WASP

Final Project Summary
Deliverable D8.4

Public Report

Information Society
Technologies

Project Number	:	IST-034963
Project Title	:	WASP
Deliverable Type	:	Report

Deliverable Number	:	D8.4
Title of Deliverable	:	Final Project Conclusions
Nature of Deliverable	:	Public Report
Internal Document Number	:	WASP_D8.4_final_project_summary.docx
Contractual Delivery Date	:	2010-08-31
Actual Delivery Date	:	2010-10-12
Contributing WPs	:	all WPs
Author(s)	:	Martijn Bennebroek, Peter van der Stok, Oscar Garcia (PRE) Kees Lokhorst, Pieter Hogewerf (WUR) Alessio Corongiu (CRF) Andreas Lachenmann, Matthias Neugebauer (EMIC) Junaid Ansari (RWTH) Laurent von Allmen, Jean-Dominique Decotignie (CSEM) Marcel Steine, Richard Verhoeven, Johan Lukkien (TUe) Matthias Andree, Holger Karl (UPB) Ramón Serna Oliver and Gerhard Fohler (TUKL) Danilo Blasi, Fabio de Ambroggi (STM) Frank Bouwens (IMEC-NL) Alessio Carenini, Gianpaolo Cugola (CEFRIEL) Antoine Fraboulet, Michael Hauspie (INRIA) Benny Lo, Louis Atallah (ICL) Bjoern Riemer, Mario Schuster, Christian Flügel, Carsten Stocklöw (FHG) François Ingelrest (EPFL) Alberto Bosisio (HTN) Laurent Gomez (SAP)

Abstract

This final deliverable describes the main objectives, solutions, results, and lessons learned by the FP6 project “Wireless Accessible Sensor Populations” (WASP) that ran from September 2006 to August 2010. The WASP project explored and developed solutions to ease deployment of Wireless Sensor Networks (WSN) and their integration in enterprise systems.

The deliverable presents a summary of the WASP integrated system and the application and system requirements architecture. The key hardware & software components, tools to ease deployment, (where possible) advance with respect to state-of-the-art and major lessons learned and useful for future improvements. Besides all the hard integration work, also quite some exploratory work has been conducted that will be illustrated as well. A brief overview of the dissemination and exploitation activities as well as future opportunities is described as well.

Keyword list

Evaluation of results, Lessons Learned, Herd Control and Elderly care scenarios and test bed integration, automotive scenarios and simulation, Generic WASP system and underlying generic components and tools to ease development and deployment, target stake holders

Table of Contents

1 Introduction

Over the past few decades, Wireless Sensor Networks (WSNs) have attracted a lot of interest due to their potential in enabling new applications and business propositions across a variety of domains. However, despite years of promising academic research, industry remains reluctant to take over research results and to start commercial (large-scale) deployments largely due to the mismatch between actual research at node-, network-, and application level on the one hand and actual industry needs on the other hand. The project "Wirelessly Accessible Sensor Populations" (WASP) aimed to narrow this mismatch by covering the whole range from node hardware and software, wireless network communication, towards the (secure) information distribution into enterprise information systems. It focussed on the exploration and derivation of cost-efficient solutions to facilitate deployment and application-driven optimization of a network of generic and flexible wireless sensor nodes. This deliverable describes the overall achievements and lessons learned from the WASP project (FP6-IST-2005-2.5.3 Embedded Systems) that ran from September 2006 to August 2010. In this first chapter, a brief introduction of the WASP objectives, activities, and consortium is given together with a outline of the following chapters.

Figure 1 depicts in cartoon style a snapshot of the main project objectives, the wide-spread technical and multi-disciplinary activities, and stakeholders addressed within WASP. The WASP objectives are depicted on the right-hand side and reflect that WASP indeed addresses industry needs by (1) considering concrete application scenarios as starting points, (2) develop technologies that matches the needs of these scenarios and generalizes beyond an individual application, allowing the wasp technology to be applied to a wider range of scenarios, (3) evaluate our approaches in concrete test beds, and (4) distill our experiences into a set of best practices. The wide-spread technical activities within WASP are depicted in the middle of Figure 1 and range from sensor node hardware and software platforms, wireless network solutions, IP infrastructure (WLAN and GPRS) solutions, WASP gateway, and the development of a backend infrastructure that enables remote (enterprise) applications for three selected domains being elderly care, herd control, and automotive.

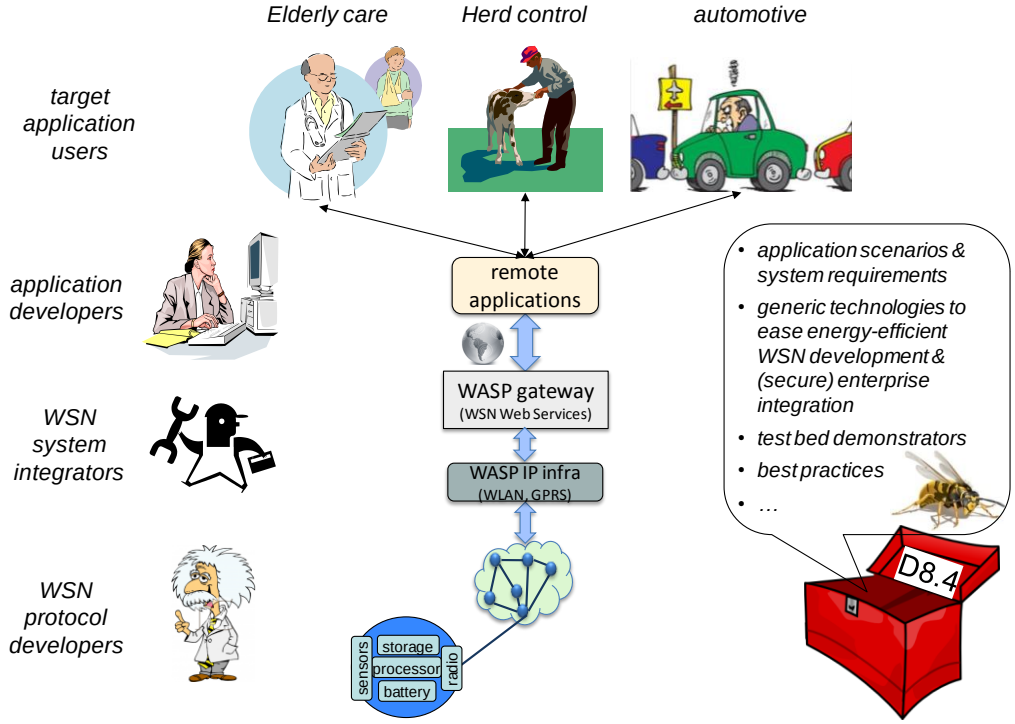


Figure 1 Cartoon depicting WASP main objectives (right hand side), technical activity range (middle), and target stakeholders (left).

One major hurdle that prevents large-scale industrial adoption of Wireless Sensor Networks (WSN) relates to the variety of disciplines and stakeholders involved when pursuing actual WSN deployments. Since all disciplines have their own professional language, constraints and solutions, the alignment of interfaces and trade-off of system-wide optimizations is quite a challenge. The left side of Figure 1 depicts the stakeholders as defined and targeted by WASP for whom WSN deployments should be facilitated by a prototypical integrated solution and tools for deployment. The **target application users** (or end users**)** refer to medical doctors and their patients, farmers and their cattle, car manufacturers and car owners who are typically non-technical experts who should obtain a real benefit in their daily business from (cost-effective, reliable, and easy-to-use) turn-key solutions for WSN deployment, data interpretation, and/or remote services. **Application developers** are the ones building (enterprise) applications by translating the end users' needs (and associated professional language) into customer-tuned data-base service offerings while matching these with the WSN services / applications. The **WSN system integrators** develop proper WSN-specific (embedded) application services and integrate these services with an application-optimized hardware platform and WSN network protocol. Finally, **WSN protocol developers** are typically wireless communication specialists that are able to develop energy-efficient and scalable network solutions that should be sufficiently reliable for the targeted enterprise applications. Within the WASP consortium, depicted in Figure 2, all the technical stakeholders are well represented and, moreover, direct links with the target application users exist which enabled the definition and validation of relevant application scenarios.

WASP consortium:

- Five industrial partners:
  - Philips Research Europe
  - SAP
  - STMicroelectronics
  - Microsoft Innovation Center Europe
  - Fiat Research Center
- Seven universities:
  - Imperial College London
  - Ecole Polytechnique Federale de Lausanne
  - University of Paderborn
  - RWTH Aachen
  - Technical University KaisersLautern
  - Technical University Eindhoven
  - Wageningen University
- Five research institutes:
  - CSEM
  - INRIA
  - Cefriel
  - FHG
  - IMEC-NL
- One SME:
  - Health Telematic Network

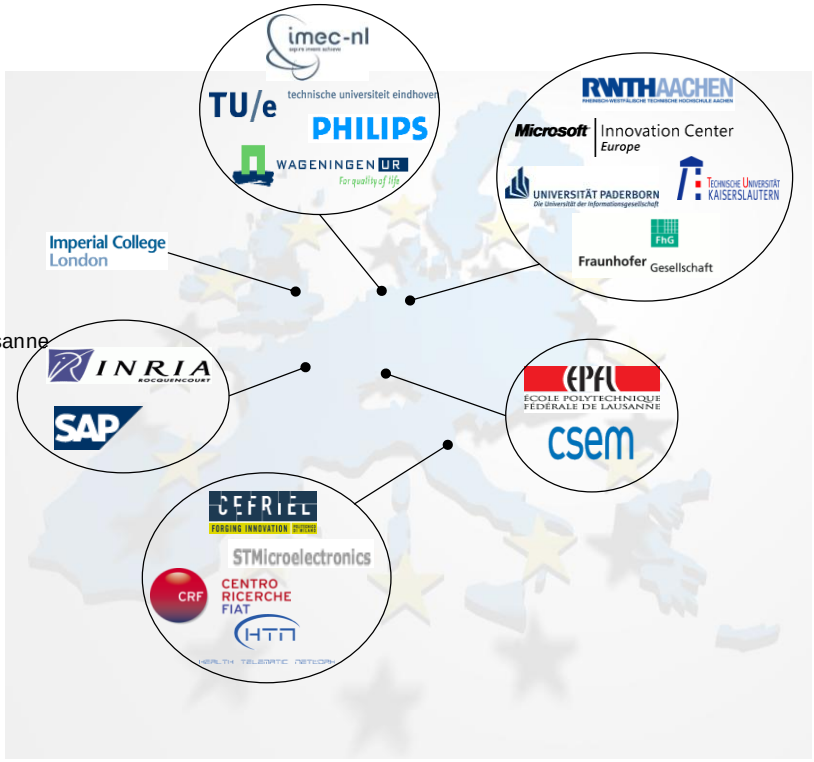


Figure 2: The WASP project consortium.

To focus the activities of such a large number of partners and to validate the use of WASP solutions, three application domains have been selected for their societal relevance and potential benefit from WSN-based solutions. The first application domain involves **elderly care** where, driven by the ageing society, novel cost-efficient solutions are needed to support the elderly in (semi-) independent Ambient Assisted Living settings as well as to provide health workers with effective means of studying transient deterioration and behaviour changes characteristic to the ageing population. The second application domain concerns **herd control**

and is driven by the increase in farm sizes in response to abandonment of the EU milk quota system in 2015. Solutions for IT-supported precision livestock farming will be required, for example, to study and enable detection of individual dairy cows that need attention in a very early phase and, thereby, prevent loss of milk production and premature carry-off of animals for slaughter while preserving costs for food, veterinary, and medicine. The third application domain concerns **automotive** where, already today, modern vehicles incorporate tens of sensors to provide information such as temperature, air quality, tire pressure, distance to nearby objects, etc. that is utilized in various vehicle control functions. With the increasing development of novel functions and applications, the number of in-car sensors could be more than a few hundred in the near future for which current sensor systems, based on shielded wires for heath and interference resilient connections, are becoming problematic in terms of cost, weight, freedom of placement. Addressing such a wide range of applications was a challenge within WASP, but it is the very challenge that actual stakeholders are faced with when attempting to break out of a single market confinement. Hence, the WASP project had to make decisions that mirror the decisions of different stakeholders. We believe that our design decisions are valuable from both a technological and a best practices point of view for actual application (end) users and technical developers.

The organisation of this deliverable is as follows. The generic WASP system, including the main application and system requirements, is described in Chapter 2. Chapter 2.5 describes the implemented WASP components in more detail together with the associated tools developed to facilitate WASP deployment for the various stakeholders. Being an Integrated Project, a significant effort has been spent on the actual integration of components and validation in the test beds but, moreover, also considerable exploratory work has been conducted. The structure of Chapter 2.5 reflects which components have actually been integrated and/or have been part of the more exploratory track within WASP. Chapter 4 describes the relevance of each of the three target application domains and the scenarios assessed. To focus the integration efforts and test bed development, and in line with the project objectives, prototype implementations to validate the use of WASP solutions have been made for two of the application domains, i.e. elderly care and herd control. For automotive, the applicability of WASP solutions has been assessed through simulations. Section 4.4 captures an extensive summary of best practices in terms of lessons learned during test bed integration and covers a number of both technical and organisational issues. This summary list complements (and sometimes partially overlaps) with lessons learned described in other sections. Throughout the project runtime, WASP results have been actively disseminated and discussed during international conferences, workshops, panel discussions around the globe as described in Chapter 5. Moreover, WASP organized three workshops for Small to Medium Enterprises (SME) to assess their questions and needs and, in parallel, to familiarise them with the WASP results and potential future applications that can be based on wireless sensor networks operating autonomously in an intelligent infrastructure. Chapter 5 also captures the main outcomes of the active interactions with related communities, e.g. around Smart Dairy Farming, Ambient Assisted Living, and some of exploitation activities triggered by the WASP developments. The deliverable ends with a summary in Chapter 6.

2 WASP system overview

This chapter has three goals: (1) to explain global system requirements, extra-functional properties and main use cases, (2) to explain how these requirements and properties are addressed in the integrated WASP system, and (3) give an overview of WASP implementation and technology results that support this system view.

Section 2.1 introduces main system and application requirements that lead to the system view presented in Section 2.2. This section introduces the WASP overall architecture. According to Kruchten [1] and the IEEE 1471 "Recommended Practice for Architectural Description of Software Intensive Systems" [2], an architecture description is organized in views that capture concerns of *stakeholders*. The views comprise models showing structure and behaviour that explain how such concerns are addressed. Within WASP these stakeholders comprise *end users* of applications, *application developers*, *protocol developers*, and *system integrators*. In this document we restrict ourselves to overviews of both the system and the software tools and we describe in text how stakeholder concerns are addressed; full descriptions are found in respective deliverables of the project.

Section 2.3 describes the main WASP design and implementation choices. The architecture admits several alternative implementations that address extra-functional concerns differently. These alternatives concern instances of the building blocks in the WSN and their relative organization. The ability to flexibly interchange components with different extra-functional properties is in fact a concern of the system integrator and a requirement to the WASP architecture.

In parallel with the component development, a tool box has evolved. During the many WASP integration trials, implementations have been optimized and further extended to enable run-time tools that can monitor network performance (e.g. packet delivery, number of packets, and topology information) and node stability (e.g. reboots per node). These tools proved extremely useful for both the WSN system integrators and protocol developers to monitor, stress test, and improve actual deployments. Besides these tools, a method of working with the corresponding tool chain has been developed which is summarized in section 2.4.

The chapter ends with a brief description in Section 2.5 of two alternative implementations where a short explanation on their impact for deployment is given.

2.1 Application and system requirements

The WASP architecture and main implementation choices have been defined in the first year of WASP by studying application requirements for the three domains (elderly care, herd control, and automotive) that have been selected for their societal relevance and potential benefit from WSN-based solutions. In addition, we have derived requirements from examining usage scenarios for the three technical stakeholders where we have integrated feedback received during the three workshops organized by WASP to involve Small and Medium Enterprises.

Common application requirements include reliable sensor read-out, sensor node life-time optimization (energy efficiency), reliable communication with the back-end, support for sensor (cluster) mobility, and synchronization of data coming from various sensors.

More specific **elderly care requirements** concern low packet loss and latencies, privacy and security of patient-sensitive data, and the ability to generate patient-specific reports to observe patient behaviour over time. A property of this scenario is that the topology is star-based. **Automotive requirements** demand high reliability in wireless communication, both in terms of data packet delivery ratio and latency, very high energy autonomy, with node lifetime of at least one year (preferably more) without changing batteries as well as security and data confidence. **Herd health control requirements** have less emphasis on security but involve

the capability to deal with relatively large-scale, mesh-based network topologies consisting of mobile clusters. Similar requirements on energy apply as with automotive systems.

Key system requirements are **adaptive behaviour and long-time deployment** that call for creating means for nodes to determine optimal behaviour, to detect flaws or failures and to act correspondingly. To that end, nodes must be capable to configure themselves dynamically, e.g., by setting thresholds or QoS parameters. They also trigger reconfiguration of the entire network. Larger changes in node behaviour require dynamic upload of program code in some form.

Application developers require system solutions for flexible communication (e.g., on demand, event based sensor data acquisition, WSN discovery) and storage (e.g. support sensor data mapping from heterogeneous WSNs to a unified sensor data format).

For a WSN system integrator, **reuse and fast deployment** require a component-based approach such that components can be selected according to application needs. By components, we refer to functional layers in the system like operating systems, network stacks and application components, both in the sensor network and in the backend. The composition procedures must facilitate application-driven optimization. WSN system integrators also need to have **good abstractions to define network behaviour** and tools to support **analyzing, simulating and debugging code** efficiently.

Protocol developers have many similar needs as the WSN system integrator but in addition have the means to **integrate a new communication stack easily** and to evaluate and optimize the network performance during deployment.

Clearly, the requirements of the three technical stakeholders (*application developers*, WSN *system integrators*, and *protocol developers*) lead to requirements both on the respective software components and to the definition of a set of tools to support development, integration and evaluation. In the next section, the developed WASP system, components, and tools will be introduced and a graphical overview is presented in Figure 4 in Section 2.4.

2.2 WASP system view

The WASP project has developed a *system view* that efficiently decouples the work of (enterprise) application developers on the one hand and the work of WSN system and protocol developers on the other hand. This decoupling is mainly achieved by the WASP gateway that allows access to the services running on the sensor nodes through a set of web-services that allows both the application developers and WSN system integrators to use development environments and programming models that match their individual purposes. Likewise, the WASP gateway allows the WSN system integrators to define WSN-specific services largely independent of the choice of wireless technologies that, in WASP, may range from (high data rate) IP-based connections to multiple (low data rate and physically-separated) WASP networks.

Figure 3 depicts a logical diagram of the WASP system that will be introduced in this section along with its main hierarchical layers and associated functionalities together with the networking technologies for communication.

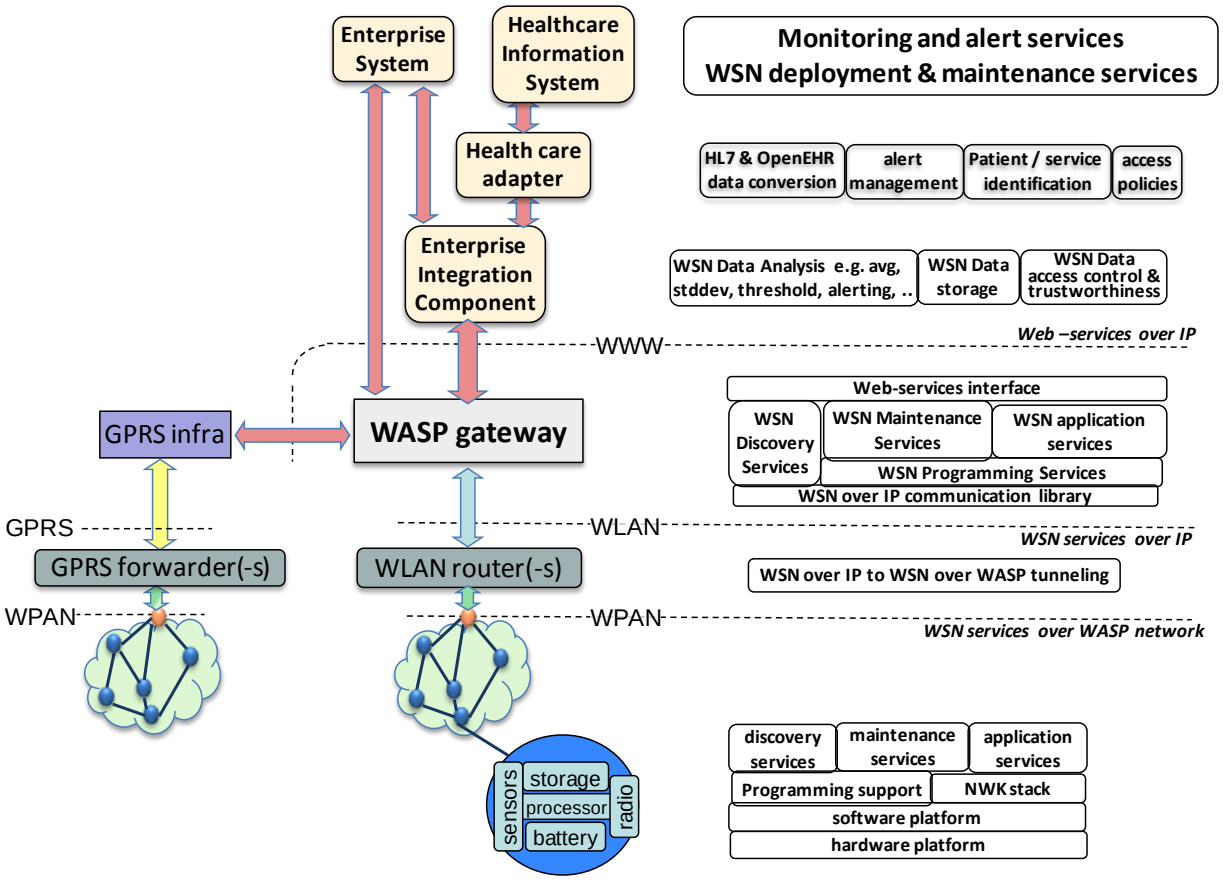


Figure 3: Abstract view of the generic WASP system. Hierarchical layers are shown on the left-hand side and the associated functionality per layer is shown on the right-hand side. Embodiments of the WASP architecture for the elderly care and herd control are shown in Figure 35 and Figure 44 of chapter 4.

The lowest hierarchical layer in Figure 3 contains **wireless sensors nodes** that use one of the Wireless Personal Area Network (WPAN) solutions developed within WASP, build on a narrow-band (low-data rate) 2.4GHz radio link to send information to one or more sink nodes (in orange). The wireless sensor nodes can be organized as mobile clusters. Sink nodes interface with the next layer containing a **WLAN router** that, by bridging the WPAN and WLAN communication, establishes a connection to the WASP Gateway. Alternatively or in parallel, a **GPRS forwarder** can be used to provide an alternative connection between wireless sensor network and WASP Gateway. This **WASP Gateway** provides enterprise applications access to wireless sensor networks through a set of SOAP-based web services. As such, the gateway plays a central role in decoupling WSN-specific system developments from enterprise application developments by providing an abstract interface to common services (such as node & router discovery, application selection, maintenance) and WSN-application specific services. Moreover, the WASP Gateway enables the use of different wireless technologies, like WLAN and GPRS, to provide high data rate, IP-based connections to multiple WASP networks that operate on low data rates and are physically separated.

On the highest hierarchical layer, the so-called **Enterprise Integration Component** acts as a generic mediation layer between (remote) enterprise systems and the WASP Gateway. This Enterprise Integration Component provides generic (data-base) functionality such as storing WSN data, assessing quality (trustworthiness), analyzing trends, and generating events (including alerts) for specified conditions. A **Healthcare Adaptor** has been developed as well

that acts as a healthcare-specific mediation layer converting WSN data to the Open Electronic Health Record (OpenEHR) format and enabling HL7-compliant communication with (professional) Healthcare Information Systems. This adaptor is an example of a domain-specific component that integrates a WSN into a particular application domain and exemplifies that the WASP system can incorporate adaptors of various kinds.

Figure 3 illustrates that the WASP architecture encompasses a rather complex distributed system where especially the functionality mapped on the wireless sensor nodes needs careful consideration given their tightly-constrained resources (battery, processor, memory, radio).

On the nodes, three main services are defined for interaction through the WASP Gateway according to a Publish/Subscribe interaction mechanism. These services include (1) **application services** to sample, process and communicate sensor data, (2) **maintenance services** to acquire and communicate (node and network) status information, and (3) a light-weight **discovery service** to advertise the presence of a node and its available (application & maintenance) services. They have their counterparts on the WASP gateway and together realize the WSN application, consisting of application program, service discovery and maintenance.

In line with the common application requirements, the WASP architecture is generic to enable (hardware and software) reuse over different application domains while allowing for application-driven optimization. This has been demonstrated by the reuse of components in the Elderly Care or Herd Health test beds where basically only the sensor hardware packages (human- or cow-resilient), the WSN routing protocol (Gradient routing or Repairing) and application services (human- or cow-specific scenarios) differ.

In the next section we discuss some of the implementation choices in these layers.

2.3 WASP implementation highlights

2.3.1 System software

Within the WASP system view, different implementation choices are possible to gear a concrete system towards it goals. In this section, we describe some choices made in the WASP project.

Efficient and flexible implementation of application and maintenance services is facilitated by a light-weight **programming support** layer (Figure 3) that allows remote uploading of new services during run-time. A virtual machine as well as a domain-specific language have been designed for that purpose and establish independence of the microprocessor instruction set architecture. Application programmers describe their programs in this language which gets compiled to virtual machine code that is uploaded dynamically. Two such languages have been developed that can coexist; the one language (referred to as ECA and based on the 'event-condition-action' paradigm) is excellent for *coordination,* while the other provides better *computation* facilities (micro DSSP). Depending on concrete application needs, either one can be used to the system integrators' advantage.

With the 2.4GHz radio transceiver being the main power consumer of the wireless sensor **hardware platform**, it is essential to minimize the use of wireless traffic to achieve low-power operation which, moreover, also favours network scalability. With this in mind, two **network stacks** have been developed, one each for star- and mesh-based topologies, corresponding to the elderly care and herd control application requirements. These stacks allow for duty-cycled radio operation, cross-layer optimization, Quality-of-Service trade-offs, and can deal with mobility.

To assist WSN protocol developers, the **WASP postmaster** implementation framework (not explicitly shown in Figure 3 but part of the box "NWK stack" on the lower right corner) has been designed. It facilitates the modification and/or replacement of protocol layers, the reuse

and/or optimization of network protocols (e.g. MAC, routing, localization, and time synchronisation) and it offers hooks for cross-optimization. Moreover, the postmaster provides the network **management information base** consisting of control hooks and monitoring functions made available to the programming support layer such that application services can be developed that influence network performance during real-time operation.

Further decoupling is achieved by a node **software platform** that provides a simple to use and consistent interface for WSN protocol developers. The software platform abstracts away differences between hardware platforms and, thereby, facilitates porting of software stacks to different hardware platforms. The node platform is accessed using the **WASP OS API,** a POSIX-like OS abstraction layer, described in Section 3.2.1.1.

These choices may be considered to be the preferred incarnation of the WASP system view and our identified best practices, without precluding alternative implementations that may be preferable for other target scenarios (see next section).

2.3.2 **Tooling**

BSN nodes have been selected as the major hardware platform as they represent a state-of-the-art solution constructed from off-the-shelf components like a MSP430 processor and (2.4GHz) CC2420 radio transceiver.

To facilitate generation of node firmware, a Build-System environment has been developed that properly links compilation and debugging tools with WASP-specific (source and library) code uploaded to our WASP SVN project repository. The Build-System environment has been made available in a Virtual Machine to circumvent host-machine dependencies experienced when integrating code released by many different parties. Tool bugs and version misalignments can be well upgraded in the Build-System environment.

Simulation tools have been made available operating at different levels of abstraction. Simulators for the programming models run on regular PCs with communication mapped onto IP protocols. The WSim node simulator is instruction and machine configuration accurate and has been developed to simulate actual compiled code for the BSN nodes, used within the WASP test beds, for further debugging and assessment of energy performance.

A tool flow is in place to generate services that can be remotely uploaded to the nodes and to create an accompanying service library to be used by the WSN gateway. Remote uploading is supported by the corresponding middleware function mentioned above, which actually is a light-weight virtual machine. Applications developed by this programming model form a graph that is composed of individual instances of services. These services can be hosted by different devices in the network, for example, a sensor node could potentially call services running on a more powerful gateway in order to optimise the use of resources. Already provided or easily implementable services include filters and functions to be calculated, gateway operation services, security and maintenance services.

2.3.3 **WSN operation outline**

The ability to upload services and threshold parameters is one of the key elements in our strategy to ease deployment. This strategy allows nodes to automatically join the network and have the proper services uploaded if needed. This is achieved by using the periodic route-building broadcast messages initiated by sink nodes that, as indicated in Figure 3, bridge the sensor network and WSN gateway (through a HW router or SW router running on the PC hosting that WASP Gateway). As soon as non-connected nodes with the proper firmware receive these periodic route-building messages, they “know” a path to a sink exists and, hence, can reach the WASP Gateway. In this phase, nodes start to send nanoSLP messages to the WASP gateway that will subsequently update the list of connected nodes and decides whether to upload (upgraded) services (based on the present node services as communicated by the nanoSLP messages). Once service upload is completed, nodes start operating and

delivering sensor, maintenance, and nanoSLP messages to the WSN gateway. Back-end applications, like the Enterprise Integration Component, will start receiving sensor information to which they subscribed at the WASP Gateway.

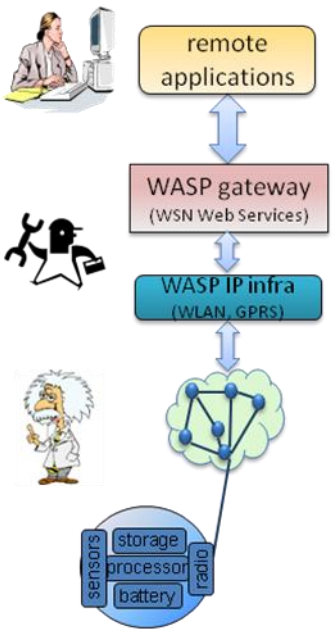
In addition to the described deployment strategy, tools to check proper functioning during deployment are also vital to ease validation of proper deployment. For this, a considerable number of monitoring tools have been developed to check proper functioning, to perform logging of network statistics, of uploaded node services, sink and router status, etc. Within the project even a 3D-viewer GUI has been developed to monitor node and network functioning (e.g. battery warning, connectivity graph, last packet received), to enable remote setting of node parameters (e.g. node type ambient or mobile), and to show the location and activity of objects in a 3D visualization of the actual environment.

2.4 WASP technology results

Summarizing from the previous sections there are three categories of technology deliverables to facilitate application development and WSN deployment for the envisioned technical stakeholders:

- **Generic components**: a set of hardware parts (nodes, WLAN routers, GPRS forwarders) and software parts (enterprise-class software, embedded operating systems and hardware drivers, network stacks, run-time programming support, etc.) from which application-specific solutions can be built and easily ported to alternative hardware platforms.
- **Toolbox for application development**: a set of tools that can be used during the application development phase where the generic WASP system will be tuned to given application requirements. Examples are tools for compilation and debugging of embedded code (integrated in the embedded SW development environment), network simulation models to assess QoS trade-offs, and programming tool chains to develop node- and gateway-level services together with communication libraries to backend services.
- **Tools to ease WSN deployment**: a set of tools to monitor and adapt actual WSN deployments. Tools include various applications that monitor node and network performance statistics, network topology, or actual service installations. These deployment monitors are actually based on (MIB and ECA) facilities integrated in the node firmware to remotely read out and modify node and network operation. Also various logging and debug options are made available to monitor actual data traffic flowing between WASP sensors and backend applications.

Figure 4 gives a graphical overview of these technology results. It has been an important goal of WASP to actually achieve these results and to validate at first hand their effectiveness in the application development activities in the test beds described in Chapter 4. In doing so, the project truly closed the gap between academic research on novel concepts and their actual contribution in solving real-life deployment and application requirements.



WASP Layers	WASP Technologies		
	Generic components	Toolbox for application development	Tools to ease deployment
Application platform		• Standard tools for service development	• Various monitors (network and node statistics, topology, uploaded services) • Maintenance GUI (3D viewer)
Enterprise Integration	• Enterprise Integration Component (EIC) • Healthcare Adaptor (HCA)		
WASP Gateway	• WSN services (gateway, discovery, maintenance, application, programming) • WSNoverIP library	• ECA tool chain • µDSSP tool chain	• logging facilities
WASP IP infrastructure	• Router HW and SW • GPRS forwarder		• SW router debug options • AT commands for remote control of router and forwarder
Networking	• Four network approaches: BAN, HC, CCBR, 6lowPAN • Link-level security key exchange • Flexible WASP postmaster	• Network simulation models (OMNET++) • X-layer trade-off models	• access to MIB to monitor network topology and performance and to modify QoS set points
Node level	• HW platform: BSNv3 nodes, novel hardware demonstrators • SW platform: WASP OS API, various driver upgrades, portability, …	Embedded SW development environment • WASP SVN project with BuildSys scripting & MSPGCC4 • WSim binary code simulation • Virtual Machine environment Other: • ReISC tool chain • NF framework	• Up loadable ECA services to test

Figure 4: WASP technology mapping towards the technical stakeholders depicted in Figure 1.

2.5 Additional implementations

The WASP system and implementation described in the previous sections have been successfully validated and refined in the integrated application test beds as further described in Chapter 4. In addition, two alternative implementations have been explored in parallel.

The first concerns an alternative implementation of the Publish/Subscribe (P/S) mechanism that is further described in Section 3.3.2.2. Here, unlike the integrated WASP system with the P/S mechanism implemented at application level, the novel Content and Context Based Routing (CCBR) approach implements the P/S mechanism in the routing layer (by adding content information to the routing header). Like Gradient routing used in the integrated herd control test bed, CCBR uses link-layer broadcast to support communication with mobile nodes but more advanced mechanisms are incorporated into CCBR, also exploiting cross-layer optimization of CCBR with a Traffic-Aware MAC (TrawMAC), to reduce the impact (and flooding) of broadcast communication on the overall network traffic. CCBR has not been integrated in the integrated WASP system depicted in Figure 3 to limit and focus the associated effort. However, integration of CCBR would not require fundamental changes but merely involve (1) node-level implementation of CCBR and TrawMAC in the WASP postmaster framework and WASP OS abstraction layer and (2) the modification of services (on both node- and gateway-level) to properly use the P/S implementation in CCBR. Only small changes to the run-time deployment tools and/or service creation cycle are foreseen.

The second alternative implementation is described in Section 3.3.2.3 and concerns an “all IP” implementation. Such implementations may become an attractive alternative over custom (partial-IP) implementations, like the WASP integrated system. This requires, however, that IPv6 has become the dominant internet protocol and that on-going developments & standardization of light-weight “IPv6-compliant” WSN protocols, such as 6LoWPAN, provide solutions to will match the requirements of real-life application. Ideally, the big advantage would be that service developers could develop WSN services themselves without depending on WSN service development by the WSN system integrators. However, in practice, this will require service developers to adopt embedded software skills and work with related development environments. As illustrated in a one-to-one comparison of the “all IP” and integrated WASP system in Figure 18 in Section 3.3.2.3, the WASP gateway functionality decreases and turns into a so-called edge router. In this manner, service development can be facilitated by (1) using well-adopted standardized protocols and socket APIs and (2) deployment supported by standard maintenance tools. Although promising, a number of issues have been identified that need further attention before “all IP” implementations become practical.

3 WASP Components & Tools

In this chapter, the major achievements and lessons learned are described and references to more detailed information in WASP-internal deliverables and/or external publications are given.

3.1 WASP hardware platform

For WASP test bed integration, use has been made of the BSNv3 platform provided by partner ICL which is briefly introduced below. Also, given the close similarities in hardware architecture, TelosB nodes (CrossBow) have been used by partners to perform protocol development and preliminary WASP integration lab tests. To enable connection between the WASP network and gateway, a WLAN router and GPRS forwarder have been developed for, respectively, indoor and outdoor applications.

Since energy-efficiency of the node hardware platform is key to wide-spread adoption of WASP applications, potential advances with respect to the state-of-the-art WSN platforms, like the MSP430-based BSNv3 platform, have been explored by the development of an energy-efficient ReISC processor that proves an attractive candidate for next-generation sensor node platforms. Also, a novel dual-processor platform approach has been explored where, next to a microprocessor for network and general control, an accompanying low-power and autonomous application processor is added to strictly decouple application from control processing. Both approaches aim to provide more (and energy-efficient) local processing abilities which can be well exploited to minimize the need to send out (raw) sensor data over the (power-hungry) wireless radio. Minimizing radio traffic is evidently attractive for scalability of wireless networks as well.

3.1.1 Integrated hardware platform

3.1.1.1 WASP sensor node platform

The BSN platform of ICL provides small-sized nodes build around a MSP430 microcontroller and CC2420 2.4GHz transceiver (both from Texas Instruments) and can be extended with add-on boards with (accelerometer, ECG) sensors and battery. On request of the WASP partners, a BSNv3 has been developed that incorporated a MSP430 processor with 10kB SRAM and 48kB Flash, replacing the MSP430 with 2kB SRAM and 64kB flash in earlier versions. A picture of the latest BSNv3 development kit is shown in Figure 5 and, during the WASP project, in total 50 of these BSNv3 kits have been provided by ICL to WASP partners. The BSN platform has also been used to actively disseminate WASP results e.g. at the BSN platform tutorials at the BSN conference 2010. More details of the BSNv3 platform are available in deliverable D2.7.

Figure 5: BSNv3 development Kit of Imperial College for WASP

Although the BSNv3 platform proved very useful during the WASP developments, the following user insights can improve a next-generation BSN platform:

- 10kB SRAM and 48kB Flash as currently provided in the BSNv3 platform is no luxury when developing advanced integrated code, especially, since it leaves little room to incorporate debug functionality. Integration of a newer MSP430 or ReISC microcontroller with more embedded memory would help to advance the WSN field.
- In the current BSNv3 platform design, the MSP430 processor uses a single SPI interface to communicate with both the radio transceiver and the external Flash. However, during WASP developments, this complicated the use of the external Flash as writing blocks to the Flash, e.g. for data logging, should not be interrupted by radio traffic whereas, vice versa, efficient radio traffic handling should not be dependent on finishing reading/writing to Flash. These issues can be circumvented by separating the physical interfaces in a redesign. In parallel, a hardware design flaw in the Flash supply input can be corrected that, by a connection to both the PCB supply line and a MSP430 digital general-purpose I/O pin, caused an excess 50mA current flow whenever the flash driver was switched off. As described in D3.6, luckily this situation could be corrected by adapting the BSN flash driver to configure the GPIO pin as input (i. e. high-impedance state).
- To achieve fully-functional nodes at small form factor, the BSNv3 has an integrated miniature antenna which proved useful during simple WASP demonstrations e.g. at workshops and tutorials. However, the current integrated antenna has a rather non-uniform radiation pattern which, thereby, complicates the tuning and deployment of networking protocols that strongly depend on RSSI values like WiseMAC and DV-distance, especially, in multi-hop set-ups and thereby limits the ability to show more advanced demonstrations. For this, the ability to attach an external antenna (with an improved radiation pattern) to the BSNv3 platform could be exploited which already proved valuable to improve range in the test beds.

3.1.1.2 **WASP WLAN router**

Within WASP, partner FhG FOKUS developed a WLAN router that connects to WASP sink nodes via serial or USB interfaces to (1) upload data from the WASP network over a WLAN (IP) network to the WASP gateway and (2) download sink commands or sensor services from the WASP gateway back to the WASP network. An IP mesh network can be formed between multiple WLAN routers and, thereby, extend the range of the WASP (sub-) networks that can be connected through multiple sinks to the WASP gateway. The WLAN router hardware is build upon the ASUS WL-500g router, depicted in Figure 6, and an externally attached BSNv2 (or TelosB) module that acts as WASP sink node. The

WLAN router application is mapped on OpenWRT Linux and more details of the “WSN over IP” software architecture used to enable IP communication can be found in deliverables D5.5, D7.4.

Besides this hardware router, a software router has been developed that runs on the same PC as the WASP gateway. Both hardware and software routers have been extensively used in the herd control and elderly care test beds.

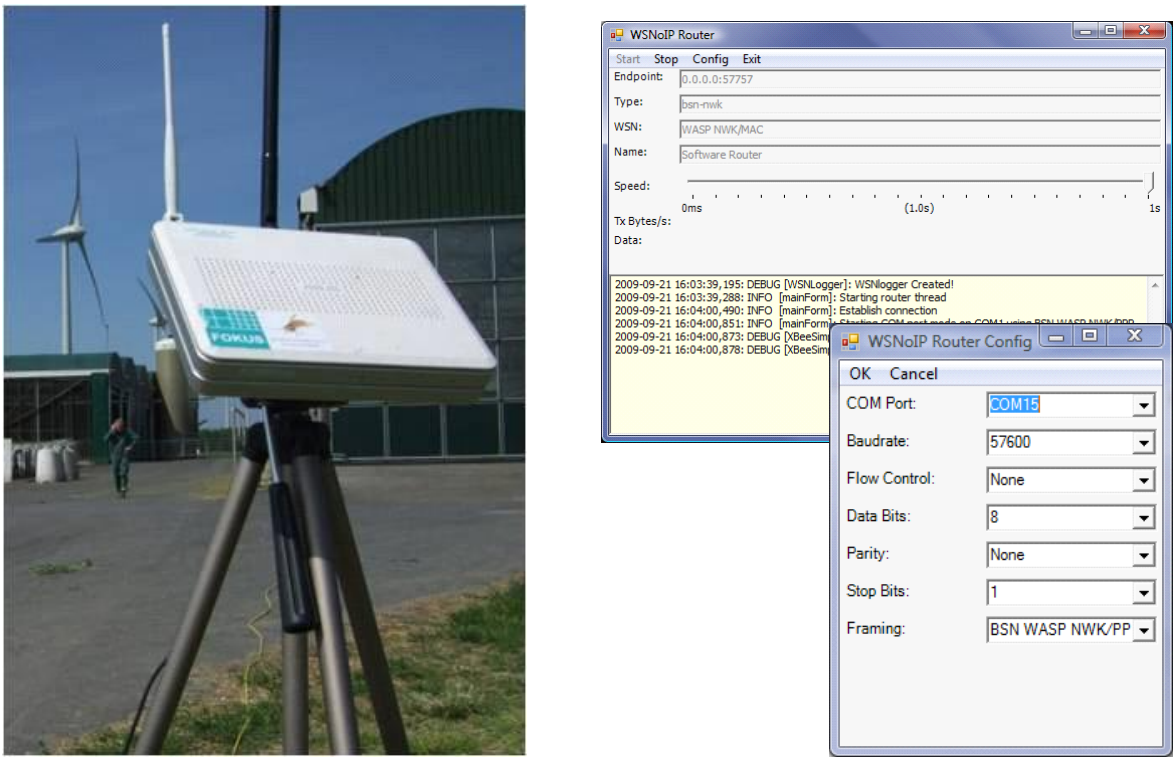


Figure 6: WASP WLAN hardware router on the herd control test bed site (left picture) and software router including COM port configuration (right figures).

Currently the WLAN router uses the non-waterproof housing which comes with the device and is fine for indoor deployment. For outdoor deployment, waterproof housing is needed that incorporates all components including the externally connected sensor node and battery. To recharge the battery, a scavenging power source like a solar-cell array seems of appropriate.

When connected a notebook with an USB-UMTS dongle, in principle, the WLAN software router could also be used as a (soft) GPRS forwarder. Similar to the (low cost) hardware GPRS forwarder described in the next section, such a (software) GPRS forwarder could be of use in (early) deployment trials when a huge storage memory and/or many debug / analysis options are required.

3.1.1.3 **WASP GPRS forwarder**

A GPRS Forwarder module has been developed by partner EPFL to enable remote monitoring in places where there is no nearby access to the internet or to the WASP (WLAN) infrastructure. As implied by its name, the GPRS forwarder uses the mobile phone network, and in particular the GPRS technology, to enable the communication between the WASP network and its gateway.

The board design of the GPRS Forwarder is depicted in Figure 7. A BSNv2 node, indicated by J5, acts as the WASP sink node and, hence, runs the WASP software stack. Packets received from the WASP network are forwarded by this sink node through a serial line to the micro-controller of the GPRS Forwarder (a Texas Instruments MSP430, indicated by U3). The board comprises a Telit GM862 GPRS module (J6), which is operated by the micro-controller to open a TCP/IP connection to the server hosting the WASP gateway. Packets received from the sink node are forwarded though this connection to the gateway. Similarly, data may be sent from the gateway to the GPRS Forwarder, and ultimately to the sink node, in order to upload new services or to change some settings.

In its current implementation, the GPRS Forwarder may be operated either in continuous or in duty-cycled mode. In the latter mode, the TCP/IP connection is regularly established for a short duration and in-between these connections the Telit module is turned off to save energy. The mode of operation and its various parameters (*e.g.,* conditions for re-connection) can be altered from the gateway with the help of special data messages, which are captured and interpreted by the GPRS Forwarder.

Figure 7: GPRS Forwarder Board Design

More information about the GPRS Forwarder module and its integration within WASP can be found in deliverables D2.9 “GPRS Forwarder” and D5.4 “Design Maintenance Services; Refinement of Service Discovery and Gateway Mechanisms”.

Figure 8: The GPRS Forwarder inside its casing, with a set of four rechargeable AA batteries. The box is certified IP67 and thus provides full protection against dust and water.

The general idea is to place a single GPRS Forwarder module in a remote pasture to enable outdoor monitoring of dairy cows. The GPRS Forwarder has been successfully tested in this configuration for livestock monitoring during the integration meetings. For the various tests, the GPRS Forwarder was enclosed inside a hermetic box, as illustrated in Figure 8. To be fully functional, the board only needs to be fed with a SIM card coming from one of the national mobile network providers (KPN has been used during the integration meetings in the Netherlands). Then, thanks to a simple text message sent from a mobile phone, the GPRS Forwarder can be configured with the IP address and the port where the WASP gateway is running.

For GPRS modules operating without power constraints (*e.g.,* connected to an AC power plug or powered by a large enough solar energy system), the continuous GPRS mode can be used, and a live stream of data is sent continuously to the gateway. However, when using the duty-cycled mode, the issue of buffering data is raised. In the current implementation, there is indeed no storage space for incoming packets in the GPRS Forwarder, except a few KB of RAM. As a result, sensor nodes have to perform local buffering until a TCP/IP connection is established. To this purpose, a special mechanism has been implemented to let all nodes in the sensor network know when data must be buffered locally and when it can be routed to the sink.

Also, in its current duty-cycled mode, the GPRS Forwarder is the sole responsible for establishing a new TCP/IP connection to the WASP gateway. There is indeed no mechanism that would allow the gateway to “ask” the GPRS Forwarder to connect to it. This constraint is fine for dairy cow (and environmental) monitoring but not optimal for instance for monitoring of elderly persons wearing a small-scale WASP body network (*e.g.,* containing activity and ECG sensor nodes). Although not tested in the elderly care testbed, the GPRS forwarder could be well used when integrated in such a wearable solution that is tuned for low-power (duty-cycled) GPRS operation. In this application case, it may be appropriate to enable remote activation of the GPRS forwarder by the WASP gateway as well to allow remote care providers real-time access in alert situations (and including GPS position).

It is to be noted that a similar GPRS Forwarder design is currently offered by EPFL start-up company “Sensorscope” (www.sensorscope.ch). This company provides a WSN solution where the GSM network is used as backbone to study of environmental parameters while minimizing the cost of setting up a complete monitoring system.

Among the various improvements that could be made to the GPRS Forwarder design, two of them are prominent:

- **Flash memory**. To allow the GPRS Forwarder to store large amount of WSN data when disconnected from the GSM network, a Flash memory chip could be incorporated into the board. For instance, an Atmel chip providing 2MB of Flash memory costs less than three Euros.
- **Support for sleep mode**. The sleep mode of the Telit chip would be useful for letting the remote WASP gateway request the GPRS Forwarder to establish a TCP/IP connection. While in sleep mode, the Telit chip indeed consumes little energy while still being able to receive text messages, which could be used to send various requests to the GPRS Forwarder. Currently, the sleep mode cannot be enabled because two RS232 pins (CTS and DTR, used for hardware flux control) have not been routed on the board.

3.1.2 Hardware platform innovations

Within WASP, two paths have been explored to improve energy-efficiency and local processing power beyond “state-of-the-art” sensor platforms like the BSNv3 platform already introduced above. The first path involved the design of a Reduced energy Instruction Set Computer (ReISC) processor that allows for instruction extensions whereas the second path addressed the exploration of a dedicated application processor that is intended to operate in a dual-processor platform approach next to a microprocessor for network and general control. A joined functional demonstrator has been presented in deliverable D2.6 “Joined ReISC and ASIP FPGA demonstrator”. Also, following the successful FPGA developments within WASP, the involved partners decided to internally fund the design and manufacturing of first test chips of which some first results are included below.

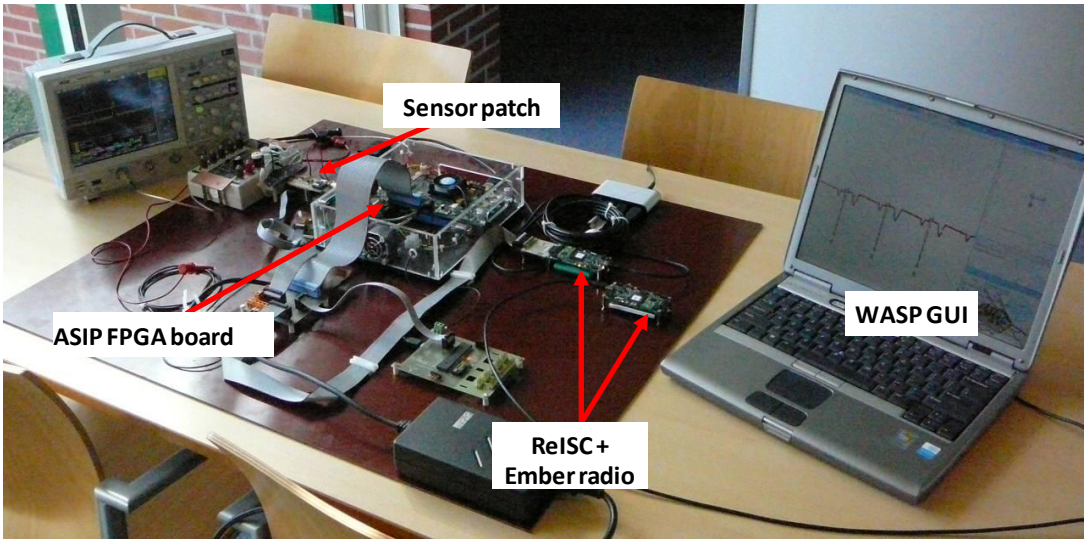


Figure 9 WASP dual ReISC and BioASIP functional demonstrator with the WASP GUI and oscilloscope to illustrate the effect of local BioASIP processing on radio transmission (taken from D2.6).

3.1.2.1 **ReISC processor design**

To advance state-of-the-art in energy-efficient microcontrollers, partner ST Microelectronics explored the design of a Reduced energy Instruction Set Computer (ReISC).

Its design is based on a pipelined 16-bit RISC micro architecture including loop buffers (to minimize memory accesses) and a fully programmable interrupt controller. The ReISC instruction set is optimized for both low dynamic and static power and, moreover, can be extended with four custom instructions. More details of the ReISC architecture, including use of its extendible instruction set and underlying co-processor, can be found in deliverable D2.3 “Processor Architecture”. Here, a first evaluation of potential energy-efficiency gains provided by the extendible instruction set & co-processor is presented by implementing dedicated FFT (radix-2 Butterfly), DWT, and MAC operations. Simulation showed that improvements in power of 43% become feasible by moving these operations from the default ReSIC instructions to the extended instructions and dedicated co-processor.

The choice of a 16-bit architecture has been addressed in deliverable D2.2 where a first FPGA implementation of the ReISC processor is presented. The FPGA demonstrator showed correct functional behavior of the novel ReISC design by demonstrating the counter, temperature sensing and interrupt operations for fast response time. By modifying the ReISC data path and synthesizing its netlist for FPGA and standard cell implementation, the efficiency of each data path configuration could be determined. Table 1 shows the results as noted in D2.2 from which a 16-bit architecture was selected as a good trade-off between performance and energy.

Table 1: Power consumption of the ReISC in the FPGA demo for various data path widths.

ReISC mode	FPGA Core Power (uW/Mhz)	Percentile difference	Netlist Core Power (uW/Mhz)	Percentile difference
8bit	787	Ref.	23.8	Ref.
16bit	938	**+19%**	29.5	**+24%**
20bit	942	**+20%**	35.3	**+48%**
32bit	1093	**+39%**	42.9	**+80%**

In parallel to ReISC hardware design, a toolchain has been developed which allows programmers to exploit all the features of the ReISC processor. As depicted in Figure 10 and further described in D2.4 “ReISC Processor Tool chain”, the tool chain contains a compiler, assembler, linker, debugger and an Instruction Set Simulator (ISS) to simulate applications on the ReISC. Within WASP, the tool chain has been used to generate code for the FPGA demonstrator and mapping of the FreeRTOS port as described in D2.8 “ReISC FreeRTOS Port”. Moreover, it has been demonstrated through a simple (blinking LED) application that the WASP OS (described in section 3.2.1.1) can also be ported on top of the FreeRTOS running on the ReISC.

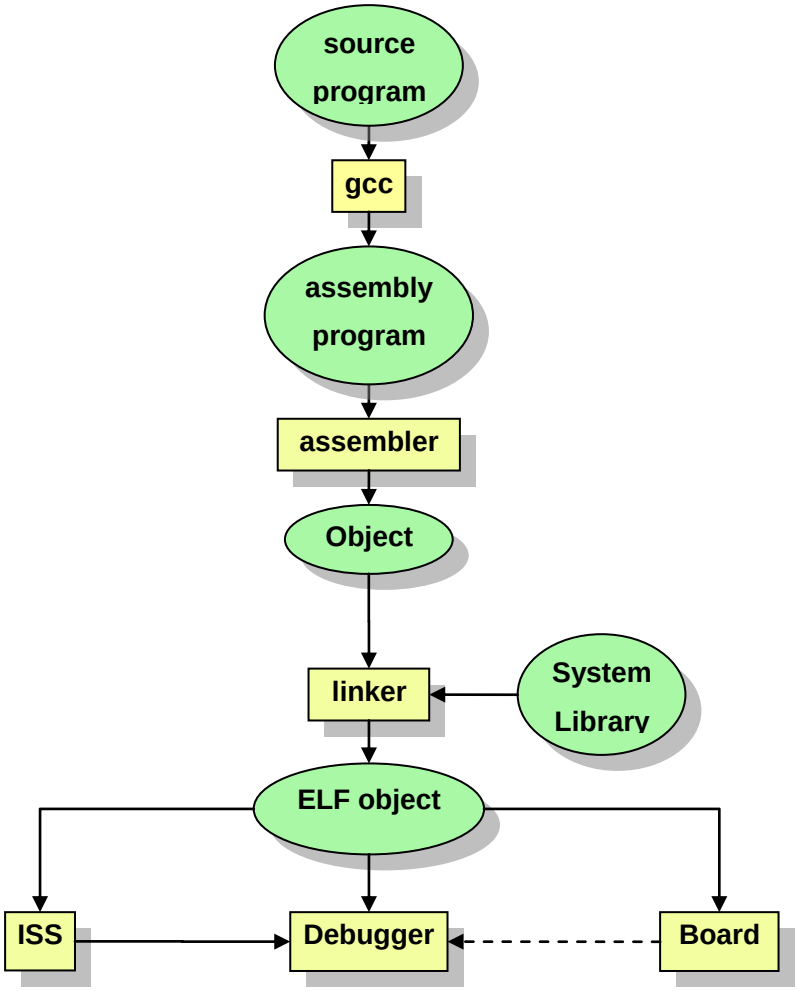


Figure 10: Workflow of the ReISC toolchain

Following the WASP design and FPGA developments, STM internally developed a SoC test chip that is currently being used for business development (in the healthcare, industrial and consumer markets). If successful, ReISC products may become commercially available through standard sale channels by 2011.

In a STM-internal study of a proprietary ECG algorithm (computing heart beat and arrhythmia detection), the 50MHz processor is busy only for a fraction of the time (1%) consuming less than 30uW (20uW dedicated to signal filtering and beat extraction, and 10uw dedicated to morph analysis).

Moreover thanks to the very fast interrupt response and internal clock gating it has been possible to implement *a 'sleep mode'* with a wakeup time of just 1us and sleep power consumption less than 1uW. This could change dramatically the duty cycle of the application (e.g. by going to sleep between the ADC sampling of the heart beat) and thus the average power consumption.

Being the leakage current of the same order of magnitude of the power consumption it would be more efficient to remain in *'sleep mode'* 99% of the time (consuming less than 1uw), the rest of the time the processor at 50MHz could easily sample and compute the heart beat signal (or when more processing power is needed, it could power up the ASIP and let it do the job).

Another achievement in this kind of application is the extension of the ReISC instruction set with DSP enhance instructions which proved very valuable and demonstrated a 10x efficiency on the filtering stage and possibly on any application specific instruction requirement.

3.1.2.2 **Biomedical application processor design**

Figure 11 sketches the exploratory dual-processor platform that includes a dedicated (and autonomous operating) application IC that is connected, in this first generation, through a serial SPI interface to a low-power micro-controller IC like a MSP430 (today) and perhaps a ReISC (in the near future).

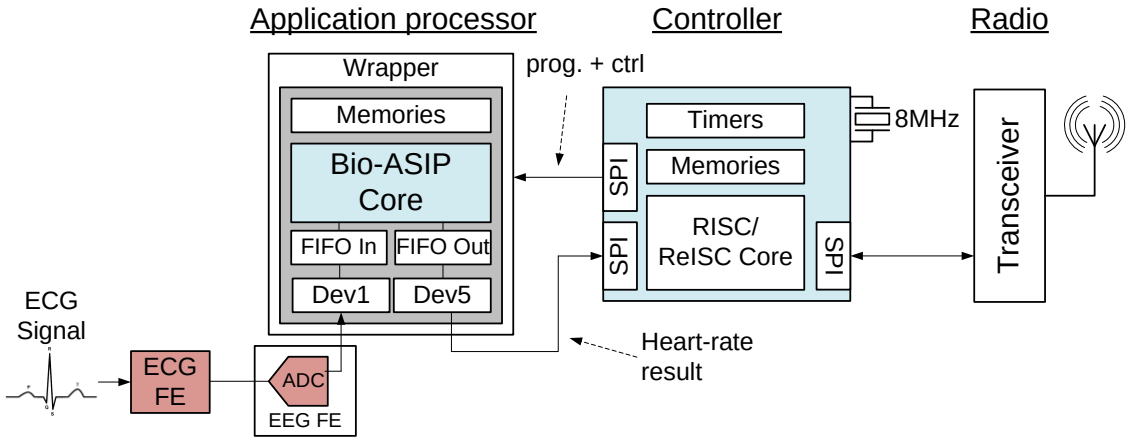


Figure 11: Dual-processor WSN platform with application processor optimized for biomedical sensors and processing and RISC control processor for platform control and network processing.

The application processor is optimized for low-power sensor data acquisition and compute-intensive computations and, thereby, off-loads the micro-controller that can be fully optimized for efficient handling of system control and radio communication. Such a strict decoupling of sensor application and network-related tasks will be useful to avoid (timing, interrupt, and memory) conflicts typically encountered in today's WSN platforms that are based on a single processor. Note that multi-processor approaches have proven their ability to significantly increase functionality within a limited power budget in other wireless systems like (smart) mobile phones. Here, dedicated processors are already common design practice to decouple compute-intense operations (like image rendering) from time-critical GSM networking to enable low-power and high-performance operation.

To minimize the energy consumption of the application processor, a novel event-driven system architecture is developed that allows for autonomous operation of sub-systems with minimum system overhead (and thus power). The idea is sketched in Figure 11 where device peripherals, like device 1 connected to the ECG front-end, are locally (de-)activated at a (configurable) frequency and captured sensor values are stored in a local FIFO buffer. Once a sufficient (and configurable) number of samples is collected, the FIFO actives the (BioASIP) core to read in the samples and start processing. In parallel, the device peripheral continues independently data collection while the processor executes a sensor application program to extract relevant sensor results, like a heart rate, that can be communicated to the micro-controller for higher-level decision making and wireless transmission to the WASP gateway and enterprise backend. In addition to the event-driven architecture, an Application-Specific-Instruction-set Processor (ASIP) core has been designed using the HiveLogic processor (design and compiler) tool platform of Silicon Hive (http://www.siliconhive.com/). This ASIP core has been designed for biomedical applications based on ECG, EMG, activity, and EEG

monitoring and more details of the system architecture can be found in deliverable D2.5 " Bio-ASIP Architecture and usage of WSim".

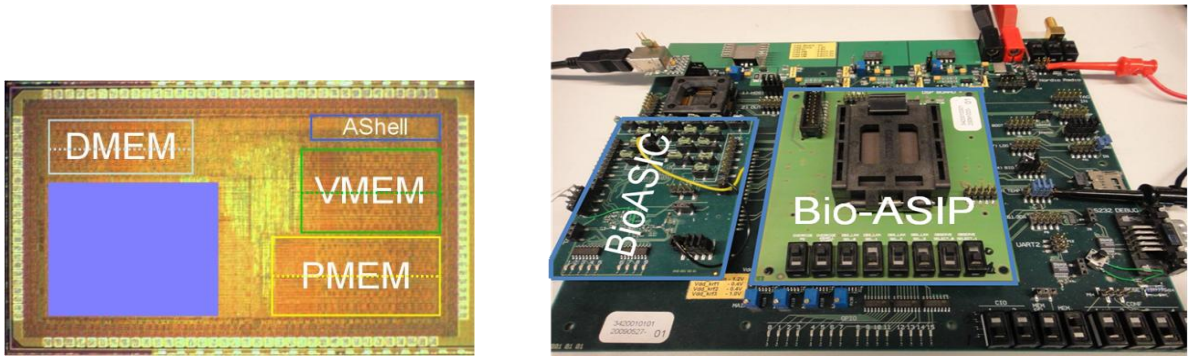


Figure 12: Die picture of the Biomedical-tuned ASIP (Bio-ASIP)and evaluation board used for functional verification and power measurements (taken from D2.5).

Following the promising first design explorations and FPGA implementation within WASP, IMEC-NL and PRE decided to internally tape-out a first test IC to prove the novel architectural concept and obtain power / performance figures for a first IC reference implementation. A picture of the test IC manufactured in 90nm CMOS is shown in Figure 12 and measurement results are included in D2.5. The test IC proved functional and confirmed for a CWT-based ECG (R-peak detection) test application that 11µW operation at 0.7V can be achieved. Several hardware (implementation) and software (mapping) improvements have been identified that can further lower power consumption in active and sleep modes and, thereby, push the average BioASIP power consumption well below 5µW. As discussed in D2.5, at such low power ranges other components start to dominate which in assessed ECG test system proved to be the ECG analog frontend and the ACD of the EEG frontend IC that we've used to digitize the ECG samples before reading in device 1.

To estimate the power comparison of a typical "state-of-the-art" single-core (MSP430 only) system versus a novel 'dual-core' (MSP430 & Bio-ASIP) system, the WSim simulation platform was used. It showed that 94% reduction in power from 1.68mW to 102µW has been estimated of which 1.39mW can be ascribed to removing the data acquisition and, especially, the compute-intensive (CWT-based) ECG processing from the MSP430-CPU to the Bio-ASIP-core. With a power budget of 102µW for the R-peak detection application, further optimization of the Bio-ASIP below 11µW seems not key priority but may become relevant for more advanced ECG applications, like multi-lead ECG systems for medical-grade arrhythmia detection, that all depend on reliable R-peak detection and, subsequent, compute-intense stages. Also, the combination with activity monitoring seems attractive to identify situations of high motion and, potential, ECG motion artefacts that reduce the application quality.

At present, partner PRE is assessing business interest within the Philips Healthcare Division. Also, results will be presented & published at the 2010 IEEE Asian Solid-State Circuits Conference, to be held in Beijing from 8-10 November and second paper has been submitted to the DATE 2011 conference.

3.2 WASP node software platform

The aim of the node software platform is to provide an infrastructure that bridges the semantic gap between hardware and protocol/application implementation, to abstract away differences between different hardware platforms to provide a simple to use, consistent hardware interface for WSN (protocol and service) developers, to expose necessary “hooks” for adaptation and optimisation of energy consumption and quality of service and, to this end, to support cross-layer optimisation. Section 3.2.1 describes the main software platform developments that have integrated in the WASP test beds and other demonstrators whereas in section 3.2.2 some of the more innovative activities are described.

3.2.1 Integrated software platform

3.2.1.1 WASP-OS API

The WASP OS API is an Operating System Abstraction Layer (OSAL) designed to ease portability of embedded software across different software and hardware platforms. In particular, it addresses the discrepancies among operating systems with respect to their functional API, hardware configuration mechanisms, resource management and peripherals handling.

As described in more detail in deliverable D3.5, the WASP OS API is a software layer set on top of the OS, which translates system primitives from the target operating system into a unified API. Thus, application builders make use of a common set of functionalities. The abstract layer embraces the management of hardware configurations and access to specific set-points which represent a major hook to performance trade-offs. It defines a subset of OS primitives which satisfies the basic application builder's requirements but at the same time, remain simple to match most target OS.

The resemblance of the subset to a POSIX OS API is motivated by the aim of reducing the learning-curve as well as the preference of a neutral reference without specific features from any particular platform. In particular, this choice allowed incorporating a number of development tools from the open-source GNU/Linux repositories to the embedded software development environment (in next section) where development and, especially, debugging is much more constrained.

Within the project, the WASP OS API has been successfully implemented for the MSP430 on top of MANTIS OS and FreeRTOS. While both versions have been tested for the evaluation of the abstraction concepts, the former has been adopted as the reference operating system for the development and deployment of WASP integrated test-beds and demos on the BSN platform. Nonetheless, an additional step towards portability has been proved with porting the WASP OS API on top of the ReISC platform, as described in section 3.1.2.1.

In D3.6, the additional cost of the abstraction layer has been analyzed in detail. The main conclusion to this regard is that the overall overhead in terms of memory and byte-code is negligible while the efforts of portability are drastically reduced.

3.2.1.2 Embedded Software Development Environment

To enable integration of embedded software components developed by multiple partners within the project, we have established an embedded software development environment. This environment essentially provides the means to compile, debug, and flash embedded code and has been has been fundamental to the successful development of various integrated demonstrators including the challenging test bed demonstrators for elderly care and herd control. Our real-life integration experience confirms that a good embedded software development environment is a key element in the tool box displayed in Figure 4 of Section 2.4

to ease WSN development for WSN system integrators and protocol developers. Key parts of the development tool box are introduced in this section.

The **Build system** framework has been WASP's central solution to build the software images to be installed on the nodes. It supports building, debugging, disassembling, tracing, and simulating of libraries and applications. It supports easy ways to select the output configuration (debug versus release; hardware type, and more). It supports flashing software images onto hardware nodes in a convenient way. The build system integrates WASP tools and libraries easily, to give new developers a quick start. Finally, it can bootstrap some of its embedded tools (such as the simulator, compiler, and debugger, i.e. WSim and MSPGCC, and GDB) from source. Details are described in D7.4 "The WASP Development Handbook". The build system has been successfully used throughout the integration phase of the project.

WSim is a full platform hardware simulator that is able to use target binary code to analyse runtime behaviour of the sensor node hardware, that is modelled by a pre-defined description of its hardware blocks and associated physical interconnections, in terms of the occurrence (and order) of block-level events and messages. By using binary code, WSim is independent from any programming language, operating system or libraries and when coupling the energy simulator eSimu energy consumption can be estimated as well. More details on WSim and its use can be found in D7.4 "The WASP Development Handbook". WSim has been adopted by WASP development partners to debug and test network protocols and embedded software stacks on the single-core (MSP430-based) BSN nodes used within the WASP test bed deployments. Various modifications have been made to support the BSNv3 platform and debug features have been added following the user feedback and use cases and these improvements are already exploited in several teaching courses for both graduate and undergraduate students at universities.

MSPGCC is an extension of GCC, "The GNU Compiler Collection", to the MSP430 target. GCC is a free-and-open-source software package that contains compilers for various languages, among them C and C++, and for various targets. There has been a prior open-source port of GCC to support MSP430 targets dubbed MSPGCC, but it was out of date when we started using it, and supported only GCC release 3.2.3. Moreover, the MSPGCC source appeared unmaintained. We have encountered various code quality problems and decided to upgrade and bug fix the MSPGCC port. We have succeeded in updating the compiler (GCC), debugger (GDB), and related binary/object file utilities ("binutils") to newer releases, namely GCC 4.4.X, GDB 7.1, and Insight (a graphical debugger). This effort was split out of the WASP development environment and made available separately through http://sourceforge.net/projects/mspgcc4/ again as free and open source software, and has attracted some public interest. Subsequently, it has been extended by an updated C standard library, and maintainer-ship of MSPGCC4 port has been passed on to Peter A. Bigot who had approached us and volunteered to take this project under his wings. His long-term plan is to revive the original MSPGCC project and merge our MSPGCC4 achievements, so that there is a central place to obtain the compiler, debugger, runtime and C support library from. The WASP project has benefited from the updated compiler because it has overcome some issues we had when intact source code was previously wrongly compiled because the more efficient code allowed us to pack more functionality into the tight memory, and because the debug information is much more accurate and useful. Details are described in Section 3.9 of deliverable D3.6.

A **WASP private Subversion (SVN) repository** has been used to facilitate multi-partner development and integration by storing all source and library codes and some parts of the documentation. Subversion is a free-and-open-source software revision control system for text and binary files that uses a central server and networked clients. A WASP *public* SVN repository has been created in August 2010 (http://wasp-public.gforge.inria.fr/) where partners will upload parts available for use in open community for research purposes.

We have used a **Virtual Machine Image** (or virtual machine environment, VME) of a Ubuntu Linux PC that is pre-installed with the WASP-related tools, development tools, and some software packages to ease communication with the host PC (for instance, it supports Windows File Sharing and allows being mounted as network drive), and came pre-installed with the WASP environment. This image is able to run with the "VMWare Player" and Sun/Oracle "VirtualBox" virtualization products that are available free of charge for several operating systems. The virtual machine has been successfully used for training inside the project, as a productive reference system for software integration and development inside WASP, and to ease installation and getting new developers started quickly.

3.2.1.3 **Implementation support**

This was a dedicated task that was laid out to provide missing functionality to other work packages. As such, it was a crucial task and indispensable for a successful integration in the test beds. Some of the work to be done was foreseen and planned through the description of work and detailed implementation plan, some of the need for support has arisen only during the course of the project while integrating. More details on the work can be found in Section 3.2.2 of this document below, and in Chapter 3 of Deliverable D3.6 "Approach for Pre-Compilation Tools; Implementation Support".

The planned work that we have completed comprises an **OS-independent over-the-network reprogrammer**, a **real-time enabled blackboard** implementation including a MANTIS OS scheduler extension for timeliness, a **MANTIS OS driver for the serial flash memory chips** found on the BSN nodes chosen for the test beds, and an **extension of the CC2420 radio driver to provide real-time time stamping** for synchronization and preamble extension for WiseMAC.

Unplanned work comprised several adjustments and improvements that had to be made to MANTIS OS, which worked well initially, but showed some scalability issues as we scaled up our test scenarios to higher numbers of nodes. Notably, we had to change MANTIS OS to **allow interrupts in the MAC's clear channel assessment** to avoid clock drift and false assertions that nodes were failing, we had to **adjust the reboot and initialization procedures** to make them more robust, and we had to **add PPP protocol support to the UART driver** to be able to receive these packets reliably. We also had to rework major parts of the MANTIS OS scheduler and timer implementations, and hardware-to-software timer mappings, in order to get accurate timing and make the MANTIS OS scheduler really pre-emptive in all circumstances.

3.2.1.4 **Main integration lessons learned**

As clarified in the previous section, we had to devote quite some time to address short-comings in the essential components we had chosen. MANTIS OS offered several advantages, and it worked well to jump-start WASP integration for small-scale deployments. However, when the deployments were scaled up, the OS exhibited some issues that we could not foresee or test earlier. These issues were time-consuming to address, so that for future integrated projects, a partner or contractor to address specifically such support and adjustment needs at a technical level should be included, so that other partners can pursue activities in their direct (research) interests.

It would have been beneficial if we had been able to deploy full integration testing from software components and changes checked into the subversion (SVN) repository, however, due to the nature of development, we could have done that only towards the end of this project, because the node software framework had not been able – unless we had resolved to use established operating systems such as TinyOS, that would have bound us in other ways to outdated software engineering practices that we have avoided.

Future projects using WASP results might exploit the results to actually set up a test bed that can automatically build the software and flash a set of nodes and actually exercise the changes, to catch quality regressions early on.

The central Subversion repository however, even if it could not provide automated testing, has still proved very useful because – after initial fixing to support the variety of desktop operating systems in use – all partners could use the same source (or proprietary object/library) code and had the same WASP development environment available.

Also, the WSim simulator, thanks to its instruction accurate simulation and its GDB interface, made debugging so much more efficient, because it did not require extending the source code by debug statements that might have skewed timing behaviour, but allowed inspection of variables at any point in time. Such debugging facilities are highly recommended for future projects.

3.2.2 Software platform innovations

3.2.2.1 Real-Time Enabled Blackboard

It was found that, although real-time systems and publish/subscribe systems are generally accepted as useful, actually used, and well-established, little research and development had been done to combine these, especially in the context of wireless sensor networks. There have been heavyweight solutions such as the Object Management Groups Data Distribution Service, however the complexity of this approach makes it unsuitable for wireless sensor networks.

In the WASP context, it was shown that even for low-end computers such as WSN nodes, it is possible to maintain the nice de-couplings in time, space/naming and synchronization that the Publish/Subscribe concept offers, while adding support for timeliness and hard deadlines. To that end, a concept and prototype have been worked out, implemented inside MANTIS OS including the required OS adaptations such as a real-time scheduler extension, and been provided to the project partners. These components will be published to the public WASP SVN repository as Free and Open Source software under a liberal (BSD-style) license. The concept was presented at a peer-reviewed workshop [37].

This real-time enabled blackboard can be used in the future to allow timely delivery of critical events, as central storage for system parameters that should be detached from a layered architecture in order to support cross-layer optimization further and leans itself towards data-centric operation of networks.

3.2.2.2 Pre-compilation tools

The study of pre-compilation tools was motivated by the problems raised during early integration stages of the WASP OS portable API. The lack of low level configuration tools that could help the system integrator during the hardware configuration phase geared us towards two main directions.

The first prototype tool that emerged from this finding is a configuration generator that can handle the whole clocking system of a micro-controller. These configurations are built using an abstract representation of the hardware registers and clock distribution on the one hand and using application requirements on the other hand. The result is a ready-to-use set of register configurations that can be included in the application in order to set the run-time parameters while optimising the running frequencies for low power consumption. The current version of the tool can only generate exact configurations that are sometimes too strict for real world application. This observation also high-lights the importance of automated tools for configuration generation as real world application have mostly complex timing requirements that need to be mapped on hardware. Doing this mapping manually is time consuming and

error prone granted that there might be no drift-free configuration available on a platform for a given application.

The second prototype done for pre-compilation tools is also a mapping application made to match and optimise the complex behaviour of low level details of MAC layer network protocols on top of the radio communication device. The current work allows to optimise the mapping for power consumption by mapping idle states to the lowest achievable energy consumption state while ensuring application timing constraints. This prototype tool can generate skeleton code for threaded OS like the WASP OS or handlers for event driven OS like TinyOS. The problem of finding an optimised path from one hardware state to another has been proven NP complete and early benchmarks using the B-MAC network layer are promising. These last observation also high-lights that there is a definite place for computer aided software integration using tools to generate optimised code.

3.2.2.3 **Portable MAC**

Over the past years, a wide variety of different MAC protocols have been designed suiting to different applications and network requirements. Many challenges exist in implementing MAC protocols across multiple transceiver hardware implementations and processor architectures, some of which are peculiar to the requirements of MAC protocols in general, and other are a result of the plethora of system and processor architectures in the embedded systems domain. In the WASP project, we investigated the design of platform independent implementations of MAC protocols that allow portability across different operating systems and sensor node platforms. We have analyzed the concepts, importance, requirements and tradeoffs of a portable MAC framework. Our framework allows the implementation of a MAC protocol independent of the underlying hardware specifics and, hence, portable across different sensor node platforms. We have carefully identified the different software and hardware dependencies influencing MAC code development and defined a bottom level API providing all the functionalities needed for a MAC implementation. We have investigated different hardware characteristics and the diversities existing among different microcontrollers. Additionally, we have also analyzed the software support aspects by the operating systems from a set of operating systems being used in sensor networks. Based on the analysis and investigation, we have proposed a list of abstractions needed by the portable MAC framework. Our suggested bottom level API provides all the underlying platform dependencies. It abstracts the platform so that the implementation of the MAC is merely reduced to the state-machine realization of the MAC and makes it fully portable. Furthermore, we defined the top level (or MAC level) API to be used flexibly by the higher layers of the networking stack. We have also considered existing MAC protocol implementations and their APIs in order to check their conformance to our suggested APIs. More information on the portable MAC activity can be found in deliverable D3.5 and/or a journal publication [1].

Within WASP, we implemented a Traffic Aware MAC protocol (TrawMAC) compliant to the portable MAC APIs. TrawMAC [3] is a low-power preamble sampling MAC protocol and part of the alternative CCBR/TrawMAC network stack described in section 3.3.2.2. This alternative network stack has been implemented (and optimized) in TinyOS that uses a single-thread & non-preemptive (concurrent) task model. In the integrated software platform used within WASP to build the test bed demonstrators, a multi-thread & pre-emptive kernel is selected and, at present, use is made of MantisOS. Porting TrawMAC to this kernel has not been performed to focus the effort and to avoid a strong dependence on the WASP software platform developments. However, a future port of TrawMAC has been greatly facilitated by the implementation lessons of WiseMAC, that like TrawMAC is based on preambles for low-power operation, and are described in section 4.4.1.2. Besides the need for proper primitives on the target software platform, a proper split of TrawMAC functionality to relocate time-sensitive parts into (portable) interrupt routines and radio drivers is essential to achieve predictable and required timing behaviour.

3.2.2.4 **Non-functional aspects framework**

The goal of the non-functional aspects framework is to provide an abstraction for the management of the extra-functional (also denoted as non-functional) aspects of an application deployed on a wireless sensor network. Such a framework is composed of data structures, libraries and tools. The main issues considered in designing and developing the framework are the following:

1. The framework implements both static configuration (i.e. compile-time) and dynamic management (i.e. run-time) mechanisms.
2. The libraries and the code generated by the toolchain have been designed to be easily portable across different node platforms, i.e. different hardware and different operating systems.
3. The libraries and the generated code have been structured to have minimal impact on the target application, both in terms of code and data sizes and in terms of execution time overheads.

The implementation of the framework requires that each component, say foo, is described by three files: foo.h and foo.c store the declarations and the implementations of the different modes, respectively, while file foo.xml collects all the non-functional information about the component in the form of an XML tree. The framework thus uses XML descriptions at compile-time only. When the static configuration is completed, the required portions of the XML descriptions of the selected component are translated into highly optimized static C data structures by means of code generation. While the details of XML descriptions are omitted here for the sake of conciseness, it is useful to shortly consider the organization of the data structure that is used at run-time.

To achieve portability, the non-functional aspects API are implemented almost completely in pure C and make a very limited use of the underlying operating system resources. The non-functional API layer is composed of two different groups of functions: the metric & mode management functions and managers support functions. The former group provides an interface for accessing the component descriptions (modes and metrics) while the second is used to perform all common operations involving the manager. The interaction among the application, the managers and these two groups of programming interfaces is depicted in Figure 13.

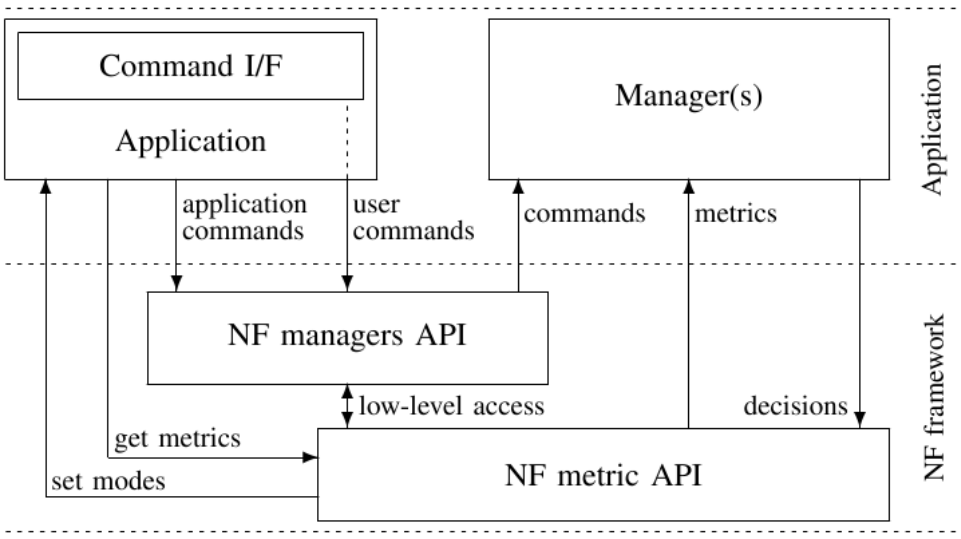


Figure 13: Interactions through layers

While executing, some functions may change their non-functional behaviour, leading to metric updates, saved in the descriptors through the NF metrics API. When invoked – either periodically or implicitly – a manager accesses the saved metrics and makes decisions on the modes to be used, which are communicated to the application, again through the NF metrics API. The NF manager API have two different goals: on one hand they implement the set-up and configuration procedures for the managers and, on the other hand they implement a communication channel from the application (either operating autonomously or under the control of a remote user) to the manager. Figure 13 shows the interactions between the non-functional layer and the application layer. As already mentioned, component management can assume one of the following forms, leading to different modes of interactions between the blocks:

- Explicit management. The mode to be used for each component is fixed explicitly at compile-time and cannot change during execution. The same component can be invoked in different modes from different points in the application. Explicit management does not require a run-time manager. It is worth noting that this type of management can always coexist with the remaining.
- Implicit management. The manager is invoked prior to each call to one of the components subject to management to decide about the mode to use for that call.
- Periodic management. The manager is invoked periodically to decide about the best modes to use for all the components under its control for the next period of time. This type of management can further be classified as autonomous or non-autonomous. In the first case all decisions are taken autonomously by the manager without any influence from other nodes or from a supervisor, while in the second case the node must have an interface for exchanging commands and information with other nodes and a manager can thus be controlled remotely. It is worth noting that, in principle, implicit and periodic management may coexist and influence different sets of functions.

To demonstrate a possible usage of the NF framework, an NF-enabled fall detector has been successfully implemented. With the demonstrator architecture depicted in Figure 14 we are able to monitor the aftermath of a fall in order to infer whether the fall has no consequences or it requires the intervention of a doctor in an elderly care scenario. More specifically, if after a fall the person wearing the node is able to walk, we can consider it as a fall without important consequences. If otherwise we don't detect any further activity after the fall, we can assume that the fall had compromised the ability of the subject wearing the node to stand and walk, and thus we can trigger an alarm.

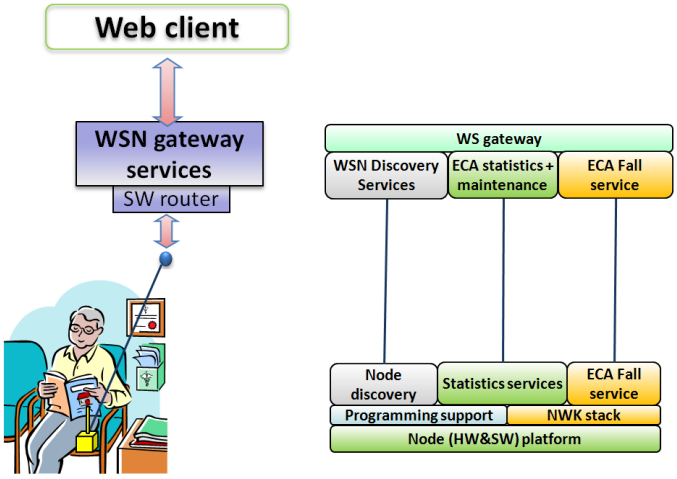


Figure 14: Non-functional demonstrator architecture

Continuously calculating the activity of a node is a power consuming task. With the aim of improving the battery life of the node, we can perform such task only after a fall has been detected, using a simple and fast fall detection algorithm for such purpose. The NF framework allows to implement such scenario, as the NF manager can dynamically decide whether to use a simple fall detection algorithm, or a complex activity classifier. By default, the NF manager lets the simple algorithm run, feeding it with the data coming from the acceleration sensor on the node. When the simple algorithm detects a fall, the manager switches the execution to the activity classifier. The activity classifier is responsible for detecting whether the person wearing the node is standing or walking. As long as the classifier keeps detecting the person as standing after the fall, we are constantly interested in discovering whether he has recovered, therefore the manager continues the execution of the activity classification function. After a fixed amount of time, an alarm is triggered to inform that a severe fall has been detected.

Warning time related to a fall can vary from person to person. In order to avoid false positives, we have to customise the time period according to the health condition of the person wearing the sensor node. Customising fall alerting through ECA actions basically enable tuning both the sampling time and the amount of time after which the alarm is reported to the base station.

The implementation of the fall detector gave the opportunity to evaluate some design and implementation choices that have been made during the project. Regarding the NF demonstrator, the chosen fall detection algorithm proved to be too much prone to false positives (raising fall alerts when the person wearing the sensor had just sat down), and required fine tuning to obtain acceptable results.

3.3 WASP network protocols

The exploration of networked protocols has been pursued along three venues: (i) improving existing protocols or creating new and better ones, (ii) cross-layer optimizations, and (iii) tuning the parameters of a stack of protocols to reach the best of the network. This resulted in the creation of tools for simulation (see Section 3.3.3) and implementation (see section 3.3.1.1) as well as actual implementation of four stacks. In Section 3.3.1, two stacks are described that have been integrated in the WASP architecture and tested in the herd control and elderly care test beds. The other two stacks have been implemented in stand-alone tests to assess alternatives to the integrated WASP solution. Section 3.3.2.2 describes the main findings of the stack based on Content and Context Based Routing (CCBR) and TrawMAC studied to explore advantages when integrating content information in the routing layer. Section 3.3.2.3 describes the findings of an "all-IP" architecture which incorporates a 6LowPAN stack. Finally, we show that another venue for optimization is to select the right protocol for the right application. An automatic procedure for this selection is presented in section 3.3.4.

3.3.1 Integrated Network Protocols

Two protocol stacks have been developed to support the WASP test bed applications requirements. One protocol stack targeted Herd Control applications in which mobility and multi-hop transmission in an ad-hoc mesh topology was needed for a relatively large number of nodes located within a barn or outside on a pasture. Elderly care needed another approach to support communication in indoor settings, with multiple nodes in a multi-hop star topology, and outdoor settings, with several nodes in a single hop to a wearable sink. For elderly care, high packet delivery ratio and short delivery latencies are very important.

To support reusability in stack development, we decided to define our own implementation architecture i.e. the WASP postmaster described in Section 3.3.1.1.

Besides being implemented, these two stacks have been used as a reference to pursue two of the optimization venues being QoS trade-off analysis and cross-layer (X-layer) optimization as reported in Sections 3.3.1.3 and 3.3.1.4, respectively.

3.3.1.1 WASP postmaster

The WASP protocol stack is designed using the postmaster framework. Here, letters (or service data units) are exchanged between protocol layers using the delivery-service of the postmaster. Additionally to the postmaster, a centralized management information base (MIB) is included to avoid data copying that allows for access of network parameters by the application services.

Figure 15 gives a simplified view of the software stack running on the sensor nodes including the Wasp Postmaster (WPM). The main idea of the WASP Postmaster stack architecture is that the different stacks are built around entities that can communicate in an asynchronous way. Each entity can be addressed using its ID which should be unique in the system. Each of the entities included in one stack should process its given tasks as fast as possible to allow an efficient system.

The stack entities could be interchanged separately between the two Stacks (Herd Control and Elderly Care) used in the WASP Project. Some of the Protocol Entities are shared between EC and HC namely these entities are the Transport and Simple-Network entity which provide a simple messaging API to the Application developers.

The MAC and routing layer entities where successfully exchanged between the HC and the EC Stack during the development of the test beds. This exchange is done very easy during the initialization of the postmaster by the application. The protocol developer (of system integrator) simply exchanges the protocol initialization functions and registers the entity id of the new

inserted protocol to the upper and lower layers. The entity id is also part of the transmitted messages so that the receiving WASP postmaster knows which protocol entity is needed to handle this message. This proves the versatility of the postmaster approach for protocol stack implementations.

The principle of the Postmaster approach is to invoke the unique layer function (entry point) whenever a letter for that layer is waiting in the letter queue. While the layer function executes, no other layer function is able to run (no pre-emption within the stack). As a consequence, the layer function execution should be kept to a minimum to maintain reactivity (in particular, wait operations are prohibited). As explained in Section 4.4.1.2, this caused problems when integrating WiseMAC that could be solved by integrating time-critical WiseMAC functionality in the radio driver.

The centralized buffer management has the advantage that it could be changed easily without the need to change every entity implementation. During the development of the postmaster and its protocol entities, the *malloc* based dynamic allocation was changed to a static buffer pool because the dynamic allocation leads to instabilities of the complete system. This is because the underlying MantisOS uses a very simple *malloc* implementation. As every events handled by the entities are send as letters, the entities must be carefully designed to return the letters as quick as possible to the WASP postmaster. If entities keep too much letters in private then the complete system runs out of free letters to deliver new packet messages. This also affects the timer system of the postmaster as Timer events are also sent as letters to the requesting entries. The number of empty letters statically pre-allocated by the postmaster should be carefully balanced to provide a stable system with as much as possible free memory left for applications.

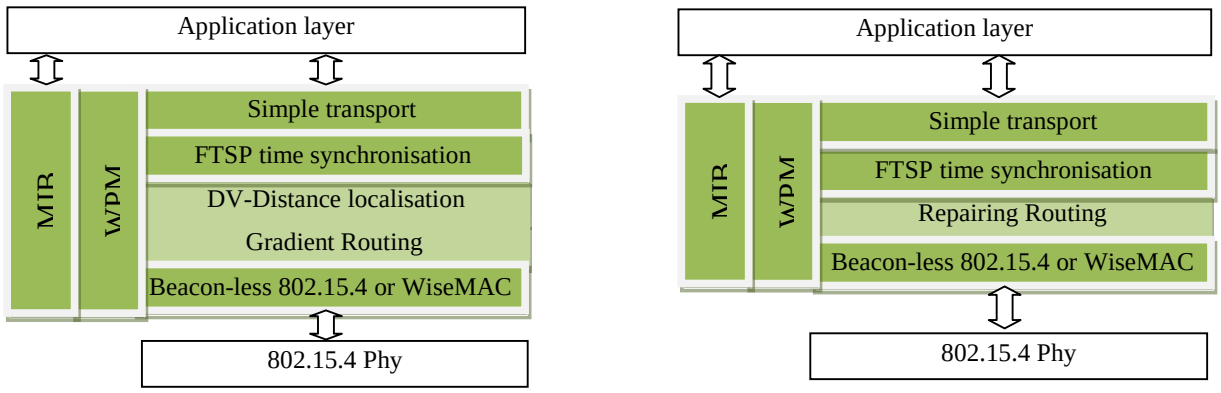


Figure 15: The WASP protocol stacks used for test bed integration. All layers are implemented using the WASP OS API on MantisOS (not shown for simplicity).

3.3.1.2 **Protocol stacks for the EC and HC test beds**

As depicted in Figure 15, both stacks have been implemented within the WASP postmaster (WPM) framework and share protocols for time synchronization, transport, and Medium Access Control (MAC). Two MAC protocols have been used in both stacks of which the **beaconless** (IEEE 802.15.4-like) **MantisMAC** proved convenient to jump-start first network and test bed integration tests while, in parallel, integration of **WiseMAC** (allowing radio duty-cycling) within the WASP multi-tasking environment could be pursued. The stacks differ in the

routing protocols being used and the inclusion of DV-distance for localization in the herd control test bed.

For herd control, a routing solution is required capable to support large-scale & mesh-topology deployments and to deal with mobile nodes. In D4.1, we have looked at various broadcast-based protocols which send duplicate packets across multiple routes towards a sink and, thereby, greatly enhance packet delivery in a mobile environment. From the protocol candidates, we have selected **Gradient Routing** because of its ability to work in a multi-sink deployment and its simplicity for integration. Moreover, previous knowledge available at partner INRIA allowed to rapidly simulate its performance and to efficiently implement it within the WASP postmaster architecture and WASP-OS API. Another important strength of the protocol is its small memory footprint and CPU usage. Based on unexpected packet losses observed in the large-scale herd control deployment trials, Gradient Routing has been extended with a "RSSI threshold" which allows Gradient to drop Gradient packets received below a given RSSI strength. Without this RSSI threshold, a low packet delivery ratio has been observed for nodes that communicate with neighbours at critical low signal strengths. For such cases, we observed that a node could receive periodic Gradient ADVERT messages from a given neighbouring node, thereby establishing its hop distance X to the sink, but wasn't able to directly transmit back to that specific neighbouring node at hop distance X-1. If other neighbouring nodes at X-1 would receive the packet, it would still make it to the sink. However, in the observed cases, other neighbouring nodes were also at X (or further away from the sink) and the packet was dropped and lost since Gradient doesn't use acknowledgements. A detailed description of the current Gradient Routing implementation can be found in D7.4 "WASP development Handbook" and a scaling exercise in Section 4.2.2.

For elderly care, the so-called **Re-Pairing Routing** protocol has been developed to support wireless communication within indoor networks with dozens of nodes. It builds hierarchical routes from sensor nodes, worn by elderly persons or fixed to walls, through a number of wall-mounted forwarder nodes, up to a single or multiple sinks. Depending on the selected route rebuilding period, at which broadcast ADVERT messages are generated by sink nodes (similar to Gradient Routing), Repairing is capable to deal with moderate mobility. Unlike Gradient routing, link-layer unicast communication with acknowledgements is used for communicate between sensor nodes and sinks. Unicast is used to avoid generation of (broadcast typical) duplicate messages thereby saving energy of (battery-powered) neighbour nodes and, moreover, minimizing network traffic and thereby delivery latency. Acknowledgements are used to improve packet delivery which for medical applications is quite an important requirement. Similar to Gradient Routing, we've introduced an RSSI threshold in Repairing to improve its robustness. A detailed description of the Re-Pairing routing protocol can be found in D7.4 and a scaling exercise is given in Section 4.2.2.

DV Distance is a distributed localization protocol that has been developed and tested for the herd control test bed. It provides mobile (blind) nodes with an approximation of its actual position by an RSSI-based estimation of distances with (at least) three neighbouring nodes that can be anchors (with predefined positions) and/or blind nodes. More details about the protocol can be found in Section 5.4 of D7.4 "WASP Development Handbook". Also, here a description can be found how the ECA services have been utilized to greatly simplify the essential calibration trials involved when tuning the RSSI-based estimations during deployment. In principle, the WASP postmaster approach enables the use of DV-distance in the elderly care stack. In practice, given the unreliability of RSSI readings within the indoor elderly care scenarios, DV-distance has not been used here.

3.3.1.3 **QoS trade-offs for HC stack**

Trade-off exploitation is the technique of using the trade-offs found within the network stack to respond to changes in the desired/provided QoS performance caused by, for example, dynamic changes in the environment. Adequate adaptation of the set-points as a response to

dynamic changes within the WSN, e.g. the running applications, and its environment, e.g. radio interference, is a potential solution to ensure that the required QoS performance is also achieved after these dynamic changes. For the trade-off exploitation, we focussed on infrequent changes within the WSN, e.g. outside – inside, day – night, which allows us to specify well-defined points, denoted as *modes*, at which we can exploit the trade-offs of the network stack. Mode changes are the interesting points in time to consider changing the set-point of nodes in the network.

Figure 16 depicts the methodology that we have introduced to support trade-off exploitation which is based on an approach of finding appropriate set-points for given modes based on simulations (see also D4.6, section 3.3). First, a selection of application scenarios with their operational modes is done. These are then simulated extensively to explore the configuration space and to derive "clouds" of set-points which are possibly grouped as "group equivalent modes". Representative set-point values for each cloud are produced next as "best configuration values" for each mode, reducing the numbers of different set-points to be considered and used at run-time.

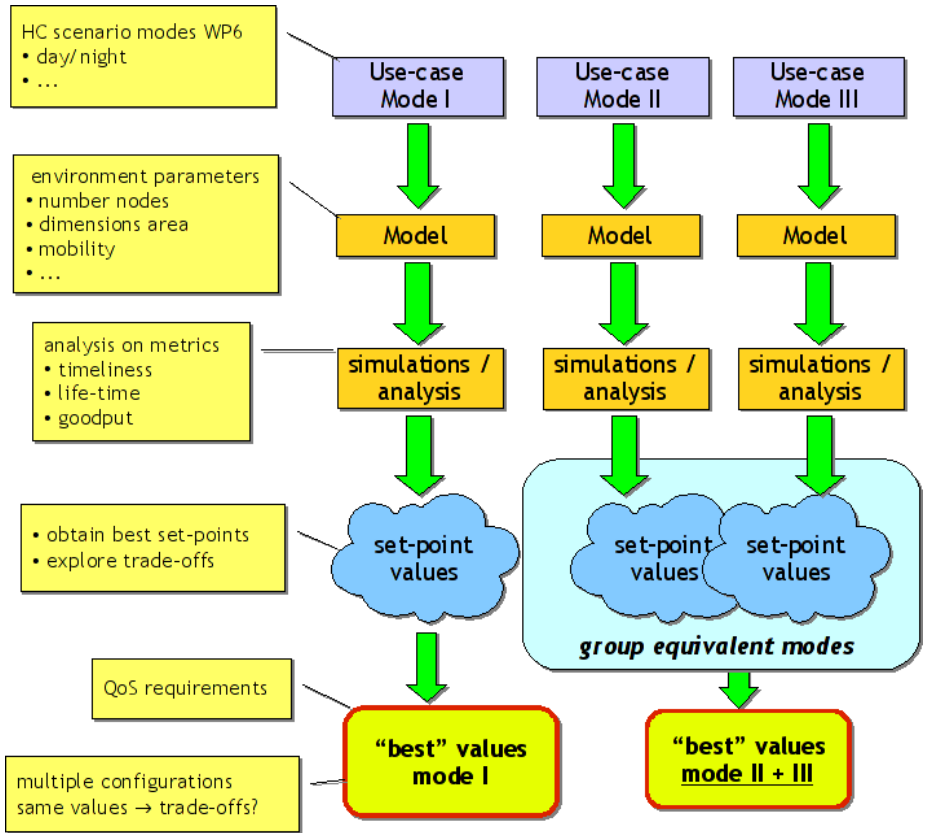


Figure 16 QoS trade-off analysis and mode selection (from D4.6)

To validate this approach and the underlying simulations, we have performed measurements on a small-scale experimental setup that allowed us to actually measure QoS performance, i.e. packet delivery, latency, and power consumption, while modifying several QoS set point within the network stack i.e. WiseMAC sampling interval, TX power, FTSP and Gradient ADVERT periods. While building this demonstrator, we have exploited the flexibility offered by the WASP system, and more specifically the network MIB and ECA services, to remotely monitor QoS performance and modify QoS set points using the Statistics Monitor views depicted in respectively Figure 26 and Figure 27 (in Section 3.4.3.1). These small-case measurements confirm that indeed significant QoS trade-offs can be made but, however,

relative large deviations can be observed between measurements at seemingly small changes in QoS set points. Moreover, as described in deliverable D4.8, future experiments with larger scale and mobile deployments should be done to determine the impact of scale on the simulator accuracy and, thereby, the validity of the proposed QoS trade-off methodology illustrated in Figure 16.

As expected, achieving simulation results close to the experimental data would require similar efforts or in fact performing the same operations, i.e., loosing the benefit of abstraction in simulations. In our study we found simulations which deviate from the experimental results only via a linear factor, which we believe to be a useful abstraction in the environments studied.

3.3.1.4 **X-layer optimization**

X-layer optimization has been addressed throughout the WASP project run time in order to guide protocol selection and keep track of stack integration. In an early phase, we have made an inventory (in deliverable D4.1) of the basic operating assumptions for each protocol but also the dependencies between protocols. The dependencies were exposed in terms of provided and required functions with the objective to identify identical functions that were provided at the different layers. Early small-scale simulations for the herd control stack have been performed (reported in D4.3a and D4.4) to quantify potential improvements in selected metrics obtainable through cross-layer optimization.

With the herd control stack maturing in the final project phase, a simulation model of the herd control stack including all relevant protocols depicted in Figure 15 and a real HC test bed scenario size has been used to evaluate the effectiveness of certain x-layer optimizations as compared to a non-optimized stack. The simulations results are included in D4.8 and show that there are significant improvements that could be gained in goodput while not negatively influencing latency of the network. This can be achieved by collecting Protocol Data Units (PDUs) of different layers into larger packets and, moreover, the send intervals of those larger packets are synchronized to the capabilities of the low layer protocols.

While a non-optimized stack including “duty-cycling” WiseMAC performs worse than the “always-on” IEEE 802.15.4 reference stack with at best 30% of all application data reaching the destination and at worst only 9%. The optimized solution can reach up to 65-70% goodput more than double that of the non-optimized version for time between packet intervals between 1s to 600s. Even the worst case performer increases goodput from 9% to more than 30%”. Our results also show that congestion of the network is a serious degradation issue at high packet generation rates of 1s and that QoS performance is correlated to the amount of protocol data that is transmitted like the periodic DV distance, FTSP, and Gradient advertisement messages.

Further cross-layer optimization options, not included in this study, concerns the use of more advanced duplicate message filters in Gradient Routing (and WiseMAC) as addressed in Section 4.1.2 and implemented in CCBR / TrawMAC. Note that the simulation setup used in Section 4.1.2 involves two groups of 10 and 50 cows that are statically and equally distributed over the simulation area (to limit simulation time) and aimed to get a feeling on practical application and deployment trade-offs. In the X-layer optimization study, a group of 12 cows has been simulated including (constant speed) mobility aiming to explore X-layer optimization potential. Hence, simulation results between the two studies are not directly comparable although the orders of magnitude match and qualitative findings can be extrapolated to some extent.

3.3.1.5 **Integration lessons learned**

- The implementation of time-sensitive protocols such as WiseMAC within a multi-tasking environment, e.g. as provided by MantisOS, requires a serious effort to avoid additional loss of packets and energy-efficiency. This is illustrated in section 4.4.1.2
- It is difficult to take into account the temporal penalty introduced by kernels in network simulators. This penalty is far from being negligible and therefore introduces an uncertainty in the validity of simulation results for multi-tasking environments;
- Many higher-layer protocols (in our case FTSP, DV-distance, Repairing Routing and, especially, Gradient Routing) use broadcast as a means to minimize the number of packets. This is adverse to low power listening MACs such as WiseMAC and TrawMAC. Aggregating payloads (see section of X-layer trade-offs) or concatenating packets (see section of TrawMAC) highly improve the match.
- The use of beaconless MantisMAC, taken from public domain, proved very useful to jump start network and test bed integration for small-scale deployments. However, MantisMAC apparently hadn't been tested for large-scale deployments and therefore required additional effort to remove scaling bottlenecks and incorrect booting behaviour as experienced in the herd control test bed and further addressed in section 4.4.1.1.
- There is little that can be done to optimize energy consumption with (non-beacon) IEEE 802.15.4 MAC which requires the radio transceiver to be continuously in (power-hungry) receive mode when not transmitting itself. Optimizations could come from the application but this, evidently, would require the application developer to understand when to switch on / off the transceiver chip.
- Strict layering and precise interfaces (such as in the postmaster approach) are key to enable the implementation of integrated stacks in a multi-site & multi-partner project like WASP.
- The Management Information Base proved very suitable to expose specific information of the routing protocols to enable remote read-out and control by ECA services and, thereby, to enable run-time performance & topology monitoring and stress testing during actual deployments.

3.3.2 **Protocol exploration**

3.3.2.1 **Lightweight security**

WASP's security architecture addresses the specific operational and technical requirements of elderly care, herd control, and traffic applications. In D4.5, we described the overall architecture, the main functional parts, and the algorithms allowing for the deployment of secure WASP networks.

At the Link layer, we introduced a very efficient distributed key agreement scheme to allow for security within the wireless sensor network. This scheme enables any pair of devices in the network to agree on a common secret out of a set of keying material and an identifier. This secret can enable basic security services at this level such as confidentiality or authentication. The application security layer builds on the link application layer enabling more advanced services running between a single sensor node and backend services. In particular, we describe an end-to-end data encryption scheme that, leveraging on the lower layers, adds complex application-specific security requirements such as end-to-end data access control to the system.

We showed in D4.7 that the studied identity-based key agreement scheme used at the link layer offers a good trade-off between computational, memory, and security requirements, converting it in a suitable option to address the different WASP use cases. It requires much

lesser requirements than related schemes for key establishment based on public-key cryptography [reference 6 in D4.7], in particular, it requires around 1 KB of flash, 20 B of RAM, and a variable amount of memory for the storage of the polynomial keying material. When compared with other polynomial schemes, this algorithm reduces the CPU needs at the price of higher memory needs. Most importantly, the keying material structure allows updating small pieces of information without overloading the communication links. Thus, the introduced key establishment algorithm can be seen as the keystone on which further security services can be built.

We furthermore showed that the application of this key establishment algorithm can actually serve to guarantee more advance capabilities when combined with routing protocols. We designed a secure version of the DYMO protocol used in the 6LowPAN stack of the "all-IP" architecture described in section 3.3.2.3 by applying cross-layer optimizations between identity-based key generation, authentication, neighbor discovery and routing as shown in the figure below. In our approach, secure route discovery between two end-hosts located n-hops away from each other requires n+1 identity-based key generation operations and 2(n+1) encryption/decryption handshakes that can be implemented in a very efficient way with our polynomial-based key establishment scheme.

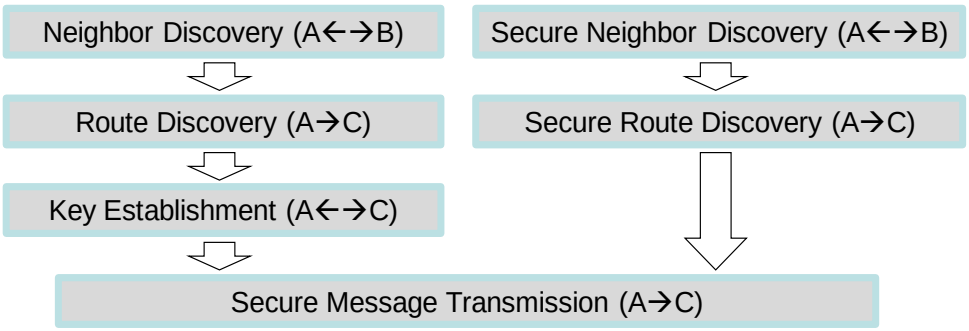


Figure 17: Concept overview: left, traditional system operation; right; cross-layer optimized system

These findings, also presented in D4.7, are relevant to many new systems based on sensor networks, and specially, to new standards, such as 6LoWPAN, in which resource-efficient security primitives are still under research. The outcomes of the security research within the scope of the WASP project will serve as input for standardization activities in the IETF scope. Furthermore, related-polynomial schemes have been applied to medical scenarios and presented at several conferences.

3.3.2.2 **Alternative routing interaction model**

In order to explore an innovative way to support node interaction in a large-scale WSN, we designed Context and Content based Routing protocol, CCBR [2]. It provides a routing solution that integrates the publish/subscribe (P/S) interaction model among nodes of a mobile & multi-sink WSN. The WASP solution also uses a P/S interaction model which, however, is integrated in the application layer through ECA services.

Together with CCBR, TrawMAC [3] has been developed which is a contention-based MAC protocol adopting advanced techniques to minimize power consumption, especially in the presence of the kind of link-layer broadcast traffic originated by CCBR. The initial experience with the two protocols, studied both in a simulated environment (OMNeT++ 4.0 using the CC2420 radio model) and on real nodes (TinyOS 2.x implementation for the TelosB/WASP nodes), has been presented at the 3rd WASP review and is described in D4.4 and D4.5. Furthermore, the representative results have been published in [4]. This shows that the two

protocols work well together and allows achieving good performance at a low cost as compared to other state-of-the-art solutions, including the well-known Directed diffusion scheme which also uses similar interaction model.

Next step was to explore the cross-layer optimization among the two protocols to obtain a fully integrated stack capable of jointly maximizing the performance of the WSN as a whole. In particular, we carefully examined the basic functionalities used by the two protocols to operate and we looked how they could be better integrated, the result was (i) to move some functions originally implemented at the routing layer into the MAC, where they could benefit of the more strict timing of such layer; and (ii) to increase the number of information originally passed from the MAC to the routing, to help the latter in making the best choice in forwarding packets. These optimizations have been described and analyzed in D4.7 and clearly shows further improvements w.r.t. the non-optimized stack. The representative results have been presented in [5].

In conclusion, the main lessons we learnt from this WASP experience can be summarized as follows:

- The publish-subscribe model, and particularly its content-based incarnation [6] perfectly fits the need of a large number of WSN applications, including those explored in WASP, allowing to limit the traffic within the network and thus reducing the energy spent for communication while enabling asynchronous communication between entities within the distributed network.
- Context-awareness, as implemented in CCBR, by allowing nodes to subscribe not only based on the content of messages sent but also to specific sources, based on some characteristics of these sources (their context), increase flexibility in the hands of application designers allowing to further reduce traffic.
- Broadcast messaging (at the link layer) coupled with opportunistic routing and soft routing state, as adopted by CCBR, performs reasonably well in presence of mobile nodes, where the traditional approach to routing, based on standard routing tables periodically renewed and unicast communication to forward messages, is not able to cope with the strong dynamic induced by mobility.
- A contention-based approach at the MAC layer better suits a mobile WSN than a time-division one, which requires strict synchronization among neighboring nodes, hard to obtain in a dynamic environment.
- Preamble sampling based MACs use various different types of optimizations in order to minimize the length of the preamble to be transmitted and received [7]. However, no particular approach shows universally optimized response. Therefore, we combined different types of preamble optimization schemes together in the design of TrawMAC. This has been theoretically proven by Bachir et al. [8].
- Cross-layer optimization among routing and MAC should involve (i) eliminating duplicated functionalities (if any); (ii) deciding which functionality implement at which layer; (iii) deciding which meta-information should flow among the two layers and (iv) eliminating duplicated packets and aggregating different packets together using MAC packet queues to minimize network traffic. When these steps are correctly implemented they allow to further reduce the traffic volumes while maximizing the performance of the stack (delivered messages). This has shown to be true even in presence of two protocols, like CCBR and TrawMAC, which were designed from the very beginning to work together.

3.3.2.3 **All-IP architecture**

In addition to the WASP architecture used for integration, an “all-IP” architecture has been explored to assess potential advantages of extending the use of IP from the back-end applications towards the applications on the sensor nodes. Details of the 6LoWPAN-based

implementation and demo set-up can be found in deliverable D4.6 where a practical firmware footprint of 35kB has been reported which leaves sufficient (13kB) memory space for application code.

The “all-IP architecture” is depicted on the left side in Figure 18 next to the WASP architecture for direct comparison.

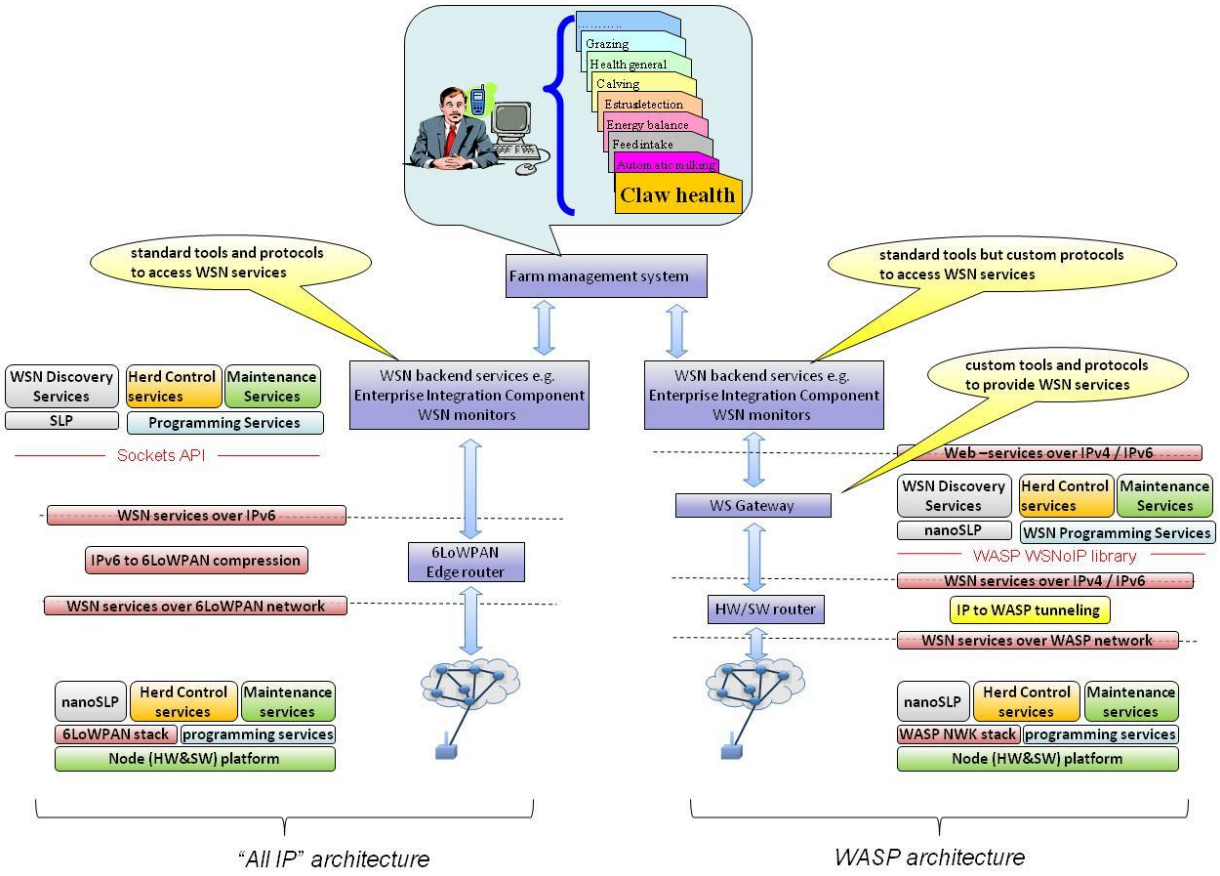


Figure 18: Comparison of “All-IP” and “WASP” architectures.

The figure shows that the “All-IP architecture” uses a 6LoWPAN Edge Router instead of the HW/SW router and the WS Gateway. Furthermore, it allows using standard IP tools to build back-end service WSN monitors for maintenance purposes.

The WASP architecture enables further optimizations in the sensor network that are not possible with a standardized protocol provided by 6LoWPAN. Furthermore, the WASP architecture is not restricted to IPv6 and its IP-based communication may use IPv4 which is still the predominant protocol version in today’s internet. Although it should be mentioned that IPv4/IPv6 conversions in the 6lowpan router are being sold today. Finally, the WS Gateway offers a higher level of abstraction to the end application developer while adding similar flexibility (i.e., support of most operating systems and programming languages for client applications) as 6LoWPAN with its socket interface.

Nevertheless, because of the well-adopted use of standardized protocols, the socket API, and standard maintenance tools, the use of an all-IP solution may become an attractive alternative over custom (partial-IP) solution like that offered by the WASP architecture once IPv6 has become the dominant internet protocol. We are convinced that standard, interoperable protocols such as 6LoWPAN will help to foster the use of sensor networks in commercial deployments. The work by the IETF CoRE working group aims at delivering the “RESTfull” architecture to be deployed on the sensor nodes. The provision of low resource consuming

web-service interfaces becomes a reality in the coming year, thus also enabling higher-level abstractions than currently possible with the socket interfaces. However, in our work on 6LoWPAN based on RFCs and ignoring ongoing Internet Drafts, we identified the following issues that still need further attention:

- Within the sensor network, 6LoWPAN still requires some work to play out its key advantage of interoperability between implementations of different vendors. For our prototype implementation, we had to make several design decisions that were not clearly specified. These were not necessarily related to the 6LoWPAN adaptation layer itself but to the protocols around it. For example, even the routing mechanisms in the sensor network were out of the scope of 6LoWPAN, and it was still under discussion if routing of IP packets should be performed (i.e., over the 6LoWPAN layer) or if MAC frames should be routed in a mesh (i.e., under the 6LoWPAN layer). In the meantime, the ROLL working group has defined the RPL protocol to address routing issues [12], but its applicability to sensor node applications still needs to be proven. Another drawback is the lack of a standardized, true low-power MAC layer. Compared to low-power MAC protocols such as WiseMAC [13], IEEE 802.15.4 – the MAC layer defined as the basis of 6LoWPAN in the RFC [14] – still consumes considerable amounts of energy because it listens continuously for radio traffic. Even some co-authors of the 6LoWPAN RFC do not use IEEE 802.15.4 but a MAC protocol based on the concepts introduced by WiseMAC [15]. For the future we see that the CoRE working group of the IETF addresses the issue of nodes having a duty cycle during operation. But still no clear decisions have been reached.
- If a sensor node becomes directly addressable and reachable from the internet, security considerations and its vulnerability to attacks become more important. It is no longer sufficient to rely for security on link-layer encryption since the attacker can come from the internet. In addition to the attacks known from PCs, it is important to note that even a small number of attackers could result in a successful denial-of-service attack by flooding the low-capacity sensor network. Such attacks may result in permanent failures due to increased energy consumption. To prevent them, a good approach might be to use firewalls and make the sensor nodes only accessible from a gateway such as the WASP WS Gateway. However, in such a combined approach, 6LoWPAN no longer has the advantage of end-to-end IP connectivity.
- 6LoWPAN only reduces the size of the IP and UDP headers but does not address the size of application-level protocols such as HTTP. These protocols are often verbose and not made for small, low-power devices. However, developers of PC applications are used to tools that are based on these protocols (e.g., web services). There are several different ways to address this issue. First, as outlined above, the WS Gateway could be combined with 6LoWPAN to provide the familiar web service protocols to PC applications while talking optimized protocols with the sensor nodes. Second, PC-based tools for the application-layer protocols of the sensor network could support the developers in a way similar to web services (but use an optimized sensor network protocol). For example, we used this approach for the .NET implementation of uDSSP that can run on a PC over IPv6. Finally, similar compression mechanisms as 6LoWPAN uses could be applied to application-layer protocols. This issue is currently being address by the CoRE Working Group of IETF. A clear consensus has emerged in the CoRE Working Group which allows building web services with minimum header overhead. Debate about the compatibility with HTTP is still ongoing. Another issue concerns the length of the Uniform Resource Identifier (URI) of internet resources which can be suppressed completely from the message but then disallows the hosting of multiple services on one node.

These insights gained through the All-IP exploration, addressed in Tasks 4.14 and 5.7, have been instrumental in specifying the application constraints on the low resources network in the context of building control and have been disseminated in the Internet draft presented by PRE during the 6lowapp kick-off (BOF) meeting in Hiroshima and during the CoRE Working Group

meeting in Maastricht where an Internet Draft has been presented on the use of URIs and group addressing.

Note that the developed 6lowPAN implementation of EMIC is planned to be made available in Microsoft .NET Micro Framework 4.2 as open source.

3.3.3 **Protocol simulation tools**

Designing, implementing and evaluating wireless communication protocols is a highly complex process. Fortunately, many techniques exist to reduce this complexity:

- the OSI layered model breaks down the communication problem into easier to address sub-problems ;
- finite state machines (FSM) provide an elegant modeling framework that helps the designer to consider all possible events and states in a systematic way ;
- debuggers and test suites enable the developers to ensure that their protocol implementations respect the FSM design ;
- simulations facilitate the understanding and optimization of the protocols in a variety of configurations.

Wireless embedded systems are difficult to debug, as they involve several independent systems and fast physical processes that cannot be stopped in time by a debugger. This makes simulation tools all the more attractive. Unfortunately, simulation models publicly available at the beginning of this project lacked important features for the evaluation of wireless sensor network protocols, making their use questionable (see references 3, 7, 8, 9 and 11 in [2]):

- power consumption models ;
- interference models (a process that affects power consumption, latency and reliability) ;
- low-power medium access control protocols ;
- wireless sensor networking routing protocols.

The discrete event public source simulation engine OMNeT++ was selected at the beginning of the WASP project. It offers basic modeling blocks, a clean separation between simulation models and configurations, and various tools for statistics generation, collection and analysis. The MiXiM (previously named mobility framework) open-source modeling framework was used as starting point for the development of the WASP simulation models, mainly for its support for modeling mobility and wireless communication.

During this project, we developed, calibrated and validated the following models [9]:

- a detailed radio model that takes into account transient states (e.g. radio reception mode setup), during which well designed low-power MAC protocols spend a non-negligible time
- a detailed Signal to Noise plus Interference reception model (see ref 13 in [9])
- a model of the IEEE 802.15.4 non beacon enabled CSMA protocol
- a model of the ultra low-power WiseMAC protocol (ported only) and TrawMAC
- a model of FTSP taking into clock drift in nodes
- numerous routing protocols.

While the algorithms behind most of those models were not new in themselves, we do not know of any other comparable work regarding the validation of simulation models (see in particular [9] and [11]) for WSN that consider simultaneously packet success rate, latency and power consumption, with such a degree of accuracy. Many of those models were later made

available open source. The OMNeT++ user community demonstrated high interest for some of these models, and they have later been integrated into MiXiM 1.2 [10]. In particular, the IEEE 802.15.4 CSMA protocol was selected as the default MiXiM CSMA implementation and is used in the examples for new users.

There is now ongoing work to merge the MiXiM wireless sensor networking framework with the INET internet protocols framework, to combine their various models. This will make the WASP simulation models accessible to an even wider community and facilitate the study of heterogeneous networks, for instance that combine multiple physical layer technologies (wired and wireless).

Finally, we observe that the WASP project significantly improved the quality of open source accurate models for WSN simulation, and that the quality and pertinence of this work was recognized by researchers outside the WASP project, working with WSN systems as well.

3.3.4 **Methodologies for protocol selection**

Most WSN are used at their limits: objectives may differ but users would like to get the best of their investment. For instance, some would like to reach the longest lifetime. Others may privilege latency. In many cases, they want to limit their investments which means they will push for the lowest node density that provides the required information and still allows communicating adequately. As a consequence, it is very unlikely that a general purpose WSN will ever succeed and be able to serve all needs.

To meet the user goals and needs, the WSN solution should be optimal towards one or more objectives. This may be achieved by selectively tuning the parameters of the different layers of the protocol stack (as in 3.3.1.3), defining new protocols (3.3.2.2) or optimizing existing ones (3.3.1.4). Here, we explored an alternative approach consisting of selecting the protocol layers to meet these needs as closely as possible. The result may be later enhanced by properly tuning the parameters (as indicated above) or by alternative methods, such as cross-layer optimizations.

Finding the right protocols for a given application using a WSN is not trivial. Among other challenges, it involves defining the application needs in a clear manner, which should then be matched with protocol properties. However, this is often not sufficient to guarantee the match for a number of reasons. First, the application needs may be incomplete and, for instance, might not include some regulatory parameters. Additionally, the selected protocols may also be incompatible. Finally, -and this is particularly the case for "near the limits" applications-, detailed calculations or simulations are necessary to check whether the selected protocols are effectively able to fill the needs.

Because of these difficulties, we proposed a 2-step approach. In the first step, the idea is to select one or more protocols for each layer on the basis of the functional aspects required by the application, as identified during the requirement analysis phase. In a second step, this short list is analyzed into details using simulations to check if the non-functional application needs (performances) may really be met. This last step is rather conventional and will not be detailed further here.

During the first step, rather than choosing the best protocols, which would imply ranking the protocols according to some criteria, we first try to eliminate or disqualify protocols that do not fit the application scenario. If possible, those that are kept will be qualified in a qualitative manner.

The protocols remaining after the selection process could not be incompatible between themselves. One way to detect the incompatibilities ahead and prevent their selection is to check on which basic assumptions the protocols are built. Different assumptions are likely to lead to incompatible protocols.

Table 2 presents the list of top selection criteria identified during the project. This list is derived from the list of protocol assumptions (from D4.1) and the list of application properties (from D4.2). The elimination process starts from the values of the criteria for the given application. It also uses the list of candidate protocols and the value for the different assumptions on which they are built (D4.1). The process looks at each criterion and checks which of the protocols are compatible with it. If the protocol is not compatible, this means it cannot be used for the given application and it is eliminated from the candidates. If it is compatible, it stays in the list of candidate protocols and may be assigned a degree of compatibility that will be used to ranks the remaining protocols. This procedure is repeated until all criteria have been scanned. The protocols that have not been eliminated possibly carry for each of the criteria a degree of compatibility. These degrees may be combined to rank the compatible protocols from most adequate to just acceptable.

Criterion	Comments
Traffic model (deadline, period, inter arrival, ...)	Load evaluation for ranking.
Reliability constraints	allows to reject solutions without retries
Maximum distance between nodes	allows to reject single hop solutions
Mobility or Immobility	allows to reject solutions based on long associations
Coexistence with other systems	allows to reject solutions that need planning
Dependence on Infrastructure or not	Allows to reject protocols that rely on this when this is not available
Single, Multiple sinks or Other patterns	Allows to reject protocols that do not support multiple sinks when this is needed by application
Energy constraints	If the constraint is on all nodes, this eliminates solutions with special coordination roles
Position referenced nodes	Allows reject protocols that need it when this is not available on the nodes
Simplicity	Ranking criterion

Table 2- List of top selection criteria

We applied our method to the HC and EC cases. The results show that it is possible to drastically reduce the number of protocol candidates for each layer using a limited number of criteria, and a method that which can be applied in a matter of minutes. Whether this process could be automated or not is a good question to which the answer is possibly positive but would need further explorations.

The protocols remaining after the elimination process are also ranked from most desirable to less desirable. The ranking is fuzzy as the technique is qualitative and some weighting of the criteria is needed to aggregate the various rankings.

Compared to our initial intuition that the process would start from a list of parameters for the application, a list of protocol assumptions (not necessarily the same) and a matching procedure, it turn out that we can have a single list and a simple matching process. It may be possible to do a finer selection process with additional criteria (not necessarily the same) but this is the subject of further study.

The method is quite simple to exercise. However, it relies on a previous analysis of the candidate protocols is necessary to extract the various assumptions made. The list we

presented in this paper highly simplifies this work, however, note that this work must only be done once.

Finally, the various criteria presented in Table 2 may greatly help application designers to identify their requirements.

3.4 WASP services

This section gives a brief overview of main technologies that underlie the creation of WASP services. In particular, it focuses on the WASP web-services gateway (WS Gateway), the programming models, and the PC applications like the Maintenance GUI and the EIC. Using concrete examples from the elderly care and herd control test beds, it shows how the developer creates WASP services and connects them with backend applications.

3.4.1 WASP web-services gateway

The task of the WASP WS Gateway is to translate between messages in the WASP sensor network and web services. Within the WASP architecture, the WS gateway raises the level of abstraction for backend developers and provides an interface they are familiar to use. Instead of building compact messages in binary encoding for the sensor network, they can use higher-level web services where they do not have to deal with byte orders and other communication issues. Furthermore, the WASP gateway decouples the sensor network from the backend applications by communicating over the internet and by enabling interoperability among different platforms with web services. Finally, it offers some common functionality that the application developers do not have to implement. For example, it monitors service discovery messages and takes care of uploading ECA programs and sending subscriptions when nodes join the WASP network or (incidentally) reboot.

Figure 19 shows an overview of the WASP gateway's architecture. Besides its runtime and the WSN over IP library, which it uses to process messages from the (hardware and software) routers and GPRS forwarder, it consists of two main parts: the "gateway services" and the "loadable application-specific services".

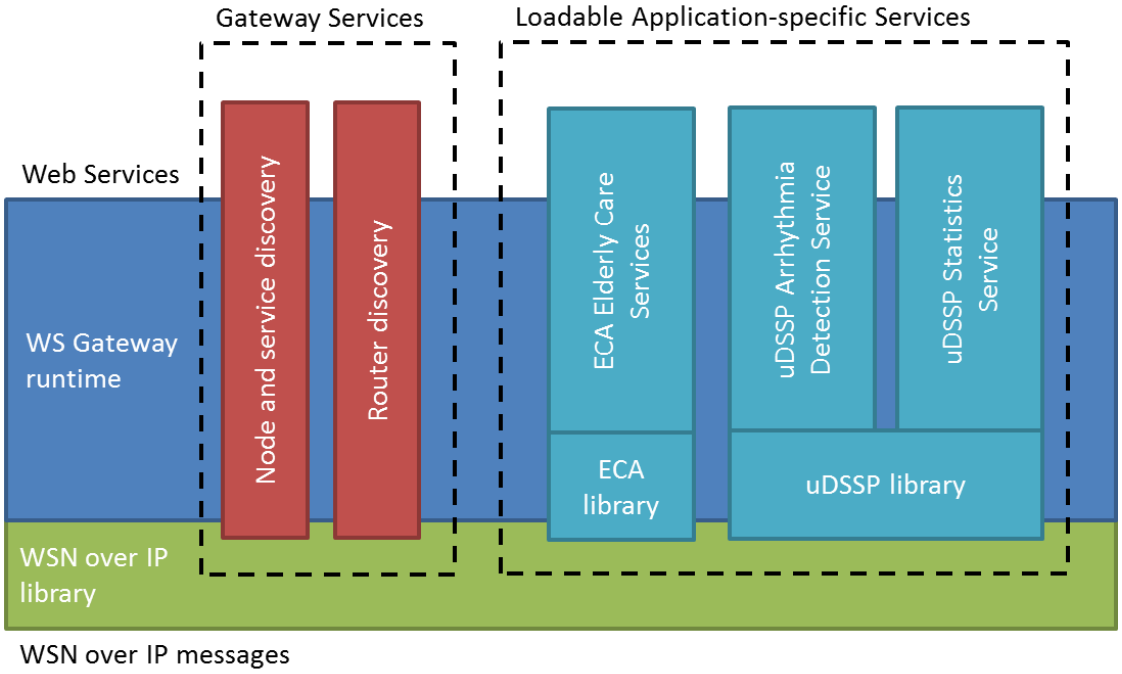


Figure 19: Architecture of the WASP web-services gateway

The "gateway services" expose web services to get information about the IP routers as well as the sensor nodes and their installed services. A web-service client application can subscribe to be updated when the available routers, GPRS forwarders, sensor nodes, and/or services on sensor nodes change. Calls to the "gateway services" do not result in messages in the sensor

network. For example, the WASP gateway internally keeps track of the connected sensor nodes by monitoring the reception of nanoSLP advertisement messages that are periodically sent from nodes to the gateway but only the availability of nodes (and their installed services) will be provided to gateway clients. If sensor nodes only rarely send nanoSLP advertisements, e.g. to reduce wireless traffic and power consumption, a client application built on top of the (continuously running) WS Gateway has the advantage that it gets the full picture of the available nodes immediately when it is started; it does not have to keep track itself of the sensor nodes over time.

The “loadable application-specific services” translate between web service and the sensor network messages and can be loaded without having to recompile the WS Gateway. At startup, the WS gateway checks all the assemblies in its working directory for classes that implement a specific interface and loads them. This interface is independent of the programming models used and, in principle, allows for the integration of various programming models in the WS Gateway. As described in section 3.4.2, code generators for two programming models, creating ECA and uDSSP services, are supported within WASP. If node memory allows, ECA and uDSSP services can run in parallel (see D5.5 for more details). Services implemented using one of these programming models can make use of additional functionality in the WS Gateway that is tailored to them in their programming model libraries. For example, the ECA library takes care of uploading services to the sensor nodes and processing the application-independent parts of the messages. Figure 19 shows the WS Gateway setup used in the elderly care test bed with one ECA web service, including all ECA node services used in one of the test bed applications (e.g., to compute the activity index and to send temperature alerts in the Elderly Care testbed), and two uDSSP services to perform ECG arrhythmia detection and collect node statistics.

The WASP gateway supports sharing of subscriptions in case multiple clients subscribe with the same parameters to a service. Without sharing, each subscription would result in the same data being transmitted multiple times in the WASP network and, consequently, waist precious battery power. By sharing multiple client subscriptions with the same parameters to a service, only one subscription is passed to the sensor network and data is transmitted only once through the sensor network and forwarded to the clients by the WS Gateway. Evidently, alignment between client applications is highly recommended to exploit subscription sharing and avoid redundant traffic within the WASP network.

In order to have the fewest requirements on web service standards that have to be supported by the client and to guarantee interoperability among different platforms, each client has to host a callback web service to receive data in response to a subscription. When data arrives, the WS Gateway will invoke this web service. Therefore, the web service platform just has to support SOAP over HTTP. We have used this mechanism both for .NET and Java clients. The data format of the messages depends on the service and is WASP-specific. In our implementation, we use an XML-string that can be easily parsed on most platforms. More details about the data format and the interface on the callback service can be found in D7.4.

Figure 19 depicts that “WSN over IP library” is used for communication of the WS Gateway with the WASP network. This approach enables easy access to multi-sink WASP networks via an IP overlay network where each sink can be connected to a PC (with IP connection and running a software router) or a WLAN router (introduced in section 3.1.1.2) or even a GPRS forwarder (introduced in section 3.1.1.3). Figure 20 illustrates the IP overlay network which is further described in D5.5. This architecture has several benefits. First, from the perspective of the WS Gateway or the applications, all three different router types are treated in the same way, which allows abstracting from the actual communication mechanism. Second, in the same way, it abstracts the number of routers that are used. Therefore, several routers can be used without requiring any changes in the application or WS Gateway. For example, with a routing protocol in the sensor network that supports multiple sinks (like, for example, Gradient Routing), this approach makes it possible to use several WiFi routers for a single sensor network. The nodes will always transmit their data to the closest sink, move traffic out of the

resource-limited sensor network quickly, and make use of the higher bandwidth and smaller loss rate of the WiFi network. Finally, by making use of IP communication, the WS Gateway does not have to be placed on the site of the sensor network but can be operated in a data center,

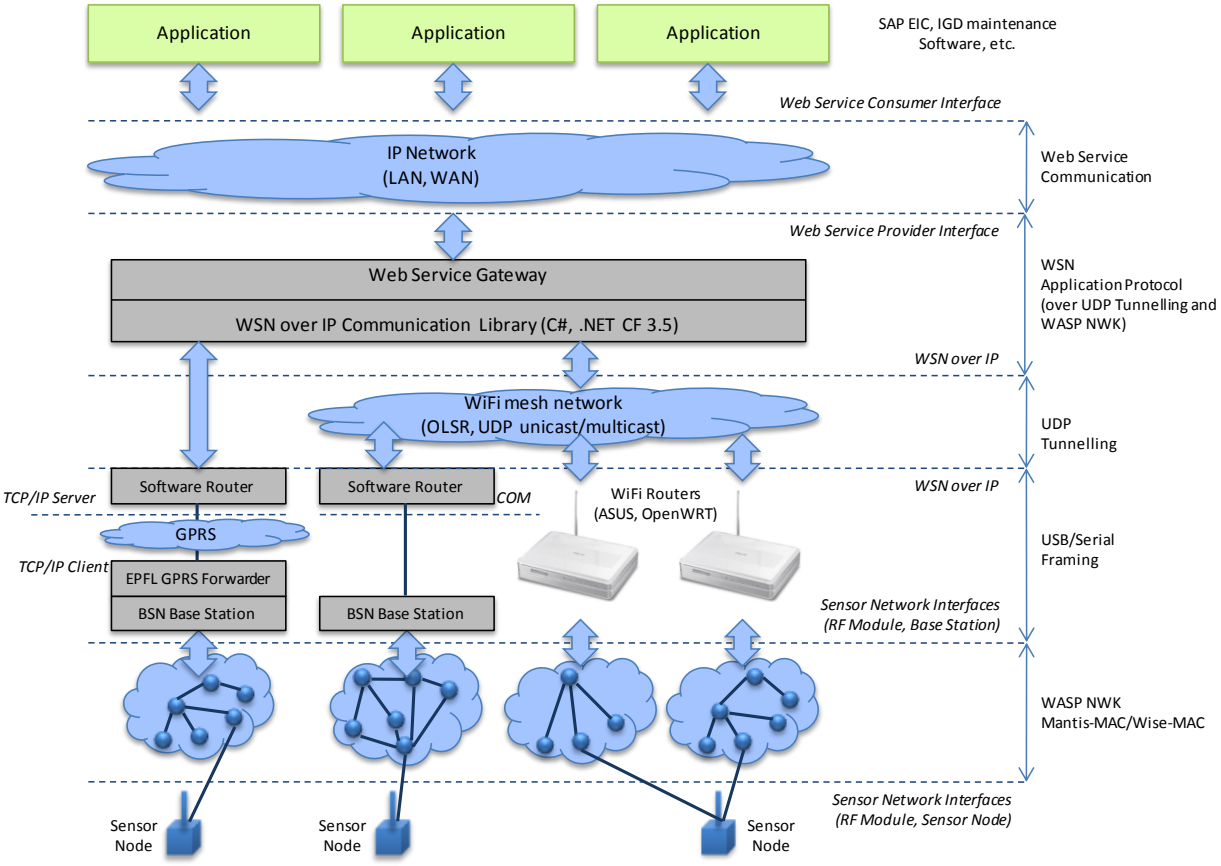


Figure 20 General setup of the IP overlay network in WASP

The availability of both hardware and software routers proved very practical throughout the WASP project. The software routers enabled partners to create the WSN over IP communication on their development PC and, thereby, connect the WASP Gateway and sensor network without having the hardware routers in the lab. Furthermore, the software router has a graphical user interface that allows the user to select several configurations (e.g. serial port and/or type of sink nodes e.g. BSN or TelosB) and to monitor the actual data passing through the SW router which proved essential during system debug. The hardware routers proved flexible to use in the various herd control integration tests as the core ASUS router with onboard XBee module enabled early system tests, based on a (temporary) simple network API and a (single hop) 802.15.4 sensor network, but could be relatively easy upgraded with an external BSN node to interface with the (actual) herd control network stack. Moreover, use of multiple hardware routers to increase the coverage of the (hybrid) WLAN and WPAN network connected to a single WASP Gateway.

3.4.2 **Service creation cycle**

Within WASP, two programming models have been developed to generate application software on the sensor nodes. Both of them provide similar abstractions such as services, subscriptions, and functions that can be called remotely. The programming models are intended for different purposes and, as mentioned above and further detailed in D5.5, can be used in parallel. The “Event-Condition-Action” (ECA) model with its virtual machine is best

suited when the service implementation is expected to change often (e.g., frequent updates to an application). The “micro Decentralized Software Services Protocol” (uDSSP), which uses native code to implement services, shows its strengths when writing computationally-intensive services. The uDSSP protocol is an adaptation of the DSSP protocol, which is used by Microsoft Robotics Developer Studio, for resource-limited devices like sensor nodes.

In this section, we describe how services are created, including their functionality both on the sensor nodes, in the WS Gateway, and in the backend application. The general steps are independent of the programming model used and shown in Figure 21.

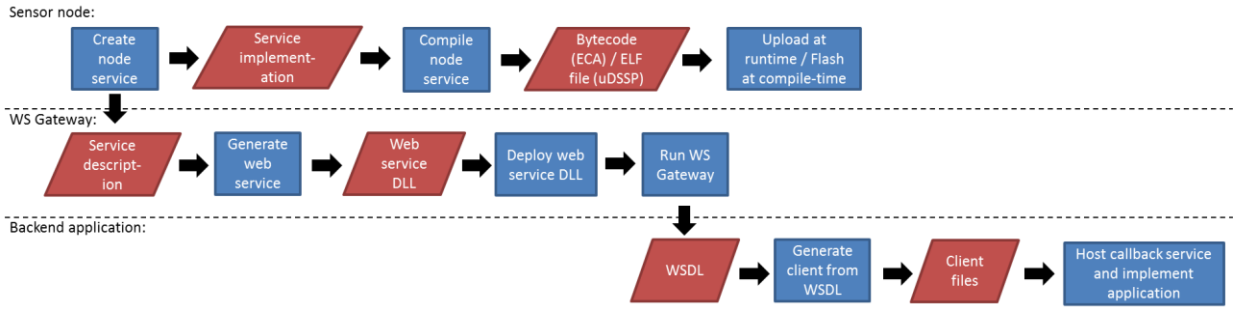


Figure 21: WASP service creation process

First, the developer creates the interface of a node service in the ECA or uDSSP programming model. This interface then has to be filled with the actual implementation of the service functionality. After the developer has finished the implementation, the node service can be compiled. For ECA services, the result of this step is a byte code file that can be wirelessly uploaded over radio at runtime. For uDSSP, in contrast, the result of the compilation is an ELF file that can be flashed to the sensor node. This process is explained in more detail in D7.4 and illustrated by the ECA and uDSSP service examples in the next sections.

For the WS Gateway, the developer generates a corresponding web service. In this step, only the node service interface is of relevance, i.e., the subscriptions, exposed functions, and data formats. Therefore, the actual node service implementation can still be modified later on without recreating the corresponding web service as long as the interface of the node service does not change. A web service code generator, that is specific to the programming model, reads the service description and creates a C# implementation file. The developer can take this file and, as described in D7.4, create a dynamic-link library (DLL) that includes the web service. At start-up, the WS Gateway dynamically loads all web service DLLs in its working directory. Therefore, the developer just places the DLL, together with the compiled ECA node services for upload (only for the ECA programming model, with uDSSP the nodes are already flashed with the service), in the working directory of the WS Gateway and, subsequently, can start the WS Gateway application.

Finally, to enable the development of back-end (client) applications, a Web Services Description Language (WSDL) file needs to be generated. The web services of the WS Gateway are configured to provide the WSDL at runtime. Therefore, any tool to generate the client code can just access the URL of the web service and get the WSDL from there. Besides generating code from the WSDL, the client application has to host its own web service with a pre-defined format for receiving the notifications with sensor network data. The format of this service is shown in more detail in D7.4.

The following subsections describe this process more concretely for some of the ECA and uDSSP services that have been developed and used within WASP.

3.4.2.1 **ECA service examples**

The ECA programming model is developed with the intention to make wireless sensor networks easier accessible for (embedded) application developers also denoted as WSN system integrators in Chapter 1. The programming model provides an abstraction from the details of the hardware, the operating system and the network protocols. The selection of the event-based programming model is described in D5.2 and its design and development is described in D5.3, D5.4, D5.5 and D5.6. The evaluation of the programming model is described in D5.7. The usage of the ECA programming model is provided in D7.4, while the programming model is used extensively in the different test beds of WP6 and several demonstrators.

The following functionality is provided to the application developer (from D5.7):

- A domain specific programming language, where the processing of sensor data and derived information can be specified. The language provides a combination of service oriented, event-condition-action and publish/subscribe models. The service oriented model allows the definition of services with state and actions that operate on that state due to remote invocations. The event-condition-action model provides a way to update the service state locally, where typically the event is a time trigger, the condition is a property of the sensor readings and the action is the transmission of a message. The publish/subscribe model provides a way to connect the event-condition-action part of one service (the publisher) with the service oriented part of another service (the subscriber). The subscriber specifies the period of the time triggers, parameters of the conditions and which actions should be invoked, while the publisher executes the ECA rules according to the requirements of the subscriber.
- A compiler for the programming language, which translates an application into configuration instructions. The application can consist of multiple services which are intended for different types of nodes, and instructions on how the nodes should interact according to the publish/subscribe model. As such, the compiler allows the developer to specify the behaviour of a complete sensor network with heterogeneous sensor nodes.
- A deployment procedure, where the configuration instructions are used to install the ECA services on the sensor hardware. Through the use of content-based addresses, each part of the application can be installed specifically on those nodes that fulfil certain properties. During the early stages of application development, the configuration instructions can be uploaded over the wireless network, which enables a fast iteration cycle for the developer. During the later stages, the configuration instructions can be included in the firmware images of the nodes, such that the upload procedure is less of a burden on the network, while the ECA services can still be updated. It is also possible to create specific nodes that perform the service upload locally within the network, such that it is feasible to run large scale networks.
- A runtime environment on the sensor nodes, which can process the configuration instructions and execute the services. The execution of a service involves the periodic execution of the ECA part when there is a subscription and the execution of an action when there is an incoming remote invocation.
- Integration with the internet, where the WS gateway provides access to sensor nodes. The output of the compiler can be used to make the services available as web services, where notifications are used to report sensor data to a backend system. The WS gateway also provides access to the actions of services, such that an application can inspect and adjust a service.

The configuration instructions and runtime environment make use of a byte code instruction set and the related interpreter. Although the interpreter introduces execution overhead, it provides abstraction, platform independence and smaller memory footprints. When the

execution overhead would be too large, the developer can provide the same functionality using machine code by extending the list of system calls, as indicated in D7.4.

One of the main reasons to provide ECA service upload functionality instead of including the services in the firmware images is the intended lifetime of the sensor nodes. For the herd control scenario, the intended lifetime is at least one year on a single battery and preferably equal to the lifetime of the animal. During such a time span, it is likely that new functionality and new hardware is added, for example due to new animals and new analysis algorithms. In the herd control test bed, replacing the firmware of a sensor node is a time consuming and technical process, which easily takes 15 minutes per node. For a medium or large farm, it is not economical to perform such an operation on a regular basis.

The ECA programming language provides the following functionality:

- The event-condition-action model allows advanced sensor processing algorithms, typically implemented by filter stages. The mode and step detection algorithms are examples of such services for the HC test bed, while an R-peak detection algorithm is available for the ECG sensor of the EC test bed.
- The periodic behaviour of an ECA service can be adjusted by the service itself, which enables the service to disable itself for a specific time. The mode detection service performs its operation less often depending on the activity of the animal.
- One service can activate another service, either on the same or on another node. The mode detection service enables the step detection when the animal is moving.
- The periodic behaviour of nodes within the network is scheduled such that traffic bursts can be avoided and network traffic is more predictable. The statistics monitor application collects network information where nodes report their status equally distributed over a long period.
- Services can access and modify network status information. For the DV-distance protocol, the location of the node can be configured. When the routing protocol indicates that no path to a sink is available, the service can prevent the transmission of packets and act accordingly. A service can add a time stamp to a packet based on the FTSP time synchronization protocol.
- Services can access operating system status information. The maintenance service can report details about the available memory and its fragmentation.
- Services can modify a number of radio characteristics. The QoS service can adjust the transmission power, the radio channel and the periodic schedule of the MAC protocol.
- A developer can easily extend the runtime environment with new functionality. The house vibe sensor and complex activity classification algorithms are made available using this method.
- A back-end application can perform remote procedure calls on all nodes. The RSSI threshold of the gradient routing protocol is adjusted using this feature.

Although each of these functionalities is useful in itself, an application developer can combine these functionalities freely, thereby creating more advanced applications. For example, an R-peak detection algorithm can start a service to stream raw ECG sensor readings on a dedicated emergency radio channel, such that the additional radio traffic does not interfere with the performance of the rest of the system.

During the preparation of the HC test bed, a setup with 127 nodes was used to test the scalability. The ECA service upload performed adequately and the system ran stable. When 90 nodes were booted all at once, the WS gateway was able to automatically install the ECA services on 89 nodes in the first upload attempt, while the 90th node was completed in the second attempt.

Within the two test beds, several ECA services are deployed, as described in D5.7. A number of services are shared between the two test beds, while others are currently only used in a specific test bed. However, with small modifications and correct calibrations, the mode

detection algorithm of the herd control test bed could be applied to the elderly care test bed. Similarly, the temperature alarm service of the elderly care test bed can be used within the herd control test bed. Except for the routing protocols (which can be installed side by side), the sensor nodes for both test beds are using the same firmware images. The hardware can therefore be shared between application domains which is important for cost reductions and off-the-shelf components.

The following listing shows the ECA statistics service, which is used in both test beds to report information about the network traffic:

```
service Statistics($Handler)
   define
      seqnr :=0
      stats[9] := []
   on event ReportStatistics when True do
      i:=0;
      while (i<8) do
        stats[i] := RoutingStatistics(i);
        i:=i+1
      od;
      stats[8] := GetMIBValue("WM_MIB_MON_PARENT_ADDRESS_1");
      seqnr:=seqnr+1;
      SendToSubscribers($Handler, NodeID(), seqnr, *stats)

   action SetRSSIThreshold(value) do
        SetMIBValue("WM_MIB_RSSI_THRESHOLD", value)

   action SetTransmitPower(value) do
      if (value>0) then
         SetPower(value)
      fi
.
```

After defining state information, one event generator is used to periodically collect the statistics information and report it to the subscriber. In addition, the service provides two actions to adjust properties of the routing protocol and the radio transmission range, which will affect the statistics indirectly.

On its own, the service does not transmit any messages. When another node or the WS gateway subscribes to this service, it will start reporting statistics about the used routing protocol. The subscriber determines the reporting period and processes the messages according to the arguments provided to the SendToSubscribers() statement.

As indicated before, the messages generated by the different nodes within the network will be distributed equally over the reporting period. By adding one statement (self.period:=0), it is possible to turn the periodic reporting service into a one-time reporting service. The nodes will report the requested information upon request, where the traffic is again distributed over the specified interval.

The ECA programming model and the runtime environment on the nodes has been used as a framework to integrate different components of the system. The installed services are reported to the WS gateway by using nanoSLP. The ECA services are made accessible to back-end applications by generating web services directly from the ECA service definitions. Parts of the operating system and the network stacks are made accessible through the ECA runtime in order to collect statistics and perform analysis. The ECA library has been used in multiple firmware images for different demonstrators.

The ECA programming model is also used within other European and national projects to provide an abstraction of wireless sensor networks.

Although the ECA runtime environment provides a very useful, there is still some room for improvements:

- The service installation is currently broadcast based, where the WS gateway uploads the services. To reduce network traffic, it would be useful when a new node can download the relevant services from neighbouring nodes, such that single hop unicast transmission can be used. It should not be too complex to add this functionality.
- In previous deliverables, mappings to IPv6 (6lowPAN) and CCBR are proposed. Due to stability problems of the radio driver of FreeRTOS on BSN2 nodes, the 6lowPAN implementation was not further pursued. The CCBR protocol is available on TinyOS, which requires considerable effort to port the ECA runtime environment and implement the mapping.
- The ECA runtime environment is currently well tested on a 16-bit MSP430, where it operates on 16-bit values. The same environment also runs on 32-bit processors, where it operates on 32-bit values. To verify the portability, the runtime environment should also be tested on different micro-controllers and different sensor platforms.
- The ECA compiler currently generates configuration messages based on a byte code instruction set. With an extension, the compiler could also generate C code, such that the step to using optimized machine code is performed automatically.

3.4.2.2 **uDSSP service ECG example**

The uDSSP programming model has several goals. It tries to relieve the developer of standard tasks and low-level details such as communication. Therefore, the developer can focus better on the actual functionality of the service. Nevertheless, the service still has access to the full functionality of the sensor node and provides the performance of native code. Finally, uDSSP tries to reduce coupling between different parts of an application and, this way, increase reuse of code.

In uDSSP, an application not only consists of the program of a single node but of distributed services in the whole network, including parts running on a PC. A uDSSP service is of similar granularity as a web service, but in contrast to that, built around its state. Communication between services is usually done by subscribing to this state and being notified when it changes. The service just modifies some local state variables and the runtime system takes care of notifying all subscribers. The parts of the application running on a PC can either directly be built upon uDSSP services themselves or be connected via the WS Gateway. The latter approach is the one we use for the WASP test beds since it offers the advantage of interoperability between different platforms and programming languages. uDSSP is described in more detail in D5.5.

In the WASP Elderly Care Test bed, we used uDSSP to implement the arrhythmia detection service and a node statistics service. To create such as service, the developer writes the abstract node service definition in an XML file. As an example, Figure 22 shows the service definition of the arrhythmia detection service. For a detailed description of this service and its state, please see D5.7.

```
<?xml version="1.0" encoding="utf-8" ?>
<service>

        <id>FC34627</id>
        <name>ArrhythmiaService</name>

        <state>
                <!-- TRUE if an arrhythmia is detected -->
                <item name="alarmMode" type="Bool" />

                <!--
                        Type of arrhythmia detected:
                        0 No arrhythmia
                        1 Ventricular Tachycardia (VT)
                        2 Supra Ventricular Tachycardia (SVT)
```

```
                        3 Undefined VT/SVT
                        4 Atrial fibrillation
                        5 Undefined, more analysis needed at upper levels
                -->
                <item name="arrhythmiaType" type="UInt8" />

                <!-- Value of last calculated heart rate -->
                <item name="heartRate" type="UInt8" />

                <!-- N-th chunk of the full ECG signal -->
                <item name="chunkIndex" type="UInt16" />

                <!-- Array of samples in n-th chunk  -->
                <item name="samples" type="UInt16[30]" />

                <!--
                        TRUE if 90s of signal have been stored and ready to be transmitted
                -->
                <item name="transmit" type="Bool" />

                <!-- TRUE to enable verbose debug -->
                <item name="debug" type="Bool" />
        </state>

        <functions />

        <partners />
</service>
```

Figure 22: uDSSP Arrhythmia Service definition

The developer calls a code generator (the uDsspTool) to generate the service-specific parts of the runtime system (e.g., to parse and build messages) and a template with functions to be implemented by the developer. The developer implements this service and creates a very basic application that just sets up the networking stack, initializes the uDSSP runtime system, and registers the service with the uDSSP runtime system. After building the code image, it can be deployed on the sensor nodes.

The same code generator can also be used to create the code of the corresponding web service. As described in section 3.4.2, the developer creates the web service assembly and generates the client code from its WSDL.

From the WS Gateway, the data of notifications is passed to the client application as an XML string. For example, for the arrhythmia detection service, the format is shown in Figure 23.

```
<vector>
  <com.sap.cmce.wasp.eic.datamodel.ArrhythmiaService>
    <theGatewayId>mobile-sisio</theGatewayId>
    <theGatewayTimestamp>1283776412093750</theGatewayTimestamp>
    <theServiceName>ArrhythmiaService</theServiceName>
    <source>2</source>
    <serviceType>264455719</serviceType>
    <serviceInstance>0</serviceInstance>
    <bitmask>
      <HaveAlarmMode>false</HaveAlarmMode>
      <HaveArrhythmiaType>false</HaveArrhythmiaType>
      <HaveHeartRate>false</HaveHeartRate>
      <HaveChunkIndex>true</HaveChunkIndex>
      <HaveSamples>true</HaveSamples>
      <HaveTransmit>false</HaveTransmit>
      <HaveDebug>false</HaveDebug>
    </bitmask>
    <state>
      <alarmMode>false</alarmMode>
      <arrhythmiaType>0</arrhythmiaType>
      <heartRate>0</heartRate>
      <chunkIndex>33</chunkIndex>
      <samples>
        <unsignedShort>33</unsignedShort>
        <unsignedShort>1503</unsignedShort>
        <unsignedShort>1531</unsignedShort>
```

```
        <unsignedShort>1531</unsignedShort>
        <unsignedShort>1517</unsignedShort>
        <unsignedShort>1519</unsignedShort>
        <unsignedShort>1523</unsignedShort>
        <unsignedShort>1523</unsignedShort>
        <unsignedShort>1555</unsignedShort>
        <unsignedShort>1671</unsignedShort>
        <unsignedShort>1559</unsignedShort>
        <unsignedShort>1559</unsignedShort>
        <unsignedShort>1523</unsignedShort>
        <unsignedShort>1559</unsignedShort>
        <unsignedShort>1547</unsignedShort>
        <unsignedShort>1551</unsignedShort>
        <unsignedShort>1549</unsignedShort>
        <unsignedShort>1549</unsignedShort>
        <unsignedShort>1517</unsignedShort>
        <unsignedShort>1563</unsignedShort>
        <unsignedShort>1543</unsignedShort>
        <unsignedShort>1517</unsignedShort>
        <unsignedShort>1579</unsignedShort>
        <unsignedShort>1533</unsignedShort>
        <unsignedShort>1553</unsignedShort>
        <unsignedShort>1555</unsignedShort>
        <unsignedShort>1531</unsignedShort>
        <unsignedShort>1559</unsignedShort>
        <unsignedShort>1555</unsignedShort>
        <unsignedShort>1499</unsignedShort>
      </samples>
      <transmit>false</transmit>
      <debug>false</debug>
    </state>
  </com.sap.cmce.wasp.eic.datamodel.ArrhythmiaService>
</vector>
```

Figure 23: Example notification string from the arrhythmia web service

3.4.3 Example client applications use within WASP

This section describes examples of client applications that interface with the web services provided by the WASP WS Gateway.

3.4.3.1 Useful client applications to ease WASP deployment

When preparing for the first fully integrated herd control test bed demonstrator, shown at the review in October 2009, it became apparent that there was a need for additional tools to monitor the nodes present, their service, the topology of the network, and their performance. Based on these needs, we have developed a set of node and web services together with a number of client applications that help in the deployment, performance monitoring, and stress testing of WASP networks.

The first application is the Node Monitor (see Figure 24). It subscribes to “gateway services” of the WASP gateway and displays the information received from the service discovery messages. It shows the nodes that are available, the services (both ECA and uDSSP) that have been installed, and the ECA service uploads that have not been received completely yet. Furthermore, it shows the point of time when nodes stop responding. This application helps to find out if all nodes are functional, have a connection to the WASP gateway, and have the expected services installed. Furthermore, using the time information, the user can determine if there is just a temporary connectivity issue (e.g., a cow lying on the node) or if the node has stopped sending service discovery advertisements for longer periods of time.

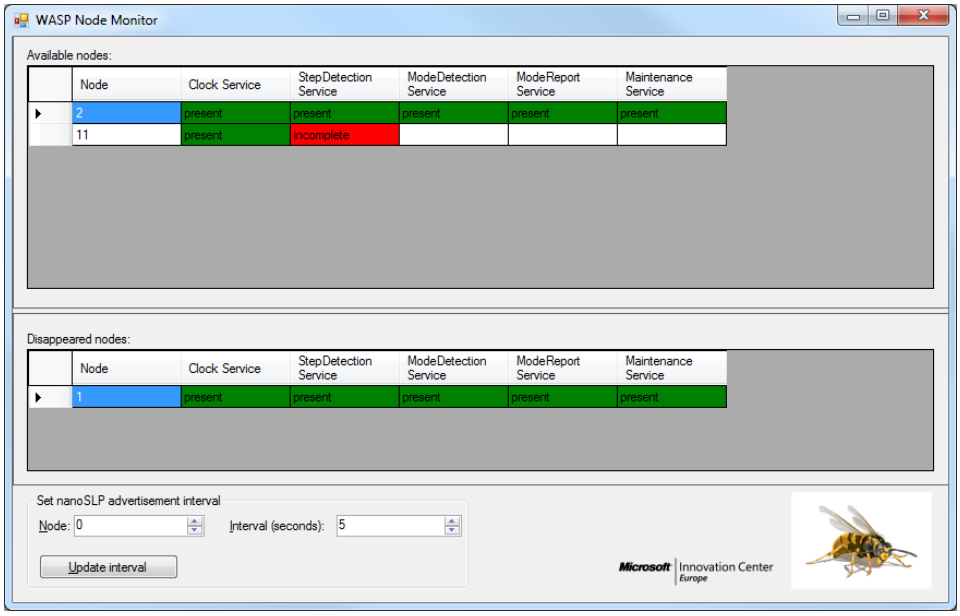


Figure 24: Node Monitor application

The Topology Monitor depicted in Figure 25 subscribes to the statistics service that provides information about the performance of the node. The only piece of data that is used by the Topology Monitor is the information about the current parent node of the routing protocol. It combines the information received from all nodes and displays it in a tree. Therefore, the user can see if multi-hop routing is actually used or if all nodes are directly connected to the sink node (also referred to as base station). The parent information is accurate for Repairing Routing as it uses link-layer unicast between node pairs in a hierarchical network tree as mentioned in section 3.3.1.2. In case of Gradient Routing, however, the parent information is not accurate and only indicative for the hop-distance to the sink. This originates from the use of link-layer broadcasts in Gradient Routing which implies that nodes do not use an explicit parent but simply forward their data to all nodes in range with a smaller hop-distance to the sink that, subsequently, forward the data further. Nevertheless, for Gradient the parent information is still useful because it indicates correctly the hop-distance of each node to the sink.

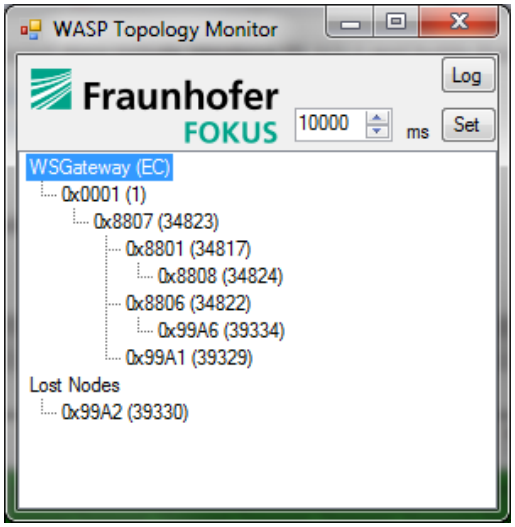


Figure 25: Topology Monitor application

The Statistics Monitor depicted in Figure 26 displays and logs information about the network performance, in particular from the routing protocol on the sensor nodes and from the WS Gateway. It helps to monitor packet loss and, hence, efficiency of the routing protocol. On the one hand, the Statistics Monitor uses information coming from the statistics service that runs on each sensor node (the packets sent and received by the node plus some additional information from the routing protocol). On the other hand, it combines this information with statistical data that is collected by the WS Gateway. This way, for example, it can compute packet loss. Furthermore, the Statistics Monitor writes all data to a log file to help finding the cause of problems and to get a more detailed view of the performance over time.

To get the data from the nodes, the Statistics Monitor subscribes to different types of web services depending on which of them are available. It supports a uDSSP implementation of the statistics service, an ECA service part of the Herd Control and Elderly Care test beds as well as an ECA service that provides additional quality of service data such as latency. Figure 26 shows a screenshot of the application while it was subscribing to the latter service.

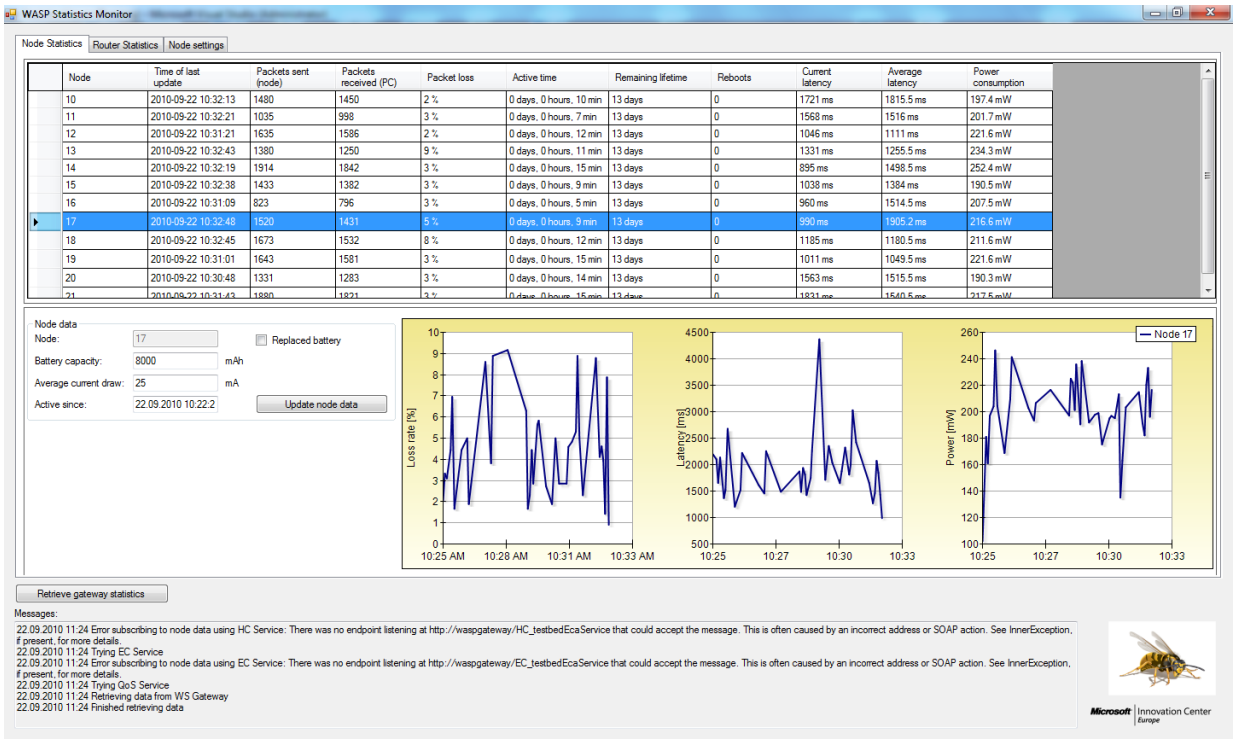


Figure 26 Node (and network) statistics view in the Statistics Monitor application. Here, per node, real-time information on (1) time of last packet received in the WASP gate, (2) number of packets sent to the WASP gateway, (3) number of packets received by the WASP gateway, (4) the packet loss, (5) node active time, (6) estimated remaining lifetime, (7) number of node reboots, (8) current delivery latency (needs FTSP activated), (9) average delivery latency, and (10) power consumption.

Figure 27 shows the “Node Settings” tab of the Statistics Monitor. With the settings shown there, the Statistics Monitor can be employed for stress testing of the sensor network. For example, the rate of service advertisements coming from each sensor node can be set here. This allows creating a constant rate of packets throughout the network that is independent of the application-specific services on the nodes. Other settings are the transmit power of each node, the RSSI threshold for Gradient and Re-Pairing Routing advertisements used to avoid unreliable links, and the period of the WiseMAC protocol.

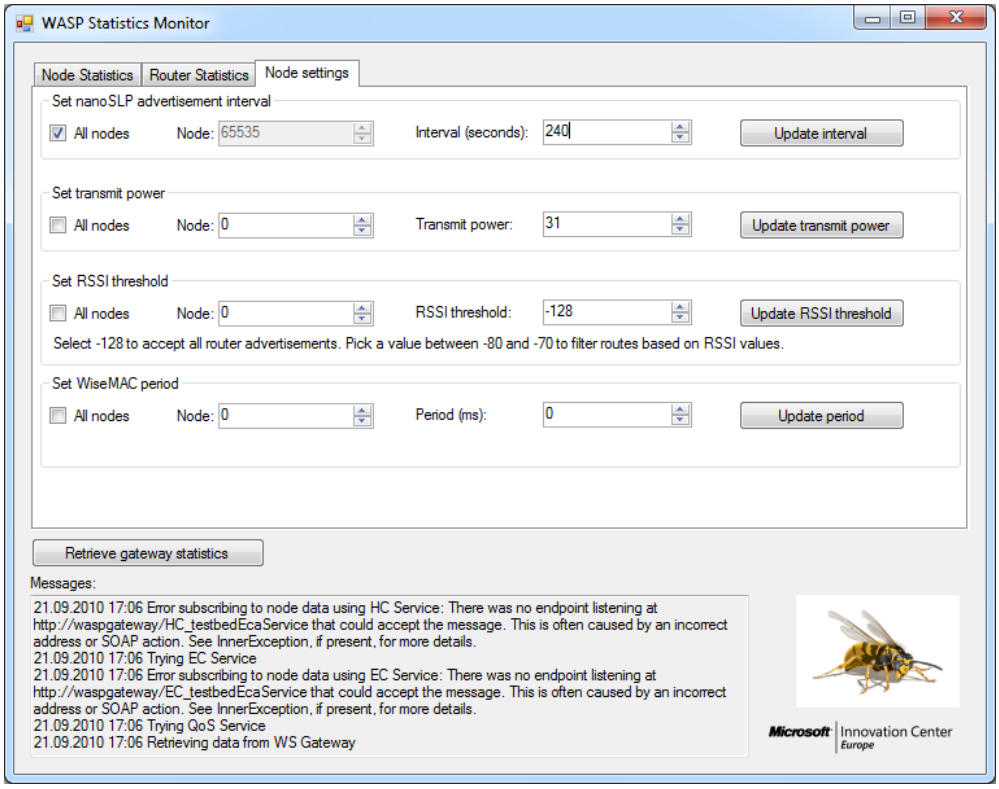


Figure 27 Node settings view in the Statistics Monitor e.g. enabling network stress tests by uploading short nanoSLP to intervals to nodes that, thereby, inject higher traffic into the network.

3.4.3.2 **Maintenance GUI**

Like the client applications in the previous section, the Maintenance GUI aims to provide users a Graphical User Interface (GUI) and 3D visualization of actual WASP test bed deployments that allows intuitive monitoring and remotely parameter setting of WASP objects.

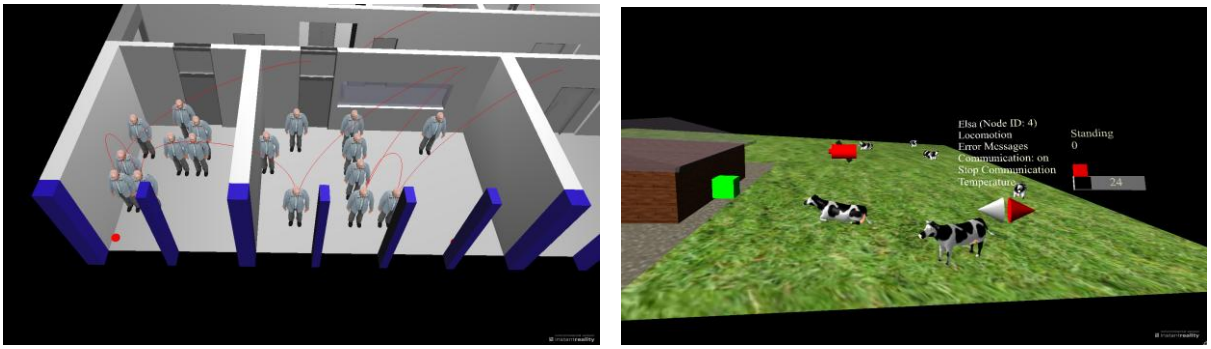


Figure 28 Illustration of the 3D viewer of the Maintenance GUI for the elderly care (left) and herd control (right) test beds.

Figure 28 shows two screenshots of the 3D virtual worlds created for the WASP test bed locations for Elderly Care (in London) and Herd control (in Lelystad). These 3D worlds are defined in VRML and a standard VRML viewer is used for display and easy navigation. Dedicated 3D models are developed to represent (non-background) WASP objects like (lying, standing, walking) cows and elderly persons and static entities like forwarder nodes, routers, and the GPRS forwarder. The actual (X, Y) position of static WASP objects like forwarder

nodes, routers, and GPRS forwarders can be set by a pop-up window accessible from the Maintenance GUI depicted in Figure 30. Dynamic WASP objects, like wearable nodes, are positioned in the 3D virtual world depending on the user-selected method which can be (1) next to the closest sink (at router or GPRS forwarder), (2) next to the closest static forwarder node, or (3) for the herd control test bed, using estimated (X, Y) coordinates derived from DV distance. The required information, i.e. parent nodes or estimated (X, Y) coordinates, are directly read-out by ECA-based node services from (the network MIB of) individual nodes and periodically reported to the Maintenance GUI. Proper 3D display of dynamic objects can be quite useful e.g. to lead a maintenance service provider to a node whose battery is about to die-out, a nurse to a patient in need, or a farmer to a cow that needs attention. Evidently, the accuracy of the 3D display relies on the underlying mechanism. Using the closest forwarder for e.g. room-level localization in the elderly care test bed strongly depends on the RF absorption of walls and ceilings, on proper positioning of forwarders nodes over rooms and corridors, and proper tuning of TRX power to limit wireless traffic within rooms and corridors. Use of DV distance for herd control also requires careful positioning of static forwarder nodes that serve as anchor nodes and the accuracy heavily depends on consistency of received RSSI values. The left screenshot in Figure 28 illustrates that a network connectivity graph can be shown which is visualized as a series of NURBS curves between a node of interest, via potential intermediate nodes, up to the receiving sink node (base station). This can be used during deployment to check connectivity throughout the test bed and to improve connectivity by adding forwarders or routers at specific (X, Y) locations. The right screenshot in Figure 28 illustrates the visualization of alerts e.g. by warning symbol for a low battery.

The Maintenance GUI can also subscribe to node activity classifier to obtain and display information about the actual activity status of an individual, e.g. whether a person (or cow) is lying, standing, or walking. This is shown in the 3D virtual world by different animations of the individual objects as illustrated in Figure 29.

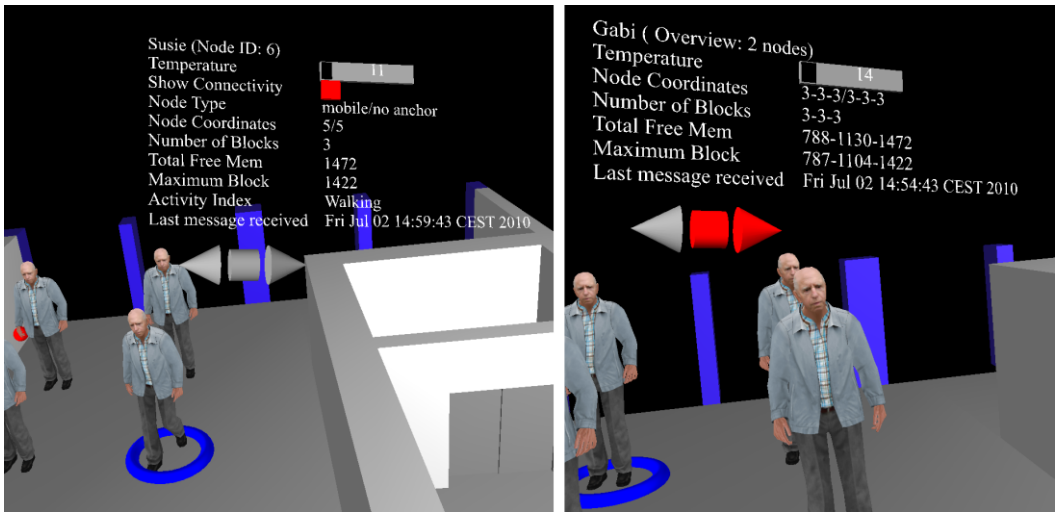


Figure 29 3D viewer pop-up menu showing service data: single node menu (left) and overview menu (right)

For all WASP objects, a menu can be shown by clicking on that object. For routers, only limited information (coming from the WS Gateway) can be shown as the attached sink (base station) nodes do not execute ECA services unlike the static and wearable nodes for which detailed information can be shown. A button menu item is available to directly interact with services on the node or services provided in the maintenance GUI, e.g. to show/hide the connectivity graph or to reboot a malicious node. To switch between different nodes of an individual, the user can select one of the arrow buttons at the bottom of the menu. An overview menu was introduced that shows service data accumulated over all nodes of an individual.

The display of information depends on the service implementation and can show an average value or more detailed information, for example in the format *minimum-average-maximum* as shown in Figure 29 for the total free memory.

Figure 30 Maintenance GUI to control 3D visualization

To control the 3D visualization, a (2D) Maintenance GUI was implemented of which a screenshot is illustrated in Figure 30. With this GUI it is possible to select a node or an individual of interest from a drop-down menu and to direct the virtual world to change viewpoint for proper display. This GUI also allows to switch 3D visualization between the various virtual worlds e.g. to show different floors of an elderly care center. Also, the localization method and node type for individual nodes can be selected as well as (X, Y) position for static nodes.

For developers, an extensive API is provided to abstract and hide the complexity of both, the underlying protocols and hardware specifics, and the visualization to prevent the developer from handling raw VRML code. This API together with an example is described in the public deliverable D7.4.

3.4.3.3 **Enterprise Integration Component**

The service-oriented Enterprise Integration Component (EIC) acts as a mediation layer between the WSN and business applications. The EIC provides the mechanisms and services for seamless and secure integration of WSNs into business applications. It meets the business applications requirements related to communication (e.g., on demand, event based sensor data acquisition, WSN discovery) and storage (e.g. support sensor data mapping from heterogeneous WSNs to a unified sensor data format).

As shown in Figure 31, the EIC is divided into three layers: WSN Interface, Data Analysis and Repository and Business Application Interface. Incoming sensor data from the WSN are sent to the WSN Interface layer, which is in charge of communicating with the WSN gateway. In the Data Analysis and Repository layer, sensor data are stored in the database and processed towards the notification broker, which forwards raw and processed sensor data to the business applications, through the Business Application Interface.

The **Business Interface** is considered as the entry point of the business application to the EIC. This layer exposes web services to business applications for discovery of available services within the EIC and acquisition services for sensor data. There are two types of acquisition services: event-based and on-demand services. Regarding, event-based acquisition service, the application subscribes to a list of alerts in the EIC as specified in WS-Notification. As soon as sensor data of interest is triggered from the sensor nodes or the EIC, it is forwarded to the subscribed application.

The **Data Analysis and Repository layer** is in charge of further data processing and a data repository for business applications. The purpose of Data Analysis is to reduce information

flooding toward applications and to provide further data processing over WSN information. A Data Repository enables business applications to process and manipulate all historical sensor data from the past, e.g., history service, statistical analysis.

The **WSN Interface** provides the communication interface between WSNs and the EIC, including data mapping. This layer is the entry point for WSN gateways to the EIC. The communication interface is implemented through a generic connector. The concept of connector enables to connect the EIC to any WSN. A generic connector can be considered as a driver for a WSN. This connector thus implements the communication protocol used by the WSN gateway.

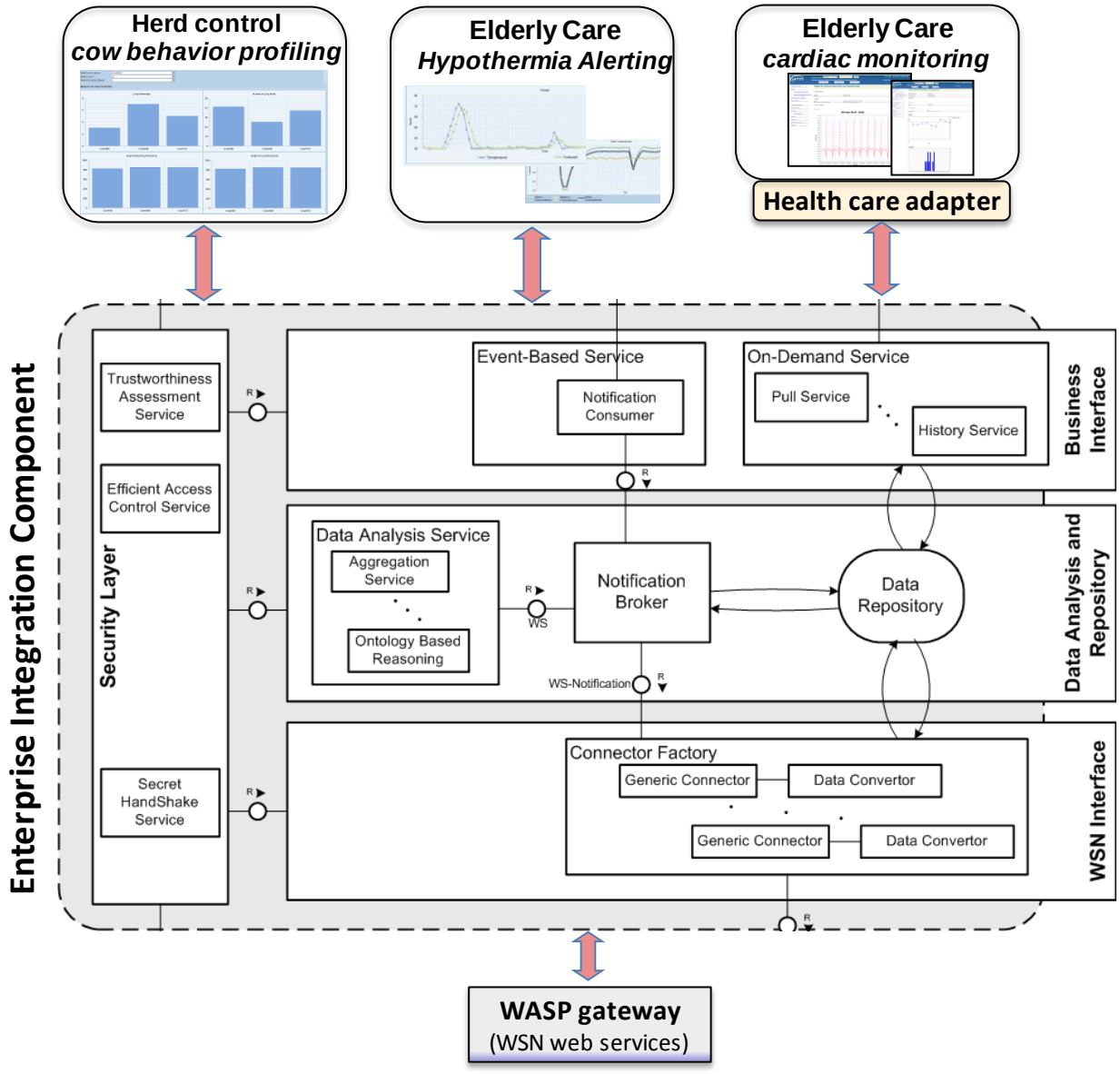


Figure 31 Enterprise Integration Component architecture and functionality together with some example enterprise applications used in the herd control and eldery care test beds further decribed in sections 4.1.1 and 4.2.1.

As indicated in Figure 31, a fourth **security layer** spans the previous three layers. Even though security is hardly seen as a functional service, this layer integrates security and trust mechanisms to provide related services to the business applications.

For secure **data access control** a light-weight node-level solution has been developed and demonstrated outside the integrated test beds [16]. The solution is based on an end-to-end symmetric encryption from sensor nodes to business applications. The key intuition is to use light-weight cryptography to provide secure data access control: if sensor data is encrypted, only the owner of the decryption key can access the data. The scheme allows the generation of hierarchical, time-bounded cryptographic keys. Sensor nodes just have to perform simple encryption operations to enforce data access control, regardless of who the listeners are and what their capabilities are. A central authority is in charge of delivering keys to applications according to the access control policy. This policy is based on sensitivity or authorization levels of sensor data and, for example, data whose disclosure does not involve high-privacy issues are mapped to low authorization levels whereas highly-sensitive data is mapped to high authorization levels. Sensor data associated to the same authorization class is encrypted with the same cryptographic material which, in line with the hierarchical organization, allows encrypting any lower class of sensor data. A so-called Access Control Module assesses the authorization level of business applications and provides them with time-bounded grants which allow them to derive the keys to decrypt sensor data from the cleared authorization classes.

A **trust assessment of sensor data** has been developed and implemented within the EIC. As described in deliverable D5.5, this assessment aims to enable business applications to distinguish between erroneous and trustworthy sensor data. For this purpose, a trust model has been defined which is based on the sensor data life cycle depicted in Figure 32.

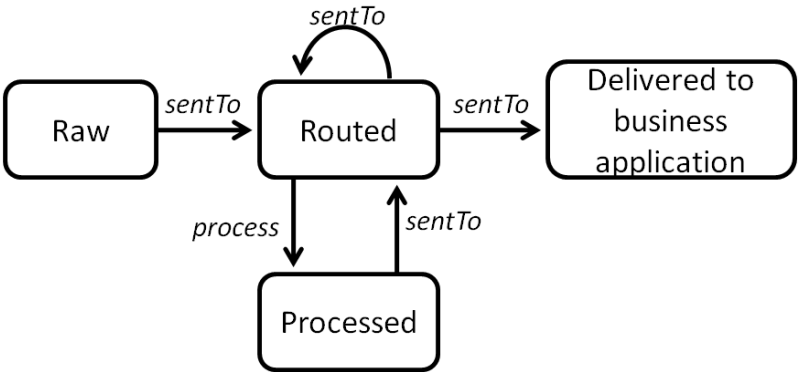


Figure 32: Sensor Data Life Cycle

In the sensor data life cycle, we identify four states for sensor data: (i) raw, (ii) routed, (iii) processed, and (iv) delivered to business application. Sensor data is raw when being sampled without being routed (i.e. sent to another node) or processed (e.g. filtered, fused or aggregated). Trustworthiness of raw sensor data is evaluated by combining the trustworthiness of several available attributes (e.g. data origin, node battery level, accuracy of the measurement). When (raw or routed) sensor data has been processed, the trustworthiness of the result depends on the processing services (e.g. fusion, aggregation) and the sensor data used as input. When sensor data has been routed from one entity to another (e.g. from its origin node, to another node; from the sink to a gateway) or delivered to business application, we consider the trust relationship between the receiver and the sender which has a clear impact on the overall trustworthiness of the sensor data. In deliverable D5.6 and D5.7, the implementation of the trustworthiness mechanism, based on subjective logic, has been described. Here and in [17], results obtained during various herd control experiments in Lelystad have been reported.

3.4.3.4 **Health Care Adaptor**

The Health Care Adaptor (HCA) is a healthcare specific mediation layer acting between external professional health care business applications and WSNs: it was introduced in the WASP architecture to support business applications in implementing a continuous patient monitoring process, integrating sensor information coming from WSNs.

The introduction of the HCA in the WASP architecture allows:

- the implementation of specific healthcare scenarios, where a **patient** could be monitored continuously subscribing to **alerts** and data notifications coming from WSNs: **raw data** and **events** coming from lower levels are then used to have further analysis to provide more accurate diagnosis and implement **specific clinical monitoring processes** (such as the arrhythmia detection and classification process used in the elderly care test bed)
- the adherence to **standards** in terms of **communication protocols** (SOAP web services over HTTP protocol), **messaging systems** (HL7 messages embedded in SOAP) and **definition of data** (use of OpenEHR as a reference model for clinical data) in order to satisfy full interoperability criteria with external applications
- delivering information only to authorized business applications in a **secure** way, providing availability, confidentiality and integrity of information

Within the WASP elderly care test bed, the HCA has been tested for the cardiac care scenario, which is further described in section 4.2.1: the TelMed Platform (HTN's telemedicine and electronic health record system) has been used as an example of external healthcare application.

Figure 33 depicts the architecture and an overview of the components of the HCA, showing the connection between services and groups of components.

The HCA can be described as a **three layers modular system**, with two layers dedicated to communication with other systems, and an internal one implementing the core services of the HCA:

- the **Health Care Interface**, exposing available services and exchanging data with external healthcare business applications
- the **Patient and Service Management layer**, providing an abstraction from WSNs on data through standardization and conversion according to a clinical reference model, and providing an implementation of specific patient monitoring services from a medical perspective
- the **EIC Interface**, providing a connector to WSNs through the EIC

Both the interfaces towards the EIC and business applications were designed as **service-oriented based layers**, exposing and making use of available services as **web services**, with exchange of *SOAP messages* according to the *WS-standard* (in particular *WS-BrokeredNotification* [18]).

Furthermore, the Health Care Interface exchanges data with external applications embedding **HL7 messages** in SOAP messages: for example for the cardiac scenario from the elderly care test bed, the HL7 Annotated Electrocardiogram [19] (*HL7 aECG)]* was selected as a reference medical record data format, since this allows submitting both the ECG waveforms and the associated measurement locations (ECG annotations) in a standardized way (it was accepted by ANSI May, 2004, being reaffirmed in May, 2009).

Further details on the design process of the HCA can be found in Deliverable D6.5 and more information on the integration process and the results for the HCA are available in Deliverable D5.7.

In summary, the integration of the HCA within the WASP architecture and the results from the elderly care test bed confirm that:

- the choice of making the HCA a dedicated integration component makes possible the implementation of continuous remote monitoring process of patients for professional health care services
- the HCA adds a valuable resource to the WASP project like the adherence to data definition (OpenEHR) and messaging (HL7) standards when communicating with external healthcare applications
- the design of the HCA in accordance to the principles of a Service-Oriented Architecture (SOA) pattern along with the use of a modular architecture where all components are loosely coupled because of the use of web services for external communication and a queue based management system for the core components can provide a solid, reliable and very good performing system
- platform and vendor independence is guaranteed by the use of standard non-proprietary technologies and standard communication protocols for all HCA components

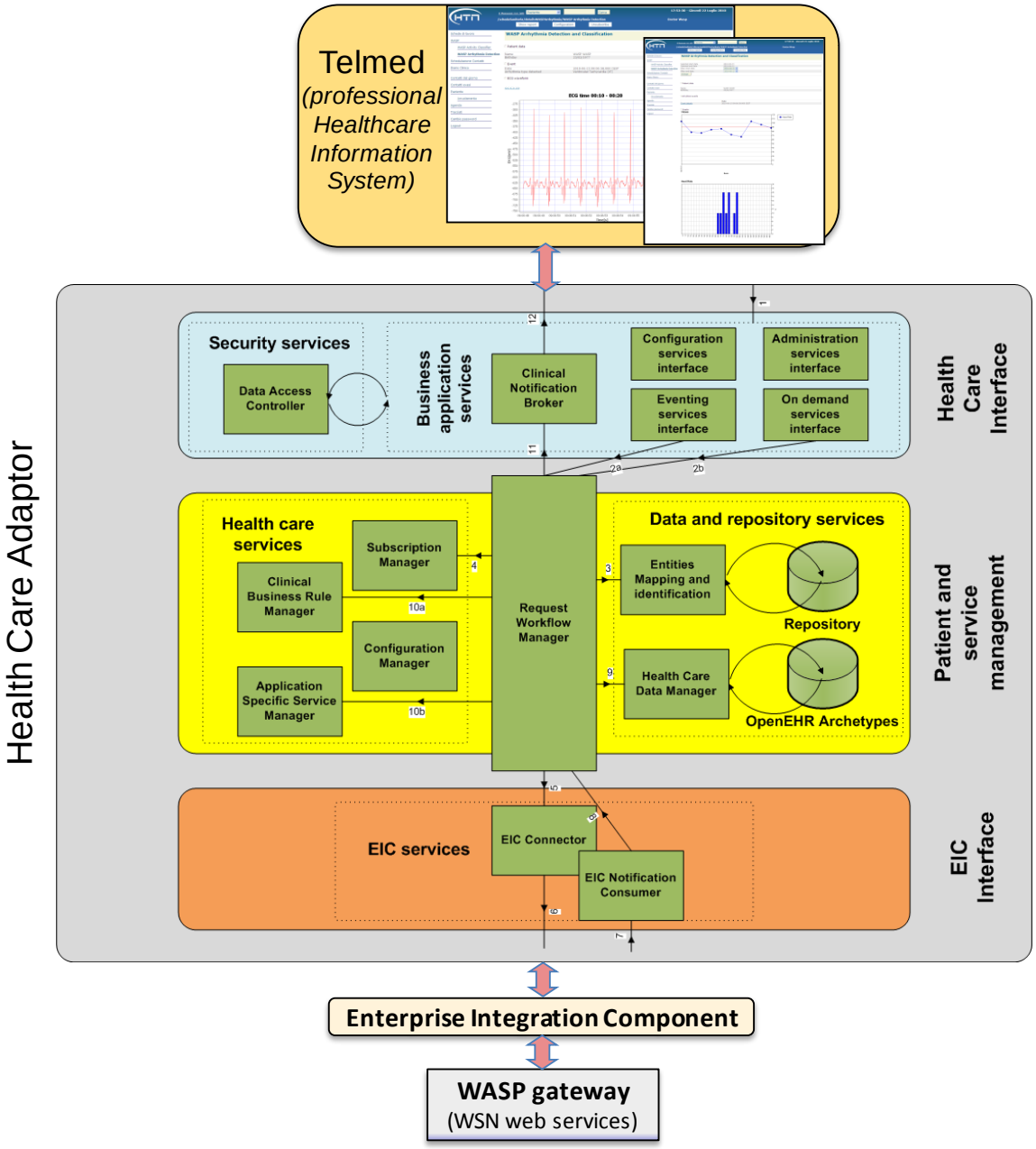


Figure 33: Health Care Adaptor architecture and components

3.4.3.5 **WASP services lessons learned**

This section summarizes some lessons learned regarding the WASP gateway and services. A more complete set of "lessons learned" and of design rules for creating efficient sensor network applications is given in D5.7.

- The WASP gateway architecture, consisting of the IP Routers, GPRS Forwarder, and WS Gateway, proved to be very flexible in enabling WSN deployment trials.
    - It physically decouples the sensor network from the gateway and its clients which, therefore, allows the back-end developer to work outside the WSN deployment area.

  - It allows to add different wireless technologies within the deployment. For example, in addition to the low-bandwidth and energy-expensive multi-hop communication on the sensor nodes, IP (WiFi) routers can be added for communication with a larger single-hop sensor network and forward messages over high-bandwidth WiFi instead.
  - It greatly simplified the development of gateway clients by adding a higher level of abstractions via web services and by putting common functionality such as node discovery into the WS Gateway.
- As the evaluation in D5.7 shows, the gateway architecture performs well and does not create a bottleneck in the system.
  - Nevertheless, we had to notice that even with modern PCs the functionality added to the critical path of the backend application has to be limited. Otherwise, this application could slow down delivery of notifications. We address this issue by separating the worker threads from the web service notifications in the backend applications and by ensuring in the WS Gateway that one slow subscriber does not slow down others. For example, in the maintenance GUI, separate threads were introduced to decouple communication with the gateway from communication with the viewer for 3D visualization. The latter communication was found to be (too) time-consuming to process all notifications from the gateway in real-time and thus, an increasing delay appeared as the gateway stores all information until it can be delivered to the backend.
  - It is interesting to note that the limiting factor is not the time spent in the WASP components but the overhead of the web service stack. If – instead of the HTTP-based web services we use for maximum interoperability between different platforms – optimized TCP channels between .NET applications were used, the web service throughput could be further increased by a factor of 10.
- A subscription-based interface as it is used by the programming models fits well the asynchronous nature of sensor network applications. By extending subscriptions via the WS Gateway to backend applications, the benefits of this interface can also be used there.

In the test beds, where we integrated the contributions of many different partners, we observed the need for comprehensive debugging facilities ranging from the nodes over the different parts of the gateway architecture to the backend application. Although several parts of the system already include debugging mechanisms (e.g., simulations with WSim for the sensor nodes, log files of the WS Gateway and the EIC) often still an expert user was needed to locate the source of a problem. Providing an efficient in-field debug solution is an area for further improvement.

4 Applications

Within WASP a Research technology prototype has been build of the advanced WSN system described in chapter 2 and using the components & tools detailed in the sections in chapter 2.5. WASP developments have been driven by three application domains. For the Herd Control and Elderly Care applications, technology prototypes have been developed in an iterative way and tested for relevant application scenarios as described in sections 4.1 and 4.2, respectively. For automotive, in-vehicle and tunnel management scenarios have been simulated to test the generality of the WASP concept as described in section 4.3.

The current integrated WASP solution demonstrated in the herd control and elderly care test beds can be well used as a research tool for animal and human behaviour scientists at ICL and WUR to enable data acquisition for periods up to several weeks to study behaviour and, in the end, come up with better models and treatments. This understanding is essential to quantify benefits and costs and, thereby, to motivate any next steps from the current WASP research technology prototypes towards industrial application prototypes and, subsequent, commercial solutions. To prepare such next steps, we've included our main lessons learned from the test bed integrations in section 4.4.

4.1 Herd control application driver

The herd control applications within WASP are based on the anticipated increase in size of dairy farms around 2015 when the current milk quota system will be abandoned within the EU and today's small-scale (50 to 100 cow) farms may no longer be cost competitive. Besides the increase in farm sizes, also more effort is needed to monitor the welfare of cows and to early warn for upcoming diseases. For all these reasons, good observation of individual cow behaviour is needed. The so-called Smart Dairy Farming (SDF) concept [20] aims to enable cost-efficient operation of large-scale (150 to 500 cow) dairy farms by automated monitoring of individual cows to identify cows that need attention in an early stage. Such cows are most expensive in food, labour, and veterinary & medicine costs and, usually, suffer from a loss in milk production and increased risk for premature carry-off for slaughter.

The implications of the Smart Dairy Farming concept are several. If individual animals and the awareness of their variation are set as a basic element and the international and interesting development in Information and Communication technology will be integrated in 'real time' management of livestock production chains then a lot of topics can benefit. These topics have all their specific interest. Some of them are mentioned shortly.

Production efficiency and energy - In daily management it is important to use the expensive production factors such as labour, food, concentrates in an efficient and sustainable way. Production is closely related to energy and the foot print. The focus on individual animals makes it possible to deal with their differences. There is a huge potential in this. Variation of 30 to 40% is still present. This becomes very important if production factors become scarce. A second part of production efficiency is that chains become more complex and that farm sizes increase. Remote monitoring makes it possible to give individual animals the attention they need. This will also be a bridge to new or changed cooperation in the production chains. New services can be developed.

Animal welfare - Animal welfare becomes more and more objective. Measuring programmes are being developed and this will be the basis for intensive communication with society. It also can be a basis for new products. These objective animal welfare programmes are based on farm visits that take place at regular moments. The first sign can be seen that there will become a need for operational checks of animal welfare, since welfare is very sensitive for the context in which animals live. The SDF concept and the tools can be helpful in developing these real time objective animal welfare characteristics.

Animal health - Discussions in the domain of animal health have different characteristics. From the perspective of individual animals and farmers there should be strong emphasis on timely detection through early warning of farm related diseases. Reduction of these health problems probably will influence longevity of animals on the farm. Cooperation with service providers and veterinarian practitioners will be beneficial for both. A lot of discussions also focus on contagious diseases. National programmes and cooperation in the animal disease chain might benefit from real time animal data. However, for this national or regional monitoring specific attention should be paid to regional modelling and location based animal data. This will lead to a more preventive and risk based monitoring task. Also in times of crisis this information will help to react quicker and more precise. The herd control scenario selected in WASP falls within this topic and, more specifically, focussed on remote monitoring of activity and locomotion of individual cows as early indicator for claw health problems.

Product quality and Food safety - Product quality is also individual animal related. So far this topic is not much in discussion but can become important when product diversification becomes more prominent. Animal specific information can become the basis for new food chains. This will have logistic consequences that should be handled by using modern ICT based support systems. The SDF concept can feed the present used quality systems with new real time elements. Then also in livestock production chains the concept of quality based tracing and tracking becomes possible.

Environmental protection and regional development - This topic has also a high importance. The connection to the SDF concept is very close related to the whereabouts of individual animal activities. Individual activities of animals are in essence the source of environmental issues. Use of food, concentrates and manuring are context and animal related. This takes place in a specific region. New concepts of farming become possible. Interaction with protected areas such as Natura 2000 regions will provide operational management. For this topic also the location awareness and the real time integration in the regional system are important.

Of course not all these aspect were used in the WASP prototype. But the relevance of observing cow activity behaviour and, more specific, the locomotion behaviour is made clear.

4.1.1 Herd control test bed

Within the WASP test bed for herd health control, the focus has been on remote monitoring of activity-related problems like claw health. Throughout various integration sessions in a dairy test farm in Lelystad (The Netherlands), of which an impression can be found in Figure 34, the components and tools described in Chapter 2.5 have been gradually introduced and tested thereby validating and refining the WASP system described in Chapter 2. The final achievement has been the successful deployment of a WASP prototype as indicated in Figure 35.

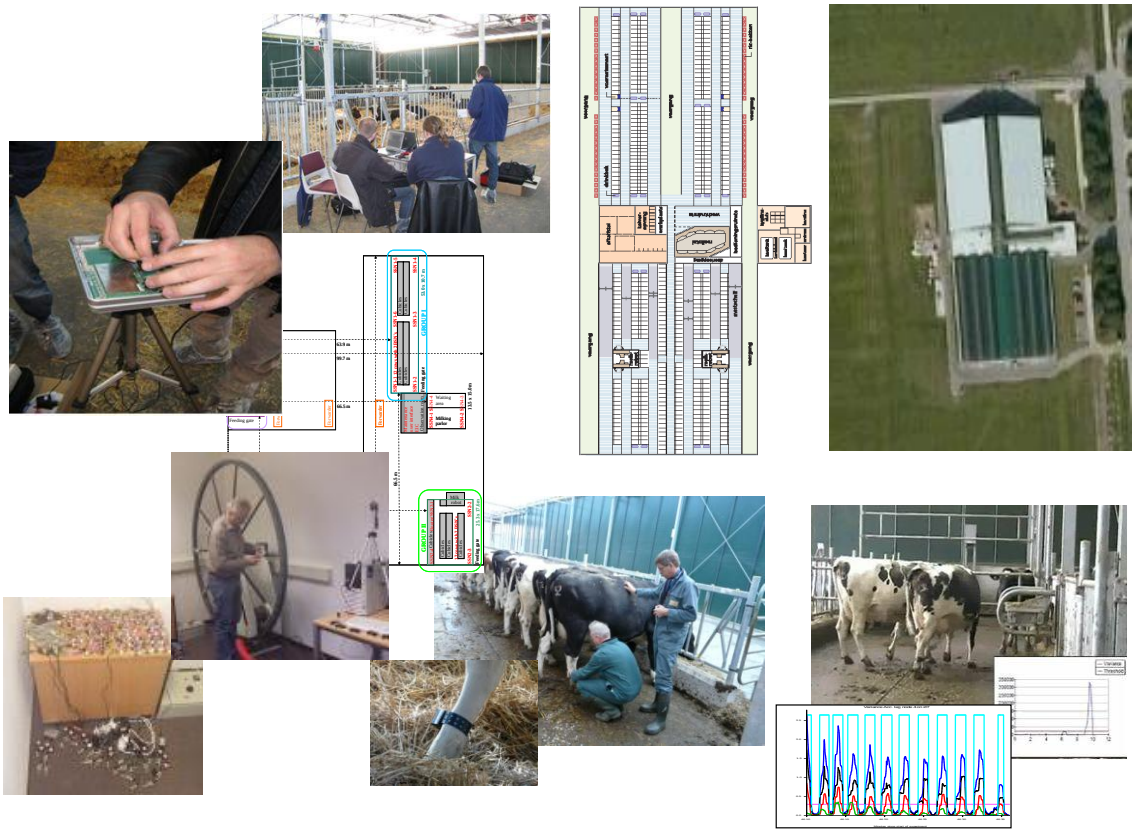


Figure 34 Impression of the herd control test bed development to validate and refine the WASP architecture and implementation during various integration sessions in the dairy test farm in Lelystad (The Netherlands)

In the prototype illustrated in Figure 35, a large-scale network of in total 79 nodes has been successfully deployed which involved 27 real cows (each wearing two nodes) 10 “virtual cows” (nodes attached on a test wheel to simulation motion and thereby to stress test the deployment), 14 stationary forwarder nodes and 1 base station node. In a separate large-scale laboratory trial, while preparation for the 79 nodes field trial, additional TelosB nodes were deployed and in total a group of 127 nodes did ran stable for more than a day.

Various deployments tools, described in section 3.4.3.1, have been used to remotely monitor the services installation on nodes (node monitor), the network topology (topology monitor), and network performance and node stability (statistics monitor) during run-time. Besides monitoring, also node settings can be remote changed to upload new services or new parameters e.g. for the Gradient RSSI threshold (to improve packet delivery) or the nanoSLP period (to perform a network traffic stress test).

For the interaction with the end user, for this moment the animal behaviour researchers of partner WUR, a prototypical Dairy Farm Information System was developed and tested. This Farm Information System is build upon the generic EIC mediation layer that was used to collect the data form the experiments and establish data trustworthiness.

From the various Smart Dairy Farm topics sketched in the previous section, our test scenario addressed animal health and, more specifically, focussed on remote monitoring of cow activity behaviour and locomotion to early detect claw health problems for individual cows. Measurements are performed with a 3D accelerator sensor that is integrated in the WASP sensor node. The measurement interval, duration and the frequency are adaptive to the

behaviour of the animal. Cow activity behaviour is translated to different modes that are used during the testing phase:

- **MODE 0:** continuous stream of 3D values, 3x10 or 3x50 measuring values per second;
- **MODE I**: Data that the cow is laying (one value, e.g. '1');
- **MODE II**: Data that the cow is standing (one value, e.g. '2');
- **MODE III**: Data that the cow is walking (one value, e.g. '3');
- **MODE IV**: Step duration, length, swing (amplitude of the sideward movement) and amplitude (four values)

MODE 0 will not be activated during default operation but has been implemented to check proper accelerometer operation by the WSN system integrator and for raw data collection by the animal behaviour scientist for off-line (activity classifier and locomotion) algorithm development.

MODES I, II, and III offer the option of switching to raw 3D-accelerometer data with a frequency of 10 Hz and a interval length of 1 s (3x10 measuring values) and in MODE IV there is an option of switching to raw 3D-accelerometer data with a frequency of 50 Hz and a interval length of 10 s (1500 measuring values). Accelerometers attached to a leg of the dairy cow can be used to record the locomotion. The feasibility of using 3D accelerometers in a wireless sensor network (WSN) for determining the locomotion of the feet of a cow was explored; a procedure to distinguish steps and to derive step parameters (length and time) was developed. It is possible to detect steps and to derive step parameters. The algorithm seems to be effective and has been tested using a test wheel and on raw data collection from three cows.

Up-scaling from a first successful small-scale (15 nodes) deployment, showing the basic WASP system functionality, to the final large-scale (79 nodes) deployment proved rather cumbersome as we encountered (seemingly non-deterministic) node stability issues that, in the end, were caused by an accumulation of several sources on sensor node level. A description of the main issues encountered when up-scaling is provided in section 4.4.1.1 as part of a dedicated wrap-up section on general integration lessons learned. A big benefit of the integrated WASP system proved to be that, since debug facilities of sensor node deployed in an actual test bed are extremely limited, the ability to build and adapt remote monitoring tools to assess during run-time installed node services, node stability, network performance (packet delivery) and topology enabled to trace to origin of multiple sources. In its current state, we are confident that the integrated WASP system can serve as research tool for animal behaviour scientists at WUR to enable data acquisition for period up to several weeks to study behaviour and, in the end, come up with better models and treatments.

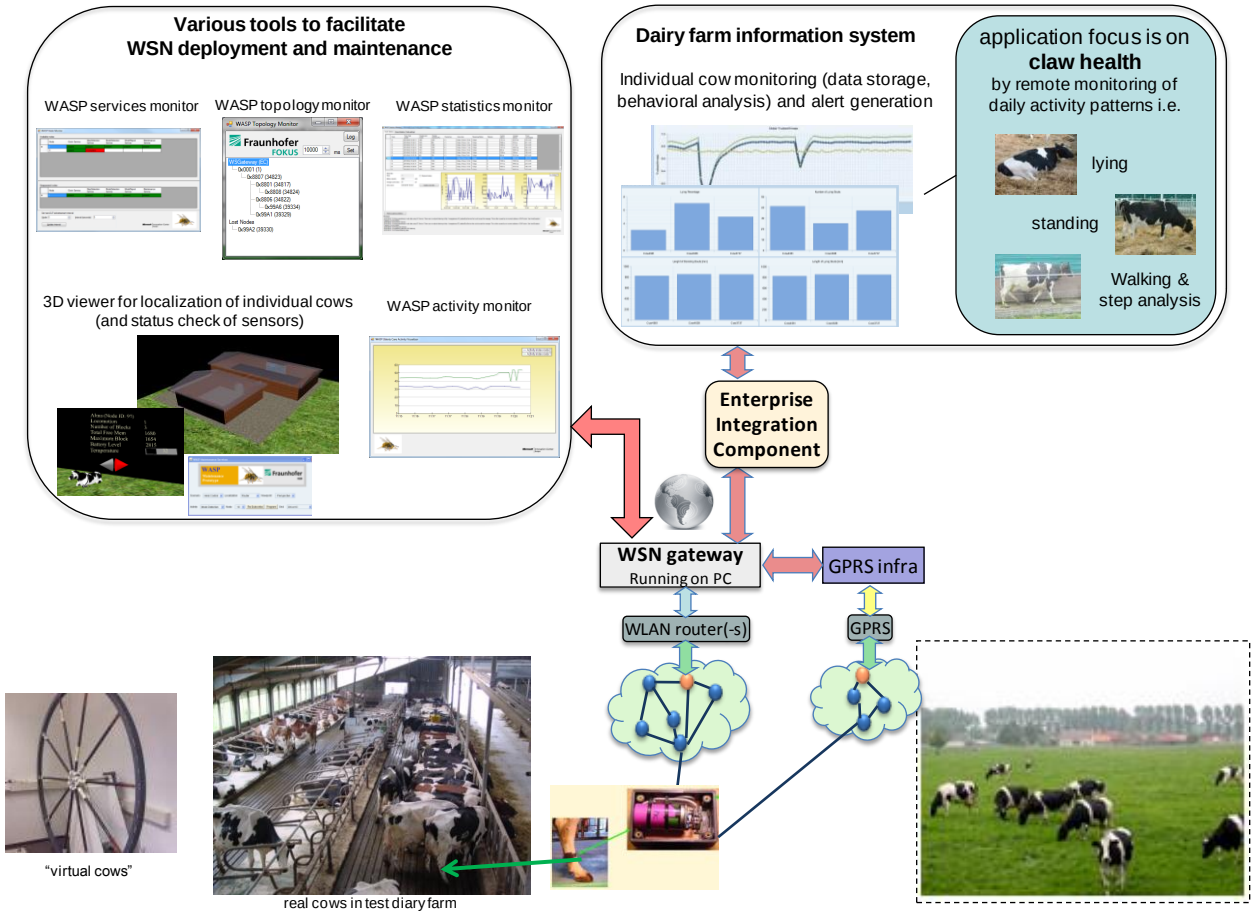


Figure 35 Herd control test bed setup for which the indoor case has been successfully demonstrated within WASP.

Throughout WASP run time, the herd control concept has been discussed with members of the Smart Dairy Farming (SDF) study group of value-chain stakeholders within the Netherlands involving Friesland Campina, CRV, Agrifirm and the NOM. These discussions contributed to the further plans of the SDF study group and assisted in selection of priorities, like "real-time" cow observations, and follow-up projects. The final herd control test bed scenario and results have also been discussed with this SDF consortium. Among several application-oriented questions, the steps needed to go from a research tool to industrial farm management information tool were of interest. Here, our thorough analysis of encountered test bed problems and solutions proved very valuable insights to this community.

4.1.2 Application and deployment trade-offs

Encouraged by the large-scale herd control deployment whose operation is limited to 10 to 12 days, we performed an exercise to assess whether it seems feasible to monitor a large number of cows over a period of a year without replacing the 8Ah batteries. Through this exercise, we wanted to get a feeling on application and deployment trade-offs like

- acceptable time between packets generated at application level
- acceptable number of battery-powered mobile sensor nodes
- required number and positioning of battery- and/or mains-powered forwarder nodes
- required number and positioning of sink nodes

that do comply within application QoS constraints like

- size (and hence energy capacity) of batteries
- packet delivery ratio
- packet delivery latency

Figure 36 illustrates two application categories and their associated time between (application) packets that are useful to monitor both healthy cows (sending health reports to the farm information system at low periodicity) and cows that need attention (and send frequent health reports and information about their location).

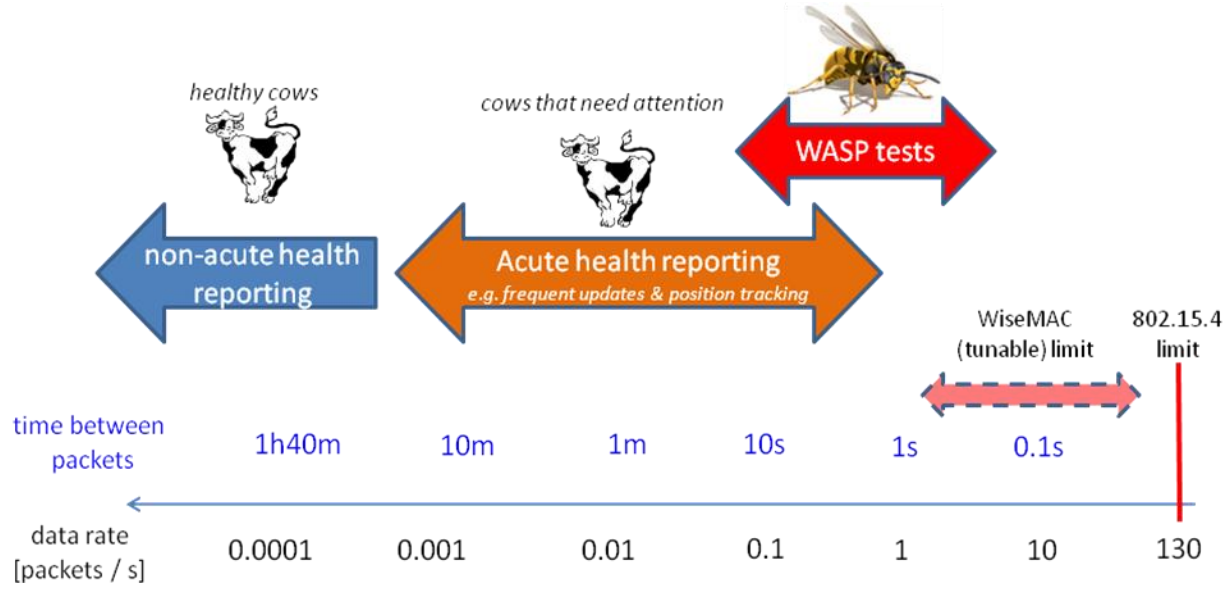


Figure 36 application ranges for herd control

The figure also indicates the application traffic typically generated in the WASP tests that are deliberately in a regime to stress test the sensor network capacity and the amount of data that can be send to the WASP gateway and enterprise system.

Network capacity is limited by the link & MAC layer and, in case of a (non-beacon) IEEE 802.15.4 MAC, typically is around 130 packets per second (assuming 120 Bytes per packet @ 250kbit/s and 50% overhead & efficiency). However, use of the (non-beacon) IEEE802.15.4 MAC, although fine for test bed development, is prohibited for long time operation as it requires the radio transceiver to be continuously active. This radio transceiver is the main power consumer (around 20mA) and causes the 8Ah batteries in the HC test bed being emptied in 10 to 12 days. Other (beacon-based) IEEE820.15.4 or alternative MACs exist but these require additional time synchronization traffic especially in mobile environments. In parallel to test bed integration using the power-hungry (non-beacon) IEEE802.15.4 MAC, the low-power (non-beacon) WiseMAC protocol has been integrated in the WASP-OS (WOS) multi-threading environment. WiseMAC is a contention single-channel MAC protocol based on non-persistent carrier sense multiple accesses that is further described in Section 5.6 of the WASP Development Handbook (D7.4). Depending on the WiseMAC sampling period, that determines the duty-cycling of radio transceiver between active and sleep mode, the throughput of WiseMAC can range typically between 1s (low but good for energy efficiency) and 50ms (higher capacity but at energy penalty) as illustrated in Figure 36.

The sink-to-router serial connection can handle up to around 9000 packets and, subsequent, router-to-gateway and gateway-to-backend links are not likely to form a capacity bottleneck as long as the right (IP-backbone) infrastructure is installed. Also, in deliverable D5.7 the WASP gateway is shown to be capable of dealing with quite some traffic when using a modern PC

and, hence, the major challenge is in deploying a wireless sensor network with manageable (= minimum) amount of sinks & forwarders.

To further assess application and deployment trade-offs within a single-sink network, we performed simulations for Gradient Routing on WiseMAC on a reference deployment shown in the figure below with 11 static forwarders and a number of cows, each wearing two nodes, equally (and statically) distributed across the (yellow) barn area to limit simulation times.

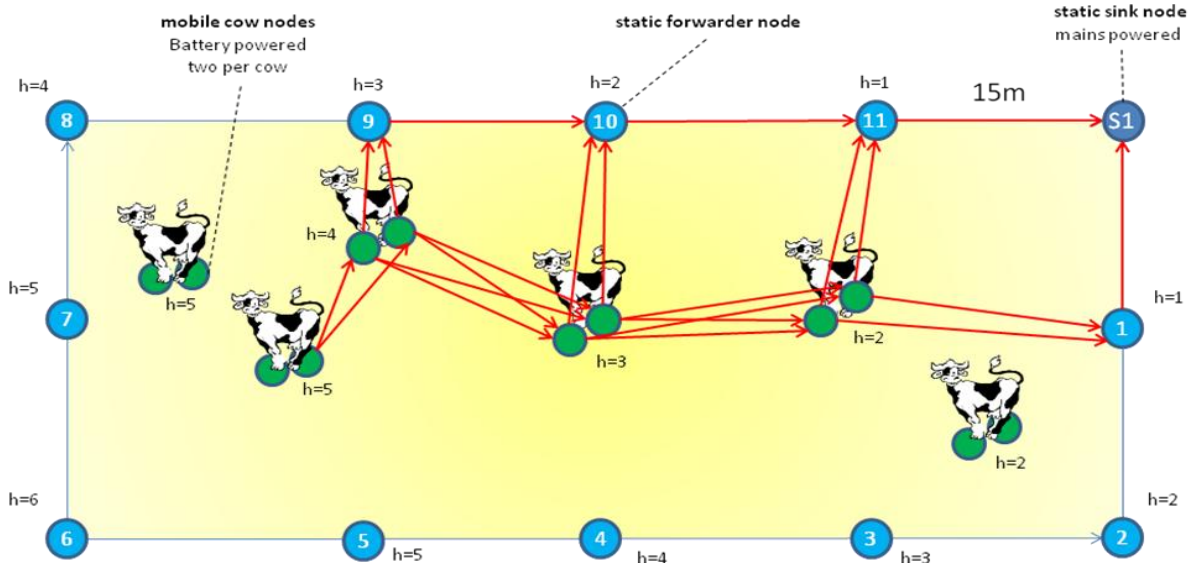


Figure 37: Simulation setup to assess application & deployment trade-offs for herd control.

Simulation results for a group of 10 cows and 50 cows are depicted in the Figure 38 below where the QoS metrics used before, i.e. average packet delivery ratio, average packet delivery latency, and average power consumption, are shown as a function of time between application packets. The results reflect that at relative long time intervals between packets, QoS performance is best which can be understood well since this corresponds to a low-traffic condition. At decreasing time intervals between packets, traffic density and, hence, competition for the radio channel increases which leads to degradation of QoS performance.

For 10 cows (i.e. 20 cow nodes), the QoS metrics are acceptable for the application requirements but when sending packets at time intervals shorter than 30s, packet delivery starts to drop below 90%, latency goes up to minutes, and power consumption breaks the 3mW constraint of (battery-powered) operation of 1 year (= 8Ah / (364 days per year *24 h per day) * 3V). For 50 cows (i.e. 100 cow nodes), the time between packets needs to be significantly increased to far above 1000s (>17 minutes) to get acceptable QoS performance including a one year operation time. Note that this QoS perfomance is already fine for non-acute reporting (say once or twice a day) for the majority of cows while enabling acute monitoring for a few cows. Further QoS improvements can be obtained by (cross-layer) optimization of Gradient and WiseMAC which, due to the various technical issues encountered during integration, has hardly been pursued. For example, currently Gradient only has a rather basic mechanism to filter duplicate packets that are so characteristic of broadcast-based routing protocols. This duplicate filter tracks whether a message from a given source with a given sequence number has already been forwarded and, if so, can be dropped. However, given memory constraints, only a limited number of message identifiers can be tracked. In large-scale deployments, the number of sources can outnumber the duplicate filter entries and thereby degrade filtering quality. This is illustrated for the 50 cow simulations in Figure 38

where simulations are shown for 32 (default used in the test bed), 64, and 200 duplicate filter entries.

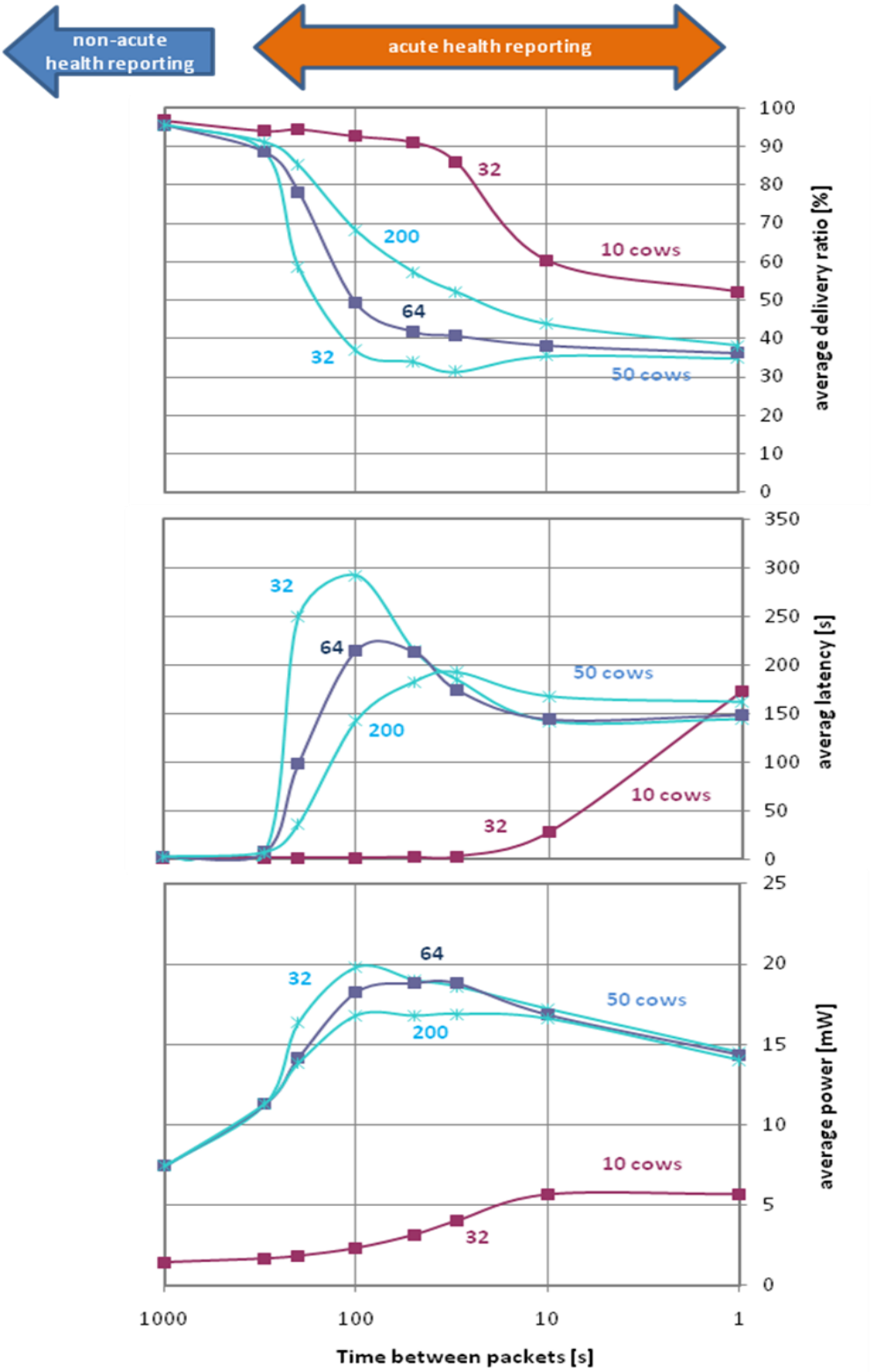


Figure 38: Simulation results for the Herd Control setting depicted in Figure 37. The numbers (32, 64, 200) next to the curves indicate the duplicate filter entries for Gradient.

The CCBR / TrawMAC stack, described in Section 3.3.2.2 and also based on link-layer broadcast communication, has been much more (cross-layer) optimized. Here, several advanced reduction mechanisms have been implemented and that confirmed the ability of greatly reducing the impact of broadcast transmission, letting a minimal number of nodes forwarding a packet and minimizing the power spent in such transmission [D4.4, D4.5, D4.7]. For example, a "delay and cancel" mechanism has been integrated that filters duplicates and favours forwarding by nodes closest to a sink. Originally this mechanism was built in CCBR [D4.4] but currently, by cross-layer optimization, it is integrated in TrawMAC [D4.7]. Also, by collecting neighbourhood information, TrawMAC is able to minimize preamble length (hence TX power consumption) and, in extreme situations, TrawMAC may choose to replace broadcast transmission with multiple unicast transmissions [D4.4, D4.7]. Finally, by adopting a queue for packets to be transmitted at the next wakeup, TrawMAC may aggregate those addressed toward the same destination (i.e., all packets in the queue for broadcast transmissions) to use a single preamble for multiple packets (thus reducing latency and power consumption) [D4.4, D4.7].

Although stack optimization is always beneficial, QoS performance can readily be improved by adding multiple hardware routers to the deployment to provide an additional sink e.g. at forwarder #6 in Figure 37. The WASP IP infrastructure allows for the deployment of multiple (hardware and software) sinks and, thereby, to divert traffic going to a single sink although adding routers evidently comes at additional (hardware) costs.

Another manner to significantly improve QoS performance would be to deploy mains-connected forwarders if allowed in the application environment. In this case, forwarders can run WiseMAC-HA (High Availability) that, by default, puts forwarder nodes in continuous receive mode, using a (non-beacon) IEEE802.15.4 MAC roughly allowing for 130 packets/s instead of around 4 packets/s for WiseMAC (with sampling period of 250ms), while switching to WiseMAC when transmitting to battery-powered (WiseMAC-based) cow nodes. When modifying Gradient (or CCBR) such that (mains-powered) forwarders act as intermediate sinks, the forwarders will be used as a preferred route with increased bandwidth to the sink.

In summary, this section explores some of the main options for application and deployment trade-offs when monitoring large numbers of cows in a basic reference configuration. Complementing these trade-offs, the cross-layer optimization and QoS trade-offs in the network layer addressed in Sections 3.3.1.3 and 3.3.1.4 can be applied to further optimize network performance as well as the network protocol selection methodology described in Section 3.3.4.

4.2 Elderly care application driver

The choice of elderly care as a WASP test-bed application is motivated by the steady decrease of the health-workers to retirees and the miniaturization of technology allowing sensors to be pervasively woven into the patients' surroundings without affecting their daily routines. This domain is of growing societal relevance as the proportion of people aged 60-plus is projected to increase rapidly in the next two decades, combined with an increase in the number of patients having chronic diseases, including cardiovascular diseases, diabetes and chronic respiratory disorders (such as COPD, Chronic Obstructive Pulmonary Disorder). Figure 39 shows the growth in elderly population over the next 50 years. In the developed world, the elderly population (over 65) is projected to be more than 30% in 2050. The increase in number for both elderly people and patients with chronic diseases requires novel methods for healthcare management that can deal with the problems posed.

The use of wireless sensors for elderly care could enable remote monitoring that allows care providers and clinicians to observe patients' health continuously instead of obtaining snap-shot assessments as currently being done during hospital or general practitioner visits. It could also provide alerts in case of symptom deterioration or a perceived threat, such as that of

hypothermia or long periods of inactivity. WASP addresses remote elderly monitoring with a special focus on activities of daily living that are important in assessing the decline of cognitive and physical functions over time and on cardiac ECG monitoring.

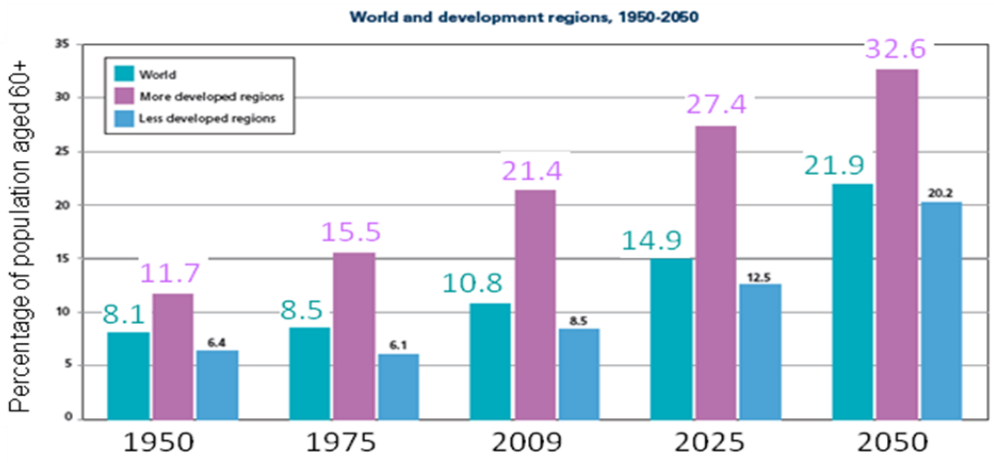


Figure 39 World-wide growth of population aged 60+. Source: /www.un.org/esa/population/publications/ageing/ageing2009.htm

Within WASP, selected EC application scenarios are centred on three main application areas: daily activity monitoring, hypothermia detection and cardiac alerting.

Activities of Daily Living - One of the major objectives of modern health care is to ensure that patients with chronic diseases, such as diabetes and COPD, as well as elderly patients are treated in a community setting and supported at home whenever possible. This would result in a lower demand on essential health care resources, release hospital capacity, and allow for more targeted care provision. The monitoring of Activities of daily living is relevant for the following reasons:

- Aiming to early detect and quantify disabilities by monitoring primary activities such as the ability to dress, use the toilet, eat … [20], Disability is among most commonly-used health indicators in the elderly population because of its close correlation with the need for social services provision [20].
- Significant correlations between the decline in levels of ADL and reduced well-being, increased social withdrawal, and reduced engagement in active and passive activities in elderly with dementia [22]
- Measuring parameters that could indicate self-care and daily function for patients with Alzheimer's disease [23].
- Mobility associated with post-operative recovery as well as rehabilitation after treatments. COPD is a good example, where mobility is linked to lung function improvement [24]

Hypothermia Detection - Hypothermia is when the core body temperature drops below 35°C. It can be classified as mild (30-32°C), moderate (22-25°C) or deep hypothermia (0-8°C) [25], [26]. Symptoms progress from shivering to lethargy to confusion, coma, and death. Risk factors for hypothermia in the elderly include reduced physical activity, immobility, dementia, under-nutrition, many other common disorders, and many commonly used drugs. Elderly patients may not recognize that they are cold and may not shiver. Symptoms may be nonspecific, and the diagnosis is easily missed. Elderly patients undergoing surgery require special precautions to prevent hypothermia. Hypothermia is known to cause about 600 deaths each year in the US [30], but this figure is probably an underestimate. For elderly patients with diagnosed hypothermia, the mortality rate is 50%. Mortality rate increases with aging, particularly after age 75 [30] as shown in Figure 40.

In Britain, officially 300 elderly people die from hypothermia each year [28] but the estimated number is much higher as the incidence of hypothermia of elderly is uncertain [29]. Also in the UK, 3.6% of elderly admissions (aged over 65 years old) are hypothermic [27]

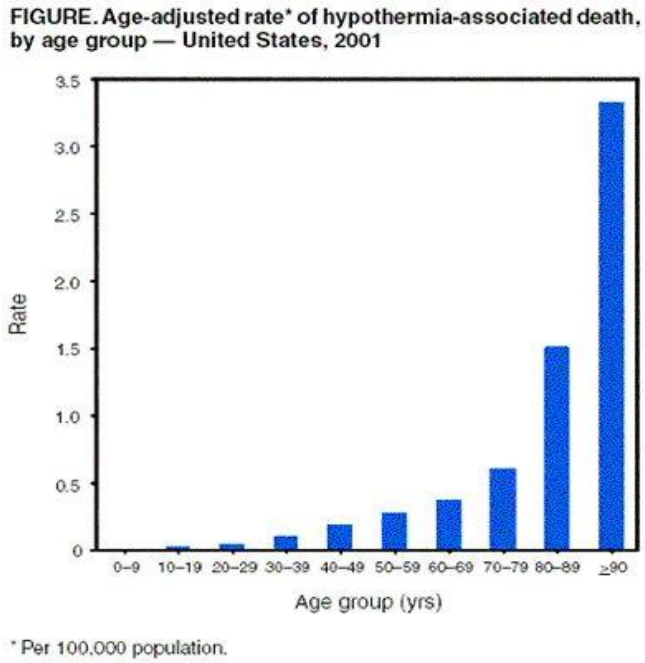


Figure 40 Figure showing the age-adjusted rate of hypothermia from; http://www.nia.nih.gov

ECG Monitoring and Arrhythmia Detection - Cardiovascular disease (CVD) is the main cause of death (30% worldwide) and disability. The main forms of CVD are coronary heart disease and stroke. About 12 million people die of heart attacks or strokes each year (but they can often be prevented). Nearly two-thirds of them die before they can reach medical care. In the U.K. alone, each year CVD costs the health care system around £1.8 billion (Source: World Health Organization and British Heart Foundation). Mortality rate of CVD (significantly) increases with age as depicted in Figure 41. In the European Union, CVD kills over 2 million people per year, costing the European economy over 192 billion Euros annually.

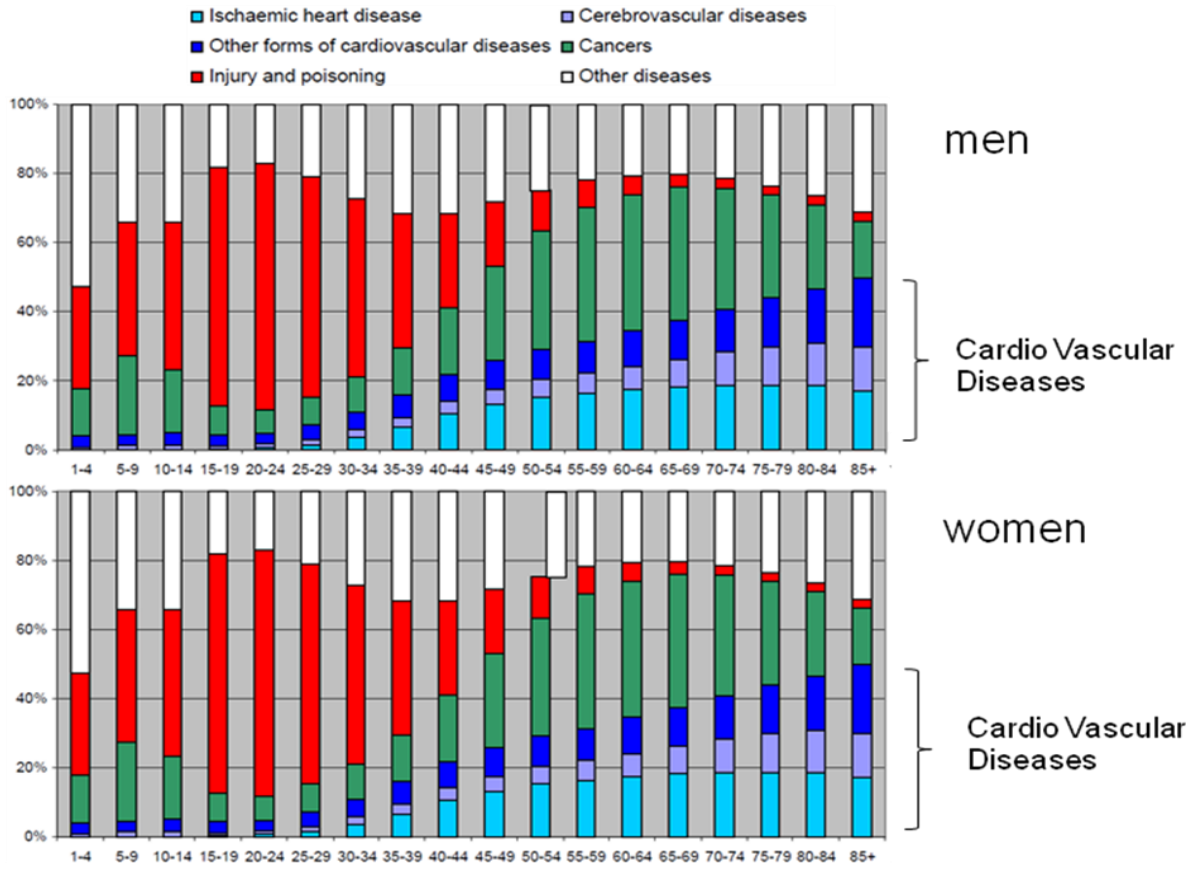


Figure 41: Proportional mortality rate of CVD within EU.
Source: EUROSTAT(http://epp.eurostat.ec.europa.eu)-2007

The above application scenarios have been used to validate the WASP prototype as a solution to provide remote services tuned to individual elderly needs. Ageing is a very complex process affecting physical and biological processes as well as demographic and psychological factors. Remote monitoring and alerting services could thus provide means of supporting independent living capable of addressing some of the problems posed by the above research areas. The following section includes a description of the test bed as well as the scenarios implemented.

4.2.1 **Elderly care integrated test bed**

Figure 42 shows an illustration of the Elderly care setup for the test bed as well as the scenarios tested. These scenarios were designed after careful discussion with medical staff affiliated with WASP partners making sure several of the relevant areas discussed above are highlighted.

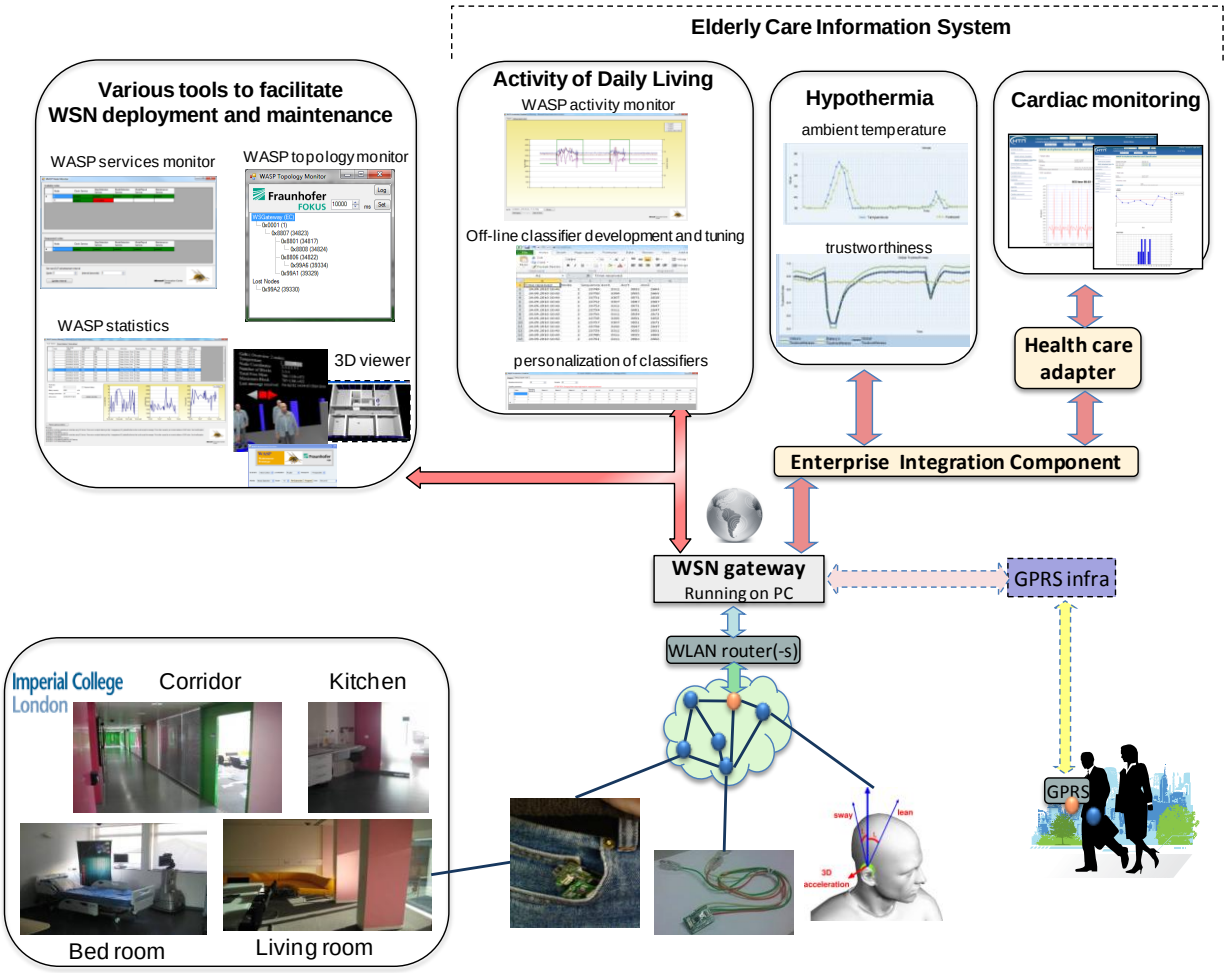


Figure 42: Elderly care test bed setup for which the indoor case has been successfully demonstrated within WASP

The tested scenarios (which are detailed in D6.7) are:

Ease of Deployment - This scenario consists of a preliminary step in order to move on to more elaborate scenarios and make sure that the nodes are reliable and can be used for the suggested demos. The following parameters are to be shown:

- **Node monitor**: Show service installation upon node power up. This would allow users to detect if all services are installed on nodes. If a node switches off or reboots, this would also be shown in the monitor as the node (and its services) would quit the list, then join once all services are back.
- **Topology monitor**: It is important to have a tool that can show network topology which as patients will be moving and we would like to observe how well the system copes with the change of distances and network structures.
- **Gateway Visualiser**: The gateway also has a visualisation screen that enables the user to observe the signals coming in from the nodes in real-time.
- **Maintenance GUI services**: These services are useful for person tracking, providing a connectivity graph and showing battery status per node.
- **Statistics service**: Enabling the user to observe packet delivery ratio as well as other statistical measures that would be useful to monitor the performance of the system.

Monitoring Activities of daily living - 4 sub-scenarios will be investigated. These are as follows:

- **Activity detection:** The aim is to be able to detect the activity of several patients at the same time. We will aim to do testing with 6 patients wearing 2 nodes each and show the data in the backend.
- **Integration of ambient and wearable sensors:** This scenario proposes to combine information obtained from wearable and ambient sensors. Data from both modalities could provide more context-awareness as activity levels could be correlated with location and information about the surrounding. For this work, we will be using a combination of wearable WASP nodes with the Housevibe radar. Housevibe is a radar-based monitoring system that provides information on human movement in a given indoor area.
- **Behaviour profiling:** This scenario will test the wearability of sensors for a long period to enable monitoring of daily patterns.
- **Personalisation with raw data service:** This scenario will show how collection of raw node data enables us to change parameters that can be used for long term monitoring. A specialised application will be developed for the gateway to enable visualisation and storage of raw data from the node. This application is of importance for gait monitoring applications as well as applications requiring a synchronised and detailed observation of body-part motion.

Detection of hypothermia conditions - Instead of measuring on-body temperature which could pose problems of wearability for elderly subjects, this scenario aims to monitor all rooms in a house and indicate if temperatures fall below a certain threshold. This threshold can be remotely set from the clinician's office (through the EIC). The system generates alerts that can be viewed on the backend (EIC), which can help a community matron locate elderly homes where there are temperature drops below acceptable levels.

Cardiac monitoring - In this scenario, arrhythmia detection will be implemented on the node allowing an alert to be sent to the backend. 90 sec of ECG data will then be sent to the backend in order to allow the clinicians to assess the arrhythmia further. Alternatively, if no arrhythmia is detected, an OK signal is send to the backend at regular intervals. The Telmed system will be used to show alerts as well as raw data from a special board created to be used with the WASP node (shown in Figure 43).

Figure 43: ECG add-on sensor board to be used with the WASP node.

Falls - Falls are a serious problem in the aging population. They can indicate deterioration in health and are related to a decrease in musculoskeletal and motor-control function, with identified rates of mortality and morbidity in the elderly [7]. After the age of 60, there is a general rise in the number and severity of the falls, with an increased tendency linked to further complications. After this age, these rates increase substantially [8]. Falls affect elderly living not only in the community, but also in nursing homes and hospitals. For elderly living in the community, however, help or medical assistance can be difficult to summon and victims could remain without assistance for many hours. In the WASP project, we used the non-

functional demonstrator described in section 3.2.2.4 to show the capability of the system to deal with falls. Here, the aim was to monitor the aftermath of a fall in order to infer whether the fall had no consequences or it required the intervention of a doctor or a clinical specialist.

Overall, EC test bed trials show successful tests of WASP solutions as research tools for behaviour scientists at ICL to enable data acquisition for long periods to study behaviour and consequently come up with better models and treatments. The WASP project also poses many solutions for issues that are of high importance for elderly monitoring. These include the synchronisation of data, mobility and dealing with scaling to a large number of sensors, as explained in the next paragraph. Future work could include the inclusion of some of these techniques for patient trials planned at ICL, including orthopaedic applications and activity monitoring.

4.2.2 Application and deployment trade-offs

Similar to the Herd Control exercise in Section 4.1.2, we performed simulations get a feeling of application opportunities (in terms of application packets per second) versus deployment constraints (e.g. maximum number of persons with nodes at given settings). Figure 44 illustrates two main application categories and their associated time between (application) packets that are useful to monitor (1) persons with non-acute health issues (sending health reports to the health information system at low periodicity) and (2) persons with acute health issues (for which frequent health reports and information about their location should be sent). Again, the figure also indicates the application traffic typically generated in the WASP tests that are deliberately in a regime to stress test the sensor network and shows the maximum data rates that can be send to the WASP gateway when using (low-power) WiseMAC or (power-hungry) IEEE802.15.4 MAC.

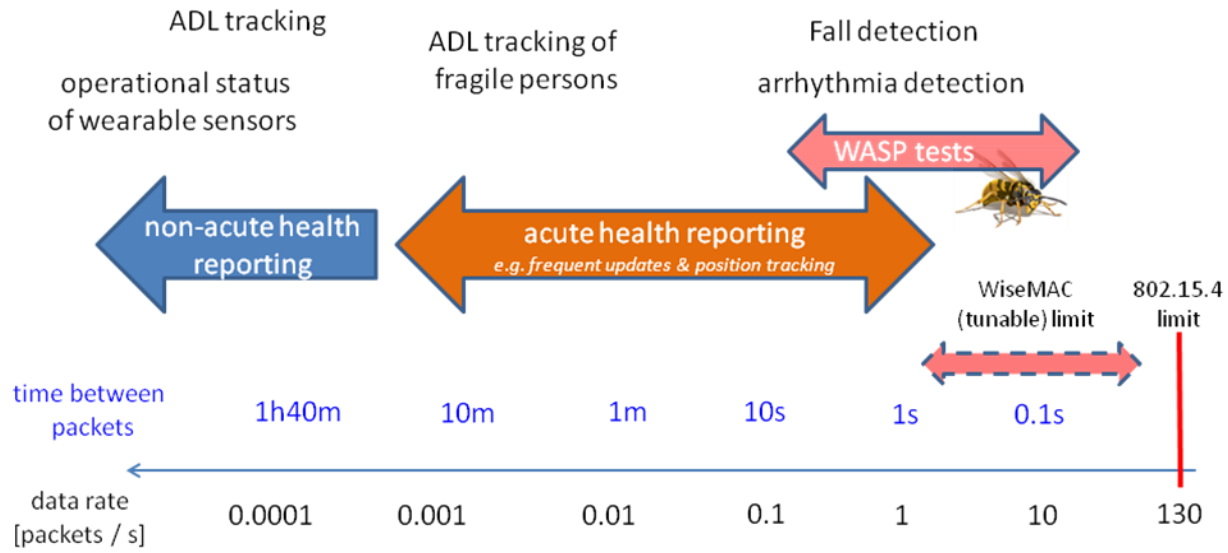


Figure 44: Main acute and non-acute application domains and associated data rates for elderly care

The simulation scenario as used for the Elderly-care scalability assessment is shown in Figure 45. As a reference, we consider a typical elderly-care home consisting of multiple private rooms as well as a larger common room for social events where many elderly persons can gather for lunch, diner, or other group events. Especially during group events, a high density of persons of which most wear nodes will stress the network and, therefore, we limit our simulation exercise to the common room with an assumed area of 20 by 20 meter. As indicated in Figure 45, nine static forwarders are distributed in a grid across the ceiling with the intention to enable sufficient paths to the mains-power sink node (indicated S1 and connected

to WLAN router) in case the default (shortest) path to the sink is temporary blocked by e.g. persons moving across the area. In the simulations, we assume that a single wearable (battery-powered) node is connected to every elderly person but, to limit the simulation effort, we exclude mobility and simply equally distribute the persons / nodes across the room. As in Section 4.1.2, use of a non-duty-cycled MAC protocol would require a relative large (8Ah) battery for 24/7 operation over 10 to 12 days. However, these 8Ah batteries are way too big and heavy for integration in “unobtrusive” wearable nodes intended for elderly persons. Hence, (low-power) WiseMAC is used to reduce radio power consumption and, thereby, enable use of smaller batteries. As figure of merit, we aim for an average power consumption below 6mW which would allow for 10 days operation time on a 500mAh (~3V) battery. In these simulations, the Repairing routing is used which (repeatedly) constructs an addressed-based routing tree and is further described in D7.4 WASP Development Handbook. To deal with potential mobility (although not included in the current simulations for simplicity), a Repairing route-rebuilding (refreshment) period of 5 seconds is considered as sufficient in this first exercise.

Simulation results for both 10 and 50 elderly persons are depicted in Figure 46 where the QoS metrics used before, i.e. average packet delivery ratio, average packet delivery latency, and average power consumption, are shown as a function of time between application packets. For each QoS metric, several simulation curves are included to illustrate the QoS trade-off impact when changing some of the QoS set points in the network stack in this case being (1) the maximum number of transmission attempts by WiseMAC and (2) the periodicity of Repairing ADVERT messages used for route building / refreshment.

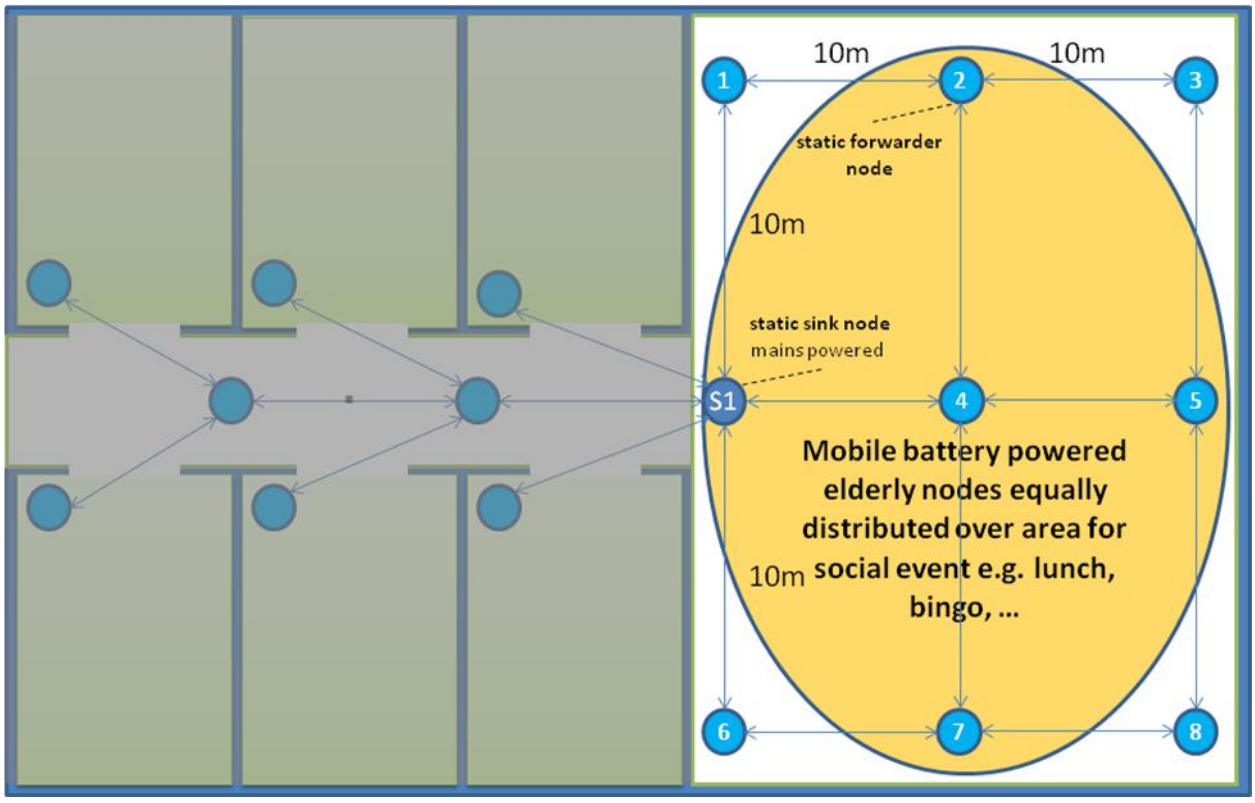


Figure 45: Simulation setup to assess application and deployment trade-offs.

For acute monitoring applications, a (very) high delivery ratio (>99%) and low latency (<3s) will be required. From the simulations, we can observe that for 10 elderly persons we are able to comply with these acute QoS requirements as long as we limit the time between packets (TBP) to be on average longer than 10s. Acute health reporting of 50 elderly persons in parallel is problematic due to the larger traffic load in the network. A Repairing route-building

period of 5s will not be able to provide delivery ratios above 50% due to the large Repairing traffic overhead.

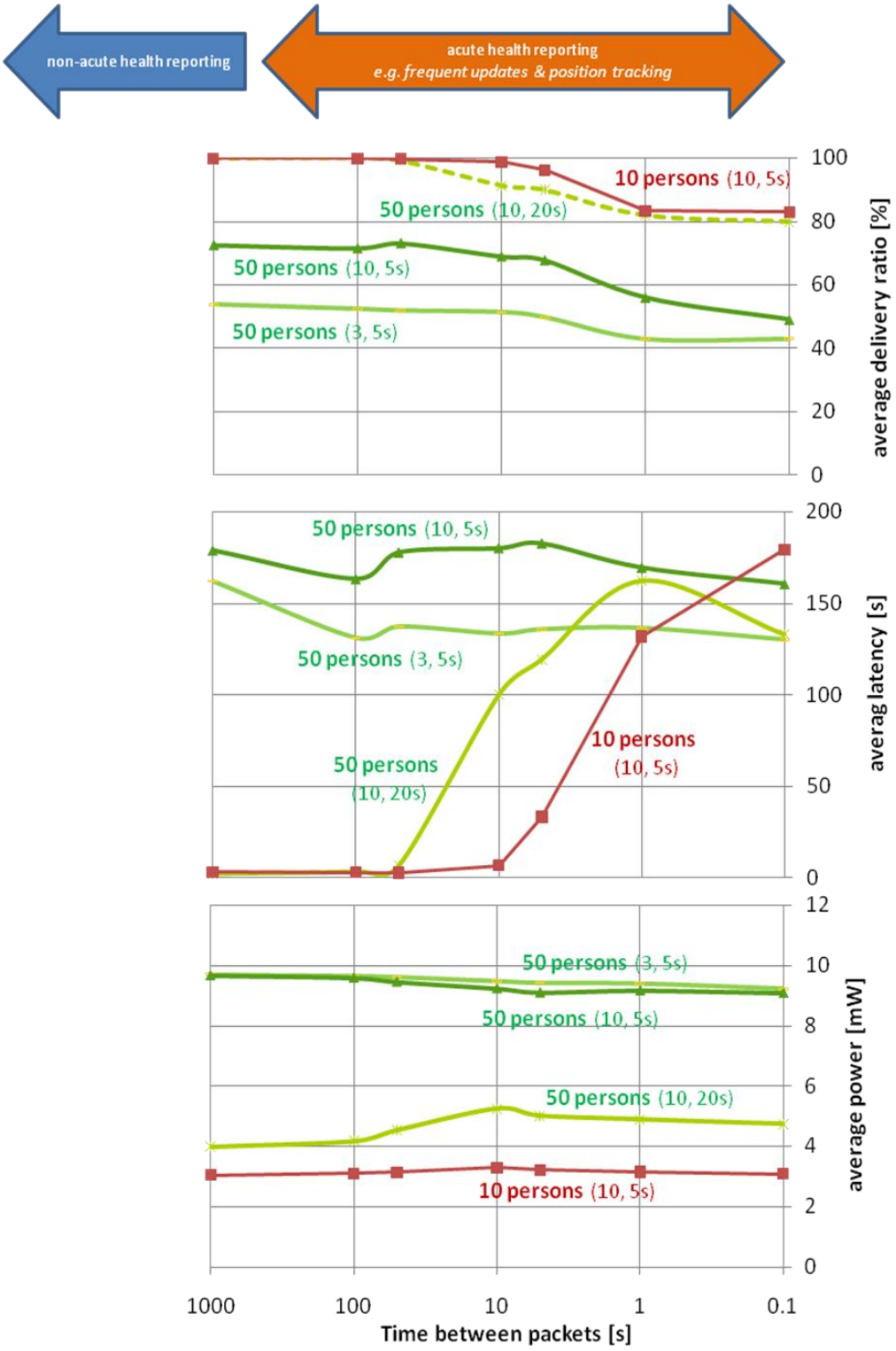


Figure 46: Simulation results for Elderly Care setting depicted in Figure 45.The numbers in brackets indicate (maximum number if transmission attempts by WiseMAC, periodicity of Repairing route-building messages)

Increasing the maximum number of WiseMAC transmission attempts from 3 to 10 attempts does improve delivery ratio up to 70% (at the expense of latency) but this level is still not

satisfactory for acute monitoring of 50 persons (which evidently serves as a worst-case / boundary scenario). A significant improvement in all QoS metrics can be achieved by reducing the Repairing route-updating period to say 20s as can be observed in Figure 46 by the curves labelled “50 persons (50, 20s)”. The downside is that the larger interval will affect the ability to deal with (fast) moving nodes as the routing tree, and thereby the appropriate parent for every node to forward packets to, is updated less frequently. In principle, acceptable QoS metrics could be obtained for an average TBP larger than 100 seconds which would imply that we cannot support simultaneous acute monitoring of 50 elderly persons. This does not have to be a bottleneck when only a limited number of elderly persons would need acute monitoring. However, the inability to support mobility at 5s intervals in case of 50 persons seems a practical bottleneck for deployment based on purely battery-powered (forwarder and wearable) nodes.

A venue for further exploration is, when designing for worst-case application requirements i.e. demanding acute monitoring of a large group of elderly persons supporting mobility at around 5s or shorter intervals, the solution already described in Section 4.1.2. In this solution, one can introduce mains-powered forwarder nodes that run WiseMAC-HA (High Availability) that, by default, puts forwarder nodes in continuous receive mode (and thereby removes an important QoS bottleneck) while switching to WiseMAC for transmission with the low-power (duty-cycled) wearable nodes.

4.3 Automotive application driver

Modern vehicles incorporate tens of sensors to provide information such as temperature, air quality, tire pressure, distance to nearby objects, etc., to the electronics control units (ECUs). The ECUs in the vehicles utilize the sensor information for various control functions and applications. In the current architecture, the sensors in the vehicle are connected to the ECUs via physical wires. The wiring harness in a car today could have up to 4000 parts, weigh as much as 40 kilograms and contain more than 1900 wires for up to 4 kilometres in length [31]. As the number of electronic components keeps increasing with the development of new features in the vehicles, the number of in-car sensors could be more than a few hundred in the near future. The physical wires, connecting the ECUs and the sensors is becoming problematic due to the following reasons:

- *Wires are expensive.* The physical connecting wires are usually shielded so they can be heat and interference resistant. The shielded copper wires in the vehicles could become a major cost component as the number of sensors increase. Moreover, it requires a significant amount of engineering effort, and hence cost, to design the layout of those wires in the vehicle. The effort needs to be repeated for every different model of vehicle.
- *Wires are heavy*. Wires and wiring harness are among the heaviest components in the vehicle. The weight of the wires could have a large impact on the fuel efficiency as the number of in-car sensors increase in the near future.
- *Wires are restrictive*. There are several locations in the vehicle where sensors cannot be deployed when using the current wired sensors, e.g., steering wheels, tires, and windshields. In addition, it requires significant effort to add a new sensor to or change the location of a sensor from the existing design; the layout of the wires needs to be modified.

It is thus imperative to create a new open architecture to support communications between sensors and ECUs. To address the aforementioned issues, many researchers are trying to design an in-car wireless sensor network, which might replace at least part of the physical wires with wireless connections. In such a network, the sensors have wireless communication capability and broadcast packets which contain sensor information to the base station(s) periodically. The base station(s) then relay the sensor information to the ECUs for further processing. The main constraints of such a solution are:

- *Cost*. One of the major motivations of building such an in-car wireless sensor network is to cut down the cost of the expensive shielded connecting wires and it would not make sense to deploy such a system if it is much more expensive. Therefore, the designs of the sensor node and the base station must be as simple as possible, and all of the following challenges need to be addressed in a cost-effective manner.
- *Longevity*. If the sensor nodes still need to be attached to a power supply via physical wires while the communication is wireless, it defeats the purpose of having an in-car wireless sensor network. Hence, the sensor node needs to be either passive, i.e., obtain the power from the electro-magnetic wave from the base station, or active, i.e., has an onboard battery. In either case, the energy available for the sensor node is highly constrained and the operation of the wireless sensor nodes, including radio transmission and reception, needs to be highly energy efficient. The lifetime of a sensor node should be in line with the designed lifetime of other components in the vehicle which is typically more than 5 years.
- *Communication Reliability.* Certain sensor information is crucial to the operation of the vehicle and needs to be delivered to the ECUs with a high success rate. The high channel loss and time-varying nature of the underlying in-car wireless channels could result in a higher packet loss rate than when using physical wires, which may require additional system design to compensate.
- *Network capacity.* Although the effective data rate of each individual sensor node is usually only up to a few *Kb/s*, as the number of sensor nodes increase in the near future the overall network load could still reach a few hundred *Kb/s*. Due to the multipath-rich in-car wireless channels and hence the low coherence bandwidth for some locations in the vehicle, the network capacity might not be sufficient to support the operation of certain sensor nodes.
- *Heterogeneous sensors.* Different from most of other wireless sensor networks, the requirements and purpose of the sensors in the vehicle are considerably different. The importance and priority of the sensor packets sent by different sensors are very different and needs to be prioritized; some need to be delivered to the ECUs before the others do. The packet sending interval of different sensors could range from 6.25 *ms* to several seconds. It is therefore necessary to adopt a design which can accommodate these heterogeneous characteristics of the sensors.

Current results and developments of WSN technologies, including WASP, and future initiatives could also represent the foundation in the development of new concept. The following step could be for example the integration of the WASP platform in a hybrid network of smart nodes inside the vehicle, based on heterogeneous technologies, where the WSN can act as a second level network, enabling the implementation of several applications related to user profile personalization and functionalities nowadays severely limited by cabling issues.

4.3.1 **Automotive simulations**

For the automotive domain, the applicability of WASP solutions has been assessed by performing simulations for some of the selected use cases.

From the list of potential scenarios provided in the first stage of the project and reported in D6.2, we have selected in-vehicle scenarios which are recognized, at present, as the most promising on the short to medium term. A network composed from several wireless sensor nodes is considered, enabling functionalities nowadays available through wired sensor solutions i.e. steering wheel buttons, temperature monitoring, passenger presence detection, side window and rear-view mirror control. Goal is to obtain a detailed insight of in-vehicle wireless sensor network performance (e.g. end-to-end latency), reliability, power consumption (node lifetime) and overall node activity for related use cases in order to find out whether (and which) wired sensors may be potentially replaced by wireless sensors.

Another automotive scenario that has been considered for assessment through simulations concerns “Wireless sensors and data replication within a tunnel” described in D6.2 as well.

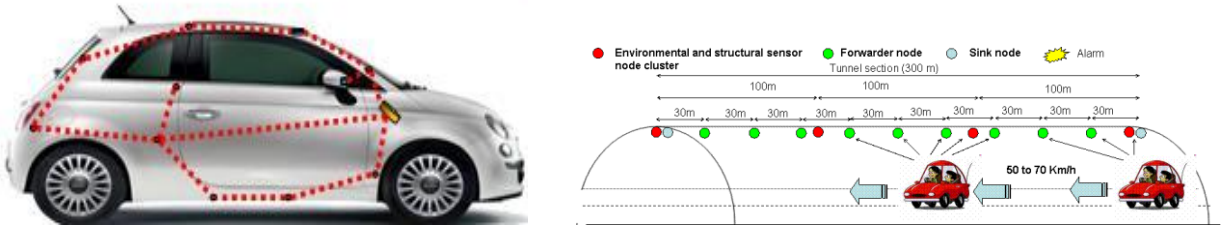


Figure 47: The automotive in-vehicle and tunnel management scenarios evaluated through simulations.

The BAN stack is adopted for simulations of the in-vehicles network applications whereas the HC stack is used for the tunnel network. Different simulation models are defined and tested for all the selected use cases in order to evaluate network performance and identify weak points. The results coming from the simulations are compared with use cases requirements. Details about reference applications, simulation model and environment, overall network design, tests performed and results obtained are reported in Deliverable D6.7.

The analysis is not focussed on specific applications. The design of the networks considers a generic deployment of nodes aimed to assess the sustainability of the overall node activity rather than to tune minutely the configuration parameters of a particular sensor application. The sensors referred in the trial have been selected just to set some domain-specific requirements. The number of nodes in the network, their deployment, the transmission rates and the interaction rules, have been set as generic as possible in order to take into account the majority of possible cases and sensors.

Results coming from the simulations demonstrate that the WASP solution, adopted for the protocol stack targeting Body Area Networks and here experimented with in-vehicle applications, guarantees very good network performances and reliability, with values of delivery ratio very close to 100% (99% delivery on average, which is less than 1% of data packet loss) and very low latency, in the order of tens of milliseconds. Collected results depict nevertheless high values of power consumption on average, resulting in a very limited autonomy of the nodes, especially considering devices sustained by batteries.

The same overall behavior results from the simulation of a network of wireless sensors for data replication within a tunnel, with a protocol stack composed by IEEE802.15.4MAC and Gradient Routing. Simulations of the same scenario based on a protocol stack composed by WiseMAC with Gradient, which actually represents the WASP solution targeting mesh networks with mobile nodes, provide on the contrary a good improvement in energy efficiency, although still far from what requested by the application requirements, but this at the expense of an overall worsening of the overall reliability of the system, both in terms of latency and delivery ratio.

We can therefore conclude that the tested WASP protocol stack architecture can be considered as a dependable solution to implement in-vehicle wireless sensor network, surely in terms of packet reception rate but also in terms of maximum packet delay, especially for all those applications where the data are not being used to make challenging actuations and mission critical decisions with sub millisecond latency requirements.

On the contrary, power consumption is at present the biggest fundamental bottleneck toward the introduction of such a technology in cars and automotive applications. This represents one of the biggest questions to be addressed and further investigated by researchers in the next

years. Further improvement in performances could be achieved for all the metrics with even more customized cross layer optimizations and protocol selection but as regards the energy autonomy, future enhancement in energy saving should be anyway coupled with energy accumulation techniques and energy harvesting to ensure an unfailing and self regenerating power supply.

4.4 Integration lessons learned

Throughout this document, and in other final WASP deliverables, lessons learned have been described at various levels of detail. In this section, a collection of main lessons has been collected to provide an overview of especially those lessons derived from the development of integrated demonstrators. The Elderly Care and Herd Control test beds proved extremely useful to identify various technical and practical issues that typically emerge only during actual deployments. Numerous technical issues appeared on WSN level and a description of the most relevant ones is included in section 4.4.1. Complicating factor is that the accumulation of various smaller and larger bugs resulted in a seemingly non-deterministic malfunctioning and a case example, encountered when up-scaling from the (first functional) small-scale to a large-scale herd control deployment is illustrated in section 4.4.1.1. Another case study reflecting the issues encountered but overcome when integrating WiseMAC in the WASP embedded environment is presented in section 4.4.1.2. Also on back-end level, various lessons learned have been captured in section 4.4.2 and we conclude with test bed integration lessons in section 4.4.3. Note that throughout D8.4, more lessons can be found for topics under discussion.

4.4.1 **WSN lessons**

- WASP SVN repository and WASP Build System to facilitate embedded code integration:
    - Proved very useful and, hence, considered essential for a IP project like WASP
    - One lesson for improvement is to install an automated regression test to filter out partner uploads that break proper functioning of integrated code that worked before. In WASP practice, components uploads were often only partially tested and any degrading effects on integrated code had to be back traced.
    - Regression tests using simulations are a good first step, however, also real node regression tests for previous functioning (stress) conditions are highly-recommended to filter out real-time degradation.
- Remove dependences upon partner development environments
    - WASP partners used various host machines and OS environments for code development which, in early integration phases of WASP, caused integration issues resulting at one point in only one partner being able to properly generate code that could be flashed on nodes.
    - By detailed comparisons and benefiting from adoption of WASP Build Systems these dependencies have been removed and, moreover, the development of the Virtual Machine Development Environment helped out well
- Converge on coding style and active use of code reviews
    - During integration of individual components, many issues occurred due to different coding styles e.g. memory allocation (static / dynamic), data structures accessible from multiple threads protected by a mutex, ...
    - In solving the issues, gradually a consistent coding style emerged and resulted in a code review check list that is included in D7.4 The WASP development Handbook.

  - For persistent problems, code reviews by other partners have been performed
  - Unfortunately, quite a few issues originated in the Mantis code adopted from the public domain which, during WASP runtime, appeared no longer to be actively supported. Analysis, isolation, and subsequent fixing of bugs especially in this code required several WASP partners to spend (significant) time outside their own commitments.
  - One lesson for improvement is to assign a dedicated or subcontracted partner for low-level tech support (such as operating system support) for components that are not developed by partners within the project

- Code integration and debug remains a time-consuming challenge especially in a multi-partner / multi-site setting
  - despite the (gradually improving) facilities and procedures described above, integration of novel components increases the number of tasks / threads that increasingly start to compete for (and require proper management of) SW and HW platform resources
  - the accumulation of various smaller and larger bugs (especially at the low-level SW support functionality) often presents itself in a seemingly non-deterministic behaviour of nodes that require in-depth analysis whereas (in-field) debug options are limited.
  - To illustrate this, two case examples are given in sections 4.4.1.1 and 4.4.1.2.
- The WASP gateway architecture, consisting of the IP Routers, GPRS Forwarder, and WS Gateway, proved to be very flexible in enabling WSN deployment trials and some lessons learned can be found in section 3.4.3.5

4.4.1.1 **Case example of large-scale HC deployment**

In October 2009, a first successful deployment of the entire WASP system in the herd control test bed has been demonstrated in the dairy test farm in Lelystad. This deployment concerned a simplified version of the targeted WASP system depicted in Figure 35 and involved a relatively small-scale (15 nodes) deployment and early versions of all key components and tools. After upgrading the HC-specific services, deployment tools and prototypical (enterprise) farm information system, we started attempts to scale-up the deployment aiming for a large-scale (79 nodes) deployment in April 2010. Early attempts were quite promising as medium-scale (~40 nodes) deployments seemed to work with reasonable stability of nodes and network performance for periods up to a few days after start-up. However, at some point in time, typically during the night, an unidentified event caused a period of multiple hours during which nodes started to collectively reboot (in a seemingly semi-continuous fashion) after which the network stability recovered automatically. This behaviour is illustrated in Figure 48 where logging data of the WASP gateway is displayed that helped in confirming that part of this instability behaviour relates to a sudden increase in traffic density which apparently caused nodes to start collective rebooting. Unravelling the underlying instability mechanisms proved quite challenging as further addressed in this section.

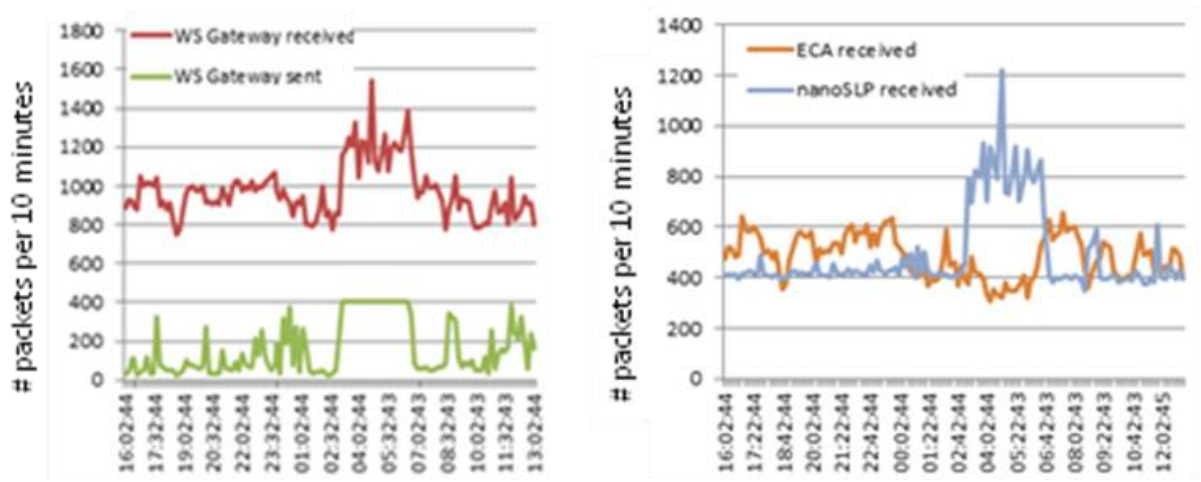


Figure 48: Traffic behaviour at the WASP gateway during medium-scale deployment trials in April 2010. The red line in the left curve shows a sudden increase in packets received by the WASP gateway between 2:30AM and 7:00AM and the right curve clarifies that this relates to a sudden increase in nanoSLP messages which reflects that many nodes are rebooting and, consequently, send out nanoSLP messages at short (5s) intervals to indicate that no services are installed yet. In response, the WASP gateway starts to send service uploads as reflected by the green line in the left curve where to (predefined) maximum upload rate of 400 packets per 10 minutes is reached for the entire period of instability.

One of the main challenges was to properly analyze the problem(-s) while having limited node-level debug options and, moreover, limited access to nodes which most of the time are non-responsive to network traffic and unable to transmit messages. For example, the firmware used in the herd control test bed did not allow the use of the debug code version (to read-out status reports over a serial line) as this exceeds the 48KB flash program memory. In addition, the encountered problems proved not practical to investigate with the simulation tool WSim due to the size of the network, the amount of traffic, the required timing accuracy, and the duration of the simulation on which the medium- and large-scale deployment problems occurred. One of the solutions was to extend the debug and logging options of the traffic actually passing through the WASP software router and received and transmitted by the WASP gateway as already illustrated in Figure 48. This worked well to identify issues but running the SW router in debug mode occasionally triggered undesired reboots of the base-station node that further delayed / complicated automatic recovery of the network from overload condition. Such base-station reboots occurred when the workload of the PC running the SW router, WASP gateway, EIC, and various deployment monitors increased to such a level that the watchdog timer guarding the serial link between base station node and software router could not be kicked in time. Run-time monitoring through the WASP deployment tools has been extended and, especially, the Statistics Monitor has been enabled for analysis of (more and more) node and network performance parameters as depicted in Figure 26. Finally, by enabling remote stress tests through the Statistics Monitor, as indicated in Figure 27, enabled us to reproduce in a consistent manner high-traffic conditions under which nodes crashed in a consistent way which pointed to several implementation issues in the node-level software.

Two implementation bugs in the CC2420 driver code proved critical in the encountered large-scale issues. The first bug was located in the CC2420 initialization code and triggered an improper boot sequence of nodes that only became noticeable under relative high-traffic conditions. Under these conditions, nodes often got locked in some deadlock state during the boot sequence even before the watchdog timer had been activated in the main thread. This

improper reboot behaviour originated in the CC2420 driver initialization code where an interrupt upon reception of incoming packets could be generated whereas interrupt status flags could not yet be properly dealt with in this phase of code initialization. The second bug was located in the CC2420 transmit function where interrupts were disabled until access to the radio channel for transmission has been obtained. Under high traffic conditions, this period proved long enough to trigger a watchdog reset. Effectively, when multiple nodes periodically transmit messages to each other, the clock drift between these nodes cause the CC2420-related interrupts to scan the active firmware of the nodes for race conditions. Eventually, a CC2420-related interrupt can occur at almost every position in the source code. For low network loads in small-scale deployments, it takes a very long time to trigger all race conditions that might exist. For high network loads in medium- and large-scale deployments, race conditions are encountered much faster, but it doesn't make the analysis any easier. Actually a complete code review with respect to interrupt handling, sections with interrupts disabled and pre-emption would be required to avoid potential race conditions together with stress tests to verify the occurrence of race conditions (without actually pin-pointing its location).

Next to fixing the above CC2420 driver bugs, avoiding the occurrence of high-traffic conditions proved beneficial to improve stability. Within WASP operation, such conditions typically occur during (burst) uploads of ECA services and, subsequent, subscriptions. These burst patterns have been reduced by relaxing the time period over which the WASP gateway uploads ECA services and by applying an offset to the periodic execution of ECA services such that synchronization between nodes is avoided. Likewise, to reduce burst high-traffic behaviour of nanoSLP service advertisements, periodically send out by all nodes, a similar offset is applied during the boot sequence of each node.

The laboratory stress tests and large-scale deployment field trials also revealed various issues in the network stack and ECA code with respect to inconsistent use of (dynamic / static) memory allocation and network buffers. To analyse the behaviour of the system with respect to stack usage, the firmware was extended to periodically report the status of all threads with respect to stack usage as follows:

```
Thread    Priority    Stack     Unused      Purpose
0            4         256        52       idle
1            3         256        53       wos_main
2            3         576       205       waspmaster
3            3         192        27       MAC802154
4            3         192        13       nanoslp
5            3         768       298       eca
```

From these reports, it became clear that stack overflows could occur as well. In addition, the sleep timer information indicated that certain periodic tasks drifted relative to each other. For example, a call-back function of a periodic timer would get executed within the scope of the interrupt service routine (ISR). When a call-back function enabled another thread with a flag to dispatch, this thread would get scheduled within the context of the ISR and induced unpredictable effects on the stack content of all threads. When a call-back function enabled another thread without a flag to dispatch, this thread would not get scheduled until the next time slice of the round-robin scheduler even if the system would be idle at that point. Since the round-robin time slice was 20ms, it was not possible to reliably and accurately run applications that require a 100Hz timer, like the ECA runtime environment that performs step detection in the herd control test bed. The solution required a complete review and partial rewrite of the code for the scheduler and how timer interrupts are processed.

The above real-life case example illustrates the complexity of debugging and improving the stability of a distributed system that involves constrained sensor nodes with limited debug

facilities during deployment. Various mechanisms degraded stability in a seemingly non-deterministic manner especially under relative high-traffic conditions. By exploiting the flexibility to define remote monitoring and stress functionality within the WASP system, we've managed to identify and solve the root causes that, unfortunately, were for a large part located in low-level software components from the public domain. This underlines one of our main lessons learned that, when aiming for advanced integrated prototypical implementations, at project start-up a dedicated or subcontracted partner for low-level technology support and/or innovation should be involved for components that are not developed by partners within the project.

4.4.1.2 **Case example of WiseMAC integration**

The low-power (non-beacon) WiseMAC protocol has been integrated on the WASP-OS (WOS) abstraction layer (built upon multi-thread & pre-emptive MantisOS) and the WASP Postmaster framework. However, integration of WiseMAC proved quite challenging and very time-consuming for a number of reasons described in this section.

MantisOS offers basic wireless communication support that combines a persistent carrier sense multiple access (CSMA) medium access control (MAC) mechanism with a minimal radio transmission routine in a single block complemented with an interrupt routine to handle reception of an in-coming packet. However, WiseMAC is based on non-persistent CSMA, to enable efficient duty-cycling of the power-hungry radio transceiver, and its (power, throughput) performance critically depends on the actual temporal behaviour of the WiseMAC implementation. First implementation attempts revealed however that WiseMAC behaved in a widely-varying, unpredictable (temporal) behaviour.

Part of the WiseMAC temporal problems where related to early versions of WASP Postmaster framework being developed to facilitate flexible integration of the network stack. This framework is intended to provide a single-thread & non-pre-emptive execution environment where event-based sequencing is used. Events may come from timers, interrupts from the radio interface (for end of transmission or start/end of reception), letters from protocol layers, and requests from the applications using the network stack. The idea is that each time an event occurs, there is only one layer invoked that performs the necessary operations in the minimum of time to avoid long (busy-wait) periods and possible pre-emption from other protocol layers. Mantis does not offer any "wait for multiple event" blocking primitives and, therefore, it was necessary to create one (based on a software queue and counting semaphores). Also, the generation of proper WiseMAC events required a rewrite of the Mantis radio interface to provide interrupts after successful completion of a frame transmission to enable WiseMAC to immediately prepare the next action. Also, proper "start-of-frame delimiter" (SFD) interrupts at the start of transmission and reception had to be written to support the FTSP time synchronisation protocols.

A more fundamental source of the unpredictable WiseMAC behaviour related to the integration in a multi-thread & pre-emptive execution environment like MantisOS. WiseMAC and similar preamble-based MAC protocols, like TrawMAC, should minimize the timing intervals between preamble frames and have a predictable response to incoming packets. This cannot be achieved within the WASP Postmaster which is just one of the active threads (listed in the previous example case in section 4.4.1.1) that all compete for execution time. We thus had to split the MAC functionality and relocate the time-sensitive part into interrupt routines. The first split concerns the transmit function. WiseMAC uses a preamble of varying size that is built by repeating the radio frame to be transmitted as many times as needed to ensure transmission to its duty-cycled neighbouring receivers. Through this variable preamble, the radio channel is simply kept busy during the entire transmission and blocked for use by neighbouring transmitters. Unfortunately, the packet-oriented hardware design of the CC2420 radio transceiver does not allow the radio chip to remain in transmit mode but it automatically switches back to receive mode at the end of each transmitted packet and, as a consequence,

preamble frame retransmissions have to be re-launched with a potential loss of channel access to neighbouring transmitters. By moving the retransmission of preamble frame copies from WiseMAC to the (transmit section of the) radio driver, timing gap between preamble frames could be minimized at a deterministic interval. In a second split, the treatment of ACKs had to be moved from WiseMAC to the radio driver. Using the postmaster transferring letters from the event frame received from WiseMAC and then having the transmission of the ACK was too slow. When implementing these splits, the generic nature of the interrupt routines and the new radio driver was deliberately preserved to support the portable MAC ambition described in section 3.2.2.3.

Another more fundamental problem arose due to the lack of a driver concept within MantisOS. In regular OSs, user code is not allowed to handle interrupts directly but, instead, interrupts are generated by the system and control is then passed to the user handler code. MantisOS does not do that and treats interrupts similar to executing user threads. As a consequence, invoking system calls from the interrupt routines did cause unexpected timing behaviour that took quite some time to isolate the actual cause. For example, the combination of this and the added "wait for multiple events" functionality was keeping the system within the interrupt routine flow of control much longer than the system clock tick. As a consequence, the system time was drifting whenever a packet was being received. This temporal issue could be solved by adding a dedicated blocking thread to deal with packets received by the radio chip.

A source of the unpredictable WiseMAC behaviour related to handling the reception timeout when using an early version of the WOS timers for which a timing uncertainty was observed that proved detrimental for WiseMAC. It appeared that initialization of the (early version) WOS timers did affect a field in the MantisOS timer structure that should have not been accessed. This caused the loss of all timers after a run time of several hours which took long to debug due to the scarce occurrence of the problem, the accumulation of several other problems, and limited in-depth knowledge of MantisOS. Only when directly using the timers offered by MantisOS, and not those from the WOS abstraction layer, this initialization error could be isolated and solved.

Some of the WiseMAC temporal behaviour problems were traced back to the dynamic allocation of memory resources within the WASP postmaster implementation. These problems have been solved by removing all dynamic allocation of memory resources and use static allocation only.

Other problems were related to the behaviour of the compiler and, particularly, with the original older 3.X versions, before MSPGCC had been ported to GCC 4. (alignment of code in memory, as shown in the Milano testing were an extra byte had to be added in WiseMAC header so that its length is an even number).

The implementation of WiseMAC within the WASP Postmaster framework has highlighted that the use of multi-thread & pre-emptive run-time execution systems like MantisOS is well feasible but requires support of proper primitives (like the “wait for multiple event" blocking primitives), a split of MAC functionality to relocate the time-sensitive part into interrupt routines, and a lot of persistence and partner collaboration to isolate and solve an accumulation of practical and fundamental issues. Given the strong impact of a run-time execution environment on time-critical protocols like WiseMAC, the exercise clearly indicates the need to involve at project start-up a dedicated or subcontracted partner for low-level tech support (such as operating system support) for components that are not developed by partners within the project.

4.4.2 **Back-end lessons**

Besides the lessons related to backend service that have already been described in Section 3.4.3.5, we have identified the following lessons for the back-end part of the WASP system:

- The development of various back-end monitoring tools (like the node, statistics, and topology monitors) that expose run-time WSN-operational details proved extremely valuable in capturing bugs in integrated tests of components (e.g. causing unexpected drop of packets in the network stack) and in actual test bed deployments (e.g. confirming that the HC up-scaling bottleneck described above indeed was traffic dependent).
    - when starting to use these monitoring tools, developers sometimes observed that no data arrived at the back-end application without a clue whether this problem was caused by the nodes or any component in between
    - for this, the generation of log files (of the router, gateway, and monitoring clients) and, subsequent, monitoring applications proved valuable to support remote bug analysis by partners
- Remote access by project partners (across Europe) to the server hosting the actual test bed deployments in Lelystad and London proved very useful to assess & improve stability of both WSN and back-end systems.
    - Enabled expert partners to observe the performance of their own components in integrated setup
    - Demonstrated that the WASP system is functional and allowing for remote uploads of adapted node-level services as well as upgrades of back-end components without involving the test bed (application) partners.
- Various issues (performance, loss of data, …) have been encountered when running multiple back-end clients in parallel, especially when different subscription parameters are used. These issues are due to a bottleneck introduced either in the sensor network or in the communication to the back-end application.
    - To avoid a bottleneck in the sensor network and transmitting duplicate data, it is advised to exploit the WS Gateway upgrade that allows shared subscriptions among different back-end clients with identical subscription parameters.
    - To avoid the second issue, the back-end application developer should be aware that the WS Gateway does not transmit notifications in parallel to subscriptions of a single application (but only to other applications with different host names or port numbers) in order to avoid that the developer of this application has to deal with concurrency. Therefore, the application should not do any complex processing when receiving data. Furthermore, since communication over web services uses a lot of resources compared to communication within a process, the application should avoid duplicate subscriptions and duplicate data rather locally if needed.
- The installation of the EIC on a remote server faced several unexpected (instability and performance) issues that required additional optimization efforts on memory and CPU consumption
    - Issues caused by the intensive use of web-service communications and database functionality when handling incoming data from the WASP gateway, data processing and storage in a local database, and sending notifications to the business application. These issues have been addressed by minimizing the code that is executed when receiving data from the WS Gateway.

4.4.3 **Test bed lessons**

Although originally not planned from July 2008 onwards, bi-annual integration meetings have been organized for both EC and HC test beds, to gradually introduce new components and move from technology testing towards actual scenario testing. Four integration meetings per year proved sufficient to make good progress although, with the core integrators also involved in WP-specific tasks, did provide some timing conflicts. The preparation and the actual

organization of the integration meetings were driven by the test bed coordinators and the project manager and, although each meeting had clear targets, the meeting schedules needed to be flexible to cope with unsuspected issues as those mentioned in the sections above. These issues basically reflected a natural process to mature technologies and align the development environments used by individual partners. Nevertheless, meetings could have been more efficient if

- A single person would have been appointed from project start-up that has the technical skills to run a well-defined basic integration test prior to an upcoming test bed meeting to identify critical bugs in advance. In WASP, we relied on volunteers to do so next to their other test bed commitments which sometimes provided timing conflicts
- We really experienced the gap between the technological developers and the scenario-driven end users. Only few persons could combine both worlds and drive the development from a holistic view. Nevertheless, big steps were made and the target application users (behavioural researchers) and the technical stakeholders worked quite well together.
- The feeling is that prototyping and test beds revealed the real issues to be solved for large scale sensor networks and that some integration time was 'lost' in discussions on best (theoretical) solutions. As an example, for long the option was kept open to use MantisOS or FreeRTOS as operating system for (test bed) integration. At the moment the decision for MantisOS was enforced, further work could be well aligned and integration really started to make progress.
- As argued in the two case examples in Sections 4.4.1.1 and 4.4.1.2, when aiming for advanced integrated prototypical implementations like in WASP, at project start-up a dedicated or subcontracted partner for low-level technology support and/or innovation should be involved for components that are not developed by partners within the project.

For a large and complex IP project like WASP with challenging (test bed) ambitions, it is strongly advisable to appoint a dedicated technical director to drive and keep momentum in the alignment and integration efforts, especially across work packages. In WASP, technical alignment in general and test bed alignment in particular involved very many topics and partners including the architectural team (to prepare alignment proposals), the test bed integration teams (to assess and implement technical alignments), the WP leaders (adjusting WP commitments to integration needs), the test bed coordinators (evaluating achievements and updating plans for next meeting) and the project manager (to keep track of agreed activities and open issues). This shared approach worked out in the end but integration would have been more efficient when a dedicated technical director, able to fully focus on the technical details of all parts of the WASP system, would have been appointed right from the start of the project to keep momentum and maintain full overview of all issues.

5 WASP dissemination and exploitation

Throughout the project run time, WASP results have been actively disseminated in journal publications and during conferences, workshops, and panel discussions across Europe and beyond (from Boston to Seoul). An overview of publications can be found on the "conferences and publications" page of the WASP public website illustrated in Figure 49.

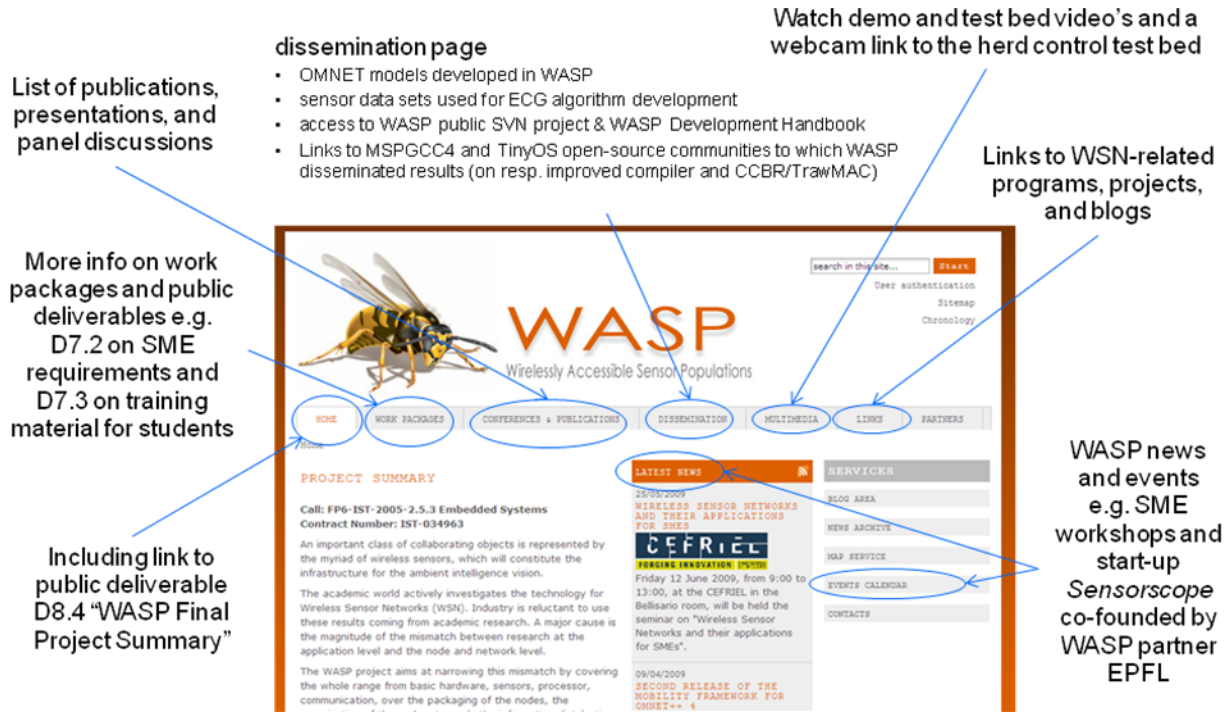


Figure 49: Screenshot of the WASP public website (www.wasp-project.org) actively used for progress and dissemination of results.

As indicated in the figure, lots of information on WASP results can be obtained through the WASP public website.

Besides external dissemination, also internal tutorials have been organized to get WASP partners acquainted especially with the various components and tools integrated in embedded software development environment described in Section 3.2.1.2. A proper alignment of needs and good understanding of consistent use proved essential for the successful integration of technologies developed in a multi-partner (multi-side) project. Material of the internal tutorials has already been used in external (conference and student) tutorials on e.g. the BSN platform, ECA model, and WSim simulator.

In the remainder of this chapter, a wrap-up of some main WASP dissemination (Section 5.1) and exploitation (Section 5.2) highlights is provided and illustrates that various initiatives and communities can benefit from the results achieved by the WASP project.

5.1 WASP dissemination highlights

In this section, highlight will be presented in view of the communities to which they relate.

5.1.1 **Link to livestock community**

Throughout WASP run time, the herd control concept has been frequently discussed with members of the Smart Dairy Farming (SDF) study group of value-chain stakeholders within the Netherlands involving Friesland Campina, CRV, Agrifirm, NOM and several SME's. These discussions contributed to the further plans of the SDF study group and assisted in selection of priorities, like “real-time” cow observations, and definition of follow-up projects. The final herd control test bed scenario and results have also been discussed with this SDF group. Among several application-oriented questions, the steps needed to go from a research tool to industrial farm management information tool were of interest. Here, our thorough analysis of encountered test bed problems and solutions proved very valuable insights to this community.

Beside the discussion with the SDF group, the WASP HC activities were also discussed with the cow group of the EU project BioBusiness (a Marie Curie project) that started in 2010. This cow group consists of the KU Leuven, the ARO Institute of Agriculture Engineering, Wageningen UR Livestock Research and DeLaval as a commercial company and is very interested in the lameness detection of individual cows.

In parallel, WASP insights have been brought forward in the ISO standardization working group (TC23\SC19\WG3) addressing animal identification for e.g. Livestock, Companion animals, Endangered species, Zoo animals, Fishes. Main conclusions are that:

- WASP solutions are flexible enough to adopt the interoperability standards ISO 11784 (numbering system), ISO 24631-6 (specification of communication of animal id information with farm equipment, management systems and other animal applications), ISO 14223-3 (object identifiers, OID). .
- ICAR (International Committee for Animal Recording) has been contacted to take the responsibility as registration authority for OID databases
- The work around pre-defined injection sites of RFID transponders (for read-out & removal during slaughter / food safety) could also be relevant for WASP in case of using implanted or injected WASP nodes☺
- The WASP project is for ISO TC23\SC19\WG3 a strong motivation to continue the OID work. Future input of WASP HC is strongly appreciated.

5.1.2 **Link to Ambient Assisted Living Community and Continua**

WASP has been actively participating in various workshops concerning healthcare and, since early 2010, WASP has become involved in various EU initiatives around Ambient Assisted Living. The outcome of the interaction of WASP partners with the Ambient Assisted Living Open Association (AALOA), that is currently being launched by the FP7 project UniversAAL, is probably most interesting to note here. This interaction resulted in a WASP presentation during the first “*Workshop on AAL Services Platform*” (WASP’10) held in conjunction with the Healthcom2010 conference (June 2010, Lyon) [33]. The presented slides are available on the WASP public home page and, in short, presented our generic WASP system and application-optimization for elderly care scenarios to the audience. By exemplifying the close link between low-power (wearable) sensor operation and proper partitioning of functionality across a distributed end-to-end service system, it complemented quite well the other contributions that addressed more middleware solutions and interoperability. During the panel discussion, interoperability was seen as one of the key enabler for the AAL domain for which active collaboration is envisioned with the Continua Alliance that addresses Health & Wellness, Disease Management, and Aging Independent.

Figure 50 illustrates a slide shown at the workshop to enable a direct comparison between the WASP system architecture and the Continua “topology”. Here, the term “topology” is deliberately used to reflect that the Continua Alliance provides guidelines for interoperability but does not have stringent implementation constraints whereas in WASP the focus was very much on system implementation without stringent interoperability constraints to provide application flexibility. Nevertheless, the figure reflects that the WASP integrated system could

well implement the Continua topology despite the (usual) difference in terminology. For example, the Continua Application Hosting Device could be implemented on a local compute device running the WASP Gateway and some local AAL applications (e.g. activity monitor) whereas the Continua WAN Device could be embodied by a remote server running the WASP backend (EIC, HCA) mediation components and a Healthcare Information System (HIS).

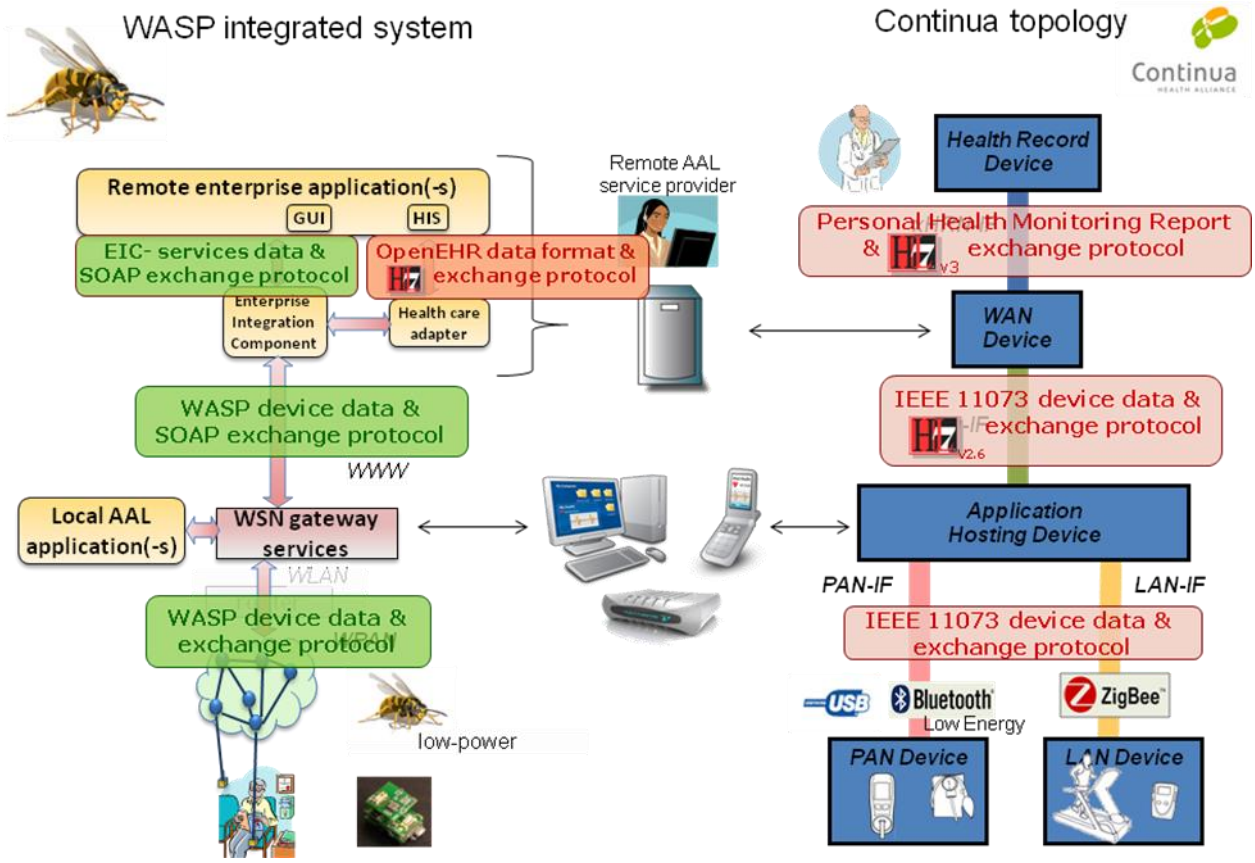


Figure 50 Comparison of the WASP system and Continua topology

Concerning interoperability, Continua promotes on device level the use of IEEE11073 device data specialization (e.g. data models and nomenclature for devices monitoring blood pressure, weight, physical activity, basic ECG, …) and IEEE11073 optimized exchange protocols (e.g. messaging protocol and data format). Between Application Hosting and WAN devices, Continua promotes the continued use of the IEEE 11073 device data specializations but now with the HL7 exchange protocols whereas on the highest level, involving data reporting to medical specialists, use of Personal Health Monitoring Reports is promoted. In principle, the components and communication mechanisms used within the WASP integrated system are flexible enough to implement these Continua guidelines. In practice, however, it will be interesting to see the effect when using the IEEE11073 standards on WASP node and network level that, at present, are tuned for low-power 24/7 wearable sensor devices (instead of the rather bulky spot check devices brought to market nowadays).

One Achilles heel in both the WASP integrated system and the Continua topology is the use of internet as communication backbone towards the Healthcare Information System. As a result, remote services become directly dependent on internet performance which may be fine for non-acute remote (ADL) monitoring services but could cause severe drawbacks when aiming for remote (cardiac, fall, hypothermia) alert services with hard timing guarantees. Such Personal Emergency Response Services (PERS) are recognized by the Continua Alliance to have a dominant market interest. A usual approach to enhance reliability is to build in

redundancy and, since already a GPRS module is foreseen for outdoor monitoring as indicated in Figure 3, integration of such a GPRS module seems a good fit in case of alerts once the internet proves congested. Evidently, the GPRS module integration should be such that the unreliable internet backbone is avoided using the telecommunications backbone to directly contact the remote care provider. Since also the GPRS backbone is not error-prone, the Continua Alliance is assessing options to team up through a PERS use case with the Telecare Services Association to improve (reliability of) alert signalling & voice communication.

In summary, the WASP integrated system is believed to be generic enough to implement the Continua interoperability guidelines, however, future work will be required to assess their applicability for wearable 24/7 non-acute monitoring devices. For acute PERS services, a solution for reliable communication of alerts signals e.g. through GPRS is expected to be essential to move this field ahead.

This lesson learned on the future challenges for the WASP integrated system, tested non-acute and alert AAL scenarios, and link to interoperability will be brought forward also after project closure e.g. in an upcoming panel “Which Features should be in AAL Platforms?" organized during the EuroSSC workshop on Nov 16 (2010) in Passau, Germany and the upcoming M&S meeting on smart buildings / spaces on November 11-12 (2010) in Portugal.

5.1.3 **Link to automotive community**

Results from WASP are expected to be exploited within FIAT Research Centre in present and future research projects, addressing implementation and use of WSN in the automotive domain and possible future design and prototyping of customized solutions for sensor nodes and intra-vehicle wireless networks.

CRF is currently involved in several research activities in the WSN domain, both in internal and co-funded initiatives, all focussed on the replacement of current wired in-car sensors by wireless solutions for a new generation of vehicle Electrical/Electronic (E/E) architecture and in the characterization of new functions. Wireless is seen as the enabling technology for functionalities nowadays not allowed mainly due to cabling, such as brake monitoring system or the introduction of intelligent tyres. In these activities, CRF could benefit of the in depth study conducted in WASP and of the possible cross-layer optimizations within different protocol stack solutions, supporting different interaction models and services. CRF could exploit and strengthen all the source code produced during the project for the implementation of networking protocols, in accordance with the WASP stack architecture routing protocol with the WASP postmaster) and for the simulation of the BAN stack within a indoor scenario with low-mobility constraints.

CRF could also exploit and further develop the simulation environment, the models and trials designed and implemented to assess the applicability and performances of the WASP solution for automotive scenarios, in terms of reliability (e.g. end-to-end latency, packet delivery ratio), power consumption (node lifetime) and overall node activity. Models and simulations results are now available for intra-vehicle networks and wireless sensors deployed inside a tunnel.

Some of the guidelines and specifications resulting from the aforementioned development have been progressively implemented in nodes deployed in a small-scale WSN trial, represented by our ‘FIAT 500 Wireless’. This prototype was conceived as a test bench to evaluate technically the technology impact within the vehicle environment, with special attention to networking issues, the compliance with the actual vehicle E/E architecture and performance measurement. At present, the prototype implements a star topology network fully integrated with the FIAT Blue&Me On-Board telematic platform through a CAN interface. This allows the driver to display the status of the single nodes and to configure the network. Amongst the nodes, we can currently number the *‘Passenger presence sensor’,* the *‘DOOR nodes’* to control the window and the rear view mirror, the *‘STEERING node’* to control the steering wheel buttons, a *‘Mobile environmental sensor’* to sense values of temperature and

humidity in different spots of the vehicle and a *'Biomedical bangle'* monitoring some vital parameters of the driver.

Tools and achievements from WASP could also represent the foundation for novel investigations and applications in the same field.

CRF will continue promoting the WASP initiative within FIAT's ecosystem of partners and suppliers, also after the end of the project, and will share results, especially with small and medium enterprise, in order to stimulate the use and further development of such platform. Future collaborations with industrial and SME partners within WASP, or even outside but based on WASP insights might be of interest for CRF, especially when aimed to real deployment and testing up to pre-engineering.

Although recent research achievements in WSN technologies (both on hardware and software side, as also confirmed by some results of the WASP project) are paving the way for a real deployment of wireless sensor networks in many application domains, different factors are slowing down their adoption in the automotive sphere. Amongst them:

- The low diffusion of the technology in the consumer electronic market (surely not comparable with the penetration of Bluetooth and Wi-Fi) that usually drives and forces the integration of wireless technologies in vehicle's systems;
- Similar to the home automation domain, WSN technology is not imposing itself as many analysts expected; this has consequently damped the interest on possible scenarios of interaction between the vehicle and the personal and home devices and networks based on such technologies;
- in spite of the several studies and experimentation, a real killer application did not yet emerged in the automotive domain for wireless sensor networks. Tire pressure monitoring systems are even today the only real customized integration for wireless sensors inside the vehicle. This can't be anyway considered as a WSN application but instead a very simple point to point connection based on proprietary protocol solutions.
- The lack of automotive certified hardware that complies with the severe requirements imposed for the electronic components deployment, limits and in some cases precludes possible applications inside the vehicle environment;
- The lack of customized and optimized solutions of energy scavenging in many cases ruins the advantages of wireless communication. Wireless is the solution to reduce costs related to wiring (wires are expensive, wires are heavy, wires are restrictive) but, even though different WSN technologies, like WASP, meet low-power requirements, in many applications the need of unfailing sources of energy from the environment is still a constraint and this calls for efficient and reliable scavenging systems;
- The global economic recession and its heavy repercussions on the car market have partially slacken efforts and activities on those research topics that are expected to have spin-offs in the medium-long term;

Despite all the aforementioned obstacles, car makers stand aloof from new achievements. FIAT's interest in the WSN domain is still high and CRF will continue overlooking the evolution of technologies, promoting and supporting all the initiatives, like WASP, that will be aimed to the development and deployment of automotive applications.

The recent incorporation of Chrysler and FIAT Groups will most likely deem necessary a redefinition of E/E architecture and this could offer the opportunity to discuss the introduction of new technologies, efficient and reliable solutions.

The European Union has recently approved a law to make tire pressure monitoring systems mandatory in 2012. This will impose the integration of TPMS on all new models as of 2012 and on all new cars as of 2014 as is already in the US. Analysts expect the market size will jump from $168 millions in 2007 to over $300 millions by 2012. This factor and current technical

challenges motivate a large number of suppliers to actively develop new systems in order to get a share of the market and this would open new opportunities for the adoption in such application of 802.15.4-based technologies.

Current results and future development of WSN technologies, including WASP, could represent the foundation in the development of new concepts. The following step could be for example the integration of the WASP architecture in a hybrid network of smart nodes inside the vehicle, based on heterogeneous technology, where the WSN can act as a second level network, enabling the implementation of several applications related to user profile personalization and functionalities nowadays severely limited by cabling issues. This concept will guide our future research in the domain of WSN for automotive.

5.1.4 **Link to SME community**

The WASP project directly involved Small and Medium Enterprises (SME) during project execution. This was done (1) to find out specific requirements for wireless sensor node applications from an SME perspective and (2) to disseminate WASP results to SME companies and illustrate potential WSN use cases. In total, three workshops for SME representatives have been organized. The first workshop was organized in Darmstadt (Germany) on September 27th, 2007 and introduced potential WSN application scenarios and available technologies to the audience. The outcome of the workshop was – in addition to contact and information exchange – was captured in public deliverable D7.2 that presents and discusses a collection of requirements and notes gathered while discussing with the workshop participants. The requirements are classified into scenario demands and technical needs. Furthermore – and most important for the WASP project – the lacks of up-to-date sensor technologies are mentioned and the expectations towards the WASP projects are stated.

The second and third workshops have been organized on March 25th 2009 in Darmstadt (Germany) and on June 12th 2009 in Milan (Italy). These workshops provided an update of project progress and results achieved at that time. Moreover, the second workshop was co-organized with the (FP7) CONET Network of Excellence and involved two parallel tracks to discuss wider aspects of WSNs (e.g. business models, programming and programming tools, applications, operating systems, standardization and deployment). All workshops were well attended and e.g. the second workshop was joined by more than 50 people from industry and science. Workshop presentations are available on the web and in deliverable D7.3 that also describes some of the main take-away messages. Examples are the need to define WSN business models in detail, to validate the maturity of WSNs by conducting large field trials and user studies, and the need for WSN standardization. Moreover, especially, WSN system integrators and (embedded) software developers currently face a (too) large variety of different operating systems, programming models and languages and/or hardware platforms. These issues have all been addressed within the WASP project and the consortium is confident that our achievements mark a clear improvement within the WSN field.

To properly close the successful SME interaction, this deliverable on “WASP Final Project Summary” will be distributed to the SME workshop attendees together with the link towards the WASP public SVN project and the “WASP Development Handbook”.

5.1.5 **Link to open-source communities**

Various WASP software results have been made available for use through the dissemination page of the WASP public website (see Figure 49).

As described in Section 3.2.1.2, a WASP private SVN project has been used to facilitate multi-partner software development and integration that stores all source and library codes. After closure of the test bed demonstrators, a WASP public SVN project will be released (already created in August 2010) where WASP partners will upload parts that they allow for use by an interested community to further develop and improve WASP achievements for research purposes. Here, the WASP development handbook, WASP Build-System, and WASP Virtual

Machine development environment are available as well to ease pick-up by a wider community. The WASP public SVN project will be announced to the SME community and, as indicated above, already has been communicated with members of the AALOA community.

Besides the WASP public SVN project, MSPGCC4 improvements achieved in WASP have already been transferred to the open-source MSPGCC4 project (sourceforge.net/projects/mspgcc4/). Also, the CCBR / TrawMAC stack has been made available to the TinyOS community and the 6lowPAN implementation has been made available in the .net micro framework open-source initiative.

5.2 WASP exploitation highlights

During WASP run time in March 2009, the start-up company "Sensorscope" (www.sensorscope.ch) was launched that is co-founded by WASP consortium members from **EPFL**. The company provides a WSN solution where the GSM network is used as backbone to study of environmental parameters while minimizing the cost of setting up a complete monitoring system. Here, part of the EPFL technology on GPRS gateways is being used both in the Sensorscope solution and in WASP.

In the scope of its exploitation activities, partner **SAP** is transferring WASP research and technical results to its Public Security Internal Business Unit (IBU). For the SAP Public Security IBU, surveillance of critical infrastructure supported with the integration of Wireless Sensor Networks is of high interest. The main motivation lies in the fact that WSNs enable critical infrastructure responsibles to maintain a real-time view on the occurring situations. It enables then to prevent and predict incidents (e.g. fire, flooding, presence detection). In addition, it supports them with the reduction of the time latency between incident detection and responsible response to those incidents. To that extend, SAP is about (i) to commercialise a Real World Integration Platform (RWIP) for the integration of smart devices within SAP application, and (ii) to use that platform in the Hoffenheim stadium and training center of Hoffenheim football team. RWIP aims at providing connectivity between any devices, including sensor nodes, and business applications. Research results related to the activities described in Section 3.4.3.3 have been transferred to that platform, with a particular focus on the implementation of trust mechanisms. In addition, with respect to the real deployment of sensor nodes, SAP will equip the Hoffenheim stadium and its training center on different business cases,including noise monitoring during events such as concert or football matches. To that purpose, the RWIP platform will serve as integration component of the sensors within an SAP BusinessByDesign system.

As stated in Section 3.1.2.1, following the ReISC architecture design and FPGA evaluations within WASP, partner **STM** internally developed a SoC test chip that is currently being used for business development (in the healthcare, imaging, and industrial markets). Also, WASP progress and achievements have been presented on various occasions to internal product divisions. Interesting to note that the subject of remote health monitoring is developed from an internal group and has brought to a strict collaboration developing a wireless cardiac monitor with a world leader clinic [34] in which some of WASP results, including the ReISC processor, might become part of a receipt to gain positions in this new and challenging market segment. If successful, ReISC products may become commercially available through standard sale channels by 2011.

With the key integration issues solved, partner **CSEM** is now considering proposals to customers to further exploit the potential advantages of multi-threading RTOS systems. Next to this, partner CSEM is convenor of the Working Group on Wireless Communications of the IEC SC65C (industrial communications) committee. This working group (#16) is in charge of establishing wireless communication standards for industrial communications and its first action has been to carry the WirelessHART standard to the level of Draft of an International Standard (IEC 62591). The experience acquired during the WASP project has been very

useful in this work although only minor changes were done to WirelessHART as this was already an industry standard. The operations of WG16 continue and new standards will possibly be created in which the WASP experience will be used.

As sketched in Section 5.1.3, partner **CRF** believes that use of WSN for in-vehicle scenarios will move beyond the use of (wireless) tire pressure monitoring systems that by European law will become mandatory in 2012. Despite an initial interest and discussion, wireless sensors technologies have been kept out of the perimeter addressed by the Car2Car consortium which considers the method to collect and transmit sensor data out-of-scope of. Standardization is considered in all the envisioned application domains as one of the key element for an effective spreading of wireless sensors. Automotive requirements are not currently driving such standardization process but the results coming from projects like WASP (and EMMA, CHOSEN and others) may feed the discussions and provide guidelines towards an effective standardization and, subsequent, exploitation of WSN technologies beyond tire pressure monitoring systems.

At present, partner **PRE** is assessing business interest within the Philips Healthcare Division in the biomedical processor design described in Section 3.1.2.2 where first results for an internally-funded (together with partner IMEC-NL) prototype IC look promising for various ECG-, EEG-, and activity-related healthcare scenarios. Also, wider WASP achievements including the system view are being discussed as a reference example of enterprise application services. Also, the 6lowPAN work is of interest given the potential advantages described in Section 3.3.2.3. Nevertheless, like for automotive, adoption of 6lowPAN technologies will await the evaluation of strong standards to which Philips contributes as described in that section.

Partner **EMIC** was successful in transferring the 6lowpan technology (Section 3.3.2.3) to the Microsoft .NET Micro Framework (NETMF) product. NETMF is an open source project and represents an implementation of the .NET Common Language Runtime to embedded devices with limited resources such as memory and CPU. Representatives of EMIC have been elected as members to the technical core team of NETMF. Based on this success, EMIC is continuing the exploitation of WASP technology along this path. uDSSP represents a natural extension of lower level communication infrastructures and therefore a perfect fit for NETMF.

6 Summary

In this final public deliverable from the WASP project, we have given an overview of the WASP integrated system, components, and tools developed in the period from September 2006 to September 2010. As described in Chapter 2, the development of the WASP integrated system has been guided by generic application requirements as well as more application-specific requirements for three selected domains being elderly care, herd control, and automotive. Also, a leading system requirement has been to derive a generic WSN infrastructure that can be easily deployed and integrated with enterprise information systems and, moreover, optimized to specific application and system needs. To guide the target group for whom WSN adoption and deployment should be facilitated, the definition of stakeholders like end application users, (enterprise) application developers, WSN system integrators, and WSN protocol specialists proved valuable.

The developed WASP integrated system and associated components and tools, described in respectively Chapters 2 and 2.5, have been successfully validated and demonstrated in real-life test bed deployments assessing relevant herd control and elderly care scenarios as described in Chapter 4. Many (integration) lessons learned have been captured throughout the deliverable and an extensive compilation is provided in Section 4.4. Here, also two examples are included to illustrate the complexity and challenges encountered when integrating advanced technologies (like WiseMAC) and moving beyond a first small-scale concept demonstrator to a large-scale research prototype for e.g. herd control. Validation of WASP solutions for automotive have been performed through simulations in order to limit the test bed efforts.

Besides the activities focused on integration, Chapter 2.5 also reports on the various explorative activities that have been undertaken to advance the WSN field e.g. with more advanced hardware components, software platform (generation) components, alternative P/S integration and (6lowPAN) system implementation.

Chapter 5 briefly reflects the active dissemination of WASP progress, results, and achievements in various journal and conference publications and, moreover, discussions in various panels and concertation meetings. Also, close links have been established with the Smart Dairy Farm group (in the Netherlands), the growing Ambient Assisted Living Community within Europe (e.g. through AALOA and universal), and parties within the automotive industry that recognize the potential (e.g. for in-vehicle applications) and challenges (technology and standardization) ahead. It is also encouraging to note that several results from WASP have already been picked up for further exploitation as described in Section 5.2 which, in some cases, reflects the high quality achieved within WASP. Also, several public-funded projects have been accepted that build upon insights and technologies developed within WASP like Sofia (Artemis), SmaCS (Dutch national project), and Vitruvius (Dutch national project).

In summary, we are quite confident the WASP project did make a significant contribution to close the gap between academic research and SME and industry adoption. Many of our approaches are advancing the state-of-the-art like e.g. the uDSSP and ECA programming models that provide the required flexibility to tune and remotely upload services for specific applications and both have been received best paper awards at recognized international conferences (see [31] and [36]). It has been hard work to integrate the generic WASP integrated system with components and tools that ease application development and support actual deployment. Fortunately, the results achieved, like in the successful test bed deployments, have been quite rewarding.

At present, the current WASP integrated system can be viewed as a Research technology prototype which is well suited to assess trade-offs at various levels of a distributed system with tightly-constrained sensor devices and network communication on the one (sensor) hand communicating with virtually non-constrained backend servers and IP communication on the other (backend) hand. The current WASP solution can be used as research tool for animal and

human behaviour scientists (e.g. at ICL and WUR) to enable data acquisition for period of several weeks to study the behaviour of individuals and, thereby, to identify the actual sensors and behavioural trends that make most sense to be remotely monitored. This understanding is essential to quantify benefits and costs and, thereby, to motivate any next steps from WASP-like research technology prototypes towards industrial application prototypes and, subsequent, commercial solutions. To stimulate potential further developments of the WASP results, a WASP public SVN project has been launched were partners can make their components available for use by third parties for research purposes. Also, various parts have been disseminated to existing open-source communities which can be found on the dissemination page of our WASP public website.

References

[1] Kruchten, Architectural Blueprints—The “4+1” View Model of Software Architecture, IEEE Software 12 (6), Nov. 1995, pp42-50

[2] IEEE-std-1471-2000: Recommended Practice for Architectural Description of Software Intensive Systems

[3] Anthony Schoofs and Phillip Stanley-Marbell, “Portability in MAC protocol and transceiver software implementations for LR-WPAN platforms”, Software: Practice and Experience, Wiley, 2010

[2] G. Cugola and M. Migliavacca. “A Context and Content-Based Routing Protocol for Mobile Sensor Networks”. In Proc. 6th European Conf. on Wireless Sensor Networks, February 11th-13th, Cork, Ireland, 2009.

[3] X. Zhang, J. Ansari and P. Mähönen. “Traffic Aware Medium Access Control Protocol for Wireless Sensor Networks”, Proceedings of ACM Mobiwac , Tenerife, Spain, 2009.

[4] M. Bennebroek, J. Ansari, A. Kovacevic, X. Zhang, E. Meshkova and P. Mähönen, "Poster Abstract: Wirelessly Accessible Sensor Populations (WASP) Deployment and enterprise integration of energy efficient wireless sensor networks". Proc. of SECON, 2009, Rome, Italy.

[5] J. Ansari, X. Zhang, C. Cugola, M. Migliavacca, M. Bennebroek and P. Mähönen, "Poster Abstract: Wirelessly Accessible Sensor Populations (WASP) Cross-layer Design of Low Power Medium Access and Content based Routing" Proc. of SECON, 2010, Boston, MA, USA.

[6] Eugster, P. T., Felber, P. A., Guerraoui, R., and Kermarrec, A. 2003. The many faces of publish/subscribe. ACM Comput. Surv. 35, 2 (Jun. 2003), 114-131

[7] Yahya, B. and Ben-Othman, J. 2009. Towards a classification of energy aware MAC protocols for wireless sensor networks.*Wirel. Commun. Mob. Comput.* 9, 12 (Dec. 2009), 1572-1607. DOI= http://dx.doi.org/10.1002/wcm.v9:12

[8] Bachir, A., Heusse, M., Duda, A., and Leung, K. K. 2009. Preamble sampling MAC protocols with persistent receivers in wireless sensor networks. *Trans. Wireless. Comm.* 8, 3 (Mar. 2009), 1091-1095. DOI= http://dx.doi.org/10.1109/TWC.2009.080130

[9] J. Rousselot, J.-D. Decotignie, M. Aoun, P. van der Stok, R. Serna Olivera and G. Fohler, Accurate Tieliness Simulations for Real-Time Wireless Sensor Networks, Proc. of the 3rd UKSim European Symposium on Computer Modeling and Simulation (EMS 2009), 2009, 476-481.

[10] http://mixim.sourceforge.net

[11] Deliverable D4.4a, "Updated version of D4.4 with full assessment of three protocol stacks", WASP consortium, Sept. 2009.

[12] T. Winter and P. Thubert. R*PL: IPv6 Routing Protocol for Low power and Lossy Networks*. Internet draft, version 11, (draft-ietf-roll-rpl-11), 2010.

[13] A. El-Hoiydi, J.-D. Decotignie, “WiseMAC: An Ultra Low Power MAC Protocol for Multi-hop Wireless Sensor Networks”, In Proc. ALGOSENSORS, 2004.

[14] G. Montenegro, N. Kushalnagar, J. Hui, and D. Culler. *Transmission of IPv6 Packets over IEEE 802.15.4 Networks*. RFC 4944, 2007.

[15] Jonathan W. Hui and David E. Culler. IP is Dead, Long Live IP for Wireless Sensor Networks. In *Proc. of the SenSys’08*, pp. 15-28, 2008.

[16] Alessandro Sorniotti, Refik Molva, Laurent Gomez, Christophe Trefois, Annett Laube, Piervito Scaglioso: Efficient Access Control for Wireless Sensor Data. IJWIN 16(3): 165-174 (2009)

[17] R. D. Mol, E. Bleumer, P. Hogewerf, and A. Ipema, “Recording of dairy cow behaviour with wireless accelerometers,” In the proceedings of the Joint International Agricultural Conference, 2009.

[18] WS-BrokeredNotification v1.0 specification, accessible through: http://www.ibm.com/developerworks/library/specification/ws-notification/

[19] HL7 Annotated ECG (aECG): http://wiki.hl7.org/index.php?title=Product_RR_aECG For a detailed description of the aECG format see http://www.amps-llc.com/UsefulDocs/aECG_Implementation_Guide.pdf

[20] Kees Lokhorst, Bart Schanssema, Frans Ettema. 2010. Intelligente Konzepte für die Milchviehhaltung – Sensoren innerhalb und auserhalb des Tieres, In KTBL-Schrift 480 Automatisierung und Roboter in der Landwirtschaft, KTBL-Vortragstagung vom 21. bis 22. April 2010 in Erfurt, pa 117-126.

[21] M. Parker and M. Thorslund, “Health trends in the elderly population: Getting better and getting worse”, The Gerontologist, Vol. 47 (2), pp. 150-158, 2006

[22] C. Ballard et al, “Quality of Life for People with dementia living in residential and nursing home care: The impact of performance on activities of daily living, behavioral and psychological symptoms, language skills, and psychotropic drugs”, International Psychogeriatrics, Vol. 13, No. 1, pp. 93-106, 2001

[23] DR Galasko, D. Bennett, M. Sano et al., “An inventory to assess activities of daily living for clinical trials in Alzheimer’s disease”, Alzheimer Dis Assoc Disord, 11(2):S33-39, 1997

[24] D Langer, F Pitta, T Troosters, C Burtin, M Decramer, R Gosselink, “Quantifying physical activity in COPD: different measures for different purposes.” Thorax 2009;64:458

[25] G. Taylor, “The problem of hypothermia in the elderly”, Practitioner, 193:761, 1964

[26] R.H. Fox, “Body temperature in the elderly: a national study of physiological, social and environmental conditions, Brit. Med. J., 1:200.1973

[27] http://www.orthogate.org/cases/trauma/hypothermia-causing-multi-system-complications-in-elderly-patient-with-femur-neck-fracture.html

[28] M.L. Mallet, “Pathophysiology of accidental hypothermia”, Q J Med, 95:pp. 775-785, 2002

[29] S. Horvath and R. Rochelle, “Hypothermia in the Aged”, Environmental Health Perspective Vol. 20, pp.127-130, 1977

[30] http://www.merck.com/mkgr/mmg/sec8/ch67/ch67b.jsp

[31] M. Ahmed, C. Saraydar, T. ElBatt, J. Yin, T. Talty, and M. Ames, “Intra-vehicular wireless networks,” in *Proc. of the IEEE Globecom Workshops*, November 2007, pp. 1–9.

[32] www.continuaalliance.org

[33] http://wasp2010.isti.cnr.it/ and the WASP contribution by Martijn Bennebroek, Andre Barroso, Louis Atallah, Benny Lo and Guang-Zhong Yang, “Deployment of Wireless Sensors for Remote Elderly Monitoring”, IEEE Healthcom 2010 Conference Proceedings, Lyon, July 2010

[34] http://mobihealthnews.com/5325/mayo-clinic-stm-developing-wireless-cardiac-monitor/

[35] Andreas Lachenmann, Ulrich Müller, Robert Sugar, Louis Latour, Matthias Neugebauer, and Alain Gefflaut, Programming Sensor Networks with State-Centric Services, in *Proc. of the 6th IEEE International Conference on Distributed Computing in Sensor Systems (DCOSS '10)*, 2010.

[36] Bosman, R.P., Lukkien, J.J., Verhoeven, R. "*An integral approach to programming sensor networks; Proceedings 6th Annual IEEE Consumer Communications and Networking Conference*", pp. 1-5, 2009.

[37] Andree, Matthias, Gebel, Alexander, & Karl, Holger. Concept and Prototype for a Real-Time Enabled Publish/Subscribe System. *2010 10th IEEE International Conference on Computer and Information Technology* (pp. 737-742). Bradford, West Yorkshire, England: IEEE Computer Society.