aWESoME: a Web Service Middleware for Ambient Intelligence

Thanos G. Stavropoulos, Konstantinos Gottis, Dimitris Vrakas and Ioannis Vlahavas

Abstract. This work presents a Web Service Middleware infrastructure for Ambient Intelligence environments, named aWESoME. aWESoME is a vital part of the Smart IHU project, a large-scale Smart University deployment. The purpose of the proposed middleware within the project is twofold: for one, to ensure universal, homogeneous access to the system's functions and secondly, to fulfill functional and non-functional requirements of the system. Namely, the infrastructure itself should consume significantly low power (as it is meant for energy savings in addition to automations), without compromising reliability and fast response time. The infrastructure should enable fast and direct discovery, invocation and execution of services. Finally, on hardware level, the wireless sensor and actuator network should be optimally configured for speed and reliability as well. The proposed solution employs widely used web open standards for description and discovery to expose hardware and software functions and ensure interoperability, even outside the borders of this university deployment. It proposes a straightforward method to integrate low-cost and resource-constrained heterogeneous devices found in the market and a large-scale placement of servers and wireless sensor networks. Different server hardware installations have been evaluated to find the optimum trade-off between response time and power consumption. Finally, a range of client applications that exploit the middleware on different platforms are demonstrated, to prove its usability and effectiveness in enabling, in this scenario, energy monitoring and savings.

Keywords. Web Services, Real-time and Embedded Systems, Ubiquitous Computing, Wireless Sensor Networks

1 Introduction

The future of computing may rely on an emerging new paradigm known as Ubiquitous or Pervasive Computing (UbiComp, PerComp). Also referred to as the third wave of computing, the term ubiquitous computing was originally coined by Mark Weiser (Weiser 1999). Ubiquitous systems are able to perceive user needs and interface with them in an intuitive way. Physical objects of everyday life become mediums for users to interface with ubiquitous systems. The use of wearable computing devices takes the place of desktop computers. As a result, the computer fades in the background and relieves the user from constantly requiring his full attention. The vision of Ambient Intelligence (AmI) slightly extends these ideas by incorporating intrinsic intelligence in pervasive systems. Ideas and methodologies of Artificial Intelligence are adapted to pervasive systems to provide intelligent automations enabling an even more hassle-free and non-intrusive user experience.

Meanwhile, another emerging paradigm, which happens to be tightly linked with AmI systems, is Service Oriented Computing (SOC). Service orientation is in essence the practice of exposing a system's methods for users or applications to consume in a universal way. A SOC-compliant infrastructure of certain roles is the Service Oriented Architecture (SOA). SOA is enabled by well-defined web standards and technologies, as the web also moves towards service-orientation. These technologies promote the reuse of existing implementations and remote collaboration between different individuals or enterprises. In AmI, SOA has proved to be of great value, as it offers both the necessary abstractions for high-level sophisticated AmI applications and remote calls. SOA has become so interconnected with AmI that it is almost considered as a requirement. Meanwhile, devices found in AmI systems become more compact and woven into everyday-life objects. Services are used to expose their data and functions and form the so-called Internet of Things.

Focusing on the interoperability layer, the primal obstacle met in Ambient Intelligence is manipulating the heterogeneous devices that provide basic functionality. Due to advancements in the industry, a wide variety of devices suitable for enabling the vision of AmI are now available at an affordable cost. Even optimal communication protocols have been designed to enable efficient and energy-saving dynamic networks of smart devices in smart homes, e.g. the wireless ZigBee[1] and Z-Wave[2] and the Power-Line-Communication-based INSTEON[3] and X10[4]. On the other hand, the industry has not and does not intend to converge on common standards concerning device manipulation. The alliances of ZigBee and Z-Wave have standardized device classes and operation protocols for devices compliant with them, breaching the biggest part of the interoperability boundaries. UPnP[5] has also become widely spread in both home entertainment and AmI applications, as it defines classes and communication standards for interoperability between different major manufacturers of multimedia systems.

Nevertheless, a middleware is still needed to provide interoperability between the different clusters of devices. Even if Z-

- *T. G. Stavropoulos is with the Aristotle University of Thessaloniki and the International Hellenic University. E-mail: athstavr@csd.auth.gr.*
- *K. Gottis is with the Aristotle University of Thessaloniki. E-mail: kgottis@csd.auth.gr.*
- *D. Vrakas is with the Aristotle University of Thessaloniki and the International Hellenic University. E-mail: dvrakas@csd.auth.gr.*
- *I. Vlahavas is with the Aristotle University of Thessaloniki and the International Hellenic University. E-mail: vlahavas@csd.auth.gr.*

[1] The ZigBee Alliance: http://www.zigbee.org/
[2] Z-Wave: http://www.z-wave.com
[3] INSTEON: http://www.insteon.net/
[4] X10: http://www.x10.com
[5] UPnP Forum: http://www.upnp.org/

Wave, ZigBee and UPnP networks of devices can each be handled in a uniform way, they need to be unified under a common API that applications can exploit. This common API also needs to include individual devices (i.e. those that do not belong to a standard family) that naturally still exist in the market, and function in unique ways. Furthermore, a manufacturer or alliance sometimes provides specific tools that may enforce constraints e.g. platform-dependent APIs. Even worse, they may provide no API at all, which is almost always the case with individual devices in the market. However, even the alliance-compliant devices often are not platform independent out-of-the-box. E.g. there is a number of different Z-Wave APIs in different programming languages that need wrappers to be utilized. All in all, a middleware can be seen as a set of drivers for different protocols that plays the role of a uniform abstraction layer between the application layer and the heterogeneous, platform-specific hardware layer.

The middleware layer, thus, has to ensure, in principle, universal access and data homogeneity for the application layer. One of the leading approaches for the first is employing a Service-Oriented Architecture, which spawns many more side benefits. Well-defined and standardized by the W3C[6] technologies promise to deliver flexible uniform access and interoperability on syntactic level, while benefiting from remote access over the Web. The leading description language and W3C standard, WSDL (the Web Service Description Language)[7], already is widely adopted in both industry and research. These XML-based descriptions distinguish and define the components of a Web Service in service operations, which can have incoming and outgoing messages which in turn can contain various data types. Ultimately the service can have a number of bindings which help locate and invoke the service by declaring its endpoints and communication protocol (usually HTTP or SOAP). WSDL sufficiently type-defines services on syntactic level which means services can be selected (i.e. matched) and composed based on syntactic criteria.

Except syntactic interoperability, uniform and remote access, the Service Oriented Architecture also introduces ways to enhance the discovery process. A Service Broker is responsible for the registering and provisioning of services to clients. The Service providers (i.e. Servers) first have to register the services that they provide on the Service Broker. The clients can browse and select the service they need from the broker's list. After they select a service, the clients directly interface with the selected service provider to invoke the service and the broker's intermediation ends. The need for a service broker is eminent due to the fact that services are essentially distributed (i.e. provided by different servers). It would otherwise be impractical for Clients to maintain an up-to-date list of all Provider locations. After all, in AmI environments, that list is dynamically ever-changing. Providers can be portable (even wearable) devices that dynamically enter and leave the system (willingly or simply collapsing).

In this paper, we present a Web Service Middleware named aWESoME which is based on a Service-Oriented infrastructure to better enable AmI applications in general and energy savings in particular. aWESoME is part of the Smart IHU (International Hellenic University) project (Stavropoulos 2010), whose goal is to develop and deploy of an AmI-oriented system that enables automation, user-comfort and last but not least energy efficiency and savings. Smart IHU entails a large-scale deployment of various devices, such as sensors of environmental conditions and actuators for switching devices or manipulating the ICT infrastructure. The devices naturally come from various manufacturers, are low-cost, platform dependent and resource-constrained (i.e. have little or no processing power and/or memory). aWESoME integrates heterogeneous devices, and provides the required abstraction layer for universal access and data homogeneity. It follows the SOA guidelines for descriptions and enhances service provisioning and discovery based on the WS-Discovery standard, to support dynamicity in a large-scale distributed AmI environment. Its internal structure allows for scalability and extensibility to more software and hardware services. Different server installations are tested to find the optimum trade-off between their own power consumption and speed (service response time). The optimum server and device deployment, given the coverage of the devices is also showcased. Except for the energy savings of the platform itself, some client applications are shown to demonstrate the middleware's efficiency and usability on desktop, web and mobile platforms.

This article is structured as follows: the next section surveys related work, i.e. other middleware designed for Ambient Intelligence. In the third section the aspects of the proposed middleware are given in detail. In the fourth section, the exact deployment of devices and aWESoME components at the Smart IHU environment is presented. Experiments and results, concerning the platform's performance and energy efficiency, are presented on the corresponding section. The final two sections present future work and conclusions from this work respectively.

2 Related Work

The most relevant of works is the Hydra middleware[8], later renamed to LinkSmart middleware. The Hydra middleware is a European project for the development of service-oriented software to expose heterogeneous device functions in a universal way and target various Ambient Intelligence domains including home automation, healthcare and agriculture. Hydra supports a wide variety of devices that can be controlled by an external communication interface such as Bluetooth, ZigBee, RF, RFID, USB etc., called Hydra-enabled devices. To integrate supported device instances, Hydra generates the required Hydra software components and web services in each case. It follows a model-driven architecture based on semantic models (ontologies) so that the middleware components for each device are automatically generated based on that meta-data (Eisenhauer 2009). The Hydra project ideally targets devices that can embed Hydra web service components and hence do not require an external server to manage them. Another approach to unify home automation protocols, over OSGi, is Home SOA (Bottaro 2008).

Hydra, Home SOA and aWESoME share a same goal: to ex-

[6] The World Wide Web Consortium: http://www.w3.org/

[7] The Web Service Description Language 1.1: http://www.w3.org/TR/wsdl

[8] The Hydra Middleware online: http://www.hydramiddleware.eu

pose various smart devices universally via Web Services and enable Ambient Intelligence environments. However, only a few of the proposed system's devices are Hydra-enabled (and even fewer by Home SOA), as it is not of course possible to support every device family in the market. Additionally, devices that can embed the services themselves are very hard to find in the market. We rather target affordable devices and employ a board microcomputer to embed the corresponding services for them (to preserve compactness, affordability and support each device equally) and present performance tests. Finally, we target a Smart University domain specifically and focus on providing additional software-services and energy-saving applications.
Semantic Spaces (Hansen 2004) is a similar, much older, approach to middleware. In this system so-called wrappers are in fact UPnP discoverable services that expose various environmental sensors and software services (CPU utilization, weather etc.). Like Hydra, this system also utilizes ontologies as a knowledge representation format. Context information or in other words the world's representation is stored in an ontology designed for the purpose. The ontology naturally enables reasoning and querying that data. In our proposed solution, WSDL services and WS-Discovery are used instead, as the leading, widely used, open standards. Similar, although more, hardware and software services are provided (i.e. smart plug actuators, embedded services and Z-Wave support).
Many more systems use ontologies either as a knowledge base or as complex service descriptions. There are even methods to generate ontologies suitable for services by the service descriptions themselves. We do use an ontology designed for this system as a knowledge base and indirect service descriptions. However, this is performed on application level to preserve low complexity and usability of the middleware for plain-WSDL industry and research clients, and is outside the scope of this work.
Emi^2 (López-de-Ipiña 2006) proposes an infrastructure quite similar to UPnP/DLNA, where each component has a distinct role: there are Emi^2 Servers, Objects, Players etc. Each user and each Object (in other words Service) are represented by interacting agents. The objects can be discovered but have to be downloaded to be executed, unlike in aWESoME and most approaches. The Emi^2 platform is kind of restricted within its own boundaries due to its implementation (no interoperability with industry services).
MEDUSA (Davidyuk 2011) is another web service middleware that enables transparent use of functions within the Smart Environment. It employs mobile phones which are equipped with RFID-tag reading capabilities. The phones are used to scan cards that represent services and form service compositions. As a result, the system provides a practical physical interface for users. MEDUSA is based on the AmIi middleware (Georgantas 2010), which defines a model and an XML-language for semantic interoperability of services. AmIi focuses on semantic specifications. It re-models information common on known models (i.e. OWL-S) and maps to BPEL, SAWSDL and more, by extending WSDL as a specification language. However, this model may be too specific, and functional only within the boundaries of AmIi compliant clients. Hence, it needs to be adopted by a wider range of the community. On the other hand, WSDL (and even SAWSDL) is simpler and widely adopted in research and industry. MEDUSA and AmIi target almost exclusively multimedia devices. The aWESoME middleware supports a range of embedded software, sensor and actuator services, while multimedia devices are not a priority for the energy-efficiency task.
Polisave (Chiaraviglio 2010) is a software system developed by the Technical University of Torino to manage the IT infrastructure. Existing technologies such as Wake On Lan, and OS-based shutdown scripts are employed to enable users assign power on and power off schedules for their computers at the university in order to save energy and reduce energy bills. The process is carried out via a web-application graphical interface. We employ the same existing techniques, such as Wake On Lan and OS-based shutdown, hibernate, sleep and restart but also integrate them as part of a Web Service middleware that provides more flexibility and availability to clients. Energy saving schedules are then implemented in independent client applications.
Some approaches focus on much different aspects of middleware. KASO (Corredor 2012) is a middleware also for Sensor and Actuator networks, but more oriented towards future internet and usage of services through the Cloud. B3G – SOM (Autili 2009) adapts mobile 3G device service to service-oriented middleware. UbiSOAP (Caporuscio 2008) is a different approach to middleware that addresses communications and optimizations on the physical layer. As mobile devices may host and/or consume services but lack computing power, delays are introduced. UbiSoap proposes that these devices function in multi-radio, and proxy servers function as bridges for the seamless integration of these devices.
A final category of middleware includes the ones that focus on performing automatic, intelligent composition of services. A complete survey of such work can be found on Stavropoulos 2011a. Some of these approaches are not oriented towards devices at all. They may even not include their own services but rather use existing ones. Scooby (Robinson 2004) is a script-like middleware abstraction language for defining service compositions. MySIM (Ibrahim 2009) is a middleware that primarily translates existing OSGi and WSDL services into an internal representation form, and spontaneously (i.e. without user request) composes the services in pairs of two. The new services are exposed on MySIM's mathematic description model, which is a set of inputs, outputs and semantic annotations for them. However, MySIM goes from usable descriptions (OSGi, WSDL) to completely non-interoperable ones: only MySIM users can invoke those services. Many more similar approaches to service oriented-middleware target autonomous service composition (Ben Mabrouk 2009, Lagesse 2010, Park 2011).
All in all, compared to existing approaches, aWESoME focuses on providing a wide range of hardware and software-based services, syntactic interoperability, simplicity, ease-of-use, fast response time and ultimately energy-savings. Numerous devices of the ZigBee, Z-Wave and RF protocols, not currently supported by the state-of-the-art, are integrated into aWESoME. In addition, different hardware has been tested. Among it, a board microcomputer has been tested and employed as a compact server to preserve low power consumption and cost, in a distributed multiple-server deployment. Finally, the WSDL and WS-Discovery open standards are used versus other approaches disregarding constraining internal representations. Knowledge representation databases of any form (ontology or other) and

composition of services are disregarded on middleware level and left out for the application level. Finally, the aWESoME middleware platform introduces significant power consumption reduction at the Smart Building, as it is utilized by energy monitoring and saving applications.

3 The aWESoME Middleware

The middleware introduced in this work[9], serves as the essential middle layer between the diverse hardware on the bottom layer and various applications on the top layer. Following guidelines in the field, the middleware provides the required level of abstraction to manipulate platform-specific, heterogeneous smart devices found in the market. In addition, implemented Web Services expose their functions universally and over the Web. A Service Oriented Architecture is set up to guarantee even more flexibility, currency and dynamicity in a large scale distributed server environment.

To ensure the middleware's extensibility to more hardware bundles, drivers are developed for each supported bundle separately. All the driver modules are engineered completely from scratch, have no dependencies and interface directly with the devices. They all presently happen to be implemented in Java, which makes them suitable for all platforms. Indeed, PCs, netbooks and Linux-powered board computers are installed in our deployment. The drivers form a first sub-layer of the middleware that can be thought of as the integration layer, presented in detail in the next subsection.

On top of the integration layer, a service layer that contains WSDL Web Services wraps each driver to expose its function over Web. The services are also presently implemented in Java. Each service is responsible for a certain device group most of the time, and its operations expose each of its supported functions. That way there are few services and many operations (instead of many services and few or one operation per service). Additionally, there are self-contained services that do not correspond to devices and do not need a driver to function. A comprehensive list of services and operations is presented on the corresponding subsection.

Following the service oriented paradigm, a service broker is employed to register and provision services scattered across distributed servers. Servers are set up by installing the essential aWESoME components for the case, i.e. the drivers and services needed. Subsequently the newly provided services are registered at the service registry. The potential service clients, then, can browse and locate providers from a list of services. The registry infrastructure is presented at the corresponding subsection.

A distributable version of aWESoME can be found online, containing some of the driver modules.

3.1 Integration Layer

The integration layer contains all the driver modules responsible for directly interfacing with the infrastructure's devices i.e. sensors and actuators. Each bundle of devices or device individuals has its own interface, communication and operation protocols. The driver modules are libraries for primitively manipulating the devices. They require no other out-of-the-box company software so they guarantee flexible deployment and swift operation. That way the middleware can first of all be decomposed and repackaged to contain the necessary drivers only. This compactness is needed for memory-restricted servers. Additionally, future support for more bundles can be added if necessary by adding suitable driver modules and services.

3.1.1 ZigBee Smart Plugs

Smart Plugs are devices for home-automation that can be attached to any electric appliance and allow both power state and power consumption readings and switching the appliance on or off. This functionality can be useful in home automation, energy-saving and generally any Ambient Intelligence scenario. We chose a commercial bundle of Smart Plugs[10] that form dynamic ZigBee mesh networks. This bundle also includes variants of the Smart Plugs that actually intersect cabling so that they can monitor and control appliances that cannot be plugged in (e.g. lighting, air conditioning). Fortunately, these variants have the exact same properties and can be manipulated in the same way as the original Smart Plugs, so they will not be considered separately from now on.

Each Plug network can handle up to around thirty nodes, managed by a gateway which is a Smart Plug itself. The gateway plug interfaces with a PC client using a USB stick, again over ZigBee. Our exact topology of Smart Plugs is presented on a dedicated section. However, the ZigBee data packets are encrypted and the devices have their own operation protocol.

Thus, the ZigBee-based protocol of the Smart Plugs had to be re-engineered, as a general-purpose open ZigBee library could not resolve the matter. The driver module developed can both form the suitable packets that perform each operation and parse back responses. Except from power consumption and power state information (i.e. on or off), the devices return additional data such as memory buffer information, firmware etc. Power consumption can also be used to calculate the energy consumption over a target period of time. The driver module is implemented in Java.

3.1.2 ZigBee Sensor Boards

The adopted Sensor Board bundle[11] serves to provide data about environmental conditions in various points on the university premises. It is comprised of four battery-powered sensor boards that each embeds a temperature sensor, a humidity sensor and a luminance sensor. The boards form a dynamic ZigBee network over a dedicated gateway that comes with the bundle. Unlike Smart Plugs, this gateway is equipped with WiFi and Ethernet so it can easily be connected to the local area network to transmit the collected data.

For simple clients, the bundle's own gateway would suffice to gather the data. However this is not the case in our system. Even though remote function is supported by the gateway (at least within the local area network), universal access and data homogeneity still needs to be ensured. WSDL Services do provide the desired syntactic interoperability.

[9] aWESoME online: http://lpis.csd.auth.gr/people/thanosgstavr/development.html#awesome

[10] Plugwise company: http://www.plugwise.com

[11] Prisma Electronics: http://www.prismaelectronics.eu

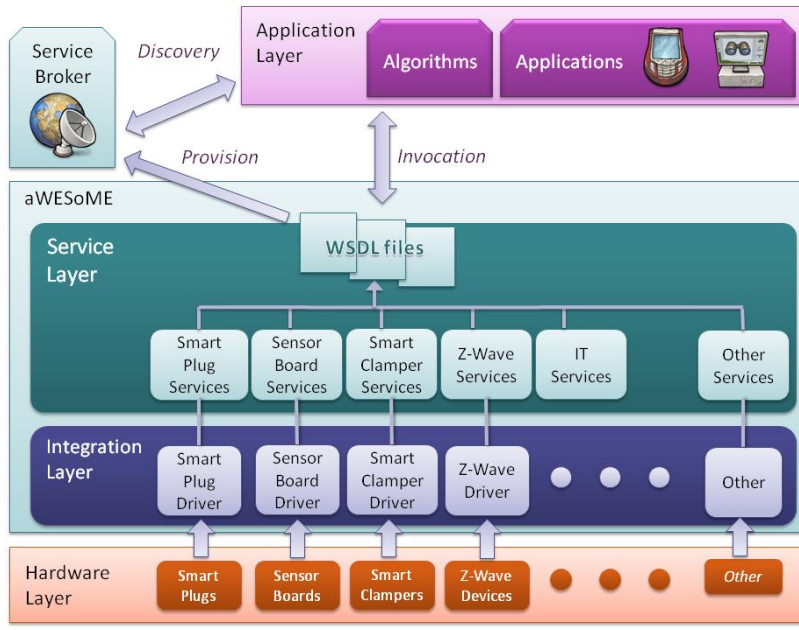

Figure 1. The aWESoME layers and components

As the gateway software cannot be modified, any computer in the local area network can access it and get the data using the bundle's software. However, this software happens to require complex and time-consuming setup every time. For the same reasons of speed and flexibility, the software was re-engineered to directly poll the gateway for data. Sensor data is ultimately transmitted over TCP/IP and no ZigBee library is required although this bundle uses no encryption on ZigBee packets. Finally, the Java Sensor Board driver can run on any Server on the local network and periodically or upon request return sensor data.

3.1.3 Smart Clampers

Smart Plugs fulfill the need for power consumption monitoring in small-scale i.e. per device. On the other hand, in the large-scale environment of Smart IHU, no conclusion is drawn for the total energy consumption of the university. Adding the consumption of Smart Plugs does not suffice and can be very misleading as many devices of massive consumption will always be left out (it is impractical to attach every single appliance to a Smart Plug).

Smart Clampers are used in home automation environments to measure the total energy consumption directly at the source, which is the main power supply. Smart Clampers usually do not plug in or intersect cables, but rather clip around cables and detect the current. However, the Smart Clampers do not suffice for a smart environment. The total energy consumption of the university is so massive that significant changes in power consumption cannot be detected in large-scale. In other words, numerous powered-on air conditioning units or lighting can go undetected. As mentioned above, Smart Plugs can also manage the power supply of individual devices, in small-scale. Thus, Smart Clampers and Smart Plugs are complementary and both need to be used.

In our deployment we chose to integrate two different commercial brands of Smart Clampers for tests and completeness. Both clampers function similarly. A transmitter is attached to the clips to wirelessly transmit data to a receiver attached to a USB port of a computer client. The first bundle's[12] receiver is also an LED monitor that shows readings. The other bundle[13] also includes an independent monitor.

The first bundle is the one used in practice (as two are redundant). It submits real-time data, and less often even historical data, in text (specifically XML) format, so that they can be directly parsed. However, the other bundle was found not to be reliable as it is prone to failure. The company's API was used but contrary to the official software, too many bugs showed up. That goes to show that only direct protocol re-engineering and interfacing with device ports can utterly be considered always reliable and fast (and not poorly supported APIs that naturally do not interest the industry).

3.1.4 Z-Wave devices

Z-Wave is another alliance protocol in home networking that sets standards and is supported by many manufacturers. It is also similar to ZigBee in terms of networking as it is also wireless and supports adequate range and many nodes per network. Z-Wave defines various device classes that are agreed upon across Z-Wave manufacturers. That ensures interoperability between devices and software client controllers within the Z-Wave network.

A great advantage of the Z-Wave family is the vast variety of smart devices provided. Z-Wave devices can control every aspect of a smart home ranging from multimedia controllers for home entertainment (TV, home theater etc.), to motors for garage doors and window blinds, home security systems (motion detection, alarms, smoke detection etc.), facilities (such as thermostats), and environmental sensors. There are also LED-screen controllers so that users can control the devices without a gateway, and USB PC interfaces for implementing PC client applications.

[12] CurrentCost: http://www.currentcost.com/
[13] OWL energy monitor: http://www.theowl.com/

For our implementation we chose some of those devices to enrich the variety of information and functions in our system. Namely, a motion and environmental sensor, an infrared sensor, a smoke detector and a USB PC interface for control. A Z-Wave library was incorporated in a Z-Wave driver module for aWESoME. Due to the extensible nature of the Z-Wave family, any other type of Z-Wave device can be added by making minor modifications to the driver.

3.2 Web Service Layer

The Web Service Layer contains all the services needed for the upper layer, i.e. applications, to interface with the middleware, and, in turn, with the system hardware. To provide functionality of the various devices, Web Services are implemented to wrap each device driver of the Integration Layer. The Service Layer also contains additional Services which do not wrap device drivers but are self-contained and software-based instead. All Services also return suitable faults when exceptions occur, so they can be handled according to client logic.

In reference to Figure 1, there are currently four services that wrap device functions and one service that does not. This categorization was preferred having in mind to preserve a small number of services and many operations for each service, instead of many services and few or even one operation per service. Services are in fact groups of inter-related operations.

3.2.1 Smart Plug Service

The Smart Plug service operations are listed on Table 1. Most service operations are based on the Smart Plug driver functions, and require the target Plug ID (referred to as the Plug's MAC Address) as input. Each instance of a Smart Plug Service manages a network of Plugs. Hence, a client has to know which Service instance to invoke for the desired target Smart Plug.

To enable various application scenarios, switching on and off is supported in two forms, e.g. Switch On, Switch Off and SwitchOnOff. Various information provided by the hardware, e.g. firmware version, internal memory addresses, internal clock etc. can be obtained through the getInfo operation.

For returning Power Consumption information, a basic operation, getPulses, returns Pulse information directly from devices. This raw information in most cases has to be converted into Watts, which is supported by the convertPulsesToPower operation. To provide a straightforward solution, the getPower operation does both.

Likewise, sometimes pulses can be fine-tuned to specific device properties. These properties are constant and can be obtained through the getCalibration operation. Then these parameters can be simply inserted into a formula and fine-tune pulse information. The contents of the formula and the meaning of these properties are outside the scope of this work. The correctPulses operation gets a power measurement in pulses and an array of the calibration information and returns the corrected value. Again, to provide an alternative straightforward solution, the getCorrectPower operation is a composite service that does all four: it gets Pulses, gets Calibration info, corrects Pulses based on Calibration and finally converts Pulses to Watts. Client applications can invoke once and store Calibration data, and use it every time for corrections. Otherwise, the complete but more time-consuming solution of getCorrectPower can be used e.g. by human clients.

Table 1. Smart Plug Service Operations

Operation	Description	Description
SwitchOff	Plug MAC	-
SwitchOn	Plug MAC	-
SwitchOnOff	Plug MAC, Power State	-
getStatus	Plug MAC	Power State
getPulses	Plug MAC	Power (Pulses)
convertPulsesToPower	Power (Pulses)	Power (Watt)
getPower	Plug MAC	Power (Watt)
getCalibration	Plug MAC	Calibration
getCorrectPower	Plug MAC	Power (Watt)
correctPulses	Power (Pulses), Calibration	Power (Pulses)
getInfo	Plug MAC	Info

Table 2. Sensor Board Service Operations

Operation	Description	Description
getNodeCount	Board MAC	Number of nodes
getMACs	Board MAC	MAC Array
getLuminance	Board MAC	Light Level (cd/m^2)
getHumidity	Board MAC	Humidity %
getTemperature	Board MAC	Temperature (Celsius)

Table 3. Smart Clamper Service Operations

Operation	Description	Description
get3PhasePower	Transmitter ID	Timestamp, Three phase power (Watt)
get3PhasePowerSum	Transmitter ID	Timestamp, Three phase sum (Watt)

Table 4. Z-Wave Service Operations

Operation	Description	Description
getNodeIDs	-	Node ID array
getBattery	Node ID	Battery %
getAwake	Node ID	Node awake state
getVersion	Node ID	Version
getSpecifications	Node ID	Specifications
getMotion	Node ID	Motion Detection value
getPIRMotion	Node ID	Infrared Motion Detection
getTemperature	Node ID	Temperature (Fahrenheit)
getLuminance	Node ID	Light level (1-40)
getSmoke	Node ID	Smoke Detection value

Table 5. IT Service Operations

Operation	Description	Description
wakeComputer	MAC, IP Address	-
pingIP	IP Address	IP online
findIP	MAC Address	IP Address
findMAC	IP Address	MAC Address
wakeComputerIP	IP Address	-
wakeComputerMAC	MAC Address	-
shutdownComputer	IP Address	-
restartComputer	IP Address	-
putComputerToSleep	IP Address	-
hibernateComputer	IP Address	-
getServerCPU	-	% CPU Usage
getServerCPUAvg	Interval (sec)	% CPU Usage

3.2.2 Sensor Board Service

All the Sensor Board Service operations are actually wrappers of the Sensor Board driver's functions. The Sensor Boards also have designated MAC Addresses which are used as operation input, to distinguish the boards. The operations shown on Table 2 are self-explanatory as each one gets the corresponding environmental parameter from the desired target Sensor Board.

3.2.3 Smart Clamper Service

Likewise, the network of Smart Clampers is accessed via the Smart Clamper Service Operations, shown on Table 3. Each Transmitter in the Smart Clamper network can have up to three Clampers attached, that measure three-phase current (one phase each). To get the Power Consumption information, the get3PhasePower operation has to be provided with a Transmitter ID. For better enabling clients, another version of this operation additionally adds up the three readings and returns the total consumption for a Transmitter.

3.2.4 Z-Wave Service Operations

The Z-Wave Service supports an initial set of devices but can later be expanded by adding more. The operations currently support the existing device classes of the platform: a PIR (Passive Infrared) Motion Detector, Smoke Detector and a Multisensor for Temperature, Luminance and Motion Detection. The supported Web Service operations are shown on Table 4. In addition to the operations that get sensor readings, there are methods to get an array of the nodes in the Z-Wave network, whether a node is awake or not and battery level for each node.

3.2.5 IT Service Operations

This service is currently the only one that does not relate to a driver module i.e. has a hardware-counterpart. Its purpose is to inform about and manipulate the IT equipment in the setting by using existing technologies. The Smart Plug operations, SwitchOn and SwitchOff are completely inappropriate for IT equipment. Computers shouldn't be abruptly shut down, and (usually) do not power on as power supply is provided (Switch On operation). This Service employs WakeOnLAN and OS-based techniques to carry out these tasks, as well as give additional information on IT equipment.

First of all, the wakeComputer operation implements a WakeOnLAN technology method to wake any PC in the LAN from Sleep, Hibernation or Off state. This is carried out by sending the standard so-called Magic Packet for WakeOnLAN that contains the target terminal's IP Address and MAC Address (for Ethernet interface only). Hence, the Addresses have to be given as the operation's input. To facilitate that process, an IP and MAC table is constructed at server-side. The operations findIP and findMAC return the IP Address and the MAC Address of a terminal based on that table. There are also operations that include that process to further facilitate clients. Possible compositions can automatically be generated on application level.

Unfortunately, while Wake On LAN is supported at hardware level, there is no equivalent for the shutdown process. On the contrary, the shutdown process is OS-dependent. The operations provided, contain OS-supported scripts to shutdown, restart, put to sleep and hibernate a computer within the LAN, given its IP Address. That enables better energy management of the IT infrastructure. Finally, the getServerCPU returns the percentage of Server CPU load for informative purposes. All operations are listed on Table 5.

3.3 Service Broker

The service oriented infrastructure of aWESoME also includes the Service Broker subsystem which is responsible for registering and provisioning the available Services. The aWESoME SOA is comprised of the various distributed aWESoME Servers that provide different instances of the aWESoME Services, Service Clients which can be human or software agents, and a Service Broker. Clients primarily browse the Service Broker's list of Services, which the Service Providers have previously registered. Overall, aWESoME's Service Broker module intermediates between Servers and Clients.

The purpose of Service Broker is twofold: to maintain a unique list of the distributed services in the environment and to refresh that list, removing Services that go offline i.e. leave the environment or collapse. The Service Broker module is based on the WS-Discovery standard of the WS-* stack. It provides the essential wrapper interface to the WS-Discovery Java implementation. When a Server deploys a new Service, it registers with the Service Broker module. That way, Clients do not have to maintain the list of various distributed (different IPs, different bindings) Services or ping them themselves. Otherwise, the Services would have to either explicitly have the same IP base (e.g. with the use of proxies) or new Services would have to register with a list of Clients (which should dynamically change).

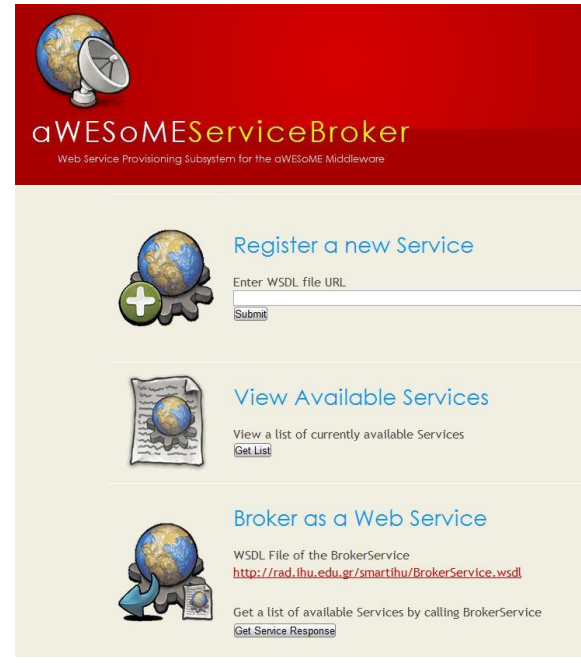

Figure 2. Service Broker Web interface

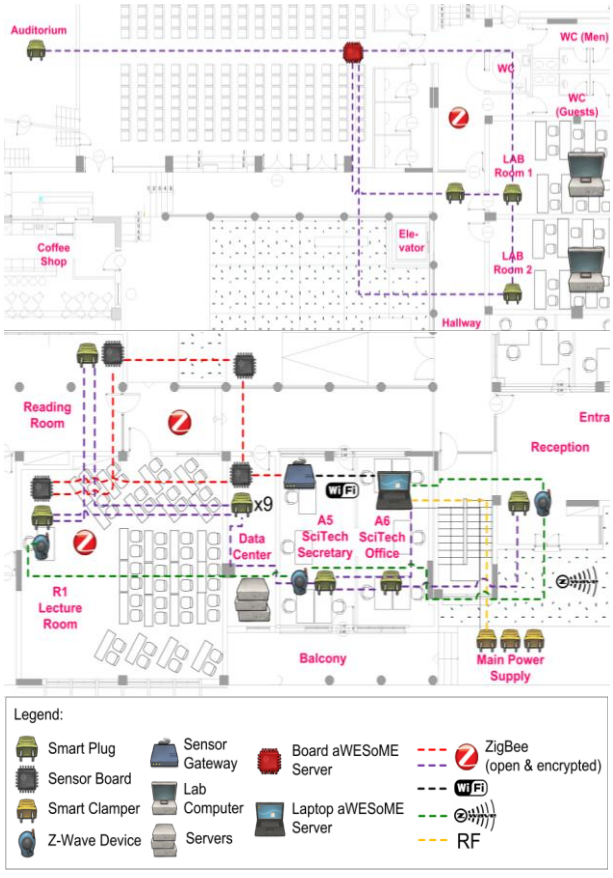

Figure 3. Current Smart IHU Deployment

Finally, the Service Broker implements a Web Service interface as well so that it can be easily used by external Clients. The Broker Service is also listed within the list, and returns a list of the currently registered and available WSDL Services. All the operations of Service Broker can be carried out via a Web UI, to facilitate the process of quickly deploying aWESoME. For compactness, all functions are presented at the home page, which is depicted on Figure 2.When the Server administrator uploads an aWESoME Web Service, he can enter the Service Broker portal and enter the WSDL URL of the Service. Human Clients or testers can directly view the Service list in HTML (which is constantly updated) by using the second function "Get List". Finally, the WSDL URL of the Broker Service itself is provided, so that software agents that exploit the Broker can be developed. For demonstration purposes a WSDL Broker Service Client is built into that page and can be invoked by using the last option "Get Service Response", which returns a SOAP message containing the Service list.

4 Smart IHU Deployment

aWESoME is in fact a set of drivers and services as described in the previous sections, so that it can support numerous smart devices and provide useful services in an Ambient Intelligence environment. A system supported by aWESoME can be deployed in numerous ways, using all sorts of different PC-Servers and deploying in each, the driver and/or service components needed.

This section presents the actual deployment of aWESoME at

Figure 4. Smart Plugs attached to the Data Center

the Smart IHU environment which was, after all, the motivation for building the middleware. The instantiation takes place at the School of Science and Technology of the International Hellenic University. At this stage, devices cover one floor, the ground floor of Building A, until the deployment is extended with more devices of the same (already supported) or different manufacturer/protocol.

To cover the whole floor, one Server was not sufficient. In order to preserve low energy consumption, cost and compactness, a board computer was chosen and set up as an aWESoME server. The final deployment verifies the fact that the topology can be extended in space with more servers of different kinds to cover more floors and buildings.

An overview of the ground floor, devices and servers is shown in Figure 3. The most interesting locations to place devices are fortunately located on this floor: the reception desk equipped with a TV and security camera, the SciTech Secretarial and Academic Assistant Office, the Data Center (Server room), Lecture Room 1, two PC-Labs and the Auditorium.

The first constraint for server placement arises from the location of the main power supply which is located at the basement, under the reception. Three sets of three Smart Clampers (one for each Phase) are placed at the main power supply. The first two sets measure the buildings total power supply and the third measures the Data Center's Power supply. A Transmitter for each set has to communicate with the Smart Clamper receiver via RF, so the receiver has to be placed relatively near. The first server is hence placed at the SciTech Office (which is also a secure place) and attached to the Smart Clamper receiver.

Evidently, this Server is assigned to cover the south part of the floor. Smart Plugs are attached to the screen and security equipment at the reception, lights, computers, fax machines, printers and heating of the SciTech Office and Secretary, to each of the nine servers in the Data Center as shown on Figure 4, to lighting and projectors in Lecture Room 1 and to the light-

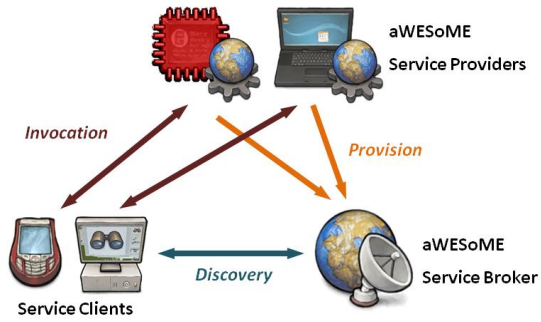

Figure 5. Broker, Provider and Client communication in current deployment

ing of the Reading Room. Sensor Boards are placed to measure environmental settings at the Reading Room, Lecture Room 1, the Data Center and outside (at the yard). The gateway-Plug is placed near the laptop server and the rest of the Plugs are distributed as evenly as possible around that point. The gateway is placed at the SciTech Office for security but can be moved anywhere in the building (as long as there is WiFi coverage). Although computers can be managed from any server in the LAN, the laptop Server, being the most powerful in the setup, is assigned to host IT Services and manage the IT infrastructure.

The Z-Wave nodes (a Smoke Detector, Motion Detector and a Multi-sensor) are placed arbitrarily on the south part of the floor. As these devices are currently few, they can be placed to serve various scenarios and then change position. E.g. the motion detector is mostly useful for detecting people at the lecture room, or a person entering the building at the reception. Various scenarios are implemented in applications and are outside the scope of this work.

The north part of the floor is covered by a second server. Unfortunately the devices themselves or their gateways cannot host their own Web Services, so aWESoME servers are required in their vicinity, usually at the center of each mesh network. In that case, a laptop is not necessary for testing or monitoring purposes, so a board computer was chosen to minimize size and power consumption. From a range of board servers, the one with the lowest memory, processing power and consumption was chosen[14]. Apparently, this is the most-lightweight computer that can host aWESoME services: it features an Atmel ARM9 at 400 MHz CPU module, and 64MB of RAM, which is just enough for hosting the services. After testing this extreme choice, any other more capable board server could offer improvements in performance if the power requirements are met. That set up also proves the extensibility of the aWESoME deployment, as any number of servers can be set up, inflicting no further delay to the system. Finally, Figure 5 shows the interaction of the two servers, the Service Broker and potential Clients, for instance desktop or Smartphone.

The north part is covered with Smart Plugs in different rooms as well, but can just as well contain any other kind of the supported devices. As long as device networks are moved as a whole, the board server contains the necessary components to provide services for the other bundles. In fact, as Sensor Boards interface through WiFi, it can also provide Sensor Board Services, but hopefully the laptop server can relieve the board server from that task as well. The same goes for the IT Services, as Computers can be managed from anywhere in the LAN. Finally, Smart Plugs are again attached to the lighting in projectors in the Auditorium, and lighting of PC Labs and the Hallway.

5 Application layer

This section presents external clients of the aWESoME middleware to prove its purpose, usability and effectiveness. aWESoME interfaces with the upper layer (application layer) through universal well-defined Web Services described in WSDL. Complementary to the WSDL files, a provisioning subsystem is used to publish and locate the distributed Service Providers. It also guarantees the currency of the available services, as new services are dynamically added and obsolete services are removed. Two clients are showcased here: iDEALISM which is a desktop application that provides a GUI for aWESoME services and added-value functionality, and PlugDroid which is a Smartphone application for the Android platform that offers similar functionality. Finally a web portal collects all sensor data for viewing purposes only. The differences between these client applications are summarized on Table 6.

5.1 Desktop Application – iDEALISM

The simplest and most straightforward client for the aWESoME middleware is the desktop application named iDEALISM (Stavropoulos 2011b). iDEALISM is directly compatible with the newer versions of the aWESoME functions, as they are WSDL-based. That goes to show the extensible character of both aWESoME and iDEALISM.

iDEALISM was designed from an energy-saving perspective. It serves two purposes: to present a usable GUI for aWESoME services and promote energy savings at the IHU premises. As

Table 6. Application functionality

Functionality	iDEALISM	PlugDroid	Web Portal
Platform	Java	Java, Android SDK	PHP
Store Service List	Locally	Locally	Globally
List of Services	Selective	Selective	Complete
View per Type	✓	✓	✓
View per Room	✓	✓	✓
Form Service Groups	✓	✓	×
Manually Invoke Services	✓	✓	×
Smart Plug Services	✓	✓	✓ (Read Only)
Sensor Board Services	✓	✓	✓
Smart Clamper Services	✓	✓	✓
Historical Data Charts	✓	×	✓
Local function	✓	×	×
QRCode scanning	×	✓	×

[14] Foxboard by Acme systems: http://www.acmesystems.it/?id=FOXG20

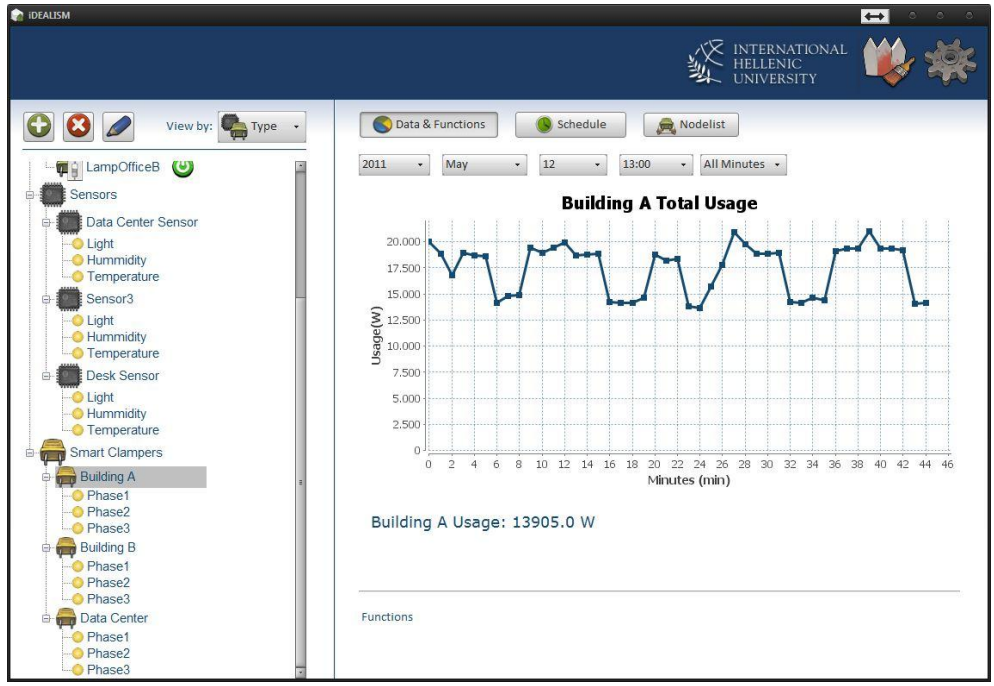

Figure 7. iDEALISM view of a Smart Clamper Service

all WSDL Web Services, aWESoME services lack a user-friendly interface for invoking the services. Not only do they exchange SOAP messages, which are difficult and impractical for non-expert users to form and to parse back, but the exchanged data itself often have no practical meaning to the user. The latter does not know what to enter to get the desired output or how to interpret the results. iDEALISM presents a user-friendly interface (Figure 7) that helps maintain a list of services, organize and invoke them periodically or per request. As desktop users do not need to be burdened with a complete list of services, they can manually enter and store the ones they are interested in. To better handle the service list, the users can form service groups and view the list per device type, room or group. All device-related services can be invoked by the iDEALISM's Web Service Client. Sensor-related Services are periodically invoked and response data is stored to form historical data charts. The actuator services can be invoked per user request. Especially operations of devices in Rooms and Groups can also be invoked collectively and carry out batch tasks.

Moreover, this extended functionality and the overall design of the application is set to promote energy savings. The ability to organize and view historical and real time service response data enables the user to comprehensively grasp an accurate view of the environmental conditions in the building in correlation with energy consumption for both past and present. Subsequently, the user is able to manage energy consumption based on this knowledge.

Despite the universal nature of Web Services, WSDL grants syntactic interoperability only. That sets a boundary for all Service clients, including iDEALISM as well. The application could reach its full potential and incorporate a form of intelligence when services get semantic annotations. This next stage is listed as future work. After that, iDEALISM could offer automatic parsing and organizing of services based on semantic descriptions and require no user action. Going deeper, that would also eliminate the need for hardcoded service behavior, as services will reference classes or groups of classes from a common model (i.e. the ontology).

5.2 Smartphone Application - PlugDroid

Smartphones are becoming increasingly used in every application and especially in Ambient Intelligence, because of their compact and portable nature. Smartphones are interesting because of their dual role: they can be used for manual user input and output but also as sensors in the Ambient Intelligence system. Smartphones incorporate numerous devices in compact size and reasonable price. The Android platform[15] has also contributed to easy adaptation and exploitation of Smartphone hardware through the open source android SDK[16].

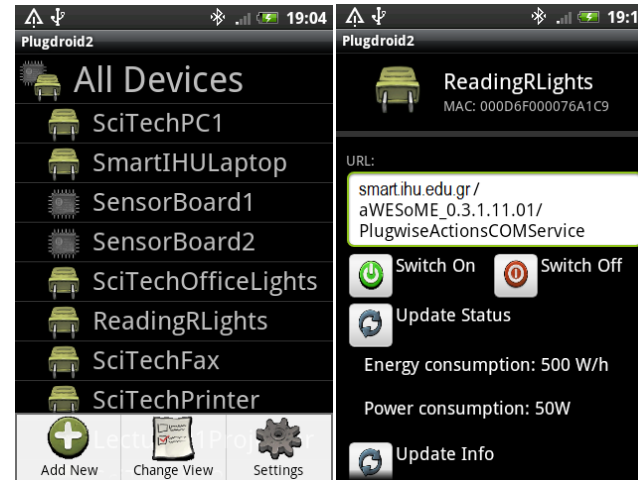

Figure 6. PlugDroid screenshots. List of all Devices (left)
Device Service view and invocation options (right)

[15] Android homepage: http://www.android.com/
[16] Android SDK: http://developer.android.com/sdk/

The PlugDroid application (Figure 6) was developed for the purposes of Smart IHU, on top of the aWESoME middleware. It can be seen as a variant of its desktop counterpart, iDEALISM, as it offers similar functionality, altered to suit its compact and portable nature. To put it simply, the functionality provided by PlugDroid is the same as iDEALISM, minus historical data (i.e. charts) and plus the aiding functions provided by the smartphone hardware.

A list of services is again maintained locally, as mobile users would be overwhelmed with a complete list of services. Instead, they can selectively add the Services they need and store them for later use. Again, the list can be organized in custom groups for batch operations and viewed categorized per Type, Room or Group. All device-related services are naturally supported by the Web Service client incorporated in PlugDroid, including the Smart Plug Services. Unlike iDEALISM, this application is rid of the historical data charts, which would need numerous services calls to fill in and burden the portable applications usage. Besides, smartphone users (not only Android users) can visit the Web Portal on the phone's browser and view all historical data carts.

The MAC Addresses can be manually entered during Service registration but in PlugDroid they can also be obtained by scanning QRCodes. QRCodes are printed on paper and placed on the appliances or near the entrances of corresponding rooms. As a result, the users can physically interface with the real-world appliances on-the-go and do not have to take notes or seek out complex MAC Addresses.

As future extension, the PlugDroid application can equally benefit from semantic service discovery and interpretation. It would then require even less user input and choice making and enable a hassle-free mobile experience.

6 Results

To measure and evaluate the robustness, responsiveness and energy efficiency of the proposed platform itself (apart from energy savings that can be achieved by applications) a series of experiments has been carried out. The services and even the servers used were evaluated in three different setups: using a PC, a netbook and a board microcomputer. The PC has a P4 1.8 GHz processor and 512 MB RAM, the netbook wears an Atom 2x1.66Ghz processor and 1G DDR3 RAM, while the microcomputer wears an ARM9 400MHz processor and 64MB RAM.

To begin with, during experiment 1, all machines were tested on different combinations, i.e. Scenarios, of mild CPU load and attached I/O devices (as they also consume energy) to measure their average idle consumptions. Scenario 1 is the most demanding one, running the aWESoME server and having a lot of devices attached. As seen on Table 7, the board's consumption is always constant and about 4% of the PC's power demand while the netbook is about 28% as demanding as the PC.

The second experiment involves invoking a given set of service operations (Switch On and Switch Off) for 120 consecutive times in each run. The average of response time for each call, over each run and per server is shown on Table 8. The performance of the netbook and the PC can be regarded as equal and was very fast (as the sensors themselves need some time to respond). However the Board PC is almost three times slower.

During the same experiment, the power consumption of each server has been logged (using the smart plug sensors and another aWESoME server). Table 9 shows the average values of these measurements. The consumption values while idle, confirm experiment 1. What is interesting is that the difference of an active PC is far bigger than the rest of the machines. Now the Board PC is consuming 2% (98% saved) and the netbook 18% (82% saved) of the PC's power demand.

Figure 8 shows the power dissipation in time during the course of experiment 2. Except from the large differences in energy savings already mentioned above, the large response time of the Board PC is also evident. As expected, the three times bigger response time of the Board PC results in a three times longer total runtime of the experiment. Note that turning the netbook's monitor off can save up to 4 Watts in average, which is shown on this figure.

Since the netbook and the PC have the exact same response time but the netbook's consumption is smaller, the PC is left out of the next comparison. Table 10 summarizes the power consumption-performance tradeoff between the Board PC and the netbook. All in all, both of them can be considered as significant improvements over the PC server.

7 Future Work

The development of the Smart IHU platform follows two directions: extending and enriching the aWESoME middleware and developing intelligent applications that exploit the middleware. aWESoME serves as the essential middleware layer that offers the required interoperability but it is not supposed to (and does not) present certain behavior. On the contrary, it serves as a library of methods (web methods) to be exploited in order to realize applications of different logic.

To provide applications with more variety and flexibility, aWESoME can be extended in two ways: more components and semantic annotations. Due to its component-based structure, newly developed drivers and/or services can be added. To integrate more devices or device families, new drivers and their corresponding services can be built. Software-based services for utility functions or internet services (e.g. weather forecast) can be added to the service layer to support more scenarios. In the future we plan to integrate more devices (e.g. a weather station) but also various Z-Wave devices to profit from existing work in the Z-Wave library.

Apart from extending the functionality provided, semantic annotations can enrich aWESoME in terms of interoperability and usability. The Semantic Web[17] technologies have reached a point where all the necessary tools and standards suffice for machine interpretable information on the Web. As web development and usage patterns shift from data to services, many efforts to apply the same semantic web methodologies to services have been presented. Some did not succeed as W3C recommendations, but provide rich expressiveness, such as OWL-S[18]. Other approaches, such as SAWSDL[19] - annotations for WSDL are based on simplicity and effectiveness. We have developed an ontology to describe entities in the Smart IHU do-

[17] The Semantic Web initiative: http://semanticweb.org
[18] The OWL-S ontology for services: http://www.daml.org/services/owl-s/1.0/
[19] SAWSDL at W3C: http://www.w3.org/2002/ws/sawsdl/

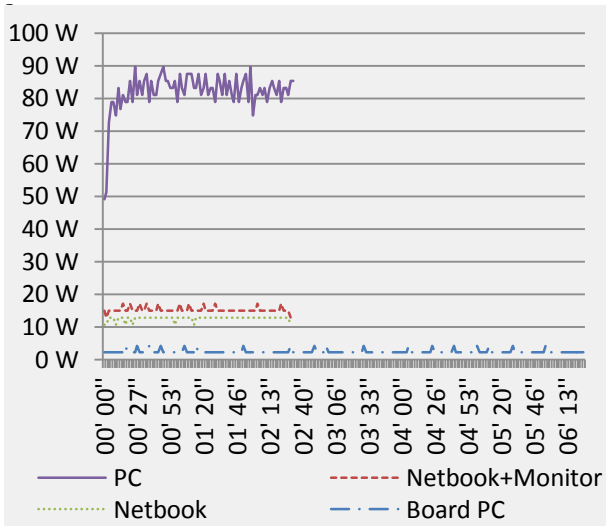

Figure 8. Server power consumption in time during experiment 2

main, and Smart Spaces in general, and plan to explore both directions towards Semantic Web Services: describe the service in a complex model (i.e. the ontology), following the OWL-S approach, and using direct annotations on WSDL, following the SAWSDL approach.

Finally, we plan to develop applications based on well-known A.I. methodologies to benefit from the infrastructure and present intelligent behavior in the system. The applications presented so far, are merely GUIs that showcase the capabilities of aWESoME and offer a user-friendly view of the system's status. However, known methodologies of A.I. have been used in research to provide automation of tasks in Smart Spaces. The most common objective for service-oriented systems is automatic Service Composition using simple matching or A.I. Planning. We plan to exploit semantic annotations on aWESoME and employ such techniques to profit from existing solutions (e.g. known planning algorithms) to automate tasks in the Smart IHU environment, towards both saving energy and raising user comfort.

8 Conclusion

This work presents a middleware for Ambient Intelligence systems, based on the Service-Oriented Architecture for interoperability, the current sensor network deployment in a Smart University system, including a prototype board Server, and finally sample client applications.

As Ambient Intelligence systems are characterized by diversity in hardware and the market has not yet reached a point where devices can be exploited by computer programs in homogeneous ways, a middleware is often employed to tackle this obstacle. Also, the paradigm shift to service computing has widely been adopted in Ambient Intelligence as it provides a useful abstraction for complex ambient applications to simply get things done. The proposed middleware is both able to expose functions and data of various devices and device families and follows the service-oriented paradigm to harvest its benefits. The driver layer of the middleware contains device drivers for ZigBee Smart Plugs, Sensor Boards, Smart Clampers and Z-Wave devices while the service layer contains services that expose the methods of these drivers and additional software-based services such as Wake-On-Lan and remote shutdown to manage IT equipment.

Table 7. Idle consumption of different servers during experiment 1

Scenario	Board PC	Netbook	PC
1	2.1324 W	13.5703 W	48.3361 W
2	2.1325 W	13.3764 W	48.4783 W
3	2.1325 W	13.3764 W	48.3361 W
4	2.1325 W	13.3764 W	47.9096 W

Table 8. Response time overall distribution during experiment 2

Response Time(sec)	Board PC%	Netbook %	PC%
1	0	82.1	82.5
2	5.417	17.9	17.5
3	90.833	0	0
4	3.333	0	0
5	0.417	0	0
Total	100	100	100

Table 9. Power consumption during experiment 2

Server	Power Consumption	
Board PC	Idle	2.133 W
	Active	2.337 W
	Difference	0.204 W
Netbook	Idle	13.376 W
	Active	15.220 W
	Difference	1.844 W
PC	Idle	48.265 W
	Active	81.998 W
	Difference	33.742 W

Table 10. Netbook and Board PC comparison

Parameter	Netbook	Board PC	Difference
Processor	2x1.6GHz	400MHz	2x41% faster
RAM	1GB	64MB	16% bigger
Average Consumption when active	15.220 W	2.337 W	65% higher
Total Response Time	142 s	358 s	40% faster
Average Repsonse Time per call	1.18 s	2.98 s	40% faster

aWESoME is deployed in a Smart University setting, namely the Smart IHU project. Each driver and/or service module can be separately installed in aWESoME servers that are distributed in the building. Thus, coverage restrictions of the smart device networks are lifted. To maintain compactness and low power consumption of the aWESoME infrastructure itself, we demonstrate a prototype board aWESoME server along with performance and consumption measurements. Current topology includes a laptop server and a board server that cover the most vital part of the School of Science and Technology (the ground floor).

A Service Broker for aWESoME has been developed to complete the architecture paradigm with provisioning capabilities, supported by the WS-Discovery standard. As the system is based on distributed servers, and services can possibly enter and leave the environment (due to portability or failure), the Service Broker serves to intermediate and locate Service Providers for Service Clients. Finally, two kinds of aWESoME clients, a desktop application and a smartphone application, are presented

to showcase aWESoME functionality and allow monitoring and managing the Smart IHU environment in a user-friendly way.

Acknowledgment

The Smart IHU project is funded by Operational Program Education and Lifelong Learning, OPS200056 (International Hellenic University, Thessaloniki, Greece). The authors would also like to thank Andrea Dimitri (MSc student at the time) as well as the undergraduate students Alexander Arvanitidis and George Pilikidis for their most valuable contribution.

References

Autili, M., Caporuscio, M., & Issarny, V. (2009, May). Architecting Service Oriented Middleware for pervasive networking. In *Proceedings of the 2009 ICSE Workshop on Principles of Engineering Service Oriented Systems* (pp. 58-61). IEEE Computer Society.

Ben Mabrouk, N., Beauche, S., Kuznetsova, E., Georgantas, N., & Issarny, V. (2009). QoS-aware service composition in dynamic service oriented environments. *Middleware 2009*, 123-142.

Bottaro, A., & Gérodolle, A. (2008, July). Home soa-: facing protocol heterogeneity in pervasive applications. In *Proceedings of the 5th international conference on Pervasive services* (pp. 73-80). ACM.

Caporuscio, M., Raverdy, P. G., Moungla, H., & Issarny, V. (2008). ubi SOAP: A Service Oriented Middleware for Seamless Networking. *Service-Oriented Computing–ICSOC 2008*, 195-209.

Chiaraviglio, L., & Mellia, M. (2010, September). PoliSave: Efficient power management of campus PCs. In *Software, Telecommunications and Computer Networks (SoftCOM), 2010 International Conference on* (pp. 82-87). IEEE.

Corredor, I., Martínez, J. F., Familiar, M. S., & López, L. (2012). Knowledge-Aware and Service-Oriented Middleware for deploying pervasive services.*Journal of Network and Computer Applications*, *35*(2), 562-576.

Davidyuk, O., Georgantas, N., Issarny, V., & Riekki, J. J. (2011). MEDUSA: Middleware for End-User Composition of Ubiquitous Applications. *Handbook of research on ambient intelligence and smart environments: trends and perspectives*, *11*, 197-219.

Eisenhauer, M., Rosengren, P., Antolin, P., A Development Platform for Integrating Wireless Devices and Sensors into Ambient Intelligence Systems in the proc. of the 6th Annual IEEE Communications Society Conference on Sensor, Mesh and Ad Hoc Communications and Networks Workshops 2009, (SECON Workshops '09).

Georgantas, N., Issarny, V., Mokhtar, S. B., Bromberg, Y. D., Bianco, S., Thomson, G., ... & Cardoso, R. S. (2010). Middleware architecture for ambient intelligence in the networked home. *Handbook of Ambient Intelligence and Smart Environments*, 1139-1169.

Hansen K. M., Zhang W., Soares G.: Ontology-Enabled Generation of Embedded Web Services. SEKE 2008: 345-350 Wang, X. H., Dong, J. S., Chin, C., and Hettiarachchi, S. R. Semantic space: an infrastructure for smart spaces. IEEE Pervasive Computing 3,3 (July-Sept 2004), 32?39.

Ibrahim, N., Le Mouël, F., & Frénot, S. (2009, July). MySIM: a spontaneous service integration middleware for pervasive environments. In *Proceedings of the 2009 international conference on Pervasive services* (pp. 1-10). ACM.

Lagesse, B., Kumar, M., & Wright, M. (2010, March). ReSCo: A middleware component for Reliable Service Composition in pervasive systems. In *Pervasive Computing and Communications Workshops (PERCOM Workshops), 2010 8th IEEE International Conference on* (pp. 486-491). IEEE.

López-de-Ipiña, D., Vázquez, J., Garcia, D., Fernández, J., García, I., Sáinz, D., & Almeida, A. (2006). A middleware for the deployment of ambient intelligent spaces. *Ambient Intelligence in Everyday Life*, 239-255.

Park, J. H., & Kang, J. H. (2011). Intelligent service processing in common USN middleware. *Artificial Intelligence Review*, *35*(1), 37-51.

Robinson, J., Wakeman, I., & Owen, T. (2004, October). Scooby: middleware for service composition in pervasive computing. In *Proceedings of the 2nd workshop on Middleware for pervasive and ad-hoc computing* (pp. 161-166). ACM.

Stavropoulos T. G., Tsioliaridou A., Koutitas G., Vrakas D. and Vlahavas I. (2010), "System Architecture for a Smart University Building", in the proc. of IEM3 workshop in conjunction with ICANN 2010, Thessaloniki, Greece

Stavropoulos, T. G., Vrakas, D., & Vlahavas, I. (2011a). A survey of service composition in ambient intelligence environments. *Artificial Intelligence Review*, 1-24.

Stavropoulos, T., Vrakas, D., Arvanitidis, A., & Vlahavas, I. (2011b). A system for energy savings in an ambient intelligence environment. *Information and Communication on Technology for the Fight against Global Warming*, 102-109.

Weiser M., The computer for the 21st century, ACM SIGMOBILE Mobile Computing and Communications Review, v.3 n.3, p.3-11, July 1999 [doi>10.1145/329124.329126]