked and the target application optimized at node- or network-level.

Model generation converts the high-level functional ADMs to *simple modules* in a node instance for the simulation environment. For the use case proposed, the `Sensor`, `Radio` and `TempAverager` functional blocks are converted to simulator simple modules, as shown in Fig. 11. An "adaptor" object is provided by the development framework as a layer that conveys the messages exchanged across the node instance borders.

Node instances are manually interconnected and configured using the simulation initialization file. It can specify configurations like field size, node receiver sensitivity, initial position and mobility. For example, it can assign values to node properties like `NodeId`, `AvgWinSize`, `SamplingPeriod` and define the properties of the

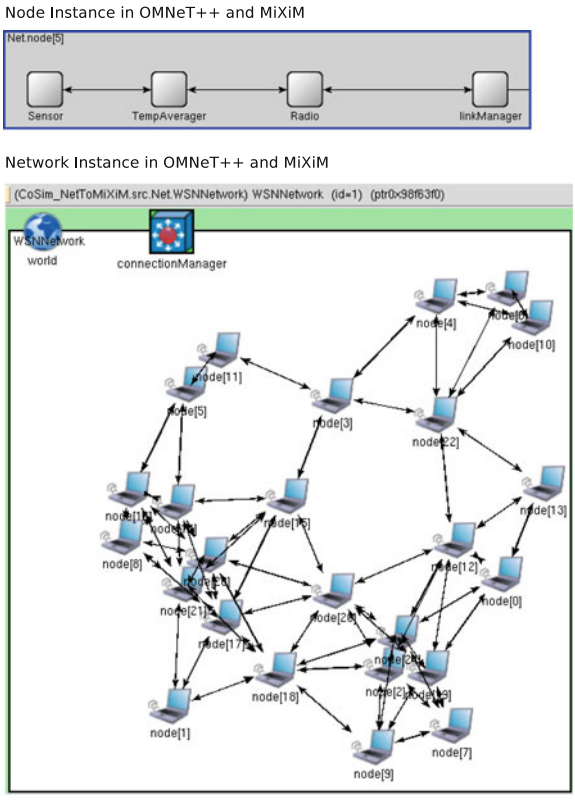

Fig. 11 Simulation in MiXiM and OMNeT++

communication channels between nodes. The network simulation environments also allow the definition of various scenarios, e.g., node mobility or variable communication channel properties.

Any algorithm for the application logic provided by the simulation environments can be easily connected to node instances allowing its performance evaluation in a large-scale simulation.

Task VI.b: Hardware-in-the-Loop Simulation and Deployment

The framework supports automatic code generation for hardware-in-the-loop (HiL) simulations for both target OSs:

TinyOS generation maps each ADM to a *Module* component. These components react and process the internal and external events, service messages, and post internal tasks. They can be manually wired together and assigned values to their attributes using the generated OS *Configuration* file.
HiL simulation can be set up for TinyOS as shown in Fig. 12. The code for the

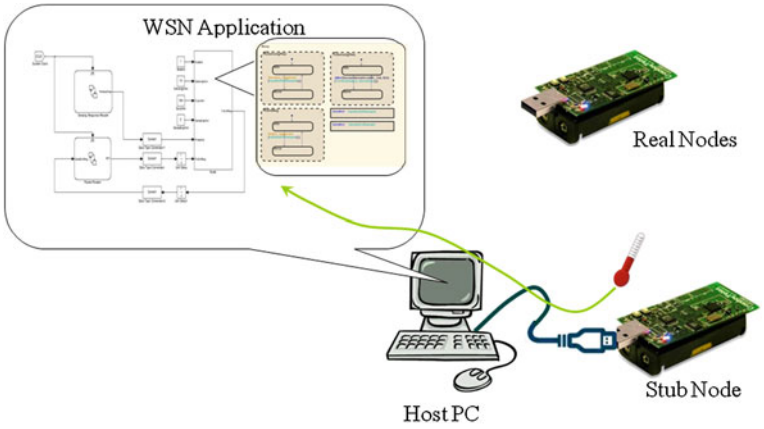

Fig. 12 HIL simulation on TinyOS platform

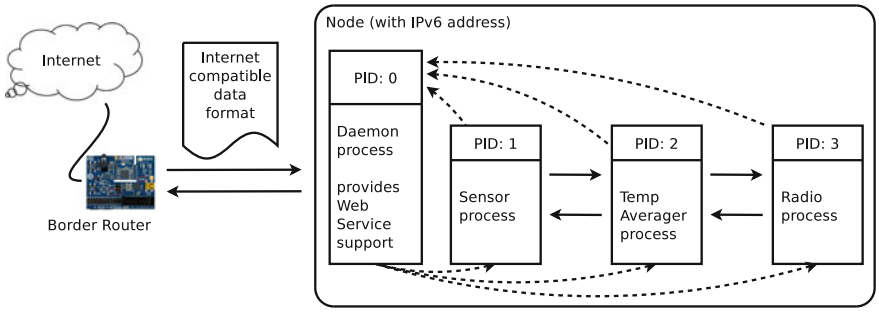

Fig. 13 Deployment on Contiki OS with REST web service support

stub node that is connected to the development framework configures it to act as a gateway bridging the virtual and hardware nodes. It forwards the physical network messages to the framework when queried and broadcasts the virtual network messages to the physical net. The stub can also transfer real sensing samples (e.g., the real temperature value) from its on-board sensors when requested by the virtual nodes.

Contiki OS generation maps each ADM to a Contiki OS protothread process, such as `Sensor`, `TempAverager` and `Radio` Processes in Fig. 13. These run in parallel reacting to internal and external events, processing the service messages using the functions from ADM behavioural description.

Particularly interesting for IoT applications, Contiki OS provides the essential support for web services. The ADM service description WSDL file can be exposed on the Internet to provide a standard remote access to the services, as shown in Fig. 13. The border router is provided by Contiki OS to bridge the Internet and the WSN.

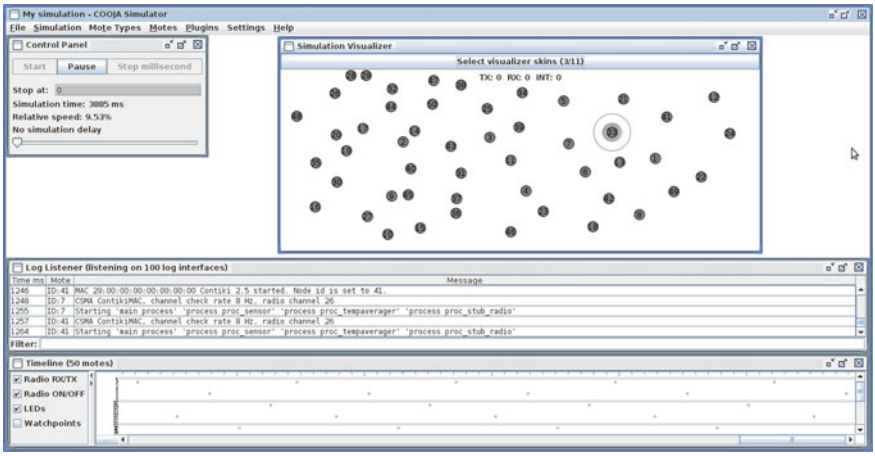

Fig. 14 Large scale network simulation in COOJA

Each Contiki node running applications with REST web service support can be internally decomposed in two type of processes: daemon processes and application processes. A daemon process (identified by PID:0 in Fig. 13) can be automatically generated by the framework. It receives and transforms CoAP Internet requests to internal events, dispatches the events to the application processes, and returns a CoAP response to the Internet requester. Each ADM-exposed service port is considered a resource of the Contiki node and can be reported by accessing the dedicated resource *discover*.

The COOJA simulator can be used to simulate large scale networks of Contiki nodes. It can trace low-level communications among virtual nodes and monitor network traffic, as shown in Fig. 14.

Once the developer validates the simulation results and the deployment performance against the WSN application requirements, the design can be refined and improved.

For instance, the sample application can be used to validate the framework capabilities. The code generation was set to transform the high-level design ADMs to nesC modules, suitable for a simple test-bed set up using Memsic Telos rev. B nodes running TinyOS.

The generated nesC modules (`Radio`, `Sensor` and `TempAverager`) are configured, interconnected and encapsulated in a wrapper nesC module that is then wired in TinyOS to adapt the existing radio communication services.

Table 3 shows the code size and memory usage measured for the binary code generated using the development framework and the same application logic implemented manually. The results show a penalty for the generated code of less than 20 % in code size and less than 7 % in RAM requirements.

Table 3 Code size and the memory usage for the use case application

	ROM [bytes]	RAM [bytes]
Hand-written	17220	492
Framework-generated	20562	526

5 Conclusion

IoT rapid evolution, diversification and pervasiveness benefit from holistic approaches and improvements of the collaboration among domain experts, developers, integrators and operators of pervasive systems.

The development tools that support shared abstractions can play a primal role in IoT application modeling and implementation at all levels.

In this context, we introduced a workflow and toolset aimed to support the definition and implementation of modular IoT systems based on WSNs, enabling hybrid simulation and HiL emulation to accelerate and simplify the evaluation of system behavior in complex, large-scale scenarios.

Acknowledgments Parts of this work were supported by ARTEMIS-JU and Governments of Italy, Spain, and Greece through the ARTEMIS project WSN-DPCM (#269389).

References

1. Ashton, K.: That 'internet of things' thing. In the real world, things matter more than ideas. Expert view. RFID J. (2009). http://www.rfidjournal.com/articles/view?4986
2. Brock, D.L.: The electronic product code (EPC)—a naming scheme for physical objects. MIT Auto-ID Center White Paper (2001)
3. Bushell, S.: M-commerce key to ubiquitous internet. Expert view. Computerworld (2000). http://www.computerworld.com.au/article/84178/m-commerce_key_ubiquitous_internet/
4. Romer, K., Mattern, F.: The design space of wireless sensor networks. IEEE Wirel. Commun. **11**(6), 54–61 (2004)
5. Smith, I.G., Vermesan, O., Friess, P., Furness, A. (eds.): The internet of things 2012 new horizons. In: IERC Cluster Book, 3rd edn. Platinum, Halifax (2012)
6. Gubbi, J., Buyya, R., Marusic, S., Palaniswami, M.: Internet of things (IoT): a vision, architectural elements, and future directions. Future Generation Comput. Syst. **29**(7), 1645–1660 (2013)
7. Akyildiz, I.F., Su, W., Sankarasubramaniam, Y., Cayirci, E.: Wireless sensor networks: a survey. Comput. Netw. **38**, 393–422 (2002)
8. Atzori, L., Iera, A., Morabito, G.: The internet of things: a survey. Comput. Netw. **54**(15), 2787–2805 (2010)
9. Chong, C.-Y., Kumar, S.P.: Sensor networks: evolution, opportunities, and challenges. Proc. IEEE **91**(8), 1247–1256 (2003)
10. Bandyopadhyay, D., Sen, J.: Internet of things: applications and challenges in technology and standardization. Wirel. Pers. Commun. **58**(1), 49–69 (2011)
11. Buttyán, L., Gessner, D., Hessler, A., Langendoerfer, P.: Application of wireless sensor networks in critical infrastructure protection: challenges and design options [security and privacy in emerging wireless networks]. IEEE Wirel. Commun. **17**(5), 44–49 (2010)

12. Durisic, M.P., Tafa, Z., Dimic, G., Milutinovic, V.: A survey of military applications of wireless sensor networks. In: 2012 Mediterranean Conference on Embedded Computing (MECO), pp. 196–199. IEEE (2012)
13. MacRuairi, R., Keane, M.T., Coleman, G.: A wireless sensor network application requirements taxonomy. In: Second International Conference on Sensor Technologies and Applications, 2008 SENSORCOMM'08, pp. 209–216. IEEE (2008)
14. Gluhak, A., Krco, S., Nati, M., Pfisterer, D., Mitton, N., Razafindralambo, T.: A survey on facilities for experimental internet of things research. IEEE Commun. Mag. **49**(11), 58–67 (2011)
15. Libelium. 50 sensor applications for a smarter world: get inspired! http://www.libelium.com/top_50_iot_sensor_applications_ranking/, Aug 2013
16. Arampatzis, Th., Lygeros, J., Manesis, S.: A survey of applications of wireless sensors and wireless sensor networks. In: Proceedings of the 2005 IEEE International Symposium on Intelligent Control, Mediterrean Conference on Control and Automation, 2005, pp. 719–724. IEEE (2005)
17. Tafich, M.: The internet of things: application domains. In: Proc. of Adv. Media Technol. seminar, pp. 37–45. Technische Universität München (2013)
18. HernÃąndez-MuÃśoz, J.M., Bernat Vercher, J., MuÃśoz, L., Galache, J.A., Presser, M., HernÃąndez GÃşmez, L.A., Pettersson, J.: Smart cities at the forefront of the future internet. In: Domingue, J., Galis, A., Gavras, A., Zahariadis, T., Lambert, D., Cleary, F., Daras, P., Krco, S., MÃijller, H., Li, M., Schaffers, H., Lotz, V., Alvarez, F., Stiller, B., Karnouskos, S., Avessta, S., Nilsson, M. (eds.) The Future Internet. Lecture Notes in Computer Science, vol. 6656, pp. 447–462. Springer, Berlin (2011)
19. Paek, J., Chintalapudi, K., Caffrey, J., Govindan, R., Sami M.: A wireless sensor network for structural health monitoring: performance and experience. The Second IEEE Workshop on Embedded Networked Sensors, 2005. EmNetS-II, pp 1–10, Sydney, Australia (2005)
20. Cheung, S.Y., Ergen, S.C., Varaiya, P.: Traffic surveillance with wireless magnetic sensors. In: Proceedings of the 12th ITS World Congress, pp. 1–13. Citeseer (2005)
21. Mohan, P., Padmanabhan, V.N., Ramjee, R.: Nericell: rich monitoring of road and traffic conditions using mobile smartphones. In: Proceedings of the 6th ACM Conference on Embedded Network Sensor Systems, SenSys '08, pp. 323–336. ACM, New York, NY, USA (2008)
22. Burke, J.A., Reddy, S., Srivastava, M.B., Estrin, D., Hansen, M., Parker, A., Ramanathan, N.: Participatory sensing (2006)
23. Yu, L., Wang, N., Meng, X.: Real-time forest fire detection with wireless sensor networks. In: Proceedings. 2005 International Conference on Wireless Communications, Networking and Mobile Computing, 2005, vol. 2, pp. 1214–1217. IEEE (2005)
24. Werner-Allen, G., Lorincz, K., Ruiz, M., Marcillo, O., Johnson, J., Lees, J., Welsh, M.: Deploying a wireless sensor network on an active volcano. IEEE Internet Comput. **10**(2), 18–25 (2006)
25. Cordova-Lopez, L.E., Mason, A., Cullen, J.D., Shaw, A., Al-Shamma'a, AI.: Online vehicle and atmospheric pollution monitoring using gis and wireless sensor networks. In: Journal of Physics: Conference Series, vol. 76, p. 012019. IOP Publishing (2007)
26. Trinchero, D., Galardini, A., Stefanelli, R., Fiorelli, B.: Microwave acoustic sensors as an efficient means to monitor water infrastructures. In: IEEE MTT-S International Microwave Symposium Digest, 2009. MTT'09. pp. 1169–1172. IEEE (2009)
27. Castillo-Effer, M., Quintela, D.H., Moreno, W., Jordan, R., Westhoff, W.: Wireless sensor networks for flash-flood alerting. In: Proceedings of the Fifth IEEE International Caracas Conference on Devices, Circuits and Systems, 2004, vol. 1, pp. 142–146. IEEE (2004)
28. Fan, Z., Kalogridis, G., Efthymiou, C., Sooriyabandara, M., Serizawa, M., McGeehan, J.: The new frontier of communications research: smart grid and smart metering. In: Proceedings of the 1st International Conference on Energy-Efficient Computing and Networking, pp. 115–118. ACM (2010)
29. Nguyen, N.-H., Tran, Q.-T., Leger, J.-M., Vuong, T.-P.: A real-time control using wireless sensor network for intelligent energy management system in buildings. In: 2010 IEEE Workshop on Environmental Energy and Structural Monitoring Systems (EESMS), pp. 87–92. IEEE (2010)

30. Kahn, J.M., Katz, R.H., Pister, K.S.J.: Next century challenges: mobile networking for âĂIJsmart dustâĂİ. In: Proceedings of the 5th Annual ACM/IEEE International Conference on Mobile Computing and Networking, pp. 271–278. ACM (1999)
31. Viani, F., Oliveri, G., Donelli, M., Lizzi, L., Rocca, P.,Massa, A.: Wsn-based solutions for security and surveillance. In: Microwave Conference (EuMC), 2010 European, pp. 1762–1765. IEEE (2010)
32. Tuna, G., Gulez, K., Mumcu, T.V., Gungor, V.C.: Mobile robot aided self-deploying wireless sensor networks for radiation leak detection. In: 2012 5th International Conference on New Technologies, Mobility and Security (NTMS), pp. 1–5. IEEE (2012)
33. Strohbach, M., Martin, M.: Toward a platform for pervasive display applications in retail environments. IEEE Pervasive Comput. **10**(2), 19–27 (2011)
34. Ping, L., Liu, Q., Zhou, Z., Wang, H.: Agile supply chain management over the internet of things. In: 2011 International Conference on Management and Service Science (MASS), pp. 1–4 (2011)
35. Mainetti, L., Patrono, L., Vergallo, R.: Ida-pay: an innovative micro-payment system based on NFC technology for android mobile devices. In: 2012 20th International Conference on Software, Telecommunications and Computer Networks (SoftCOM), pp. 1–6 (2012)
36. Forcolin, M., Fracasso, E., Tumanischvili, F., Lupieri, P.: EuridiceâĂTiot applied to logistics using the intelligent cargo concept. In: 2011 17th International Conference on Concurrent Enterprising (ICE), pp. 1–9. IEEE (2011)
37. Anderseck, B., Hille, A., Baumgarten, S., Hemm, T., Ullmann, G., Nyhuis, P., Potthast, J.-M., Schulz, R., Monecke, J., Zadek, H., et al.: Smarti: deploying the internet of things in retail supply chains. Integration **10–11**, 2013 (2012)
38. Carullo, A., Corbellini, S., Parvis, M., Vallan, A.: A wireless sensor network for cold-chain monitoring. IEEE Trans. Instrum. Meas. **58**(5), 1405–1411 (2009)
39. Brizzi, P., Conzon, D., Khaleel, H., Tomasi, R., Pastrone, C., Spirito, M.A., Pramudianto, F.: Bringing the internet of things along the manufacturing line: a case study in controlling industrial robot and monitoring energy consumption remotely. In: IEEE International Conference on Emerging Technologies and Factory Automation—ETFA 2013. IEEE (2013a)
40. Wark, T., Peter, C., Sikka, P., Klingbeil, L., Guo, Y., Crossman, C., Valencia, Philip, S., Dave, S., Bishop-Hurley, G.: Transforming agriculture through pervasive wireless sensor networks. IEEE Pervasive Comput. **6**(2), 50–57 (2007)
41. Chaudhary, D.D., Nayse, S.P., Waghmare, L.M.: Application of wireless sensor networks for greenhouse parameter control in precision agriculture. Int. J. Wirel. Mob. Netw. (IJWMN) **3**(1), 140–149 (2011)
42. Scalera A., Brizzi, P., Tomasi, R., Gregersen, T., Mertens, K., Maselyne, J., Van Nuffel, A. Hessel, E., Van den Weghe, H.: The pigwise project: a novel approach in livestock farming through synergistic performances monitoring at individual level. In: Proceeding of Conference on Sustainable Agriculture through ICT innovation—EFITA 2013, Jun 2013
43. Brizzi, P., Conzon, D., Pramudianto, F., Paralic, M., Jacobsen, M., Pastrone, C., Tomasi, R., Spirito, M.A.: Bringing the internet of things along the manufacturing line: a case study in controlling industrial robot and monitoring energy consumption remotely. In: IEEE International Conference on Emerging Technologies and Factory Automation—ETFA 2013. IEEE (2013b)
44. Wheeler, A.: Commercial applications of wireless sensor networks using zigbee. IEEE Commun. Mag. **45**(4), 70–77 (2007)
45. Hubert, T., Grijalva, S.: Realizing smart grid benefits requires energy optimization algorithms at residential level. In: Innovative Smart Grid Technologies (ISGT), 2011 IEEE PES, pp. 1–8 (2011)
46. Dohr, A., Modre-Opsrian, R., Drobics, M., Hayn, D., Schreier, G.: The internet of things for ambient assisted living. In: 2010 Seventh International Conference on Information Technology: New Generations (ITNG), pp. 804–809 (2010)
47. Frederix, I.: Internet of things and radio frequency identification in care taking, facts and privacy challenges. In: Wireless VITAE 2009. 1st International Conference on Wireless Communication, Vehicular Technology, Information Theory and Aerospace Electronic Systems Technology, 2009, pp. 319–323 (2009)

48. Caporusso, N., Lasorsa, I., Rinaldi, O., La Pietra, L.: A pervasive solution for risk awareness in the context of fall prevention. In: 3rd International Conference on Pervasive Computing Technologies for Healthcare 2009, PervasiveHealth 2009, pp. 1–8 (2009)
49. Talzi, I., Hasler, A., Gruber, S., Tschudin, C.: PermaSense: investigating permafrost with a WSN in the Swiss Alps. In: Proceedings of the 4th Workshop on Embedded Networked Sensors, EmNets '07, pp. 8–12. ACM, New York (2007)
50. Szlavecz, K, Terzis, A., Ozer, S., Musaloiu-Elefteri, R., Cogan, J., Small, S., Burns, R.C., Gray, J., Szalay, A.S.: Life Under Your Feet: An End-to-End Soil Ecology Sensor Network, Database, Web Server, and Analysis Service. CoRR, abs/cs/0701170 (2007)
51. Wireless Sensor Network Development, Planning, Commissioning, and Maintenance (WSN-DPCM). http://www.wsn-dpcm.eu/. Accessed Oct 2011
52. Hui, J.W.: An extended internet architecture for low-power wireless networks—design and implementation. PhD thesis, EECS Department, University of California, Berkeley(2008)
53. Sugihara, R., Gupta, R.K.: Programming models for sensor networks: a survey. ACM Trans. Sen. Netw. **4**(2), 8:1–8:29 (2008)
54. Mottola, L., Picco, G.P.: Programming wireless sensor networks: fundamental concepts and state of the art. ACM Comput. Surv. **43**(3), 19:1–19:51 (2011)
55. Hill, J., Szewczyk, R., Woo, A., Hollar, S., Culler, D., Pister, K.: System architecture directions for networked sensors. SIGPLAN Not. **35**(11), 93–104 (2000)
56. Gay, D., Levis, P., von Behren, R., Welsh, M., Brewer, E., Culler, D.: The nesc language: aholistic approach to networked embedded systems. SIGPLAN Not. **38**(5), 1–11 (2003)
57. Cheong, E., Liebman, J., Liu, J., Zhao, F.: TinyGALS: a programming model for event-driven embedded systems. In: Proceedings of the 2003 ACM Symposium on Applied Computing, SAC '03, pp. 698–704. ACM, New York (2003)
58. Han, C.-C., Kumar, R., Shea, R., Kohler, E., Srivastava, M.: A dynamic operating system for sensor nodes. In: Proceedings of the 3rd International Conference on Mobile Systems, Applications, and Services, MobiSys '05, pp. 163–176. ACM, New York (2005)
59. Han, C.-C., Goraczko, M., Helander, J., Liu, J., Priyantha, B., Zhao, F.: CoMOS: An Operating System for Heterogeneous Multi-Processor Sensor Devices. Technical Report MSR-TR-2006-177, Microsoft Research (2006)
60. Greenstein, B., Kohler, E., Estrin, D.: A sensor network application construction kit (SNACK). In: Proceedings of the 2nd International Conference on Embedded Networked Sensor Systems, SenSys '04, pp. 69–80. ACM, New York (2004)
61. Levis, P., Gay, D., Handziski, V., Hauer, J.H., Greenstein, B., Turon, M., Hui, J., Klues, K., Sharp, C., Szewczyk, R., et al.: T2: a second generation os for embedded sensor networks, Telecommunication Networks Group, Technische Universität Berlin. Technical, Report TKN-05-007, (2005)
62. Kasten, O., Romer, K.: Beyond event handlers: programming wireless sensors with attributed state machines. In: Information Processing in Sensor Networks 2005, pp. 45–52 (2005)
63. Welsh, M., Mainland, G.: Programming sensor networks using abstract regions. In: Proceedings of the 1st Conference on Symposium on Networked Systems Design and Implementation, vol. 1, NSDI'04, pp. 3–3. USENIX Association, Berkeley (2004)
64. Bhatti, S., Carlson, J., Dai, H., Deng, J., Rose, J., Sheth, A., Shucker, B., Gruenwald, C., Torgerson, A., Han, R.: Mantis os: an embedded multithreaded operating system for wireless micro sensor platforms. Mob. Netw. Appl. **10**(4), 563–579 (2005)
65. McCartney, W.P., Sridhar, N.: Abstractions for safe concurrent programming in networked embedded systems. In: Proceedings of the 4th International Conference on Embedded Networked Sensor Systems, SenSys '06, pp. 167–180. ACM, New York (2006)
66. Dunkels, A., Gronvall, B., Voigt, T.: Contiki—a lightweight and flexible operating system for tiny networked sensors. Local Comput. Netw. **2004**, 455–462 (2004)
67. Nitta, C., Pandey, R., Ramin, Y.: Y-Threads: supporting concurrency in wireless sensor networks. In: Proceedings of the Second IEEE International Conference on Distributed Computing in Sensor Systems, DCOSS'06, pp. 169–184. Springer-Verlag, Berlin (2006)

68. Levis, P., Culler, D.: Maté: a tiny virtual machine for sensor networks. SIGOPS Oper. Syst. Rev. **36**(5), 85–95 (2002)
69. Levis, P., Gay, D., Culler, D.: Active sensor networks. In: Proceedings of the 2nd Conference on Symposium on Networked Systems Design and Implementation, vol. 2, NSDI'05, pp. 343–356. USENIX Association, Berkeley (2005)
70. Yu, Y., Rittle, L.J., Bhandari, V., LeBrun, J.B.: Supporting concurrent applications in wireless sensor networks. In: Proceedings of the 4th International Conference on Embedded Networked Sensor Systems, SenSys '06, pp. 139–152. ACM, New York (2006)
71. Koshy, J., Pandey, R.: Vmstar: synthesizing scalable runtime environments for sensor networks. In: Proceedings of the 3rd International Conference on Embedded Networked Sensor Systems, SenSys '05, pp. 243–254. ACM, New York (2005)
72. Gu, L., Stankovic, J.A.: T-kernel: providing reliable os support to wireless sensor networks. In: Proceedings of the 4th International Conference on Embedded Networked Sensor Systems, SenSys '06, pp. 1–14. ACM, New York (2006)
73. Kothari, N., Gummadi, R., Millstein, T., Govindan, R.: Reliable and efficient programming abstractions for wireless sensor networks. SIGPLAN Not. **42**(6), 200–210 (2007)
74. Newton, R., Arvind, Welsh, M.: Building up to macroprogramming: an intermediate language for sensor networks. In: Proceedings of the 4th International Symposium on Information Processing in Sensor Networks, IPSN '05, Piscataway, IEEE Press, NJ (2005)
75. Newton, R., Morrisett, G., Welsh, M.: The regiment macroprogramming system. In: Proceedings of the 6th International Conference on Information Processing in Sensor Networks, IPSN '07, pp. 489–498. ACM, New York (2007)
76. Chu, D., Popa, L., Tavakoli, A., Hellerstein, J.M., Levis, P., Shenker, S., Stoica, I.: The design and implementation of a declarative sensor network system. In: Proceedings of the 5th International Conference on Embedded Networked Sensor Systems, SenSys '07, pp. 175–188. ACM, New York (2007)
77. Madden, S.R., Franklin, M.J., Hellerstein, J.M., Hong, W.: Tinydb: an acquisitional query processing system for sensor networks. ACM Trans. Database Syst. **30**(1), 122–173 (2005)
78. Mottola, L., Picco, G.P. : Programming wireless sensor networks with logical neighborhoods. In: Proceedings of the First International Conference on Integrated Internet Ad hoc and Sensor Networks, InterSense '06, ACM, New York (2006)
79. Bakshi, A., Prasanna, V.K., Reich, J., Larner, D.: The abstract task graph: a methodology for architecture-independent programming of networked sensor systems. In: Proceedings of the 2005 Workshop on End-to-End, Sense-and-Respond Systems, Applications and Services, EESR '05, pp. 19–24. USENIX Association, Berkeley (2005)
80. Madden, S., Franklin, M.J., Hellerstein, J.M., Hong, W.: Tag: a tiny aggregation service for ad-hoc sensor networks. SIGOPS Oper. Syst. Rev. **36**(SI), 131–146 (2002)
81. Marron, P.J., Karnouskos, S., Minder, D., The CONET Consortium.: Roadmap on Cooperating Objects. Kluwer Academic Publishers, Luxembourg (2009)
82. Song, Z.Y., Lavagno, L., Tomasi, R., Spirito, M.A.: A service-driven development tool for wireless sensor network. In: C. Benavente-Peces, F.H. Ali, J. Filipe (eds.) PECCS, pp. 87–95. SciTePress, Rome, Italy (2012)
83. IBM. Restful web services. http://www.ibm.com/developerworks/webservices/library/ws-restful/
84. Eclipse wsdl editor. http://wiki.eclipse.org/index.php/Introduction_to_the_WSDL_Editor. Accessed 8 Feb 2013
85. Liquid. http://www.liquid-technologies.com/WSDL-Editor.aspx. Accessed 10 Feb 2013