Bringing pervasive embedded networks to the service cloud: A lightweight middleware approach

Iván Corredor , José F. Martínez, Miguel S. Familiar

A B S T R A C T

The emergence of novel pervasive networks that consist of tiny embedded nodes have reduced the gap between real and virtual worlds. This paradigm has opened the Service Cloud to a variety of wireless devices especially those with sensorial and actuating capabilities. Those pervasive networks contribute to build new context-aware applications that interpret the state of the physical world at real-time. However, traditional Service-Oriented Architectures (SOA), which are widely used in the current Internet are unsuitable for such resource-constraint devices since they are too heavy. In this research paper, an internetworking approach is proposed in order to address that important issue. The main part of our proposal is the Knowledge-Aware and Service-Oriented (KASO) Middleware that has been designed for pervasive embedded networks. KASO Middleware implements a diversity of mechanisms, services and protocols which enable developers and business processing designers to deploy, expose, discover, compose, and orchestrate real-world services (i.e. services running on sensor/actuator devices). Moreover, KASO Middleware implements endpoints to offer those services to the Cloud in a REST manner. Our internetworking approach has been validated through a real healthcare telemonitoring system deployed in a sanatorium. The validation tests show that KASO Middleware successfully brings pervasive embedded networks to the Service Cloud.

1. Introduction

1.1. Motivation

During the last two decades, the Ubiquitous Computing philosophy has been used in a number of network designs. Such a concept was introduced by Mark Weiser [1]. Recently, early idea about Ubiquitous Computing is being broadly applied in the form of pervasive networks because of two major trends in embedded devices. First, hardware components are becoming cheaper, more integrated and powerful due to advances in nanotechnology. According to the Internet of Things paradigm [2], MEMS (micro-electro-mechanical) technologies could provide tiny sensor nodes for communication and computation capabilities that will be able to interact and cooperate with their surrounding environment. Secondly, many software companies have become interested in addressing traditional problems through service-oriented technologies. The Internet of Service paradigm [3] assumes that simple to complex computational processes can be accessed in a highly distributed fashion through standardized interfaces. Typically, services-oriented technologies focused on creating architectures to provide services using the Service Cloud, which are made up powerful elements, e.g. company networks, racks, data base servers, etc. However, in the past few years we have been facing a new trend in which the service-oriented systems cross the border between physical and virtual world, providing great expectancy over real-world aware applications. The performance of these kinds of applications heavily depends on an efficient collaboration of heterogeneous, pervasive and networked embedded devices among themselves and with business systems [4].

Wireless Sensor and Actuator Networks (WSANs) is a basic pillar in the provision of *real-world* services. These kinds of networks are made up of a number of embedded devices using sensors and actuators. WSANs are able to work autonomously in carrying out activities as monitoring physical parameters (e.g. temperature, vibrations, sound, movement, or gases [5]).

Trends indicate that some critical challenges have to be solved in the Future Internet, mainly to deal with a much more varied infrastructure, which will need a number of service interactions. In such manner, new approaches for managing and using mash-ups of services will have to be proposed. The baselines of those proposals have to be designed according to a cross-layer vision in which collaborations in two different planes will be performed; first, "horizontal" interaction among devices with no human

involvement, and second, "vertical" interaction between devices and external entities, i.e. applications or services running on other networks and systems. The latter will be able to directly access functionalities offered by underlying devices without intervention of proprietary drivers, or through gateway wrapping approaches that hide their functionalities. Middleware architectures over the embedded platforms are necessary in order to optimize "vertical" communications and facilitate tools to deploy services involving "horizontal" interactions.

In the Future Internet, real-world services provided by embedded networks will be one of the key challenges. Embedded nodes will be able to offer those services by using the most broadly used standards in Service Oriented Computing (SoC) domain: SOAP-based Web Services or RESTful APIs [6]. The resource costs (in terms of memory, CPU and bandwidth) necessary to support current implementations of Web Services and RESTful architectures can be done by conventional devices. However, those requirements are not feasible for embedded networks made up of tiny resource-constrained nodes. In that sense, the major challenge is to reach similar capabilities in order to be suitable for those tiny nodes in a cost effective way.

The early Wireless Sensor Networks were deployed for military objectives such as vehicle tracking in battlefield. Nowadays, WSANs are deployed in civil applications: energy harvesting, logistics, security and healthcare, are common applications. The latter will be a relevant topic to consider in the coming decades. In 2020 and beyond, demand in healthcare will increase because of aging. Studies performed by The World Health Organization indicate that at least 1 billion of the world population will be 60 years and over in 2025 and 80% will be residing in developed countries [8]. According to other reports about the market of WSN [9], from 2005 to 2011, an increase of $4.1 billion will be spent on systems and services based on WSN. Consequently, development and deployment in real-world services over embedded devices have become a relevant research topic and a promising business opportunity.

1.2. Contributions of this research

According to our experience in developing and deploying real-world services over WSANs [10], a set of requirements to design a whole SOA-based architecture for pervasive embedded networks can be provided:

(a) *Reduced Service Overhead:* Devices offering real-world services usually have very limited resources, so typical service-oriented solutions can generate an overload. Those devices have to implement lightweight mechanisms and protocols to work according to an optimized service-oriented paradigm.

(b) *Reduced cost from discovery mechanisms:* Nodes have to be able to expose their services in a specific repository by using discovery mechanisms. The discovery process has to comply the "plug and play" paradigm, which means human intervention will not be needed or reduced to minimum during this process. Devices should provide minimum information when registering their services, and are expected to provide more details in information if necessary.

(c) *Resource-aware service orchestration:* Context-aware applications are potential consumers of real-world services. Those types of applications need complex services involving other simple services. Dynamic and efficient allocation of resources has to be performed in order to orchestrate complex real-world services over pervasive embedded networks. In such manner, mechanisms that explore best combination of resources have to be designed.

(d) *Simple programming paradigm for rapid prototyping:* The pervasive nature of networks offering real-world services can hinder the development and deployment of business logic over the network nodes. Simple and flexible programming tools have to be provided to developers in order to allow fast prototyping of complex scenarios based on sensor and actuator networks.

The work presented in this research paper is partially based on previous existing works that dealt with adapting classical service approaches to embedded systems [11,12]. The main contribution of our research is the design, development and evaluation of a middleware platform that enables Service-Oriented Computing in pervasive embedded networks by providing lightweight protocols and mechanisms making both "horizontal" and "vertical" interactions easier (see Fig. 1). In such manner, a Knowledge-Aware and Service-Oriented Middleware (KASO Middleware) for pervasive embedded networks is proposed, especially for those managing sensors and actuators like WSANs. This innovative architecture complies with the requirements explained above as follows (*requirements from a to d*): Every node offers its functionalities by means of a service paradigm which reduces service overload (*requirement a*) regarding typical service-oriented solutions; for this aim, two paradigms for service dispatching are provided: on-demand and event-based. In order to enable service discovery with minimal resource cost (*requirement b*) a Micro Inter-Knowledge Protocol (μIKP) that allows providing service exposure and discovery mechanisms to agents running over KASO Middleware has been designed. Provision of contextual service through combinations of simple services has been addressed by means of a distributed, dynamic and resource-aware orchestration engine (*requirement c*). Developers are provided with a simple programming environment (*requirement d*) based on an agent pattern. The development and deployment of new agents is provided by means of well-specified development models. Finally, an information model describing every system element as well as their relationship is specified. This information model is mapped over an ontology, which allows formalizing a representation of the real-world services and other resources offered by the pervasive embedded network.

The rest of the paper is organized as follows: Section 2 discusses related work and background emphasizing current drawbacks to be solved. Section 3 defines the foundation of our SOA-based architecture for embedded networks. Section 4 describes the conceptual model of the KASO Middleware architecture. Section 5 shows the development model of KASO Middleware through examples. Section 6 describes the validation results of the overall architecture over a real healthcare scenario. The results of this research and some possible future work are pointed out in Section 7.

2. Related work and background

Several efforts have been done in order to achieve the requirements described in the previous section. The diversity of transactions related to real-world service has increased due to the introduction of novel context-aware applications that require more complex contextual information from pervasive embedded networks. This way, and taking into account the limited resources of pervasive embedded networks, a number of reasoning mechanisms and protocols have been proposed in recent years. Usually, those solutions have been addressed in two approaches: (a) in-network data processing; (b) data processing using a middleware layer interceding in communication between the pervasive network and business applications.

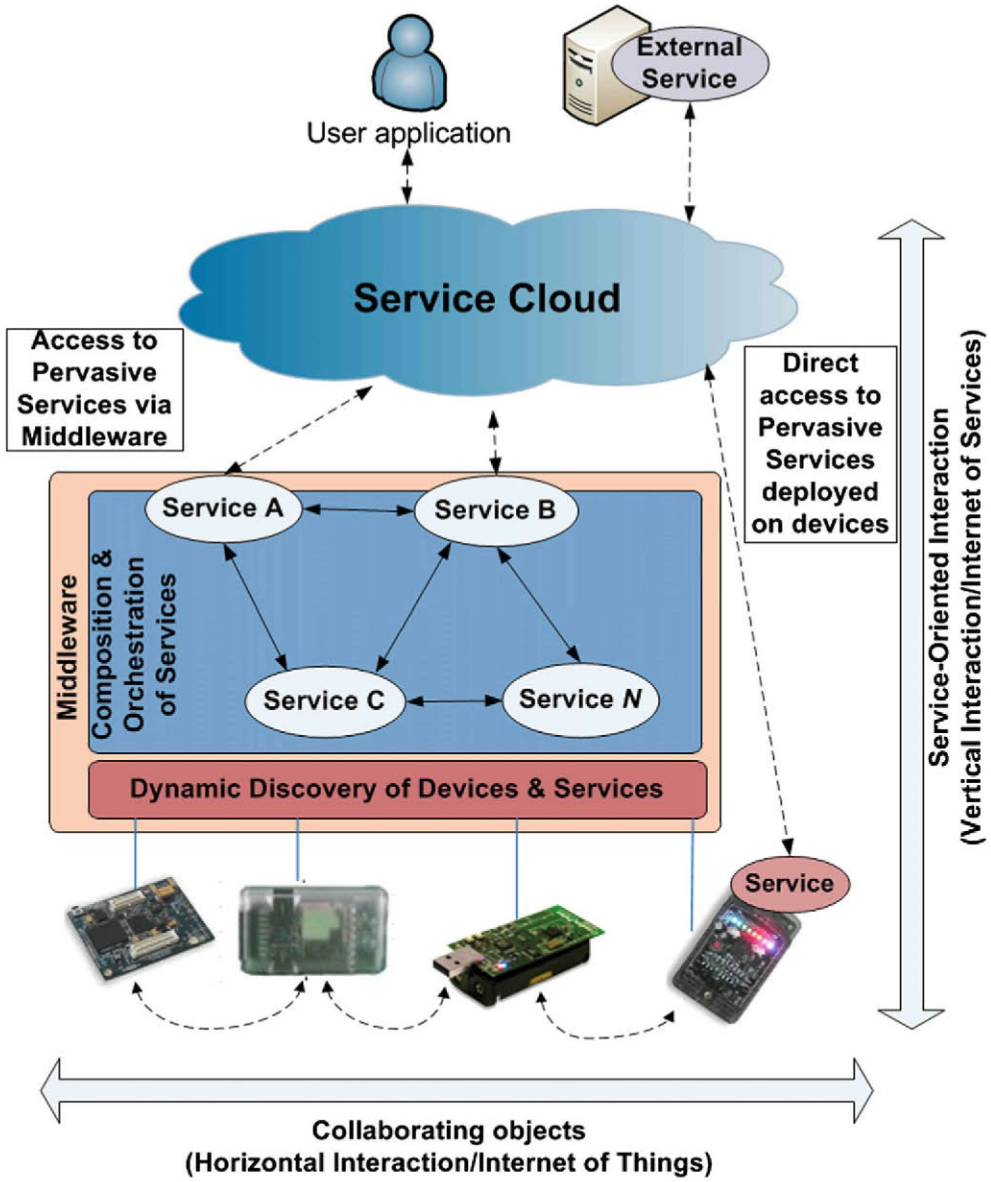


Fig. 1. Integration of pervasive networks into the service cloud of the future internet.

In-network data processing is general procedure for gathering data and routing through the network in order to optimize the resources of the network, particularly from the point of view of energy consumption that optimizes the system lifetime [13]. In this respect, a number of researches were performed during the past decade such as those related to aggregation, metadata negotiation or data fusion [14–18]. However, in most cases, in-network processing is specifically designed for a single type of tasks and very often it is not generic enough to support multiple services demanded by the current context-aware applications that deal with a variety of objectives.

An alternative to early ad hoc approaches is to support and provide a diversity of functionalities through a well-specified application-level interface while taking advantage of in-network data processing background. This proposal can be achieved by using middleware platforms that allow developers to deploy applications disregarding low-level issues (e.g. resource allocation, node topology, routing, etc). A middleware-based paradigm enables an easy development of novel sensing-based applications as well as its deployment and maintenance in pervasive networks [19].

Some interesting proposals have been found in the literature related to middleware approaches for embedded devices. For instance, RUNES project (Reconfigurable, Ubiquitous, Networked Embedded System) [20] tries to solve the common challenges (mostly maintenance) using a component-based programming model. Another approach, MiLAN (Middleware Linking Applications and Networks) [21] allows the applications to specify their QoS needs and adjust the network characteristics to increase WSN lifetime while still meeting those requirements. In spite of early efforts in middleware for embedded platforms, it still shows some unacceptable drawbacks for current SOA-based pervasive networks. Data aggregation is the only in-network data processing performed by those middleware proposals. Furthermore, network node dynamic behavior and node mobility are not clearly supported. They impede the development of more robust and generalist environments that are needed in order to support service-oriented paradigms.

Big efforts have been done in order to model high level contextual information and translate it in order to be used in a service-oriented manner. Those service-oriented approaches are usually

achieved through three phases managing the contextual information: *discovery*, *acquisition* and *reasoning*. An early approach for this issue can be found in [22]. That proposal is based on a middleware platform for *contextual agents*. This middleware is designed to make up an execution framework suitable for agents in ubiquitous computing environments, allowing the use of various reasoning tools like first order logic and temporal logic to process contextual information. More recent approaches like [23] and [24] provide frameworks to hide the platform heterogeneity by means of interfaces of its various components. In [23] and [25] contextual information is tagged semantically and mapped into ontologies. Those semantic mechanisms allow discovering and processing contextual information in the network. According to service-oriented computing paradigms, the semantic information models are necessary to integrate pervasive embedded networks into the Service Cloud. However, those models have to be standardized and become generic enough so as to describe a wide range of domains.

Service-oriented approaches described above solve issues such as context information acquisition, classification and processing. However, their major drawback arises when dealing with high dynamic environments. Unlike traditional Internet services, real-world services are provided by embedded and resources-constrained devices. These devices create high dynamic environments where services can appear, vanish, and re-appear. According to this specific characteristic, it is necessary to use mechanisms and protocols to discover devices and services, as well as their effective management. It could seem a good idea to implement SOA standards to solve this issue; however, these standards were designed taking into account service deployed on static and no resource limited networks e.g. WS-$*$ specifications.

A proposal to adapt SOA standards to pervasive embedded networks has been presented in [11]. This proposal deploys Web services directly over embedded devices by using the Web Service Description Language (WSDL) [7]. Moreover, it uses SOAP over UDP in order to achieve high performance in service transactions as well as WS-Eventing [26] to enable integration with Web service based on Internet applications. However, this proposal introduces too much overhead when using WSDL to describe services, as well as SOAP as application protocol, since both use XML which is a redundant language. Using more lightweight languages that consider limited resources of embedded devices e.g. JavaScript Object Notation (JSON) [27] can solve this issue.

Another proposal to implement Web services in embedded devices is SOCRADES [28]. Its major goal is to assist the developers to the discovery of real-world services by means of the Real-World Service Discovery and Provisioning Process (RSDPP). The access to real-world service is achieved by deploying two alternative approaches in gateways: Device Profile for Web Services (DPWS) [29] and Representational State Transfer (REST) APIs [6]. SOCRADES seems a good global solution to provide real-world services through pervasive embedded networks; however, the processes carried out to deploy and discover services over the embedded network have not been completely specified enough from developer's point of view.

The approaches previously described have positively contributed to the State of the Art of SOA-based pervasive embedded networks; although, there are many issues to be solved in this field yet. Firstly, lightweight information models have to be designed. We propose the use of JSON to describe services and tag contextual information. This offers less semantic redundancy than XML language, which can better performance by saving computing resources and bandwidth. Secondly, a development model for rapid prototyping in embedded networks does not successfully address this point. Our proposal aims a framework to model and deploy agents similar to the Plain Old Java Objects (POJOs), which allows rapid deployment and reusability. This development model enables an easy integration and maintenance of loosely coupled software parts. Moreover, underlying mechanisms and protocols to discover and orchestrate services have been designed in order to hide resource allocation issues to developers, creating an Internetworking environment without the need of application gateways.

The following sections do not aim to provide a comparison between our approach and other proposals that enable internetworking capabilities; rather, the objective is to present an original SOA-based proposal for pervasive embedded networks and an evaluation that helps comparing it with other service-oriented solutions.

3. General principles of the architecture

One of the major foundations of KASO Middleware is a set of Knowledge Management (KM) services that depends on an Information Model of the System. This Information Model is handled to expose network resources in a service-oriented manner in order to achieve the major challenge of our research: to narrow the gap between pervasive embedded networks and the Service Cloud.

In this section, the general principles of the KASO Middleware architecture are explained. These principles are based on the following points: a *basic deployment infrastructure* over which pervasive service will be deployed and run; the *Information Model* in order to define an abstract representation of the System environment; *Perceptual Reasoning Agents* which are logical units implementing business processes to provide pervasive services; and *external entities*, i.e. user applications and external services, which are deployed over conventional networks taking advantage of pervasive services.

3.1. Basic Deployment Infrastructure

The basic deployment infrastructure is made up of various types of hardware devices deployed around specific spaces. These devices range from wireless embedded nodes, to Gateways. Fig. 2 shows a scheme of this deployment infrastructure. As shown, nodes in the embedded network play three roles: (i) Sensor and Actuator (SA), (ii) Contextual (C), and (iii) Sink. SA Nodes offer simple services that are discovered and orchestrated on C Nodes. Sink Nodes collect descriptions of these services and transmit them directly to the Gateway. Furthermore, there are application servers running on the Gateway which enable performing any service transaction over pervasive services offered by the embedded network.

Sensors and actuators connected to the nodes of the WSAN will allow providing a number of real-world services (e.g. environmental measurements and HVAC of a building).

3.2. Information Model of the System

An Information Model of the System has been designed in order to describe both low-level resources (e.g. sensors, actuators, memory, battery, etc.), and high-level resources (simple and composite services, service composition rules and workflow plans). Data characterizing such resources is stored in *Knowledge Bases*; this data generates instantiations of the Information Model of the System.

Knowledge Bases (KB) are distributed between the devices of the overall System (see Fig. 3). The KBs are planned in different tables which are explained in following sections. In this base scheme there are three kinds of KBs:

Mote Knowledge Base is deployed on Sensor and Actuator nodes (motes) and stores information about simple services, service com-

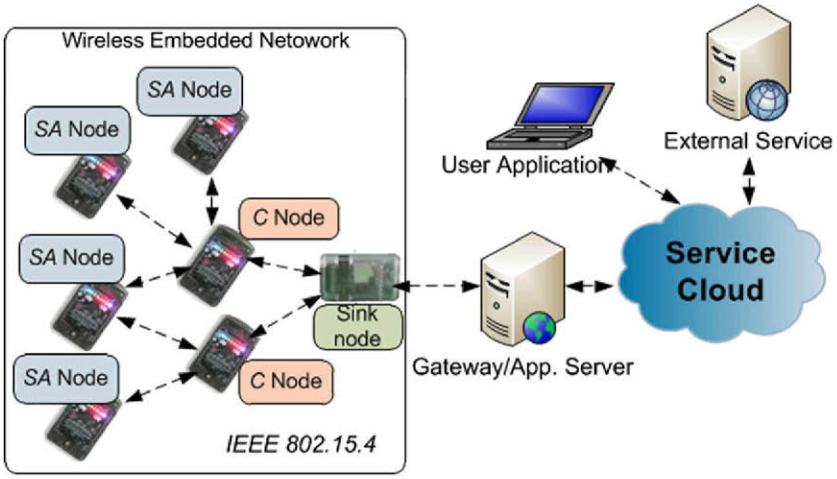


Fig. 2. The Basic Deployment Infrastructure.

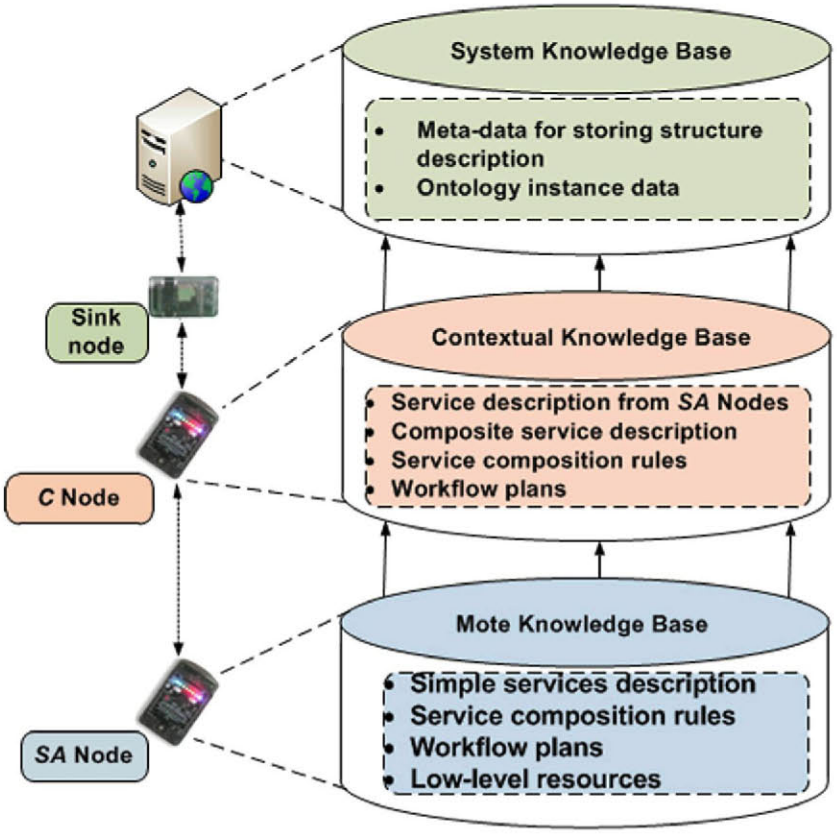


Fig. 3. Knowledge Bases mapped over the infrastructure devices.

position rules and workflow plans to manage the orchestration of simple services and available physical resources.

Contextual Knowledge Base is deployed on Contextual nodes and stores information about simple and composite services, service composition rules and workflow plans to manage the orchestration of composite services.

System Knowledge Base is deployed on Gateways and collects semantic pieces in order to generate the Information Model of the whole embedded network.

As previously described, the repository on the top of the hierarchy is deployed on the Gateway that interconnects the embedded network with the Service Cloud. This KB is intended to be structured as ontology over a RDF/OWL 2 [30,31] document.

3.3. Perceptual Reasoning Agents

KASO Middleware is designed from an agent paradigm called *Perceptual Reasoning Agents* (PRAs). PRAs are deployed both in SA Nodes and C Modes according to the network hierarchy; they are called *M-PRAs* and *C-PRAs*, respectively. A programming model has been designed to facilitate the development and deployment of those PRAs into the execution environment of the KASO Middleware.

Among other facilities, the KASO Middleware provides developers a common API to register PRAs in the system which allows hiding the complexity of underlying mechanisms, e.g. service discovery, resources allocation, orchestration and other control operations.

3.4. External entities

Internetworking mechanisms implemented by KASO Middleware allows external entities to access pervasive services offered by the embedded network. These services are mapped over standardized REST Web services, which are described and exposed through a WSDL 2.0 document [7].

Internetworking functionalities implemented by KASO Middleware allows performing actions over every level of a pervasive embedded network by invoking standard Web services. In Section 5.6, the Web service method encapsulation used by KASO Middleware is shown through an example. External entities have access to: (a) low-level resources by setting up parameters on hardware components (e.g. radio signal strength, behavior of sensors and actuators); (b) middleware level to control the life-cycle of PRAs (e.g. load/unload and start/stop them); (c) information modeling level to augment or restrict the Information Model of the System.

In this section, principles of KASO Middleware architecture are explained. In the following sections, specific features of KASO Middleware architecture are explained. Those features are the cornerstone in providing Knowledge Management services from data collected by sensors and actuators nodes and offering this knowledge by means of SOA mechanisms.

4. Design specifications of the Knowledge-Aware and Service-Oriented Middleware

4.1. Architecture overview

KASO Middleware architecture takes advance from SOA approaches that have become very popular in the current Internet [32–34]. Moreover, semantic models defining ontologies [30,31] have been taken into account in order to design Knowledge Management services. Fig. 4 shows a block diagram of subsystems making up KASO Middleware architecture.

KASO Middleware is based on a layered architecture in which each subsystem encapsulates different functionalities. Each of these layers is explained in the following sections. Firstly, the Multi-functional Embedded Layer, which is common to most approaches for pervasive embedded networks, is explained. Secondly, the middleware layer, which provides services-oriented and Knowledge Management capabilities to the embedded environment, is pointed out.

4.2. Multi-functional Embedded Layer

The *Multi-functional Embedded Layer* is the lowest layer of the architecture. In this layer, the hardware platform, which consists of a variety of hardware modules (e.g. CPU, radio, memory, etc.), is located. Recently, many products for Wireless Sensor and Actuator Networks have emerged, in both commercial sector [35,36] and open-source sector [37,38].

A Real-Time Operating System, which manages most of the processes running at low-level, is implemented in this layer. Its major aim is to hide the hardware heterogeneity from higher layers. Several open-source OS for wireless embedded devices such as TinyOS [39], Contiki [40] or LiteOS [41], have consolidated as valuable solutions.

The Network Protocol layer uses mechanisms and protocols for transmitting packets in a multi-hop communication taking into account some essential factors as network topology, energy consumption, and QoS requirements [42–44]. Moreover, standardized routing protocols have to be considered, as those implemented in 6lowpan [45] and *uIP* [46], in order to guarantee compatibility with external conventional network, e.g. LANs or Internet.

The *Multi-functional Embedded Layer* provides basic functionalities, which are complemented by a middleware layer. This middleware layer was designed keeping in mind the major objective

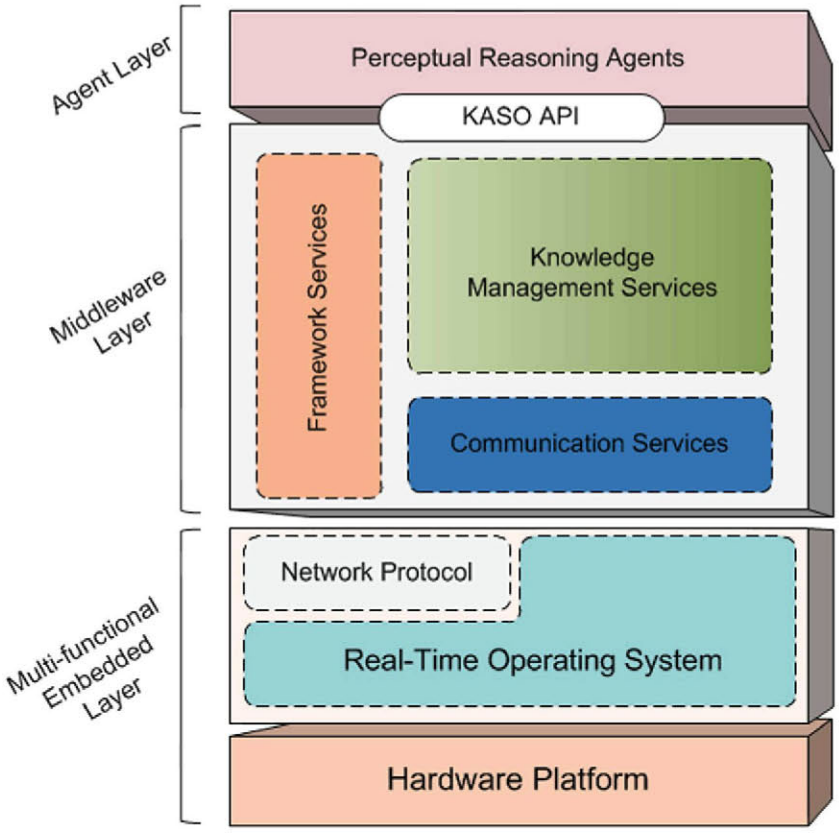


Fig. 4. Subsystems making up KASO Middleware architecture.

of this research: to provide Knowledge Management functionalities in a service-oriented framework. Major features of this middleware layer are explained in the following section.

4.3. Middleware layer

The middleware layer is made up of three subsystems. Fig. 5 shows each one of these subsystems.

The main subsystem offers *Framework Services* which create an execution environment to control the life-cycle of components and agents. The *Query Service* allows PRAs and external services to request information about specific parameters by using a limited set of SQL queries. The *Runtime Manager Service* is in charge of load/unload and star/stop components and PRAs. The *Security Service* offers procedures and algorithms to manage various security issues, majorly those regarding information ciphering and service access permissions. The *In-node Configuration Service* allows setting up parameters at low-level such as allocation of resources (e.g. memory).

The second subsystem provides *Communication Services*. These services are distributed in three modules enabling low-level resource discovery, service exposure and discovery, and internetworking communications. The first module implements *Resource Monitor*. This module provides mechanisms to establish bidirectional communication with physical resources of the node by means of OS primitive callings.

Two modules complement the device communication functionalities in this middleware approach. Each one of them implements application protocols: *Micro Inter-Knowledge Protocol* (μIKP) and *Compressed HTTP over PANs* (CHOPAN) [47]. μIKP offers service discovery mechanisms to nodes belonging to the same embedded network. μIKP is based on a subset of WS-Discovery standard [48]. Furthermore, a REST endpoint is implemented based on the CHOPAN protocol specification. CHOPAN is designed from a binary specification of HTTP with the aim of saving bandwidth and energy consumption in resource-constraint embedded networks. An adaptation layer is necessary in order to translate CHOPAN messages into compatible format for KASO Middleware.

One of the major contributions of this research comes from the subsystem providing Knowledge Management Services (KMS). The business logic behind KMS is implemented by a Broker and an Orchestrator of services. The function of Brokers and Orchestrators depends on the Information Model of the System and its instantiations which are stored in Knowledge Bases (KB) (see Section 3.2). The Broker implements an API that is offered to developers of PRAs in order to register services in the System. Table 1 shows this API.

API shown above hides low-level issues to developers; they have to be only aware of managing simple and composite services, which will be offered by PRAs. The Broker requires three parameters in order to be able to manage a service; two descriptive documents (*Service Description* and *Service Composition Rules*) and a structure that controls two callback methods (see Table 1).

Orchestrator mechanisms need a set of *Service Composition Rules* stored in the KB. This kind of documents indicates which resources have to be allocated so as to provide a service as well as how that service has to be orchestrated is case of being activated. This process is explained in depth in Section 5.4.

5. Programming pervasive services for embedded networks

In this section, the main tools to program pervasive services by means of the development of Perceptual Reasoning Agents are explained. Explanations are illustrated with real-world examples in order to facilitate its comprehension; to this end, a healthcare application based on Wireless Sensor and Actuator Networks (WSANs) has been used as reference. This section finishes with a proposal to interconnect a KASO-based WSAN with the Service Cloud.

5.1. Discovery and mapping of low-level resources

Before KASO Middleware can offer pervasive services, it has to be aware of low-level resources in the node platform, majorly those with sensorial and actuation capabilities. Those resources are locally discovered by means of *Resource Monitor* of the Communication subsystem. *Knowledge Management* subsystem models every low-level resource like sensors or actuators. In order to model such low-level resources, a lightweight description language is proposed: the *Sensor and Actuator Mapping Description* (SAMD).

Listing 1. A SAMD document to describe low-level resources related to a healthcare scenario.

```
{
  "sensors":{ //Defines a list of sensors connected to the node and its capabilities.
    "battery":{ //Battery is defined as a sensor.
       "unit":"volts", //Defines representation units of the sensor.
       "accuracy": "0.2", //Defines the accuracy of the sensorial hardware.
       "resolution": "0.001", //Defines the resolution of sensor samples.
       "eventing":{"bySamplingPeriod", "byThreshold"} //Define event-based behavior.
    },
    "Heart_Monitor":{
       "unit":{"beat_per_second"},
       "accuracy": "2",
       "resolution": "1",
       "eventing": {"bySamplingPeriod", "byThreshold"}
    },
    "Sphygmomanometer":{
       "unit":"mmHg",
       "accuracy": "1",
       "resolution": "2",
       "eventing":{"byThreshold"}
    }
  },
  "actuators":{ //Defines a list of actuators connected to the node and its capabilities.
    "defibrillator":{
         "actions":[ //Defines a list of available actions
      //Configuration attributes (minValue, maxValue and resolution)are defined for each action.
           {"name":"defibrillation", "minValue":"5", "maxValue":"400", "resolution":"1"},
       ]
    }
  }
}
```

Table 1
API offered by the Broker.

`PRA_ID=registerPRA(Service_Description, Service_Composition_Rules, Service_Processor)`	This method registers PRAs into the System by means of its three gears: description of services provided by such PRA, composition rules for those services, and a structure consisting of two callback methods to receive final results from service request and partial results from a service workflow that has to be controlled by the PRA.
`deregisterPRA(PRA_ID)`	This method deregisters a PRA from the System.
`updatePRA(PRA_ID, Service_Description, Composition_Rules)`	This method updates information about a registered PRA.

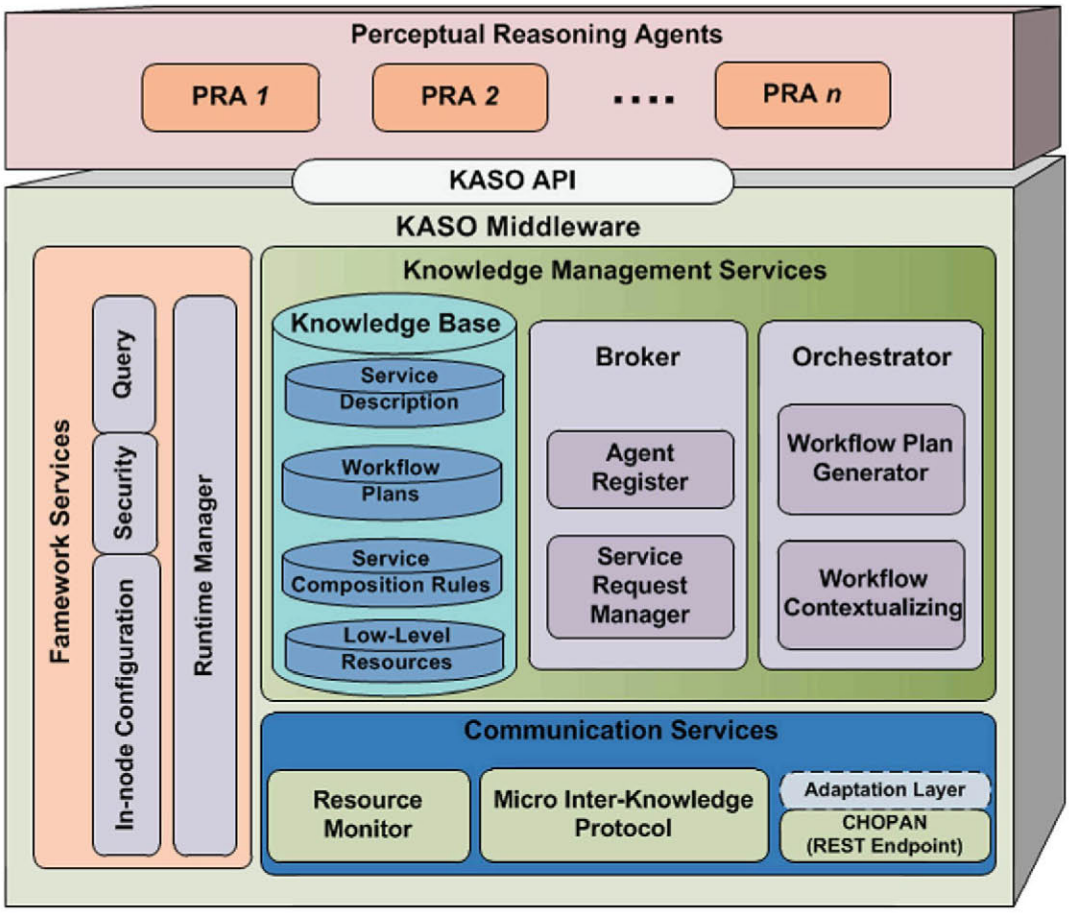


Fig. 5. Detailed diagram of KASO Middleware subsystems.

An example of SAMD document is shown in Listing 1.

SAMD documents describe parameters such as representation unit (`''unit''`), asynchronous behavior (`''eventing''`) or sensorial accuracy (`''accuracy''`) can be described.

Low-level resources are discovered through protocols managed by the Resource Monitor during node's bootstrapping process and runtime. Finally, a SAMD document is instantiated with semantic description about low-level resources and stored in the *Contextual Resources Table* of the Knowledge Base.

5.2. Specifications of the Perceptual Reasoning Agent

Perceptual Reasoning Agent (PRA) specification is based on two characteristics: *independency* from the platform, and *reusability* in multiple kinds of scenarios. A specific programming model is proposed to expedite design, develop, and deploy PRAs while consuming few low-level resources (e.g. low footprints and CPU use).

The development of PRA is tackled by means of a model-driven approach. This way, a UML (Unified Modeling Language) model is designed as a PRA *template* to be used by developers. That interface facilitates a seamlessly management and scheduling of the PRA's lifecycle by Framework and Knowledge Management subsystems. The UML model is illustrated in Fig. 6.

The PRA's life-cycle can be manage through control methods: `load()`, `run()`, `stop()` and `unload()`. An invocation to `load()` method allows instantiating a copy of the Framework context. Moreover, when invocating `load()` method, two semantic structures, which are related to services provided by the PRA, are instantiated: the first one (*Service Description*) describes the service itself; the second one (*Service Composition Rules*) describes how the service has to be composed and orchestrated. After an invocation to `run()` method, PRAs are registered into the System by means a calling to `registerPRA()` method of the Broker API. This step establishes a service point between the PRA and the Knowledge Management Subsystem. When nodes are being shutdown, PRAs running on them are deregistered and semantic structures regard-

ing services are liberated from the KB; Such operations are performed by means of invocations to `stop ()` and `unload ()` methods.

5.3. Register of pervasive services

As explained in Section 4.3, when registering PRA services, developers have to perform an invocation to the method `registerPRA ()` of the Broker's API. Such method needs several parameters to complete a registration, among others, a service description document. That document has to be written according to the *Service Mapping Description* (SMD) [49]. SMD documents are *compact, simple* and *readable;* they are represented as a JSON Object describing specific features of Web services. The Broker collects and stores every SMD document in the *Service Description Table* of the Knowledge Base. Listing 2 shows an example of a service description document based on SMD.

Listing 2. Service description for a healthcare application based on the SMD proposal.

```
1{
2    "transport": "REST",        //The service request should be sent using
3                                //standard HTTP methods (GET, POST, PUT and DELETE).
4    "envelope": "URL",          //The response should be value returned from the method call.
5    "target": "/eHealthPRA", //Defines the URL to connect the service.
6     "SMDVersion": "2.1",
7     "exposure": "level2",      //Defines the exposure level of the service. To be used by uIKP.
8
9    "services": {
10       "BloodPressure": { //Definition of an event-based service
11           "composition": false, //Defines availability to be aggregated in composite services.
12              "eventing": "byThreshold", //Defines event-based behavior.
13           "parameters": [
14               //Defines parameter for the subscription to events.
15               {"name":"operator", "type": "string"},
16               {"name":"value", "type": "number"},
17               {"name":"expiration", "type": "number"},
18           ]
19              "additionalParameters": [
20                   //Defines Additional parameter for the subscription to events.
21                 {"name":"operator", "type": "string"}
22                 {"name":"value", "type": "number"}
23              ]
24              "return":{"type": "number"}
25       },
26       "getHeartRate": { //Definition of an on-demand service.
27            //Redefines the transport and envelope for this specific service.
28           "transport": "POST", //It uses POST as the transport.
29           "envelope": "JSON",  //It uses JSON object as the envelope.
30           "composition": true,
31           "parameters": [
32               {"name": "mode", "type": "string"},
33               {"name": "minThreshold", "type": "integer", "optional": true},
34               {"name": "maxThreshold", "type": "integer", "optional": true}
35           ]
36                  "return":{"type":"number"}
37       }
38   }
39 }
```

The JSON schema of the SMD has been extended in order to improve service-oriented capabilities of the KASO Middleware focused on *discovery*, *composition* and *dispatching mechanisms*. This way, three new attributes have been added to the original schema of the SMD: "`exposure`", "`composition`" and "`eventing`". The definition of these attributes is mandatory; if they do not exist in the SMD document, default values will be obtained.

The "`exposure`" attribute is defined to manage service discovery by means of the μIKP protocol. This attribute can get the following values:

- "`Level0`": Services are not exposed to any entity of the system. This is the default value.
- "`Level1`": Services are exposed to other entities of the embedded network, i.e. other SA and C nodes.
- "`Level2`": Services are exposed to entities making up the whole system.

The ''`composition`'' attribute indicates if a simple service can be composed and orchestrated into a workflow. This attribute can get ''`true`'' or ''`false`'' value. If it is instantiated to ''`false`'' value, the service will not be able to be aggregated into a composite service. Such pervasive services are intended to be atomically invocated. The default value for the ''`composition`'' attribute is ''`false`''.

The ''`eventing`'' attribute defines two models of service dispatching mechanisms: *on-demand* and *event-based*. Furthermore, two methods are supported for event-based services, which are available through values assigned to ''`eventing`'' attribute. Those values are the following ones:

- ''`byThreshold`'': Events are thrown when results of service procedures comply specific threshold conditions defined in a subscription. This kind of subscriptions use two parameters to define threshold conditions: "`operator`", which permitted values are "`equal`", "`more Than`" and "`less Than`"; and "`value`", that defines the value from which the threshold is defined.
- "`by Sampling Period`'': Events are thrown when finishing a period of time defined in the subscription. This kind of subscriptions uses the "`period`" attribute in order to define the sampling period to throw results to event consumers.

Event subscriptions are also defined by the ''`expiration`'' attribute. This attribute indicates the period of time during which

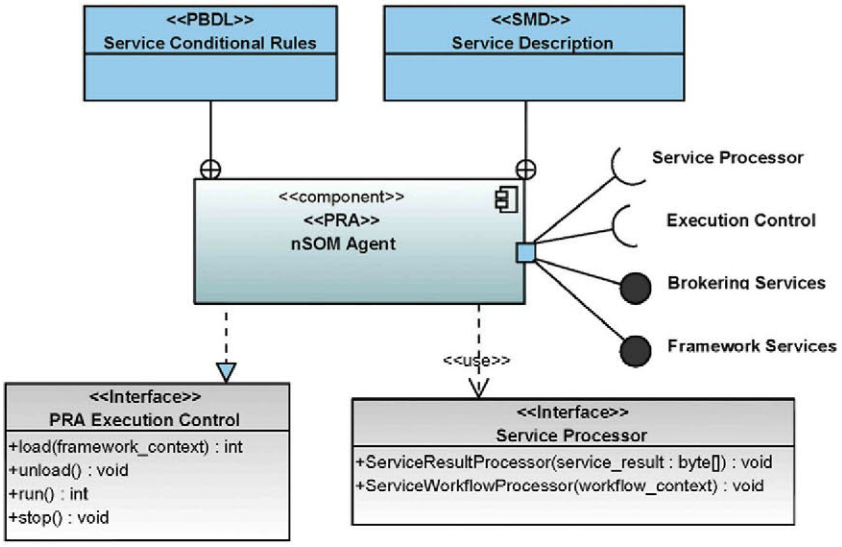


Fig. 6. UML model of the Perceptual Reasoning Agent.

a concrete subscription is valid. Every subscription has to be renewed by event consumers in order to restart the expiration counter; otherwise, if the subscription is not renewed and that counter is expired, the subscription is automatically removed.

To define on-demand services, the ''eventing'' attribute has to get ''false'' value. That is the default value for ''eventing'' attribute.

5.4. Composition and orchestration of pervasive services

Knowledge Management Subsystem of KASO Middleware is able to compose and orchestrate pervasive services from low-level resources and simple services by means of complex business processes. Composition and orchestration are supported by the Information Model of the System (see Section 3.2). The smallest units of

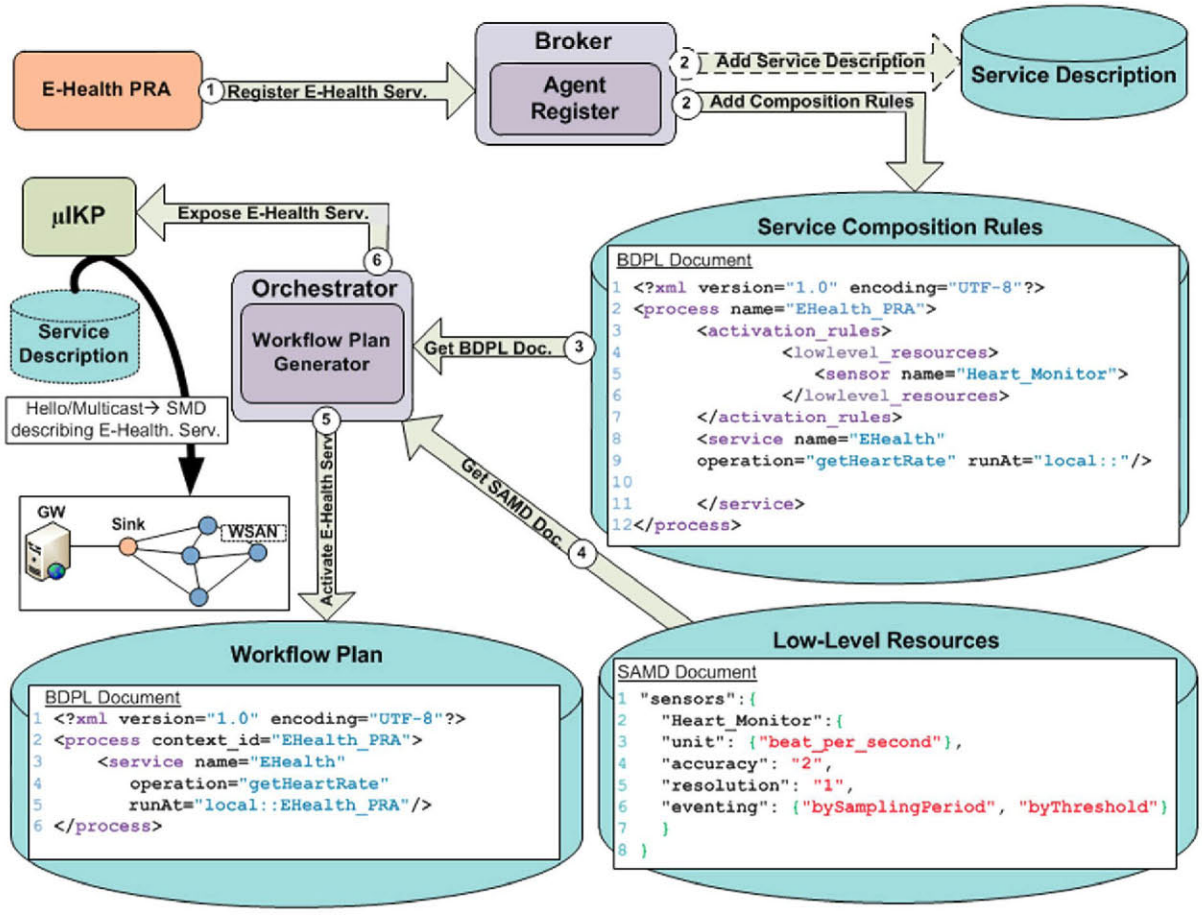


Fig. 7. Procedures to activate and expose a simple service to provide e-Health.

Listing 3. <Activation_Rules> section for a composite service related to e-Health.

```
1<activation_rules>
2
3 <simple_services>
4    <service operation="getECG" runAt=" externalNode::ECG_PRA"/>
5     <service operation="getBloodPressure" runAt="externalNode::blood_PRA"/>
6     <service operation="getHealthStatus" runAt="localNode::health_PRA">
7               <parameter value="context.ECG"/>
9              <parameter value="context.heartRate"/>
10   </service>
11   <service operation="setDefibrillator" runAt="externalNode::emergency_PRA">
12
13  </simple_services>
14
15</activation_rules>
```

this Information Model are mapped over low-level resources from which simple services can be provided. On the other hand, simple services can be aggregated into composite services. Such composition tasks are carried out by the Orchestrator through its Workflow Plan Generator module.

The KASO Middleware has its own mechanisms to discover both high and low-level resources. To this aim, KASO Middleware dynamically explores the network for resources; a specific service is just activated if necessary resources to compose that service are available. The only requirement is to provide a document describing service composition rules to the Broker.

A new lightweight XML-based language has been designed in order to define composition rules and workflow plans for the orchestration of services. That language has been called *Pervasive Business Definition Language* (PBDL). PBDL documents describing both composition rules and workflow plans are stored in the *Service Composition Rules Table* and *Workflow Plans Table*, respectively.

Let us illustrate a basic service composition through an example (see Fig. 7). In this scenario a PRA registers a simple service in order to provide an e-Heatlh service. Meanwhile, the Workflow Generator has found a heart monitor through the Resource Monitor sensor. Finally, the `getHeartRate` operation is activated and exposed by means of μIKP protocol.

The *Service Composition Rules Table* is made by PBDL documents which are structured around two tags: `<activation_rules>` and `<service>`. `<activation_rules>` tag is used to describe resources necessary to provide concrete services. `<service>` tag is used to describe business processes managing resources to obtain a result from a specific request. If rules in *<activation_rules>* section are satisfied, i.e. necessary resources to be able to provide such service are available, a PBDL document describing the workflow plan is created and inserted in the *Workflow Plan Table*. Regarding PBDL documents in *Service Composition Rules Table*, PBDL documents describing workflow plans instantiate the `runAt` attribute of the tag `<service>` (to a local or external PRA) and removes the `<activation_rules>` section.

Composition and orchestration of composite services are performed by C Nodes (see Section 3.2). This process takes place when a C-PRA registers a composite service in the System. Activation procedures for composite services are carried out in the same way as those for simple services, but taking into account simple services instead of low-level resources. The latter issue is illustrated in Listing 3.

In Listing 3, the activation requirements to activate composite services are defined within the section `<activation_rules>`. The business process orchestrating simple services, which are aggregated to a composite service, is described in a workflow plan.

Listing 4 shows a workflow plan for a hypothetical composite service (`Set_Emergency_Assistant`) which would be activated by activation rules shown in Listing 3. In Listing 4, the first tag is `<fork>`, which has two groups (tagged as `<group>`) in the body. It divides the workflow and parallel invokes `getBloodPressure` and `getECG` operations that are provided by PRAs running on different nodes (`Blood_PRA` on Node A, and `ECG_PRA` on Node B, respectively). Then, `getHealthStatus` operation of the `Health_PRA` running on local Node is invoked. Finally, if current health status of the patient is critical, the operation `setDefibr-`

Listing 4. A workflow plan describing a business process for an e-Health service.

```
1 <process name="Set_Emergency_Assistant">
2
3   <fork>
4         <group>
5            <service operation="getBloodPressure" runAt="NodeA::Blood_PRA"
6              context="context" returnVar="context.bloodPressure">
7                   <parameter value="context.pollingMode"/>
8             </service>
9              <service operation="getECG" runAt="NodeB::ECG_PRA"
10              context="context" returnVar="context.ECG"/>
11               <parameter value="context.pollingMode"/>
12            </service>
13        </group>
14  </fork>
15
16    <service operation="getHealthStatus" context="context"
17       returnVar="context.currentHealthStatus" runAt="localNode::Health_PRA">
18         <parameter value="context.bloodPressure"/>
19         <parameter value="context.ECG"/>
20    </service>
21
22      <if test="{context.currentHealthStatus == context.critical}">
23         <service operation="setDefibrillator" runAt="NodeC::Emergency_PRA"
24         context="context"/>
25       </if>
26
27 </process>
```

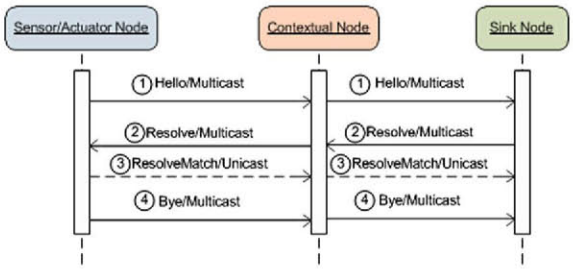


Fig. 8. Messages Interchanged by μIKP.

`illator` provided by `Emergency_PRA` that is loaded in Node C is invoked.

The management of the memory allocated for service workflows is performed by the Workflow Contextualizing of the Orchestrator; that management includes the collection and storage of returned values from invocation to operations, partial invocations to `ServiceWorkflowProcessor(workflow_context)` method of the specific PRA every time an operation returns a value, and frees memory when the service workflow is finished, between other tasks.

5.5. Exposure of pervasive services

The embedded network uses the Micro Inter-Knowledge Protocol (μIKP) in order to expose and discover pervasive services defined in SMD documents. The μIKP is a reduced specification of the WS-Discovery standard [48] used by DPWS; however, μIKP is not still compatible with those traditional Web service architectures. Three roles are managed by μIKP: *SA* Nodes work as service targets, Sink Nodes work as service clients and *C* Nodes have a combined behavior (service client or service target). Message interchanged by μIKP are shown in Fig. 8.

Initially, *SA* Nodes and *C* nodes send multicast *Hello* messages (1) to expose services they can provide when joining the network. Hello messages can be sent again during runtime in order to update the service profile stored in KB of other nodes. This usually happens when a new service is activated into the node. *SA* Nodes and *C* Nodes may also receive multicast *Resolve* messages (2) to find a particular service at any time and send a unicast *ResolveMatch* message (3) if some of their services are the searched ones. Finally, before a node leaves the network, it sends a multicast *Bye* message (4) in order to remove information from KBs regarding to its services.

A *Semantic Engine* running on the Gateway translates every SMD document into a RDF model. Those information pieces are used to build the System's ontology, which is structured in OWL. The System's ontology is evolved according to the reception of SMDs and stored in the System KB of the Gateway.

Fig. 9 it is shown a proposal based on a synergy between embedded networks and RESTful [6] architectures. This approach allows exposing pervasive services deployed in the embedded network as Web services in WSDL 2.0 documents. Those WSDL 2.0 documents are automatically generated by means of a parsing process from an instance of the System's ontology.

5.6. Requests for pervasive services

When finishing exposition and discovery procedures, PRAs and external entities can access to pervasive services that are provided by the embedded network. These services are accessed through

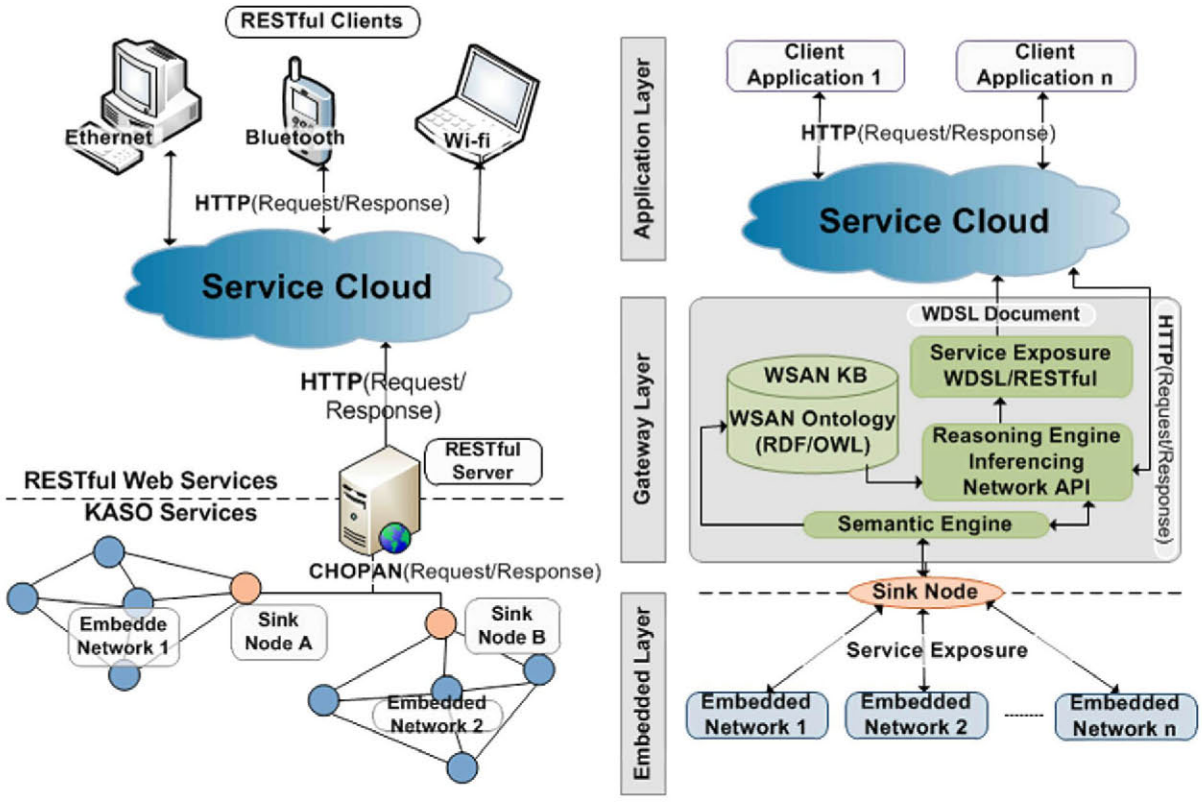
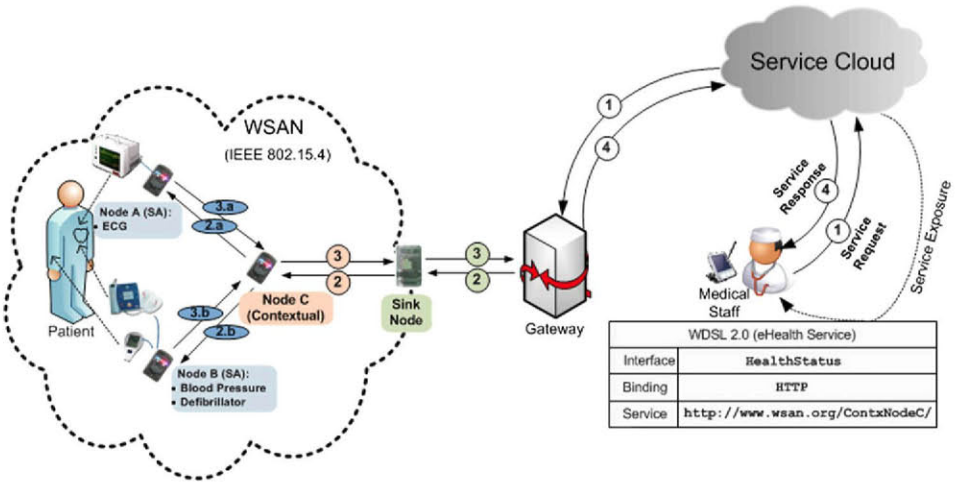


Fig. 9. An Internetworking approach based on RESTful technologies.



Message Interchange for a Request to eHealth Service

① HTTP [GET /ContxNodeC/HealthStatus HTTP/1.1 Host: www.wsan.org]	④ HTTP [HTTP/1.x 200 OK Content-Type: text/plain; charset=ISO-8859-1 device: ContextualNodeC resource: /ContxNodeC/HealthStatus method: GET <?xml version="1.0" encoding="UTF-8"?> <wsan id="wsan"> <serviceResponse value="getHealthStatus"> <origin id="ContxNodeC"/> <payload id="Health" value="X"/> <serviceResponse/> <wsan/>]
② CHOPAN [POST /ContxNodeC/HealthStatus {"method":"getHealthStatus", params":["byThreshold", Health<normal],"id"="x"}]	
2.x CHOPAN [POST /nodeX/(a) ECG, (b)BloodPressure {"method": "getECG", "getBloodPressure" "params":"polling_mode", "id"="y"}]	
3.x CHOPAN [{"result":{(a)"volts",(b)"mmHg)": X } } ,"id"="y"}]	
③ CHOPAN [{"result": {"Health": X } ,"id"="x"}]	

Fig. 10. Message Interchange during a service transaction in our Internetworking approach.

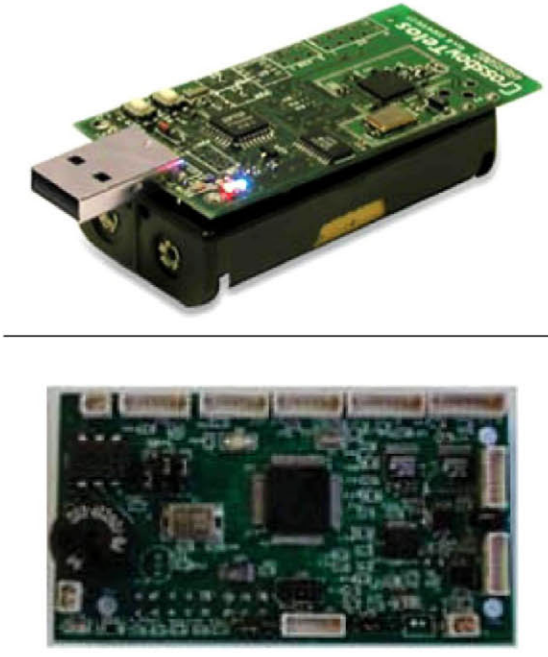


Fig. 11. On the top, a TelosB node; on the bottom, the top face of a SSB.

HTTP methods (*GET, PUT, POST* and *DELETE*) according to RESTful mechanisms. Services interfaces are obtained through a WSDL 2.0 document.

The Gateway is in charge of receiving service requests and performs some translations, e.g. SOAP messages are translated into JSON format. Moreover, HTTP header fields are translated into a binary format according to CHOPAN specification, which can be interpreted by REST endpoints of each node. After this processing, CHOPAN PDUs are routed to the specific node which has to provide results to those service requests.

An example of a service transaction is shown in the Fig. 10. The ''HealthStatus'' service that is described in Listings 3 and 4 is used in this example. In this use case, medical staff can monitor patients by means of a WSAN which exposes its services to the Cloud Service. Pervasive services are described in a WSDL 2.0. Those services can be accessed through REST methods by using traditional devices as PDAs or Laptops.

6. Validation of the KASO Middleware architecture

The motivation scenario to validate KASO Middleware architecture was focused on a set of healthcare applications. The necessary deployments were performed in the Sanatorium Versmė in Birštonas, Lithuania. In this scenario a KASO-based WSAN was integrated in a hypothetical Service Cloud. Since the major aim of this section is to show the validation results of the KASO Middleware, explanations are mainly about results of the tests performed over the WSAN.

6.1. The embedded network infrastructure

The healthcare scenario was aimed taking into account three points: (i) surveillance of the Sanatorium's perimeter, (ii) tracking of patients and medical staff, (iii) and monitoring of critical vital

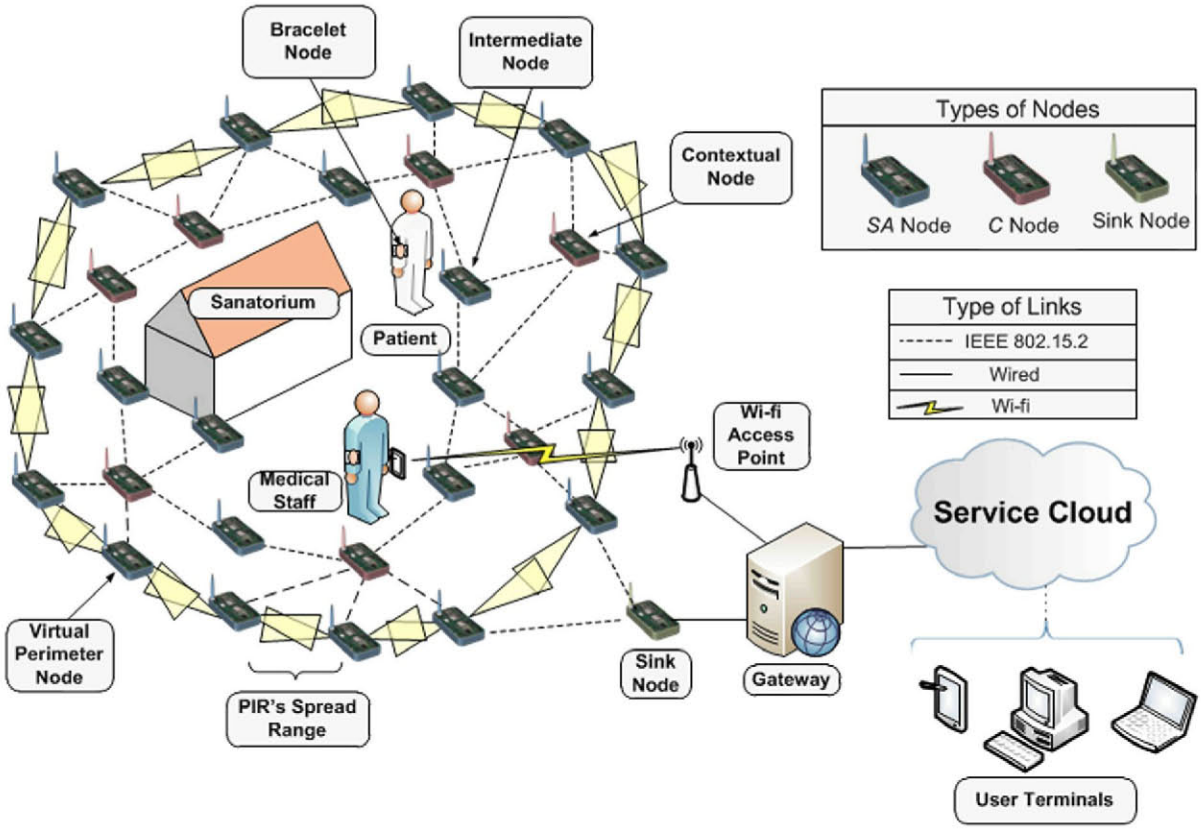


Fig. 12. The scenario infrastructure deployed to validate KASO Middleware architecture.

Fig. 13. A Virtual Perimeter Node deployed on the Sanatorium perimeter.

signs. The deployed System was able to provide services in order to manage almost every non-common and emergency situation regarding healthcare and safety in Versmè Sanatorium. The deployed infrastructure is shown in Fig. 12.

The WSAN deployed around the Sanatorium area was made up of SA nodes with different roles offering different services. The hardware platform chosen for the WSAN deployment was the Crossbow TelosB [50]. TelosB's features are 16 bits RISC processor (MSP430 microcontroller), wireless interface (IEEE 802.15.4), 48 Kb of Flash memory, 16 Kb of Configuration EEPROM, and 10 Kb of RAM. The energy supply of this platform is by means of 2 AA batteries. An additional hardware module was used, a Smart Sensor Board (SSB). The SSB was designed to connect a number of sensors and actuators through the USART interface of TelosB nodes (Fig. 11).

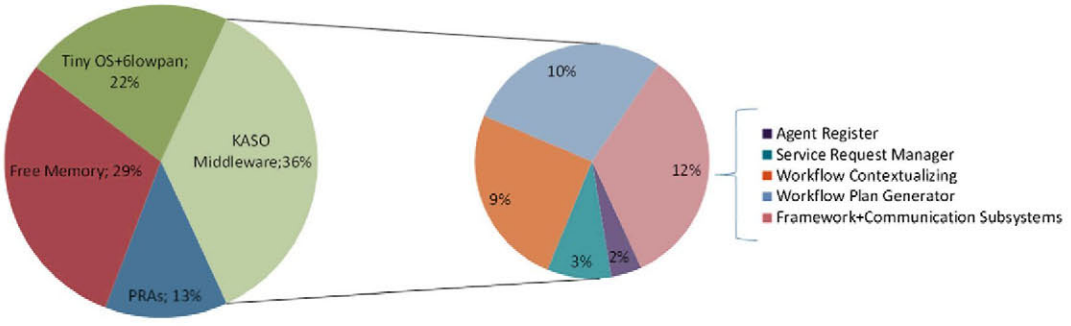


Fig. 14. ROM footprint of the overall architecture.

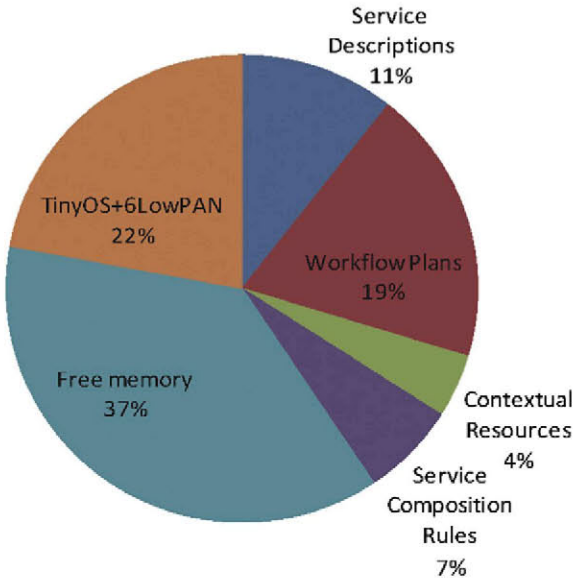


Fig. 15. RAM footprint of the Knowledge Base.

Table 2
Event-based services and corresponding PRAs.

Service name	Provider PRA	Event trigger
Suitable environment	Statistical monitoring C-PRA	Average temperature threshold of an area
Perimeter intrusion	Surveillance M-PRA	No authorized person crosses the Sanatorium perimeter
Health emergency	Critical monitoring M-PRA	Heart rate of a patient passes a threshold
Technical supervision	Technical M-PRA	Energy level of batteries passes a min threshold

Table 3
On-demand services and corresponding PRAs.

Service name	Provider PRA	Returned result
Environmental monitoring	Statistical Monitoring C-PRA	Instantaneous average humidity level of an area
Health status	Critical Monitoring M-PRA	General patient's health status according to several parameters: blood pressure, body temperature and heart rate.
Technical control	Technical M-PRA	Instantaneous energy level and RAM occupied.

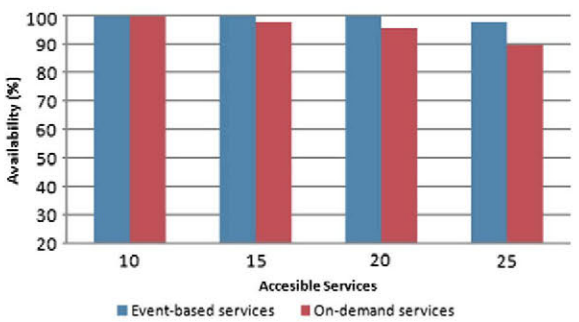


Fig. 16. Service availability in four test cases.

TelosB is a very resource-constrained platform compared to more recent like Imote2 [51] or Sun Spot [36]. Taking into account such restrictions, the testing on memory footprint, latency and availability carried out with this type of nodes are decisive to check the feasibility of the KASO Middleware in embedded devices.

Tiny OS 2.0 [39] and 6lowpan [45] protocol were implemented in the Multi-functional Embedded Layer (see Fig. 4). The interface between KASO Middleware and hardware platform was well addressed through the event-driven paradigm used by Tiny OS. Networking services were also appropriately managed by means of the 6lowpan protocol; its major advantage was the provision of internetworking capabilities to the System, which improves interaction between the WSAN and external networks.

The WSAN infrastructure was made up of five kinds of nodes. Each one played a specific role within the network (see Fig. 12):

- *Virtual Perimeter Node:* A Virtual Perimeter was deployed around the Sanatorium. Nodes making up such perimeter (Fig. 13) were supplied with GSN PATROL-701 Passive Infrared Sensors (PIR) [52]. *Surveillance* M-PRA was running on Virtual Perimeter Nodes. These agents were able to detect perimeter crossings and identify the person crossing the perimeter, i.e. patients or medical staff. For this purpose, a personal Id was provided by a Bracelet Node, which uniquely identified each user.
- *Bracelet Node:* This kind of node was worn by both patients and medical staff. They integrated two biomedical sensors: heart-monitor and body temperature. The major objective of the Bracelet Node was to monitor vital signs of patients while guaranteeing their privacy. By using bracelet nodes, the patients' mobility is notably improved since medical staff can monitor patients' health status wherever they are.
- Services provided by Bracelet Nodes were managed by *Critical Monitoring* M-PRAs which were able to collect data from biomedical sensor and to infer the patient's health status. This Critical Monitoring service could operate under two mechanisms: on-demand or event-based.
- *Intermediate Node:* This kind of node was deployed all around the Sanatorium area. Their main objective was to monitor several environment parameters by using three automation sensors: temperature, humidity and light. From the data extracted with those sensors, some inferences could be performed e.g. if it is advisable for patients go out to take a walk depending on their profile which is determined by their kind of illness or comfort preferences.
- The monitoring of environment parameter was carried out by *Environmental Monitoring* M-PRAs running on Intermediate Nodes.
- *Contextual Node:* The aim of Contextual Nodes was to discover simple service in its neighborhood in order to aggregate them into a workflow plan and activate some composite service (e.g. max/min and average of temperature measurements). These functionalities were provided by a *Statistical Monitoring* C-PRA.
- *Sink Node and Gateway:* These infrastructure components are essential pieces to interconnect the WSAN with external networks. The only objective of Sink Nodes was to resend information from the WSAN to the Gateway through a USB port. On the other hand, the Gateway is in charge of collecting semantic information from the WSAN and building the System ontology based on OWL, as well as to manage service transactions for those services deployed on the WSAN.

6.2. Validation results

A set of test was planned according to the specific infrastructure used in the deployment and the use cases of the System. The Sys-

tem deployment was made up of 106 *SA* Nodes and 21 *C* Nodes. Moreover, 20 patients and 5 medical staff of the Sanatorium Versmè contributed to perform these validation tests since they provided valuable support in order to carry out some of the validation tests.

6.2.1. Memory footprint

Firstly, the footprint of a full implementation of the KASO Middleware architecture over the TelosB platform is analyzed. A node profile consisting of 1 C-PRA (Statistical Monitoring) and 2 M-PRA (Surveillance and Environmental Monitoring) are taken as reference. This profile creates a demanding execution environment, which is useful to measure the impact of KASO Middleware architecture in a real deployment scenario.

The footprint analysis was performed according to the used and available memory both in ROM and RAM. Fig. 14 shows the percentage of used memory according to the total ROM available in TelosB platform i.e. 48 Kb.

As it can be seen in previous Figure the footprint of the KASO Middleware in ROM takes 36% of the total, and the PRAs, which were running on such node, needed 13%. The Multi-Functional Embedded Layer consisting of Tiny OS 2.0 and 6lowpan required 22% of the ROM. The remaining 29% of the ROM (more than 14 Kb) was available to future improvements of the KASO Middleware architecture as well to deploy new PRAs on the node. Since the PRA programming model was designed to optimize memory such remain memory space can be enough to deploy around 4 C-PRAs and 6 M-PRAs.

Moreover, the footprint of the Knowledge Base of that node in RAM during run-time is analyzed. The results of the footprint in RAM are shown in Fig. 15.

The overall architecture used for this test consumed 63% of the total RAM i.e. 10 Kb. This footprint allowed reaching good general performance since the free RAM (37%) was enough in order to manage service workflow contexts, service request tables or internal buffers for 6lowpan.

6.2.2. Availability and Latency

The availability and latency of the system was tested by deploying PRAs all around the WSAN's nodes which provided services by means of two paradigms: *event-based* and *on-demand*.

The services deployed by PRAs were the ones shown in Tables 2 and 3.

Tests to measure both availability and latency were carried out by sending requests to event-based and on-demand services from user terminals of the medical staff (PDAs or laptops). Four scenarios were designed to analyze availability and latency of services. Those four scenarios were designed according to a number of accessible services (10, 15, 20 and 25, respectively), for both event-based and on-demand services. During tests, service requests were randomly sent from 5 user terminals for 7 h. Results of these test cases are shown in Fig. 16 and Fig. 17.

For on-demand services, availability level was correctly obtained from service results returned in the specific terminal. For event-based services, availability level was obtained considering events correctly received from those expected to be received in a concrete situation.

Both tests showed high service availability in general. Availability level was particularly reduced in two cases: When the WSAN dealt with 25 event-based services, the availability decreased slightly to 98%; on the other hand, when the WSAN had to deal with 15 on-demand services, the availability decreased to 96%. In wireless embedded networks, it is difficult to find factors producing the fall of service availability from values near to 100%. In described cases, the main problem comes from the pervasive services own nature deployed on embedded nodes. Usually, when on-demand services are requested, the node blocks resources related to the specific service for short periods of time (e.g. when providing humidity monitoring service, humidity sensors are blocked for 0.5 s). During those periods of time, on-demand services are unavailable to dispatch more requests. Once service result is obtained, it is only returned to the requester entity. Similarly, event-based services block node resources when they are performing operations; however, results obtained from those kind of service (events) can be simultaneously dispatched to more than one requester, i.e. to those all have subscribed to that service. In most cases, this situation explains the higher availability of event-based services in embedded networks.

The analysis observed from lantency tests also provided better behavior for event-based services. In this case, the results are directly related to the traffic traveling through the network: the more the packets are transmitted by nodes, the more the latency is due to the overload of network buffers (both Tx and Rx). This factor has to be especially taken into account when using so bandwidth-constrained devices as IEEE 802.15.4 compliant platforms. It can conclude that to use *one-way* traffic paradigms, like the ones based on events, are more advisable than using *request-reply* paradigms since the traffic can be reduced more than 40%.

According to the results previously shown, we concluded that on-demand services must be only used in specific situations (e.g. health emergencies) in order not to block the node resources for long periods of time. Business process designers should plan service workflows to reach a good event-based strategy according to the applications requirements, particularly in those cases in which the System has to manage critical events, e.g. in deployments for hospitals.

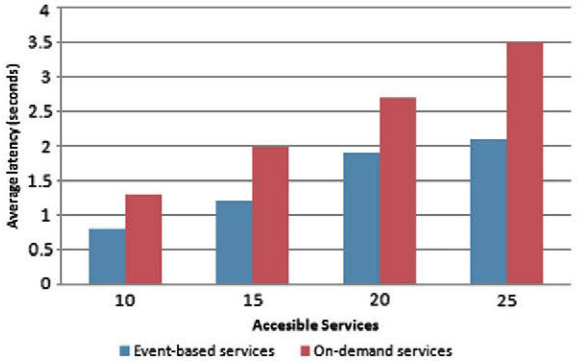


Fig. 17. Latency in four test cases.

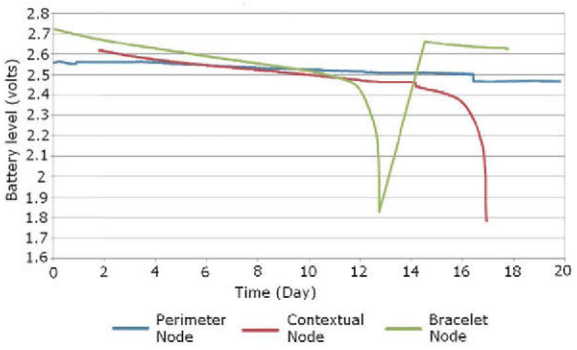


Fig. 18. Energy consumption of three nodes.

6.2.3. Energy consumption

A complete study about energy consumption using this validation scenario could be complex and extensive because of its heterogeneous consisting of a great variety of nodes and services. Therefore, a brief analysis of energy consumption is described in this section. To this end, three nodes of the WSAN in order to sample their respective battery levels for 20 days have been taken and are mentioned as follows:

- A Virtual Perimeter Node exposing simple services to provide environmental measurements and presence events produced around the Sanatorium perimeter.
- A Contextual Node exposing a contextual service to provide aggregated environmental measurements.
- A Bracelet node worn by a patient and exposing critical monitoring services.

Each of these nodes was supplied with 2 Ni-MH batteries characterized by a nominal charge of 2400 mAh at 1.2 V.

The energy consumption results are shown in the Fig. 18.

The Bracelet Node was the first to run out of battery (day 13th) which was changed on the 14th day. The Contextual Node run out on the 17th day, and their batteries were not changed. The Virtual Perimeter Node was still working when the test was done. From the test results, it could be concluded that the node lifetime was directly related to the number of service requests it dispatched; however, there were other factors that contributed to the node lifetime, e.g. number of sensors being managed by the node or number of packet forwarded by the network protocol. The first factor affected majorly to the Bracelet Node because of the number of biomedical sensors it had to control and the criticity of the service it provided. The second factor mostly affected the Contextual Node since it was deployed in a place near the Sink node. This is always a critical point in multi-hop networks as WSANs since packets from all around the network converge in order to be routed to the Sink node. The deployment of alternative sink nodes is usually a good solution to balance traffic since it allows providing better performance and a longer lifetime to the nodes deployed on those regions of the network.

7. Conclusion and future work

This research paper describes a novel middleware architecture for embedded networks, called *Knowledge-Aware and Service-Oriented Middleware* (KASO Middleware). The major aim of KASO Middleware is to integrate embedded networks in the future Service Cloud which will provide pervasive and real-world services (i.e. those related to sensors and actuators) to achieve high performance of the *Internet of Things* paradigm. KASO Middleware is made up of three major subsystems: *Framework, Communications* and *Knowledge Management*. Those subsystems allow managing pervasive services provided by an new agent paradigm called *Perceptual Reasoning Agent* (PRA). PRA-based programming model is provided to developers in order to ease design and deployment of services in pervasive embedded networks. PRAs provide services by means of SOA mechanisms which allow managing pervasive services according to following phases: (i) register, (ii) exposure and discovery, (iii) composition and (iv) orchestration.

KASO Middleware was subjected to a test-bed based on a WSAN infrastructure. The WSAN infrastructure was designed to support a healthcare telemonitoring application and was deployed in a Sanatorium. This real deployment allowed performing many valuable tests in order to validate major features of KASO Middleware architecture. From those validation tests, some strong points were found in KASO Middleware. Its footprint over TelosB platform was considerably acceptable since a full implementation of the proposed architecture left free memory in order to allow future improvements and extensions of the architecture. Moreover, good behavior was observed in service availability and latency majorly related to event-based services.

A future research in order to solve weak points of the KASO Middleware architecture is planned. Such research will focus on improving the semantics describing some important aspects of the System as low-level resources, pervasive services, service composition and business processes. The major objective of this future work is to design a lightweight language founded on *Notation 3* syntax (subject–predicate–object) encapsulated over JSON documents as well as specific ontology to properly describe all the important aspects previously mentioned. Reaching better footprint and energy consumption in the architecture have also been taken into account. To this end, service management issues and size of documents that describe low-level resources, services, service composition rules and workflow plans will be optimized so as to achieve better performance.

Acknowledgments

This work was supported in part by the European Commission under the IST-034642 μSWN: Solving Major Problems in Micro Sensorial Networks research project within the Sixth Framework Programme (FP6) (http://www.uswn.eu); and under the ITEA 2-08005 DiYSE: Do-it-Yourself Smart Experiences (http://www.diyse.org).

References

[1] M. Weiser, Hot topics: ubiquitous computing, IEEE Computer 71 (1993) 72.
[2] M. Conner, Sensors empower the "Internet of Things", EDN 32 (2010) 38.
[3] Z. Brayan, Benefits of service oriented architecture (SOA), University of Applied Science of Northwestern Switzerland, School of Business, 2009.
[4] P.J. Marrón, S. Karnouskos, D. Minder, Research Roadmap on Cooperating Objects, Office for Official Publications of the European Communities, 2009.
[5] M. Friedemann, The design space of wireless sensor networks, IEEE Wireless Communications 3 (2004) 54–61.
[6] Fielding, Roy Thomas, Architectural Styles and the Design of Network-based Software Architectures. Doctoral dissertation, University of California, Irvine, 2000.
[7] R. Chinnici, J-J. Moreau, A. Ryman, S. Weerawarana, WSDL 2.0: a W3C Recommendation, 2007.
[8] World Health Organization, Global age-friendly cities, 2007.
[9] M. Hatlet, D. Gurganious, C. Chi, M. Ritter, WSN for Smart Industries. OnWorld Study, 2007.
[10] J.F. Martínez et al., Composition and deployment of e-Health services over Wireless Sensor Networks, Mathematical and Computer Modelling Journal 53 (2010) 485–503.
[11] N.B. Priyantha, A. Kansal, M. Goraczko, F. Zhao, Tiny Web Services: Design and Implementation of Interoperable and Evolvable Sensor Networks, in: Proceedings of Sixth Conf. Embedded Network Sensor Systems, ACM, 2008, pp. 253–266.
[12] W. Drytkiewicz, I. Radusch, S. Arbanowski, R. Popescu-Zeletin, PREST: A REST-Based Protocol for Pervasive Systems, in: Proceedings of Int'l Conf. Mobile Ad-Hoc and Sensor Systems, IEEE, 2004, pp. 340–348.
[13] L. Gomez, L. Laube, A. Sorniotty, K. Wrona, Secure and trusted in-network data processing in wireless sensor networks, Journal of Information Assurance and Security 2 (2007) 189–199.
[14] C. Intanagonwiwat et al., Directed Diffusion: A Scalable and Robust Communication Paradigm for Sensor Network, in: Proceedings of 6th annual international conference on Mobile computing and networking, ACM, Boston, 2000, pp. 56–67.
[15] W. Heinzelman, J. Kulik, B. Balakrishnan, Adaptive Protocols for Information Dissemination in Wireless Sensor Networks, in: Proceedings of 5th annual international conference on Mobilen computing and networking, ACM, Seattle, 1999, pp. 174–185.
[16] J. Kulik, W. Heinzelman, H. Blakrishnan, Negotiation-based Protocols for Disseminating Information in Wireless Sensor Networks, Wireless Networks, Springer, 2002. pp. 169–185.
[17] C. Chih-Min, H. Tzu-Ying, Design of Structure-Free and Energy-Balanced Data Aggregation in Wireless Sensor Networks, in: Proceedings of 11th IEEE International Conference on High Performance Computing and Communications, IEEE, Washington, 2009, pp. 222–229.

[18] F. Xiufang, X. Zhanwei, A Neural Data Fusion Algorithm in Wireless Sensor Network, in: Proceedings of Pacific-Asia Conference on Circuits, Communications and Systems, Chengdu, 2009, pp. 54–57.
[19] K. Römer, C. Kasten, F. Mattern, Middleware challenges for wireless sensor networks, Mobile Computing and Communication Review 2 (2002) 59–61.
[20] F. Oldewurtel, J. Riihijarvi, K. Rerkrai, P. Mahonen, The RUNES Architecture for Reconfigurable Embedded and Sensor Networks, in: Proceedings of 3th conference of Sensor Technologies and Application, Athens, 2009, pp. 109–116.
[21] W. Heinzelman, A. Murphy, H. Carvalho, M. Perillo, Middleware to Support Sensor Network Applications, IEEE Network Magazine 18 (2004) 6–14.
[22] A. Ranganathan, R.H. Campbell, Use of ontologies in a pervasive computing environment, Vol. 18, Cambridge University Press, London, 2003. pp. 209–220.
[23] I. Mudasser, B.L. Hock, W. Wenqiang, Y. Yuxia, A Service-Oriented Model for Semantics-based Data Management in Wireless Sensor Networks, in: Proceedings of International Conference on Advanced Information Networking and Applications Workshops, Bradford, 2009, pp. 395–400.
[24] A. Taherkordi, Q. Le-Trung, R. Rouvoy, F. Eliassen, WiSeKit: A Distributed Middleware to Support Application-Level Adaptation in Sensor Networks, in: Proceedings of 9th IFIP International Conference on Distributed Applications and Interoperable Systems, 2009, pp. 44–58.
[25] M. Lionel et al., Semantic Sensor Net: An Extensible Framework, International Journal of Ad Hoc and Ubiquitous Computing (2009) 157–167.
[26] D. Box et al., WS-Eventing, W3C Member Submission, (2006).
[27] JSON specification, (2010) URL: "<http://www.json.org/>".
[28] G. Dominique, V. Trifa, S. Karnouskos, P. Spiess, D. Savio, Interacting with the SOA-based internet of things: discovery, query, selection, and on-demand provisioning of web services, IEEE Transactions on Services Computing 3 (2010) 223–235.
[29] D. Dan, M. Antoine, Devices Profile for Web Services (DPWS) Version 1.1. OASIS, 2009.
[30] RDF Core Working Group, Resource Description Framework (RDF), 2004.
[31] S. Peter F. Patel, P. Bijan, Boris Motik (Eds.), OWL 2 Web Ontology Language: Structural Specification and Functional-Style Syntax, W3C Recommendation, 2009.
[32] Microsoft Corporation, DCOM Technical Overview, 2010.
[33] Object Management Group, Catalog of OMG CORBA®/IIOP® Specifications, 2010.
[34] R. Fielding, R. Taylor, Principled design of the modern web architecture, ACM Transactions on Internet Technology 2 (2002) 115–150.
[35] Crossbow Technology, Inc., (2010) URL: "<http://www.xbow.com>".
[36] SunSpot. Sun Microsystems, Inc., (2010) URL: "<http://www.sunspotworld.com/>".
[37] Arduino, (2010) URL: "<http://www.arduino.cc/>".
[38] WaspMote, Libelium Comunicaciones Distribuidas S.L., (2010) URL: "<http://www.libelium.com/>".
[39] W. Munawar, M.H. Alizai, O. Landsiedel, K. Wehrle, Dynamic TinyOS: Modular and Transparent Incremental Code-Updates for Sensor Networks, in: Proceedings of IEEE International Conference on Communications, 2010, pp. 1–6.
[40] A. Dunkels, B. Gronvall, T. Voigt, Contiki – a lightweight and flexible operating system for tiny networked sensors, in: Proceedings of 29th Annual IEEE International Conference on Local Computer Networks, 2004, pp. 455–462.
[41] Qing Cao, T. Abdelzaher, J. Stankovic, Tian He, The LiteOS Operating System: Towards Unix-Like Abstractions for Wireless Sensor Networks, in: Proceedings of International Conference on Information Processing in Sensor Networks, 2008, pp. 233–244.
[42] J.F. Martínez et al., Trade-Off Between Performance and Energy Consumption in Wireless Sensor Networks, in: Proceedings of 2th International Workshop on Self-Organizing System, 2007, pp. 264–271.
[43] He Tian, J.A. Stankovic, L. Chenyang, T. Abdelzaher, SPEED: a stateless protocol for real-time communication in sensor networks, in: Proceedings of 23rd International Conference on Distributed Computing Systems, 2003, pp. 46–55.
[44] F. Emad, L. Chang-Gun, E. Ekici, MMSPEED: Multipath Multi-SPEED Protocol for QoS Guarantee of Reliability and Timeliness in Wireless Sensor Networks, IEEE Transaction on Mobile Computing 5 (2006) 738–754.
[45] Internet Engineering Task Force (IETF), IPv6 over Low power WPAN (6lowpan), 2010 URL: "<http://datatracker.ietf.org/wg/6lowpan/charter/>".
[46] A. Dunkels, Full TCP/IP for 8-bit architectures. in: Proceedings of the 1st international conference on mobile systems, applications and services, New York, 2003, pp. 85–98.
[47] Internet Engineering Task Force (IETF), Chopan - Compressed HTTP Over PANs, 2009, URL: "<http://tools.ietf.org/html/draft-frank-6lowpan-chopan-00>".
[48] OASIS, Web Services Dynamic Discovery (WS-Discovery) Version 1.1, 2010.
[49] Proposal by anonymous author, Service Mapping Description Proposal, (2008) URL: "<http://groups.google.com/group/json-schema/web/service-mapping-description-proposal>".
[50] Crossbow Corporation, Inc. TelosB specification, 2010.
[51] Crossbow Corporation, Inc. Imote2 specification, 2010.
[52] GSN Corporation, Inc. PATROL-701 PIR device, 2010.

Iván Corredor is a Ph.D. candidate and a researcher of the Department of Engineering and Telematics Architectures (DIATEL) at Technical University of Madrid (UPM). He received the B.Sc. degree in Telecommunications Engineering in 2007 and M.Sc. in Service Engineering for Information Society in 2009, both at the UPM. His research interests include design of service-oriented architectures for embedded networks and knowledge management for the Internet of Things.

Dr. José F. Martínez received his Ph.D. degree in Telematics Engineering from the Technical University of Madrid (UPM), Spain in 2001. He is an Associate Professor in the Department of Engineering and Telematics Architectures (DIATEL) of the same University. His main areas of interest and expertise are new services for wireless sensor networks, service management, advanced software architectures, component-based distributed applications, and telematics services for next-generation Internet.

Miguel S. Familiar is a graduate student at Technical University of Madrid (UPM). He received the B.Sc. degree in Telecommunications Engineering with honours in 2009, and is currently working towards his M.Sc. in Telematics Engineering at the Universidad Carlos III de Madrid. He is member of the Department of Engineering and Telematics Architectures (DIATEL) at the UPM. His research interests include next-generation networks and services, including service-oriented computing, and middleware systems for mobile, ad-hoc sensor and dynamic networks.