EAI Endorsed Transactions
on Ubiquitous Environments Research Article **ICST.ORG**

Reuse of pervasive system architectures

Dennis J.A. Bijwaard[1,3,*], Berend Jan van der Zwaag[2], Nirvana Meratnia[2], Hylke W. van Dijk[4], Henk Eertink[5], Paul J.M. Havinga[2]

[1]Inertia Technology, Hengelosestraat 583, 7521 AG Enschede, The Netherlands
[2]Pervasive Systems, University of Twente, P.O. Box 217, 7500 AE Enschede, The Netherlands
[3]Ambient Systems, Colosseum 15d, 7521 PV Enschede, The Netherlands
[4]NHL Hogeschool, Postbus 1080, 8900 CB Leeuwarden, The Netherlands
[5]Novay, Capitool 15, 7521 PL Enschede, The Netherlands

Abstract

Developers are often confronted with incompatible systems and lack a proper system abstraction that allows easy integration of various hardware and software components. To try solve these shortcomings, building blocks are identified at different levels of detail in today's pervasive/communication systems and used in a conceptual reasoning framework allowing easy comparison and combination. The generality of the conceptual framework is validated by decomposing a selection of pervasive systems into models of these building blocks and integrating these models to create improved ones. Additionally, the required properties of pervasive systems on scalability, efficiency, degree of pervasiveness, and maintainability are analysed for a number of application areas. The pervasive systems are compared on these properties. Observations are made, and weak points in the analysed pervasive systems are identified. Furthermore, we provide a set of recommendations as a guideline towards flexible architectures that make pervasive systems usable in a variety of applications.

Keywords: pervasive computing; middleware; wireless sensor networks

Received on 18 March 2012, accepted on 19 October 2013, published on 17 November 2014.

doi: 10.4108/ue.1.3.e3

1. Introduction

Historically, networks first supported voice, then other media. Future networks will add data of things, i.e. intelligent context-aware and context-producing devices and services that can communicate peer-to-peer, via centralised or federated services. Pervasive systems are considered important constituents of these networks. They contain a large number of collaborating tiny sensing, actuating, routing, and processing devices.

A great number of pervasive systems have emerged over the last decade, most of which serve dedicated application areas such as environmental monitoring, supply chain monitoring and remote health monitoring [1–4]. Some generic architectures have been developed in research projects that unleash the potential of pervasive systems to multiple applications. Unfortunately, none of these generic architectures have matured into a widely accepted architecture yet.

Our main objective is to provide a mechanism for shared usage of pervasive systems in multiple applications. This is because the effective use of legacy systems in new contexts and integrating them in new systems turns out to be tedious. It is difficult to exploit opportunities of reuse and to identify and implement effective measures for modifications. Recently, technologies like web services and Service Oriented Architecture (SOA) have been adopted in pervasive systems architectures to provide service discovery and service composition. In some cases [5] these technologies have even been used within clusters of sensor nodes in a pervasive system. Although these technologies offer great flexibility, they can jeopardise the efficiency of these systems (see Section 3.6).

In general, a common reasoning framework in which existing pervasive systems architectures can be assessed in multiple dimensions is missing. Such a framework should support: (i) comparison of existing systems; (ii) effective integration of heterogeneous systems; (iii) modification of existing systems; (iv) easy system deployment in new contexts; (iv) identification of and fixing weak spots.

Unified Modelling Language (UML), a well-known and widely used modelling language, does not yet provide a view in which the important properties of pervasive systems can be modelled together in a comprehensive way. Furthermore its precise notation adds a level of detail that is not required for the conceptual framework advantages listed above, and can therefore be distracting.

[*]Corresponding author. dennis@inertia-technology.com

To this end, it is our objective to create a simple conceptual framework for reasoning about pervasive systems, as well as for comparing and integrating such systems. For doing so, it is obviously essential to derive the key entities and principles of pervasive system architectures and their modules. Therefore, we identify generic entities, called *resources*, and interactions between them found at different levels of detail in pervasive systems. The general applicability of these resources is validated by decomposing a number of generic pervasive systems into their *resource compositions* in Section 3. A resource composition gives an overview of the resource interactions at different levels, to which we refer as the *communication view*.

In addition to the communication view of each pervasive system, we analyse the required properties for scalability, efficiency and pervasiveness in different application areas. These properties are also used to compare the analysed pervasive systems. From these comparisons and observations, the disadvantages of these systems become apparent, which helps us to define a guideline towards an overall architecture that diminishes the identified weaknesses. In Section 4, an improved pervasive system is created from existing ones to show the strength of our reasoning framework, and to identify the integration points.

To summarise, the contributions of this paper are threefold: (i) we propose a conceptual framework for pervasive systems (Section 2) and analyse the requrired properties of pervasive systems in different application areas, (ii) we use this framework to decompose and compare a number of existing pervasive system architectures with respect to the required properties (Section 3), (iii) we use this framework to integrate existing pervasive systems by indicating integration points (Section 4). We conclude with directions towards a flexible architecture for shared use of pervasive systems (Section 6).

1.1. Related work

A great number of pervasive architectures are available nowadays. Some focus on context retrieval from sensors and databases like Aura [6], others also add context reasoning and awareness like Gaia [7]. In this paper we add attention to the sensing and actuation in pervasive architectures, such as with a Wireless Sensor and Actuator Network (WSAN), and how these can be used by applications. A recent survey [8] highlights advantages of WSAN middleware: providing system abstraction, shared and deployable functionality, and resource management. As noted in that survey, the main WSAN challenges, i.e. context-awareness and data-centricity, have not yet properly been addressed.

Recently, the pervasive middleware design direction shifted towards web-based and service oriented architectures [9–12]. Another survey [13] addresses existing pervasive architectures and evaluates them in terms of context abstraction level, communication model, reasoning system, extensibility, and reusability. This survey concludes that most of the current architectures are centralised and application-dependent, and suffer from lack of generality and single point of failure. In this paper we analyse more generic pervasive architectures and explicitly model the communication type. We make it explicit where combined systems can interface. Gajjar et al. [1] give design directions for WSANs in different application areas. In this paper we focus on architectures using WSAN and extend their work with architecture requirements per application area.

Our proposed conceptual reasoning framework builds on modelling concepts and model-driven design. UML provides a good starting point for modelling pervasive systems. Unfortunately, it only provides limited support for modelling behavioural patterns like synchronous and asynchronous communication, push and pull. A recent modelling paper [14] proposes to use stereotypes to express these architectural primitives, an idea that we adopt in our framework.

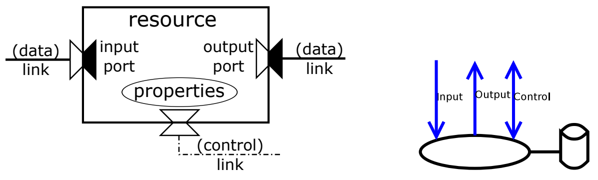
Another UML-based modelling language for pervasive systems is PervML [15]. It is used to visually and platform-independently model a pervasive system from the point of view of both system analysts and system developers. Since PervML targets the generation of Open Services Gateway Initiative (OSGi) code and only two example models are available, its generality to model any kind of pervasive system is not obvious.

2. Conceptual framework and system properties

We propose a flexible conceptual framework that allows: (i) assessment of pervasive systems using a common notation, and (ii) identification of how to incorporate different modules into an existing system. In what follows, we explain the concept and building blocks of our conceptual framework as well as their types and properties, and analyse them in different application areas.

2.1. Conceptual framework

Our conceptual framework builds on the concept of *entities.* An entity is identifiable and consequently it has a set of properties. Some properties of an entity can be good in one context and may be bad in another. The main entities that can be identified in pervasive systems (and communication systems in general) are: *resource*, *port*, and *link.* A basic resource has an input port and output port and transforms input events or streams into output events or streams. Links can carry data or control signals. Data links connect output ports to input



(a) Main Entities (b) Interactions

Figure 1. Main entities and interactions of resource.

ports, whereas control links are used to schedule actions or access the properties of an entity.

Figure 1a illustrates the principal entities of a resource. One may notice that not all details of this model are always equally relevant. Therefore, in the remainder of this paper, we will use a compressed form as shown in Figure 1b. The model implicitly holds properties in a container, which are made explicit whenever appropriate.

Resources may be compound or basic. A compound resource can contain other resources that are connected via links. Figure 2b provides an example of a compound resource. When a compound resource uses other external ports than the ones contained resources require or provide, a transformation is required. A dedicated resource called *transformer* is introduced that takes care of this transformation and makes it more explicit. To distinguish the transformer from other resources, it is drawn as a rectangle instead of an ellipse.

The operation of a compound resource is coordinated by a *manager* , which is a dedicated resource that can manage the configuration of the contained resources, their execution, resource scheduling and maintenance. Note that there can only be one manager per compound resource. In embedded devices, the manager is usually the operating system or scheduler. In bigger software systems, an execution environment, event loop or scheduler are also candidate managers. To distinguish the manager from other resources, it is depicted as a rectangle with rounded edges.

Figure 2a provides an overview of all the resource and link types. It also shows the inheritance relation between the different resource types. Storage is added to denote a resource dedicated to information storage and retrieval. The link types and notation of interaction protocols and interfaces are further detailed below.

In order to make architecture comparisons presented in the following sections easier, we will use our compressed notation and borrow notations like multiplicity and inheritance from UML where appropriate (appropriate notations can come from different types of UML diagrams). Multiplicity is denoted by numbers on either end of the communication link between resources (like UML associations in a class diagram) and specifies how many instances can exist on each side of the relation.

Note that the **communication link** between resources are drawn as in UML sequence diagrams:

- an **open arrow head** denotes an asynchronous message. Asynchronous messages are normally used to push information (including requests). The "pull" stereotype [14] can be added to the link to explicitly specify information pull.
- a **solid arrow head** denotes a synchronous message, the corresponding return message is not drawn. Synchronous messages are mostly used to pull information and can contain information in the request message. The "push" stereotype [14] can be used to explicitly specify information push.

The **interaction *type*** is specified as a label on the communication link between resources, using the following EBNF notation:

$$\begin{aligned}
type &= [M``:"]P[``/"T] \\
M &= (method|message)[``,"(method|message)]* \\
P &= (protocol|interface)[``,"(protocol|interface)]* \\
T &= transport[``,"transport]*
\end{aligned}$$

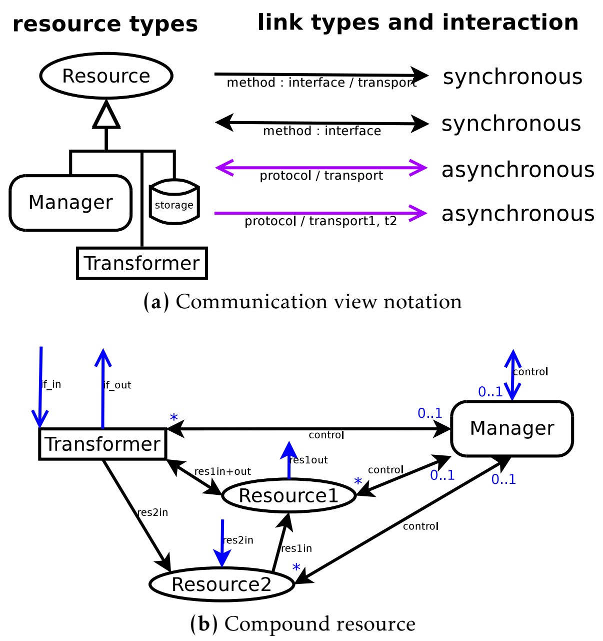
A *method* is usually a synchronous call to a resource according to an *interface* specification. A *message* is usually an asynchronous *protocol* message. *Protocols* and *interfaces* can be used over different *transports*, such as TCP, SSL, HTTP, Simple Object Access Protocol (SOAP) and RS232.

In Figure 2b, the "0..1" on the *Manager* side specifies that *Resource1*, *Resource2* and *Interface* can only have 0 or 1 managers, just like associations in UML class diagrams. The *Manager*, on the other hand, can control any number of *Resource1*, *Resource2* and *Transformer*, denoted by "*" on the connecting arrow. Inheritance is denoted by a directed hollow arrowhead as shown in Figure 2a), just like in UML class diagrams. Stereotypes can be added to the arrow to specify the type of inheritance. From now on we will only explicitly model the multiplicity where this adds value, e.g. when there are many-many or one-many relations. From here onwards, the compressed notation is called the communication view.

Since different groupings of resources into compound resources are possible, a similar grouping should be made to allow for comparison and integration of models. The models in this paper therefore use a grouping that at the highest level identifies components that communicate over TCP/IP, serial lines or via radio links.

2.2. Resource types

Pervasive systems tend to have recurring and hierarchical categories of resources. The main categories include: Sensors, Actuators, Repositories, and Services. Sensors



(**a**) Communication view notation

(**b**) Compound resource

Figure 2. Notation and compound resource example

observe physical signals such as light, temperature and movement, whereas Actuators control these physical signals. Repositories are used to store parameters, services, programmes, and other data. Finally, a Service is a (possible) combination of sensors, actuators, or repositories.

Transformers are often bi-directional and may be in the form of Software–Software, Hardware–Hardware, Hardware–Software, and Human–Machine. A software–software transformer transforms data from one domain to another, for example a software module. Hardware–hardware transformers are merely physical channels, whereas hardware–software transformers can for instance be a transition between analogue continuous domain to digital quantised domain; sensors and actuators fall in this category. A human-machine example is a Graphical User Interface (GUI) that shows a graphical representation on a screen or captures user actions from input devices like a mouse and a keyboard.

2.3. Properties of pervasive systems

To date, pervasive systems have been used in various contexts and still have the potential to be deployed in many more. These systems present different properties and attributes, which make them suitable for specific application areas or contexts. Based on our own experience and literature study (e.g., [14, 16–18]), we identify the following groups of properties (scalability, efficiency, pervasiveness, and maintainability) that characterize pervasive systems:

Scalability. Scalability of a system is high when it can grow while still being manageable and responsive. Scalability is low otherwise. For scalability we define the following properties and parameters (inspired by [16] amongst others):

- **Resource binding**: Explicit binding between applications and resources is discouraged, since it requires a WSAN node to signal and process on a per application basis. This is related to the notion of localised scalability [16]. The binding can be *tightly* coupled, *loosely* coupled, or *uncoupled*. A looser binding enhances scalability.

- **Node configuration**: Examples of WSAN resource configuration are setting sampling and reporting periods, and alarm thresholds. The most static case is that it requires *reprogramming* of the node. *Online* configuration using an open protocol is considered better. The most scalable is *controlled* configuration, in which authentication and authorisation are checked and concurrent requests are managed. This is related to the notion of Effective Use of Smart Spaces [16]. When possible, also the typical frequency of re-configurations is shown.

- **IP Reachability**: IP Reachability refers to being able to work through firewall and private network boundaries (and associated Network Address Translation (NAT) traversal) to reach a WSAN (e.g. for remote node configuration or actuation). When components are not reachable via IP, additional measures are required per installation (for instance adding firewall rules, or STUN) to make the system work across the Internet, which makes the system less scalable. When possible, also the typical frequency of remote maintenance is shown.

- **IP Mobility**: IP Mobility controls the changes of network attachment of devices and their contained applications, services, and context sources. Mobility is an integral part of pervasive systems [16]. Scalability increases when devices can remain connected while moving around or reconnect without much deployment or communication overhead. When possible, also the typical number of mobility changes is specified.

- **WSAN Mobility**: WSAN Mobility refers to changes of attachment of sensor and actuator nodes within and across WSANs. Scalability increases when nodes can move freely within the WSAN coverage area while still being reachable and able to send measurements. Scalability increases even more when the WSAN supports multiple hops and when nodes can also move between WSANs. When possible, also the typical number of mobility changes is specified.

- **#Domains**: Estimated number of supported peering domains or clusters. Peering enables applications to transparently use resources from another domain or cluster, and increases scalability.
- **#WSANs/domain**: Estimated number of WSANs supported per domain or cluster (or in the overall system when peering is not supported).
- **#Nodes/WSAN**: Estimated number of nodes supported, per WSAN when applicable, else in the overall system.
- **#Apps/node**: Supported number of concurrent applications using sensor information from one or more nodes. This is typically 1 for dedicated pervasive systems. Increasing this number may limit the efficiency of the interactions (see below).

Efficiency. Efficiency of a system increases when latency and bandwidth overhead in communication decrease. With respect to efficiency we define the following properties (inspired by [8, 18] amongst others):

- **Application type**: The application type can be *generic* or *dedicated* and can run close to the WSAN (*local*) or elsewhere. A dedicated pervasive application can be fine-tuned to be more efficient than a generic pervasive system that has support for many different applications running in parallel. When possible also the typical number of applications is shown per application type.
- **Dependencies**: Systems may be stand-alone or depend on another component, like *WSAN*, *WAN/LAN* connectivity, and applications (*apps*). A stand-alone system can be more efficient since it does not need to interact.
- **Interaction complexity**: The order of complexity of the interactions is defined in terms of number and size of messaging required. Lower interaction complexity leads to improved efficiency.
- **WSAN link**: This refers to the interaction protocol that is used among WSAN nodes (including the gateway). Messages are sent either with or without a response (acknowledgement) message. Bidirectional asynchronous messaging is the preferred type of interaction, since it allows reaching dormant WSAN nodes statelessly and thus reduces the overhead in WSAN nodes. When undetermined, *proprietary*, *de facto* or *open* is used to indicate the typical level of standardization.
- **Application link**: This refers to the interaction protocol between the application and the WSAN or device. Asynchronous messaging is a natural extension of WSAN messaging, whereas synchronous messaging, like request/response in web services, involves additional logic (processing of responses) and associated energy consumption and bandwidth. When undetermined, *proprietary*, *de facto*, or *open* is used to indicate the typical level of standardization.
- **WSAN multicast**: This refers to the ability to efficiently send messages to a group or to all nodes in a WSAN. Link layer broadcast and/or multicast support for messages towards WSAN nodes makes communication in WSANs much more efficient in terms of energy consumption and bandwidth. When undetermined, the *beneficial* effect of multicast can be indicated.

Pervasiveness. Availability of the following properties make the system more pervasive [13, 17] (note that mobility is already covered under scalability):

- **Sensing**: Specifies whether sensing is supported.
- **Actuation**: Specifies support for actuation. Actuation may also be controlled (authenticated, authorised and management of multiple requests).
- **Reasoning**: Reasoning refers to aggregation, merging of sensor data or context. Reasoning can be done through algorithms, programs, rules or inferencing in different parts of the system.

Maintainability. Maintainability properties relate to the changeability of components, and the maturity/stability of the system and its components:

- **Changeability**: Specifies the minimum level of interchangeability and stability of hardware, ranging from prototype, specific, various (i.e. a mix of specific and standardized) to standardized hardware.
- **Maturity**: A pervasive system can be a model, a simulation, a prototype, or a commercially available product.

2.4. Typical application area requirements

This section describes the typical needs of different application areas with respect to pervasive systems, using the properties described in Section 2.3. The following application areas are considered [4]:

- **Cool chain logistics**: In the cool chain market, it is important to optimise the quality of perishable products by ensuring optimal storage and transport conditions. In addition, assets can be tracked when they enter or leave certain areas.

- **Environmental/habitat monitoring**: In this application area, monitoring is done in the environment or the habitat of living beings. The monitoring is usually for extended periods where user-intervention is either expensive or disturbing. Data mules are sometimes used to collect the sensor information when no wireless coverage is available. In habitat monitoring also the animals themselves can wear a sensor node.
- **Surveillance**: Building, vehicle and infrastructure monitoring to detect forcefully opened or unlocked doors/windows, theft and damage.
- **Smart spaces**: Smart spaces adapt to the needs of the users that enter and leave. They typically contain sensors and actuators that can be monitored and controlled by applications running in the environment and on user devices.
- **Remote eHealth**: In remote eHealth, sensor networks consist of few wireless sensor nodes on or around a living being's body. Typically, these nodes are integrated with a smart phone or a stationary device at home. Monitoring vital signs, and tracking are the main objectives of these sensor networks. Analysis is often done offline but increasingly becomes real-time.

Analysing the different application areas. Note that the scoring of requirements on the different properties is an educated guess based on a number of papers [1–4], since the use of sensor networks in most application areas is still in the early stages of deployment. The typical items used in the application areas are shown in Table 1. The typical application area requirements are depicted in Table 2. The specified values for scalability, efficiency, pervasiveness and maintainability properties depict the minimum workable option in each application area. For scalability, four new parameters have been added for a WSAN. The *IP transfer period* indicates how often sensor updates are made available to applications. The *Sampling period* indicates how often WSAN nodes take measurements. The *WSAN IP messages per second* equals *Number of nodes per WSAN / (60 * Sampling period)*, multiplied by the *Number of WSANs per domain* this gives the *Domain IP messages per second*.

3. Generic pervasive systems

This section describes a number of generic pervasive system architectures from research projects. It uses the conceptual framework and the properties from Section 2 to decompose (using a similar grouping) and compare them. Other popular pervasive architectures like SOCAM [9], Gaia [7] and Aura [6] could be decomposed in a similar manner and scored against the properties using available literature.

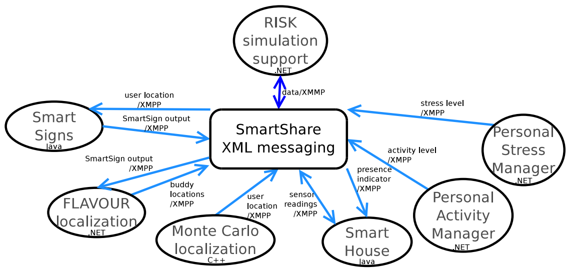


Figure 3. Smart Surroundings communication view

3.1. Smart Surroundings

One of the main goals of the Smart Surroundings project [19] was to support many diverse ubiquitous and context-aware applications concurrently and remain open towards unforeseen use cases. The architecture design aimed at enabling execution of many software components developed in different languages and running on different platforms across distributed and embedded devices and technology platforms. Its integration technique has two steps, i.e., (i) defining an eXtensible Markup Language (XML)-like message exchange interface on top of TCP/IP, and (ii) designing a publish-subscribe mechanism on top of Extensible Messaging and Presence Protocol (XMPP). Result of this project is an integrated demonstrator composed of multiple heterogeneous prototypes and systems developed within the project that exchange their real-time data. Figure 3 illustrates the communication view of Smart Surroundings.

3.2. Hydra

The goal of the Hydra [20] project is to develop a middleware that enables any device to be detectable and usable from Hydra applications and to develop tools for solution providers of ambient intelligent applications using such devices. A complementary goal is developing tools for device producers to enable their devices to be part of an ambient intelligence environment.

The Hydra architecture is divided into middleware[1] for applications and devices. This middleware has three stacked layers: network, service, semantic, and one cross-layer for security. The middleware uses web services based on SOA and the semantic web. The service and semantic layers make devices available as services that advertise their capabilities and properties using a device ontology. Applications can then search and use these device services remotely and context-aware applications and workflows can be composed of application and device services.

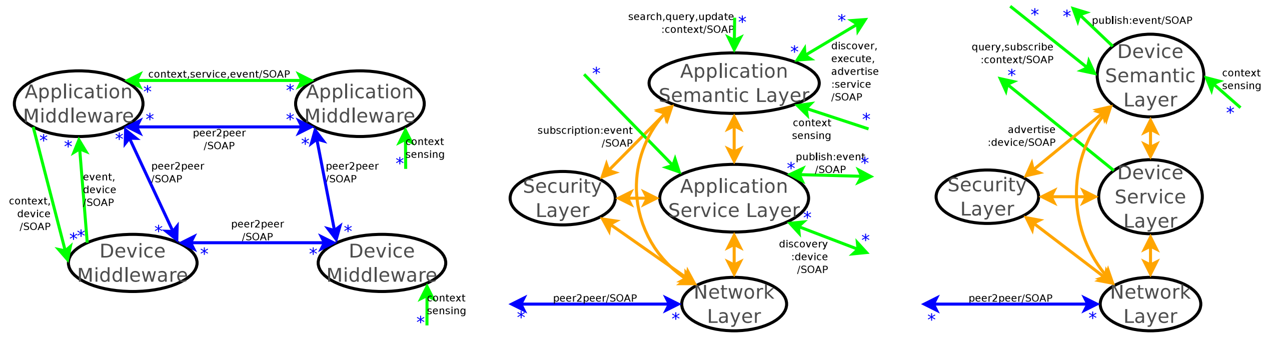
[1]Hydra middleware has been renamed to LinkSmart middleware.

Table 1. Typical associations in different application areas

Application area /association	Cool chain logistics	Environmental monitoring	Surveillance	Smart spaces	Remote eHealth
Mobile entity	truck, node	data mule, node	vehicles	user-device, object	smartphone
domains	depot, warehouse	geographic area	building, infrastructure	place	clinic
WSANs	areas, trucks	sub-areas	vehicles, areas, different types	different types	patients
Nodes	roll container	animal, object	door, window	object	object, user device
Apps	views, triggers	views, triggers	views, triggers	experiences	views, feedback

Table 2. Typical requirements for different application areas

	Framework /property	Cool chain logistics	Environmental monitoring	Surveillance	Smart spaces	Remote eHealth
Scalability	Resource binding	loosely	none	loosely	loosely	loosely
	Node config	monthly	monthly	yearly	per minute	weekly
	IP Mobility	#reconnects	#reconnects	#reconnects	#reconnects	#reconnects
	WSAN Mobility	#movements	#movements	#movements	#movements	#movements
	IP Reachability	monthly	monthly	hourly	per minute	weekly
	IP Transf period	per minute	per day	per minute	per minute	per hour
	#domains	$\leqslant$100	$\leqslant$1000	$\leqslant$100	$\leqslant 10^6$	$\leqslant$1000
	#WSANs/domain	$\leqslant$1000	$\leqslant$1000	$\leqslant$10	$\leqslant$10	$\leqslant$10000
	#Nodes/WSAN	$\leqslant$2000	$\leqslant$10000	$\leqslant$100	$\leqslant$100	$\leqslant$10
	#Apps/node	$\leqslant$10	$\leqslant$100	$\leqslant$10	$\leqslant$100	$\leqslant$10
	Sampling period	10 minutes	30 minutes	1 minute	1 minute	10 minutes
	WSAN IP mesg/s	$\leqslant$3.333	$\leqslant$5.556	$\leqslant$1.667	$\leqslant$1.667	$\leqslant$0.01667
	Domain IP mesg/s	$\leqslant$3333	$\leqslant$5556	$\leqslant$16.67	$\leqslant$16.67	$\leqslant$166.7
	Overal	medium-high	high	low-medium	medium	medium
Efficiency	Application type	dedicated, optional local	$\leqslant$10 dedicated	dedicated local, optional central	$\leqslant$100 generic	dedicated, optional local
	Dependencies	all	all	all	all	all
	Inter.complexity	low-medium	low	medium	medium	medium
	Application link	proprietary	standard	proprietary	standard	standard
	WSAN link	proprietary	proprietary	proprietary	proprietary	proprietary
	WSAN multicast	benefits	benefits	no	benefits	no
	Overall	medium/high	high	medium	medium	low/medium
Pervasive	Sensing	+	+	+	+	+
	Actuation	–	–	+/–	+	+
	Reasoning	–	+/–	+	+	+
	Overall	low	medium	high	high	high
Maintain.	Changeability	standardized	various	specific	various	specific
	Maturity	product	prototype	product	prototype	product
	Overall	high	low	medium	low	medium



(**a**) Hydra interactions (**b**) Application middleware (**c**) Device middleware

Figure 4. Hydra communication view

Additionally, applications can subscribe to context or other middleware events related to a specific topic, independently from the application or device that generates them. The communication view in Figure 4 depicts how application and device middleware resources in Hydra interact.

3.3. Daidalos

The goal of Daidalos [21] was to enable seamless pervasive access to content and services via heterogeneous networks that support user preferences and context. Daidalos demonstrated its integrated concepts in December 2008, in which a prototype from the UbiSec&Sens [22] project was used as sensor network.

The Daidalos architecture enables federation between multiple domains; each domain hosts a Daidalos platform that consists of service provisioning and pervasive service support. The corresponding communication view is depicted in Figure 5a.

Context information can have various forms in Daidalos, i.e., it can be from a Network Monitoring Entity (NME) in the Mobile Terminal or in one of the routers, from the Session Initiation Protocol (SIP) user agent (MMSP-UA) or sensors in the terminal, it can also be sensor information from a device in the Access Network. Additionally, a complete WSAN can be attached to one of the Access Routers.

Context Managers (CMs) in both Mobile Terminal, Access and Core Network provide a means to stream context information from all context sources to a distributed database. Also higher-level context information can be stored or inferred. In these Context Managers, context is stored using ontologies. Applications can query and obtain context information from any of the Context Managers when they are permitted based on their credentials.

3.4. Ambient integration middleware

Recently Ambient middleware [2] was developed to enable easy integration with applications and to enable remote monitoring and maintenance. The communication view in Figure 6a shows the interaction between the different components. Note that AmbientStudio, the ConnectBox and Interconnect all share the same Ambient middleware (AmbientMW).

Multiple WSAN Gateways (GW) can be connected via RS232 using the AmbientMW in a ConnectBox device or AmbientStudio on a PC. This makes GWs, MicroRouters (MR) and SmartPoints (SP) remotely available.

The AmbientMW offers the ConnectAPI to ease integration with third-party applications using asynchronous XML messages over a TCP/IP connection (optionally encrypted with Secure Socket Layer (SSL)).

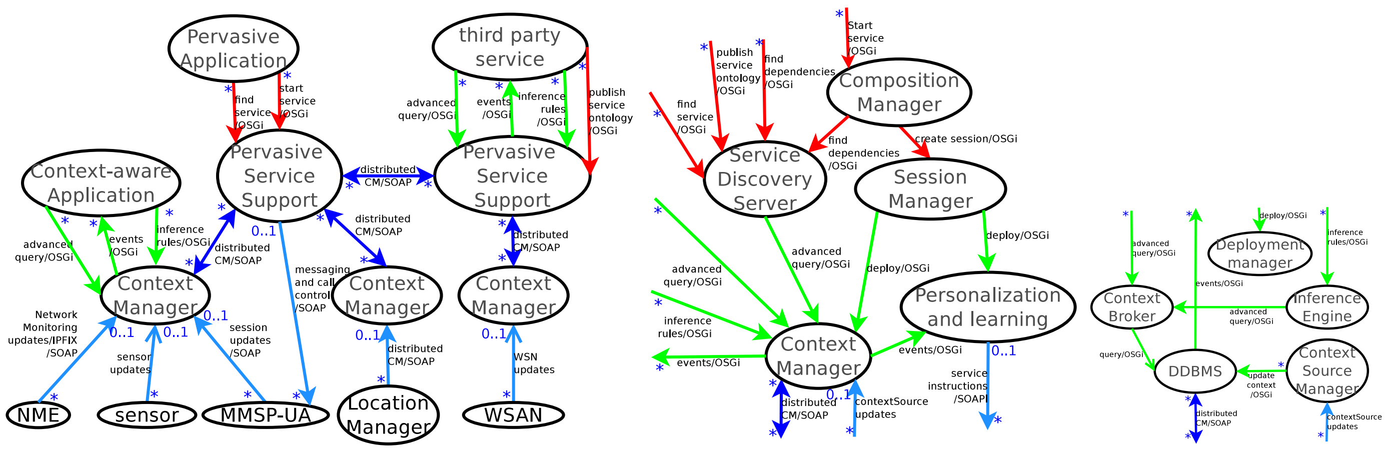
The AmbientMW also offers AmbiLink to ease remote monitoring and maintenance of sensor networks using compact asynchronous binary messages over optionally SSL encrypted TCP/IP connections.

The SmartView web application enables monitoring temperature, humidity and/or shelf-life of static location or cargo with an attached SmartPoint. The SmartView Bridge uses the ConnectAPI to receive sensor measurements from AmbientStudio or ConnectBoxes and forwards the samples to SmartView.

3.5. SENSEI

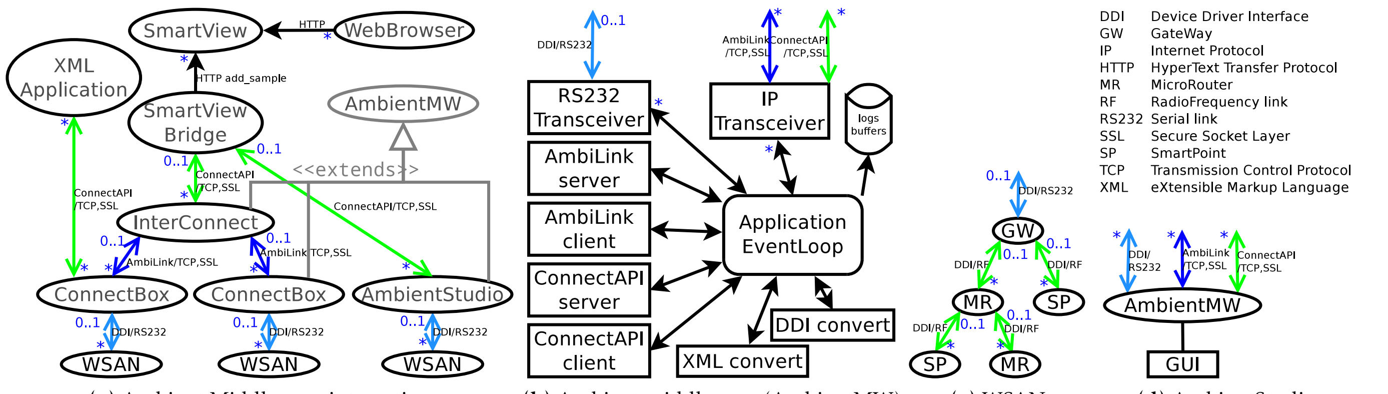
The Integrated European project SENSEI [5] developed a common framework to make WSANs available to services and applications. SENSEI can handle sensor, actuator, and context resources and is able to create compound resources from basic ones. It also provides a common interface for integrating existing sensor platforms.

SENSEI uses a Representational State Transfer (REST)-style architecture over HTTP, which allows



(a) Daidalos interactions **(b)** Pervasive Service Support **(c)** ContextManager

Figure 5. Daidalos context related communication view



DDI	Device Driver Interface
GW	GateWay
IP	Internet Protocol
HTTP	HyperText Transfer Protocol
MR	MicroRouter
RF	RadioFrequency link
RS232	Serial link
SSL	Secure Socket Layer
SP	SmartPoint
TCP	Transmission Control Protocol
XML	eXtensible Markup Language

(a) Ambient Middleware interaction **(b)** Ambient middleware (AmbientMW) **(c)** WSAN **(d)** AmbientStudio

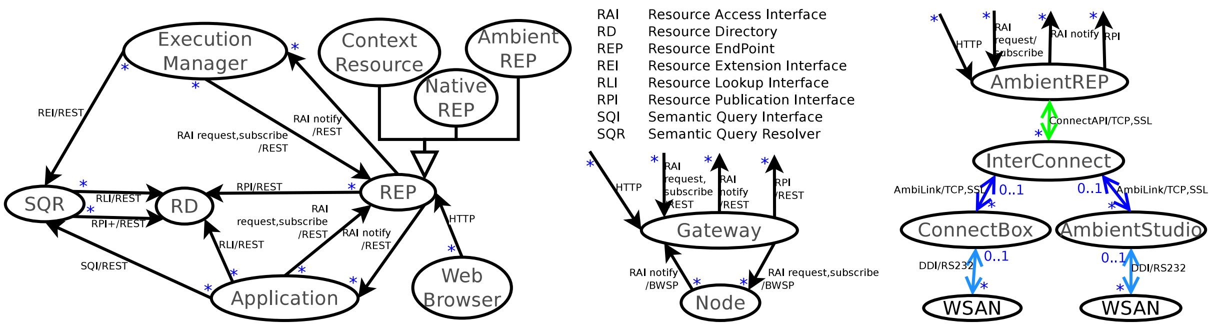
Figure 6. Ambient Middleware communication view

sending requests and returning a response. Requests and responses can be simple text, but also complete XML documents. Resources are identified with a URL, which can be used to request the resource information, or to post a command.

The interaction between the different SENSEI components is depicted in Figure 7a. The project created prototypes for most components and has demonstrated their integration.

Native SENSEI. In a native SENSEI sensor network, a similar REST-style communication is used between the SENSEI Gateway and the Nodes in the sensor network using IPv6 over Low power Wireless Personal Access Networks (6LoWPAN). The SENSEI Gateway translates REST requests/responses to/from the Binary Web Service Protocol (BWSP), which is used inside the sensor network. BWSP uses User Datagram Protocol (UDP) as transport and can optionally disable generation of response messages from the gateway. XML messages are greatly reduced in size using Efficient XML Interchange [23] (EXI). As a future option, the SENSEI Gateway may also mediate when multiple subscriptions are done for the same Node resource. The interaction between the Native SENSEI Gateway and Node is depicted in Figure 7b.

AmbientREP. The Ambient Resource EndPoint (AmbientREP) connects Ambient sensor networks with the SENSEI framework via the AmbientMW (see Section 3.4). So far, temperature, humidity, battery level, and location have been integrated in the AmbientREP using a subset of the wide variety of Device Driver Interface (DDI) drivers available on SmartPoints, MicroRouters, and Gateway. Also LEDs can be turned on the MicroRouters and Gateway. The interaction between the different components is depicted in Figure 7c.



(a) SENSEI interactions (b) Native REP (c) Ambient REP

Figure 7. SENSEI communication view

Since the underlying Ambient network provides asynchronous updates of sensors, the AmbientREP simply remembers the latest measurement and requests a new sample when no sample is available yet. For more frequent sensor updates, an application can subscribe for sensor updates and will be notified when a matching DDI message is received by AmbientREP.

3.6. Reflection and observations

This section reflects on how the generic pervasive systems perform on the properties defined in Section 2.3. Furthermore it describes the observations that can be made when comparing these systems.

Daidalos. Daidalos is very scalable, both horizontally between domains and vertically between different access technologies, networks, and third party application providers. Daidalos treats WSANs as just another context source and supports distributing this context via a chain of context managers within and across domains. It does not yet consider actuators in the WSAN, although it does send feedback to network management entities. With respect to reasoning, there is context inference from low-level context to more complex notions describing the environment. Daidalos requires creation of a Context Source Manager to send sensor/context to a Context Manager which allows various WSANs to be integrated. An application can both subscribe to context events and request specific context. The order of interaction complexity of the Daidalos system is low, as sensor information is just fed into the context managers without delay. Interested applications will then get an (optionally inferred) context update. There is no direct coupling between context producers and context consumers and configuration/management of WSANs is not yet considered.

Smart Surroundings. Application mobility is supported in the sense that application can run anywhere and on any type of platform as long as it can connect to the Smart Surroundings framework. Since XMPP supports a chain of servers between clients to exchange messages over several hops, private networks with NAT and firewalls can be easily traversed. Simple asynchronous messaging in Smart Surroundings through XMPP offers low interaction complexity. However there is a tight binding between context producers and applications using the context. In addition parallel execution of applications is not supported. Therefore the scalability is low. Reasoning occurs at the application level and is not supported as a feature of the systems itself.

Hydra. Subscribing for types of notifications is powerful and unbinds producers from consumers. The distribution of events in Hydra is done at the application layer and therefore multiplies the involved SOAP messages. Hydra supports most kinds of federation, but does not yet consider sharing of WSAN data between peers. Hydra therefore offers medium scalability; no evaluation results have been found with multiple applications that use a variety of devices concurrently, and WSANs are not yet considered. The order of interaction complexity of Hydra is high, each application and each device needs to publish itself as a web service, all communication between applications and devices is based on SOAP, which adds considerable processing and bandwidth overhead.

Ambient middleware. There is no binding between context producers (sensor resources) and context consumers (applications). The mobility support depends on whether the client or server role is used. In principle, an AmbiLink server supports mobility of ConnectBoxes and associated WSANs, while the ConnectAPI client and server offer mobility towards applications. Since a great number of Ambient WSANs can be merged with AmbiLink, made available to multiple applications, and managed remotely, the scalability is medium. The order of interaction complexity is low, as each sensor update goes asynchronously to application(s) without much delay. The communication via AmbiLink adds small latency, since there is an additional hop between application and WSAN, and improves scalability. The communication via ConnectAPI adds some latency by converting to XML and improves flexibility for connecting with applications.

Native SENSEI. Since each sensor update needs either arequest to fetch it from the sensor resource or a subscription to the specific sensor resource (i.e. the sensor node), the interaction complexity is high. Context producers and consumers are therefore tightly coupled. The execution manager makes this binding less strong by working on behalf of applications. Mobility is covered across WSANs, this means that moving a node between WSANs does not update subscribers with the new location (e.g. for updating/stopping the subscription). Reachability can be a concern for SENSEI when parts are behind firewalls or in private networks. On the positive side, native SENSEI is able to connect to a number of WSANs and be used by multiple applications and a number of framework components can peer across domains (most notably the resource directory). All things considered, the scalability of native SENSEI is low/medium.

AmbientREP. The order of interaction complexity of AmbientREP is medium, since the requests and subscriptions for sensor updates are handled within the AmbientREP and do not reach the WSAN nodes. Therefore the context producers and consumers are loosely coupled. The scalability of AmbientREP is medium, i.e. a multitude of Ambient WSANs can feed into one AmbientREP and they become available to multiple SENSEI applications, and its resources are locatable across domains via the peering Resource Directory. Other properties are inherited from Ambient middleware and SENSEI.

Comparison and observations. The generic pervasive systems architectures comparison are listed in Table 3 for scalability, efficiency, pervasiveness and maintainability properties. Comparing these systems yield the following observations:

- **Explicit resource binding**: Two pervasive systems, i.e., Smart Surroundings and native SENSEI, make an explicit **binding between applications and sensor** resources. Such a tight binding is not so bad for small systems, but as the system grows and sensors are used by multiple applications this will impact energy consumption of sensor nodes and consumed bandwidth in both the WSAN and the IP network, and thus hinders scalability.

- **Web services for applications**: Protocols like REST and SOAP offer great flexibility at the application level. Choosing one over the other usually depends on bandwidth and processing constraints, and protocols that are already in use. Requests and subscriptions require reachability of the WSAN that normally sends a stream of measurements. This can be impractical when that WSAN is within a private networks or behind a restrictive firewall, e.g. when the WSAN is in a vehicle. Therefore, a mobile web-enabled WSAN usually polls for reconfiguration and actuation and needs a server for subscriptions and requests.

- **Web services within WSANs**: Energy and bandwidth limitations in the WSAN make a poor match with the added complexity of web services. Precious bandwidth and energy is used to handle acknowledgements for sensor measurements and to notify multiple subscribed receivers. Moreover, latency is added by per-message connection set-ups. Protocols like BWSP help reduce the web services overhead in the WSAN and make the communication asynchronous. However, combined with its high interaction complexity, the efficiency of Native SENSEI is still barely enough to support the considered application areas.

- **Instant messaging for applications**: XMPP and SIP for Instant Messaging and Presence Leveraging Extensions (SIMPLE) are standardised and open candidates for conveying **presence information** (which can be any context) across the Internet. However, efficient one-to-many messaging still poses a great scalability problem for both. On the other hand, distributed Context Managers in Daidalos, Eventing in Hydra, and Subscription in SENSEI offer more tailored messaging, driven by context inference or subscriptions. Note that the size of XMPP and SIMPLE messages are quite high, which makes their usage in energy and bandwidth constrained WSAN nodes impractical.

- **Bandwidth limitations**: The uplink from a WSAN towards the application can have limited (and often costly) bandwidth, for instance a General Packet Radio Service (GPRS) link has typical upload between 9 and 18 kilobit/s (download upto 52 kilobit/s), which severely limits the size and amount of messages that can be sent. GPRS may therefore be too limited to use instant messaging and web protocols with bigger WSANs [2]. Of course a lot can be gained by compression of aggregated messages, this would be practical for messages that are not time-critical. The communication view helps to make the protocol type and the number of messages explicit using the multiplicity on links from sender to receiver. This makes it easier to estimate the required bandwidth in each communication direction.

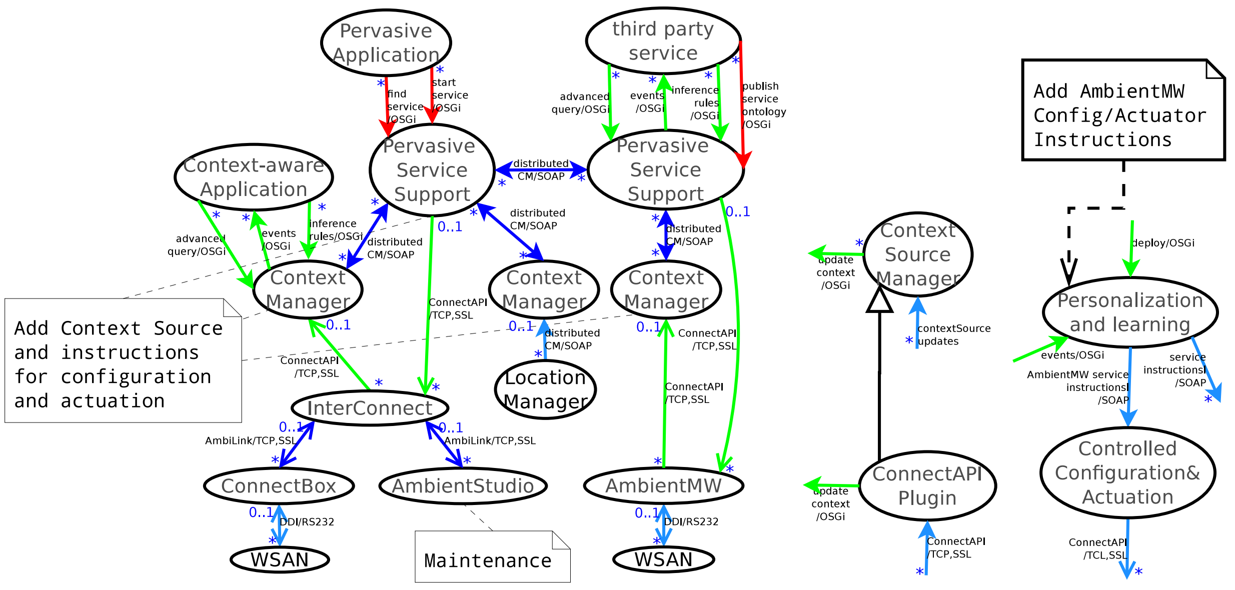
Table 3. Comparison of generic pervasive system architectures

	Framework /property	Smart Surround.	Hydra	Daidalos	Ambient MW	Native SENSEI	Ambient REP
Scalability	Resource binding	tightly	loosely	none	none	tightly	loosely
	Node config	–	–	–	+	+/–	+ (AmbientMW)
	IP Mobility	+	+	++	+/–	+/–	+/–
	WSAN Mobility	n.a.	n.a.	external	++	++	++
	IP Reachability	overlay	overlay	MobileIPv6	overlay	issues	issues
	#Domains	n.a.	n.a.	$\leqslant$10	n.a.	$\leqslant$10	$\leqslant$10
	#WSANs/domain	n.a.	n.a.	$\leqslant$10	$\leqslant$10	$\leqslant$10	$\leqslant$10
	#Nodes/WSAN	$\leqslant$10	$\leqslant$10	external	$\leqslant$2000	$\leqslant$100	$\leqslant$2000
	#Apps/node	$\leqslant$10	$\leqslant$10	$\leqslant$10	$\leqslant$100	$\leqslant$10	$\leqslant$100
	Overall	low	medium	med./high	medium	low/med.	medium
Efficiency	Application type	generic	generic	generic	generic	generic	generic
	Dependencies	none	WSAN	WSAN,appl.	WSAN,appl.	appl.	appl,AmbientMW
	Inter.complexity	low	medium	low	low	high	medium
	Application link	XMPP	SOAP	OSGi	DDI/XML	REST/XML	REST/XML
	WSAN link	–	–	various	DDI	BWSP	DDI
	WSAN multicast	–	–	–	+	–	+
	Overall	medium	medium	high	high	low/med.	medium
Pervasive	Sensing	+	+	+	+	+	+
	Actuation	+/–	+/–	–	+/–	+	+/–
	Reasoning	none	rules	inference	none	rules	rules
	Overall	low	medium	medium	medium	med./high	med./high
Maintain.	Changeability	various	various	various	Ambient	various	Ambient
	Maturity	prototype	prototype	prototype	product	prototype	prototype
	Overall	low	low	low	medium	low	low

- **Asynchronous communication**: Only Ambient middleware explicitly uses asynchronous communication between the WSAN, other middleware instances, and applications. On the application link, both OSGi, ConnectAPI, and XMPP provide asynchronous communication, which aids the system scalability and reduces latency since message providers do not need to wait for responses, the provider resources are not necessary tied to its consumers, and messages can more easily be sent to multiple consumers without impacting the producer. Content-based routing [24] could improve efficiency with multiple interested applications.

- **Quality of information**: The importance of accurate information and reliable message transfer differs per application area. Asynchronous protocols could offer the flexibility to only sent acknowledgements for messages that require it, or sent one acknowledgement for a group of messages. With REST, retries are sent when an acknowledgement does not arrive while the message may have arrived. Compared to REST, SOAP offers additional security features, atomic transactions, and reliable messaging. The stronger reliability is usually only necessary for online purchases, business processes and surveillance. They may however also become important for remote configuration and actuation.

- **Shared control and reasoning**: Although a number of pervasive systems support **remote actuation and configuration** of WSAN nodes, only SENSEI started defining specific modules that resolve issues when multiple parties are controlling/configuring the same sensor/actuator. Additionally, most pervasive systems lack proper authentication and authority checks to make sure that WSAN configuration and actuation are only invocable with proper credentials. Only SENSEI defines a security mechanism to support fine-grained authentication, authorisation, and accounting. So, only the analysed SENSEI-based systems offer the required pervasiveness properties for surveillance, smart spaces, health



(**a**) Ambient/Daidalos interactions (**b**) Context source (**c**) Shared control

Figure 8. Combined Daidalos/Ambient communication view

and well-being. Smart surroundings and Ambient MW lack the required reasoning for environmental monitoring, but an outlier-detection module [25] may prove enough to satisfy this requirement.

- **Application area scalability**: The generic pervasive systems with tight resource binding are clearly not scalable enough, and only Ambient middleware supports remote configureability without reachability issues but lacks reasoning. Therefore, it makes sense to combine the strengths of pervasive systems (see Section 4).
- **Mobility of WSANs and their nodes**: Multiple WSANs should be able to coexist in the same area [26]. For sharing sensor information in different applications, compact asynchronous messaging has the best efficiency properties [26].
- **Maintainability**: All analysed generic pervasive systems offer a sufficient platform for deployment in environmental monitoring and smart spaces. Only the Ambient MW provides a sufficient platform for deployment in all considered application areas, except for the required standardization of the hardware platform.

4. Combining pervasive systems

This section combines pervasive systems into new ones with combined strength. For integrating multiple systems it is important that these systems are modelled on the same level of abstraction at the point where interaction between the systems is expected. This means that components involved in this interaction and their candidate links must be explicitly modelled. This will make it easier to analyse if and where transformations need to be done between the systems.

The combination of different pervasive systems is done at a conceptual level. For actual integration, the details of protocols and interfaces may need to be modelled as well. One example, AmbientREP in Section 3.5 illustrates a combination of SENSEI with AmbientMW that has been prototyped as well.

4.1. Combining Daidalos and Ambient middleware

When combining Daidalos with Ambient middleware, a Context Source Manager instantiation is needed (see Figure 5c) to convert ConnectAPI messages to the applicable context type, and let the ContextManager store and/or inference from them. A way to combine these systems is shown in Figure 8. Since Daidalos does not support sensor network configuration, AmbientStudio can be used to maintain them as depicted in the figure. However, it would be better to add functionality for controlled and authorised configuration and actuation to the Personalisation and Learning module of the Pervasive Service Platform using the virtual identity mechanisms of Daidalos and let it expose an interface for actuation and configuration. Therefore, a controlled configuration and actuation component (see Figure 8c) is added to Pervasive Service Support (see Figure 5b), in order to control service interaction for multiple users and at the same time and translate the service instructions from SOAP to the ConnectAPI protocol.

Since Ambient middleware focuses mostly on remote configuration of and sharing information from mobile WSANs, it is complementary to the Daidalos scalability and efficiency properties. With respect to pervasiveness, Daidalos adds context inferencing and Ambient middleware adds limited support for actuation. The combined system allows thousands of nodes per WSAN but not enough WSANs per domain to support all the application areas. This is mainly limited by the context manager. In order to support upto 5556 sensor readings per second per domain (see Table 2), the scalability can be increased by:

- Distributing the WSANs over multiple context managers. But this will increase operational costs.
- Using node configuration and reasoning within the WSAN to send only meaningful events [3, 27].
- Using context inferencing in the Daidalos context manager to derive meaningful events for applications, such as temperature variations during transport, while (off)loading and during storage.

5. Future work

To further increase the usefulness of the conceptual framework, it can be enhanced in the following ways:

- Decompose and compare other popular pervasive architectures with the conceptual framework such as SOCAM [9], Gaia [7] and Aura [6]. In principle, the conceptual framework could be applied to communication architectures in general.
- Enable embedding of rules in each modelled resource that describe the interaction behaviour of the resource. These rules could for instance be modelled as UML interaction diagrams of the resource and its direct peers.
- Enable further specification of interaction links between resources, such as typical message size, bit-rate and round-trip time.
- Enable execution of model instances such that quantitative measurements can be obtained for different architecture deployments, i.e. where the multiplicity of components is fixed.

6. Conclusion

We introduced a conceptual framework with flexible building blocks for composing pervasive systems. These building blocks provide an effective way to compare different pervasive systems and modules. We exemplified the use of this conceptual framework by decomposing various pervasive systems. The mapping turned out to be straightforward for the selected pervasive systems. We also showed how easily models can be combined into an integrated model and identified where integration work is required.

Over the last years, progress has been made in a number of areas with respect to the properties of pervasive systems. Dedicated pervasive systems show excellent properties for their task. The structured approach of our conceptual framework helps identify and evaluate these properties. Further, it clearly indicates opportunities for integration and as such exploits individual qualities of existing systems in merged ones.

Despite the good progress in different application areas, there is not yet a candidate architecture that prevails on all properties. The identified weaknesses in the compared architectures are:

- Synchronous messaging is not very efficient for WSANs. An intermediate component can be used to distribute a stream of asynchronous messages from WSANs to multiple applications (as a web service or by messaging protocols like XMPP or SIMPLE) without having the energy and bandwidth impact on the WSANs and its uplink.
- A number of pervasive systems lack reasoning support, only few support remote WSAN configuration, and most lack support for shared but controlled actuation and configuration.
- Message reduction in size and number is an effective technique to limit resource utilisation. Unfortunately only a few systems have native support for these techniques, which is essential for the scalability of WSANs and its remote usage.
- Most of the pervasive systems are still in prototype stage or dedicated to a specific task. They are often limited to specific sensor data and context. Interoperability between different WSANs is usually not possible.
- Binding sensor resources to applications hinders scalability, and should be avoided.

Progress on these weaknesses and interoperability between pervasive system architectures will enable efficient and scalable architectures for shared use of pervasive systems in multiple applications.

Acknowledgement. This paper describes work in part undertaken in the context of the SENSEI project, “Integrating the Physical with the Digital World of the Network of the Future” (sensei-project.eu), the ILAND project, “mIddLewAre for deterministic dynamically reconfigurable NetworkeD embedded systems” (iland-artemis.org), and the Sen Safety project, “Heterogeneous Sensor Networks for Safety and Security in Public Places” (sensafety.nl).

References

[1] Gajjar, S., Pradhan, S. and Dasgupta, K. (2011) Wireless sensor network: Application led research perspective. In *Recent Advances in Intelligent Computational Systems (RAICS), 2011 IEEE*: 025 –030. doi:10.1109/RAICS.2011.6069266.

[2] Bijwaard, D.J.A., Kleunen, W.A.P., Havinga, P.J.M., Kleiboer, L. and Bijl, M.J.J. (2011) Industry: Using dynamic WSNs in smart logistics for fruits and pharmacy. In *Proceedings of SenSys'11, November 1-4, 2011, Seattle, WA, USA.*, SenSys '11 (New York, NY, USA: ACM): 218–231.

[3] Avci, A., Bosch, S., Marin-Perianu, M., Marin-Perianu, R.S. and Havinga, P.J.M. (2010) Activity recognition using inertial sensing for healthcare, wellbeing and sports applications: A survey. In *23th International Conference on Architecture of Computing Systems, ARCS 2010, Hannover, Germany* (Berlin: VDE Verlag): 167–176.

[4] Meratnia, N., van der Zwaag, B.J., van Dijk, H.W., Bijwaard, D.J.A. and Havinga, P.J.M. (2010) Sensor networks in the low lands. *Sensors* **10**(9): 8504–8525.

[5] Sensei (Last visited Jan 2011), Integrating the physical with the digital world of the network of the future, `www.sensei-project.eu`.

[6] Garlan, D., Siewiorek, D.P. and Steenkiste, P. (2002) Project aura: Toward distraction-free pervasive computing. *IEEE Pervasive Computing* **1**: 22–31. URL `http://citeseerx.ist.psu.edu/viewdoc/summary?doi=?doi=10.1.1.15.7105`.

[7] Román, M., Hess, C., Cerqueira, R., Ranganathan, A., Campbell, R.H. and Nahrstedt, K. (2002) Gaia: a middleware platform for active spaces. *SIGMOBILE Mob. Comput. Commun. Rev.* **6**(4): 65–67. doi:10.1145/643550.643558, URL `http://dx.doi.org/10.1145/643550.643558`.

[8] Wang, M.M., Cao, J.N., Li, J. and Das, S.K. (2008) Middleware for wireless sensor networks: A survey. *Journal of Computer Science and Technology* **23**(3): 305–326.

[9] Gu, T., Pung, H.K., Zhang, D.Q. and Wang, X.H. (2004) A middleware for building context-aware mobile services. In *In Proceedings of IEEE Vehicular Technology Conference (VTC)*. URL `http://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.4.540`.

[10] Kansal, A., Nath, S., Liu, J. and Zhao, F. (2007) Senseweb: An infrastructure for shared sensing. *IEEE MultiMedia* **14**: 8–13.

[11] Pastrone, C., Spirito, M., Tomasi, R. and Rizzo, F. (2008) A jabber-based management framework for heterogeneous sensor network applications. *International Journal of Software Engineering and Its Applications* **2**(3): 9–24.

[12] Khedo, K. and Subramanian, R. (2009) A service-oriented component-based middleware architecture for wireless sensor networks. *IJCSNS International Journal of Computer Science and Network Security* **9**(3): 174–182.

[13] Miraoui, M., Tadj, C. and Amar, C.B. (2008) Architectural survey of context-aware systems in pervasive computing environment. *Ubiquitous Computing and Communication Journal* **3**(3).

[14] Kamal, A. and Avgeriou, P. (2008) Modeling architectural patterns' behavior using architectural primitives. In Morrison, R., Balasubramaniam, D. and Falkner, K. [eds.] *Software Architecture* (Springer Berlin / Heidelberg), *Lecture Notes in Computer Science* **5292**, 164–179. doi:10.1007/978-3-540-88030-1_13.

[15] Javier Munoz, Vicente Pelechano, C.C. (2006) Implementing a pervasive meetings room: A model driven approach. In *Proccedings of the International Workshop on Ubiquitous Computing (IWUC 2006)*: 13–20.

[16] Satyanarayanan, M. (2001) Pervasive computing: vision and challenges. *Personal Communications, IEEE* **8**(4): 10 – 17. doi:10.1109/98.943998.

[17] Kolos-Mazuryk, L., Poulisse, G.J. and van Eck, P.A.T. (2005) Requirements engineering for pervasive services. In *Second Workshop on Building Software for Pervasive Computing. Position Papers., San Diego, California, USA*: 18–22.

[18] Wang, Y., Liu, X. and Yin, J. (2006) Requirements of quality of service in wireless sensor network. In *Networking, International Conference on Systems and International Conference on Mobile Communications and Learning Technologies.*: 116. doi:10.1109/ICNICONSMCL.2006.185.

[19] Smart Surroundings (Last visited April 2010), Future ambient systems, `www.Smart-Surroundings.org`.

[20] Hydra (Last visited May 2010), Middleware for networked embedded systems, `www.hydramiddleware.eu`.

[21] Aguiar, R.L., Sarma, A., Bijwaard, D., Marchetti, L., Pacyna, P. and Pascotto, R. (2007) Pervasiveness in a competitive multi-operator environment: the Daidalos project. *IEEE Communications Magazine* : 22–27.

[22] UbiSec&Sens (2006), Ubiquitous sensing and security in the European homeland, `www.ist-ubisecsens.org`.

[23] W3C (Last visited Jan 2011), Efficient xml interchange(exi) primer, `www.w3.org/TR/exi-primer`.

[24] Banavar, G., Chandra, T., Mukherjee, B., Nagarajarao, J., Strom, R. and Sturman, D. (1999) An efficient multicast protocol for content-based publish-subscribe systems. In *Distributed Computing Systems, 1999. Proceedings. 19th IEEE International Conference on*: 262 –272. doi:10.1109/ICDCS.1999.776528.

[25] Zhang, Y., Meratnia, N. and Havinga, P. (2009) Adaptive and online one-class support vector machine-based outlier detection techniques for wireless sensor networks. In *Proceedings of the IEEE 23rd International Conference on Advanced Information Networking and Applications Workshops* (Bradford, United Kingdom: IEEE Computer Society Press): 990–995.

[26] Bijwaard, D.J.A., Havinga, P.J.M. and Eertink, E.H. (2011) Analysis of mobility and sharing of WSNs by IP applications. *International Journal of Distributed Sensor Networks* **2012**: 923594.

[27] Zhang, Y., Meratnia, N. and Havinga, P.J.M. (2010) Outlier detection techniques for wireless sensor networks: A survey. *IEEE Communications Surveys & Tutorials* **12**(2): 159–170.