# The Smart IHU Project Architecture for Energy and Ambient Intelligence Applications

Thanos G. Stavropoulos, Dimitris Vrakas and Ioannis Vlahavas

**Abstract—. This paper presents a complete architecture for a Smart University Building. The real-world deployment is based on a wide range of wireless sensor and actuator networks, integrated by a middleware based on the Service-Oriented Architecture of Web Services. The middleware provides the necessary basis for various energy monitoring, management and savings applications as well as Intelligent Agents in the context of Ambient Intelligence.**

***Index Terms*—smart building, smart grids, sensor networks, web services, knowledge management, semantic web, ambient intelligence**

## I. Sensor Layer

The physical layer of the proposed architecture hosts a wide variety of wireless sensor and actuator networks. All hardware was carefully selected to optimally resolve the tradeoff between necessary requirements, availability and affordability. First of all, the devices would have to operate over a large range and offer a variety of functions, to cover the scale and the diversity of parameters in the building. On the other hand, they should remain widely available and affordable, especially in large quantities. As the market is yet far from convergence in a common communication protocol and data format, the devices were not expected to interoperate. This issue of interoperability is rather resolved on the middle layer of the architecture.

The selection of deployed devices mainly operates over wireless communications and comes from different manufacturers and suppliers. Most devices are also widely available in the market and affordable to retail customers, which supports the feasibility of the proposed system.

First of all, as the target system revolves around energy monitoring and manipulation, the first set of devices covers these aspects. The company Plugwise[1] offers a commercial bundle of devices that is equally popular amongst retail buyers and research [1], [2]. The fundamental version of Plugwise products is a sensor/actuator device that will be referred to as Smart Plug. Smart Plugs are attached between a wall socket and any electrical appliance, allowing users to measure and control its power supply. In detail, their sensor functions can measure the attached appliance's status (on or off), its power supply (in W) while the actuator function can switch it on or off. Another commercial version of Smart Plugs, handles non-pluggable appliances, as it intersects power cables between source and appliance. They allow the exact same capabilities so both versions will be referred to as Smart Plugs to preserve generality.

In terms of communication, the Smart Plug platform follows the ZigBee wireless communication protocol, one of the optimally suitable protocols for smart home applications. The Plugs form encrypted ZigBee networks of up to thirty nodes in mesh topology. Each network is coordinated by an augmented, router Smart Plug which collects and propagates all network communications to and from a USB stick PC interface. Consequently, PC users can monitor Plug data and invoke actuator operations. Smart Plugs are passive devices which means they have to be polled to perform an action or return a measured value in a bidirectional manner. Each network's range depends on how evenly the mesh network is distributed. Users are prompted to ensure a circular distribution of plugs around the coordinator to avoid long lines of hops but rather provide short, alternative paths for all nodes. To adequately cover the building's appliances, we have deployed approximately forty-five Smart Plugs, grouped in three Plug networks. One network covers the eastern part of the building, whereas the other two overlapping networks cover the southern part.

While Smart Plugs are responsible for small-scale, per appliance measurements, large-scale power usage has to be monitored by a different set of devices. Especially in the case of a large building, it is naturally impossible to monitor its total consumption by using just Smart Plugs. On the contrary, it is much easier to monitor total consumption directly at its source, the main power supply. Smart Clampers are affordable sensors that clip around main power supply cables, without intersecting them, and inductively measure the current. Due to their affordability and ease of installation, various manufacturers provide retail Smart Clamper solutions. The selected Smart Clamper bundle offered by Current Cost[2] additionally offers an open data format and support for multiple transmitters. Each Current Cost digital 433MHz SRD band transmitter can be attached to up to three Smart Clampers for the measurement of three-phase current. Data of up to ten transmitters

T. G. Stavropoulos is with the Computer Science Department, Aristotle University of Thessaloniki, Greece (e-mail: athstavr@csd.auth.gr).
D. Vrakas is with the Computer Science Department, Aristotle University of Thessaloniki, Greece (e-mail: dvrakas@csd.auth.gr).
I. Vlahavas is with the Computer Science Department, Aristotle University of Thessaloniki, Greece (e-mail: vlahavas@csd.auth.gr).

[1] Plugwise Online, http://www.plugwise.com

[2] Current Cost online, http://www.currentcost.com/

Table 1 Hardware technical, communication and networking aspects

| Device | Manu | Capabilities | Sampling Frequency | Number of Nodes | Protocol | Maximum Nodes Per Network | Range | Topology | Networks Deployed |
|---|---|---|---|---|---|---|---|---|---|
| *Smart Plug* | Plugwise | Sensor/ Actuator | 1s | 50 | ZigBee (encrypted) | 25 | 10m | Mesh | 2 |
| *Sensor Board* | Prisma Elec-tronics | Multi-Sensor | Any | 20 | ZigBee (open) | 10 | 10m | Mesh | 2 |
| *Z-Wave* | Various | Sensor | Any | 20 | Z-Wave | ? | 10m | Mesh | 1 |
| *Smart Clamper* | CurrentCost | Sensor | 8s | 3 | RF | 10 | 10m | Star | 1 |

are collected and displayed on a monitor/receiver that also provides a PC USB-interface. In our deployment, two three-phase main power supplies are measured by corresponding clampers. The sum of these two measurements returns, thus, the total consumption of the building. For disaggregation purposes, a third clamper and transmitter bundle measures the Data Center’s i.e. server room’s power supply, which is, of course, included in the first two measurements.

Apart from energy, the proposed smart building applications need to monitor and correlate various environmental parameters. Hence, a wide selection of sensors had to be deployed. The first bundle of sensors is manufactured by Prisma Electronics[3] and comprises of two types of nodes. Quaxes are Sensor Boards that embed a microcontroller, a ZigBee module and various arrays of sensors. The microcontroller can be manually programmed to periodically parse data from sensor arrays, and transmit them over ZigBee. Optionally, they can also act as routers for other Sensor Boards, forming a ZigBee mesh network. The second type of nodes, are ZigBee gateways that collect Sensor Board data and transmit them over Ethernet or Wi-Fi to the LAN’s PC clients. The devices are active which means they actively transmit data one directionally.

For the purposes of our deployment two networks of ten Sensor Boards each, have been distributed across the building. The sensors attached to the boards measure temperature, luminance and humidity. Again, one network was assigned with the southern part and one with the eastern part. The Sensor Boards were set in non-routing mode, so each network follows a star topology instead of mesh. This allows the modules to enter hibernation between transmissions, prolonging battery life. For the same purpose, transmission interval has been set to ten minutes. Two gateways collect data from each network and make it accessible from anywhere in the LAN.

Finally, to complement the variety of measured environmental data, we integrated sets of devices complient to the Z-Wave alliance. The Z-Wave alliance constitutes one of the efforts to unify data formats and communications between smart home automation devices at hardware level. So far, the alliance has managed to provide a wide variety of sensors, actuators, network controllers/coordinators, remote controllers etc. achieving interoperability between numerous manufacturers. Transmissions are very similar to ZigBee in nature, as they form wireless mesh networks. In our deployment we made use of four $CO_2$ air level sensors, fourteen motion sensors and three smoke detectors, each coming from a different manufacturer. All data is gathered by a USB-Stick controller and PC-interface. A single mesh network of these devices was installed to cover the whole building.

Table 1 compares different aspects of all device families in the proposed deployment.

## II. Middleware Layer

The intermediate level of the proposed architecture hosts a middleware specifically tailored for Ambient Intelligence applications, based on the Service Oriented Architecture. The nature of these applications operating on top of a dynamic, real-time environment and heterogeneous devices presents several requirements. Service-orientation has been proved to be extremely suitable for such environments and used widely in literature [3], resolving application development issues. Namely, one benefit of Service-orientation is being able to program on a high-level of abstraction, agnostic of platform-specific and communication-protocol specific device programming. This also adds to extensibility, as new devices can be integrated via the authoring of new middleware-plugins which simply translate to new services. All services comply to a web-wide universal API, the W3C WSDL language, which syntactically defines service operations, resolving heterogeneity. The service descriptions are further described semantically using Semantic Annotations for WSDL (SAWSDL) rendering them machine interpretable. To resolve dynamicity in the environment, the Service Oriented Architecture supports the notion of service provisioning. A Service Broker is responsible to list available services at any given time, which means that mobile providers that interleave the environment do not have to be handled explicitly during application programming.

After studying current state-of-the-art guidelines and existing implementations, a WEb Service MiddlewarE (aWESoME) was developed, tailored to the system’s needs. The middleware itself entails three distinct layers, one for hardware integration, one that implements and provides (syntactically described) services and a semantic description layer on top. Secondly, it reuses and extends existing implementations as much as possible. Motivation behind building a middleware from scratch has been based on defining semantics more efficiently and integrating hardware modules that existing middleware does not. However, some existing modules have been used on the hardware integration layer, when possible, to interface with part of the hardware.

[3] Prisma Electronics online, http://www.prismaelectronics.eu

### A. Hardware Integration Layer

This layer consists of separate modules each of which interfaces with a target device platform. The so-called driver modules can be considered as plug-ins developed for each device family to be integrated into the Smart IHU system.

*Smart Plug Driver:* The Smart Plug Driver, implemented in Java, is invoked every time a get or set operation needs to be performed over the Smart Plug network. Smart Plug operation is bidirectional, since they need to be polled to read a sensor value or to perform an actuator action. Thus, the Smart Plug Driver is more a library rather than a module, invoked each and every time needed by the corresponding Smart Plug Service operations. To implement the library, the Smart Plug encrypted ZigBee protocol commands had to be reversed engineered to implement the essential polling functions and decrypt response packages.

*Sensor Board Driver:* Unlike Smart Plugs, Sensor Board communication is one-directional. The Sensor Board Driver module is a C# daemon that constantly receives data, parses them according to the open package format provided by the manufacturer and stores them for the Sensor Board Service to retrieve. The module also offers a GUI to allow administrators monitor the multiple gateways and nodes of the sensor board network.

*Smart Clamper Driver:* Quite similarly, the Smart Clamper Driver is a Java module that passively receives all Smart Clamper data, parses and stores them to be retrieved by the Smart Clamper Service. It also presents data on a GUI for monitoring and administration purposes.

*Z-Wave Driver:* Being the largest and the most diverse family of devices, the Z-Wave Driver makes use of the open source Open Z-Wave library to handle potentially any Z-Wave device. To again provide administrators with a GUI, another open source solution was selected. zVirtualScenes is a GUI application project which already incorporates the Open Z-Wave library. Thus, both projects were merged and modified to our needs. The Z-Wave driver provides a GUI to monitor and manage a Z-Wave device network. Unlike other devices, Z-Wave nodes receive configuration parameters such as sleep interval over the network, so this function is also provided on the administration GUI. It also presents all values received from connected nodes and stores them for the Z-Wave Service to retrieve on-demand. Virtually any type of Z-Wave device is supported by the Open Z-Wave library and in turn by the Z-Wave Driver.

### B. Web Service Layer

This layer hosts the Web Service APIs themselves, implemented as JAX-WS web services, syntactically described in WSDL format and hosted on instances of the Glassfish server. The design methodology followed by all web service modules was to associate each service with a single device family bundle. In turn, each service provides as many operations as the corresponding hardware offers. This choice was not straightforward, as, in general, even a single service can hold many operations encapsulating code to handle every device family. On the other extreme, numerous single-operation services could provide a single function for each module. As hardware is physically separated in device networks of the same brand or communication protocol, services were made to reflect that physical distinction, also matched essentially by the driver modules. Note that since drivers already perform protocol unification of data coming from the devices, a single service could indeed handle more than one device type. Instead, this integration is allowed to be resolved on the semantic layer, in a more efficient and machine-interpretable way.

*Smart Plug Service:* The service provides sensor operations to poll and get each Plug's on/off binary status, read power consumption in 1 or 8 sec interval, and various hardware information such as internal clock and firmware. It also provides actuator operations to switch them on or off. All operations require a target Plug ID as input.

*Sensor Board Service:* This service provides numerous sensor operations that receive a target board ID and return the value of measured temperature, humidity, luminance or the board's battery level.

*Smart Clamper Service:* The Smart Clamper Service provides basically one operation to receive the target Clamper's ID and return its power measurement.

*Z-Wave Service:* Similarly, operations of this service accept a target node ID parameter and each return measured temperature, humidity, luminance, motion detection and $CO_2$ air concentration level.

### C. Semantic Layer

The semantic layer of the architecture consists of two components, the ontological infrastructure and the semantically described service endpoints. The ontological infrastructure generally entails one or more ontologies that serve as a lexicon of interrelated concepts, meant to semantically describe entities. After a thorough review of state-of-the-art, BOnSAI

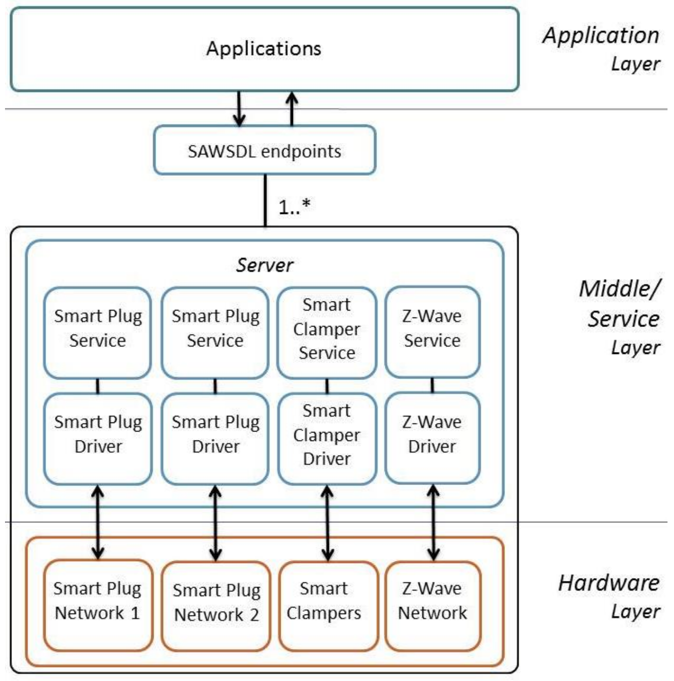
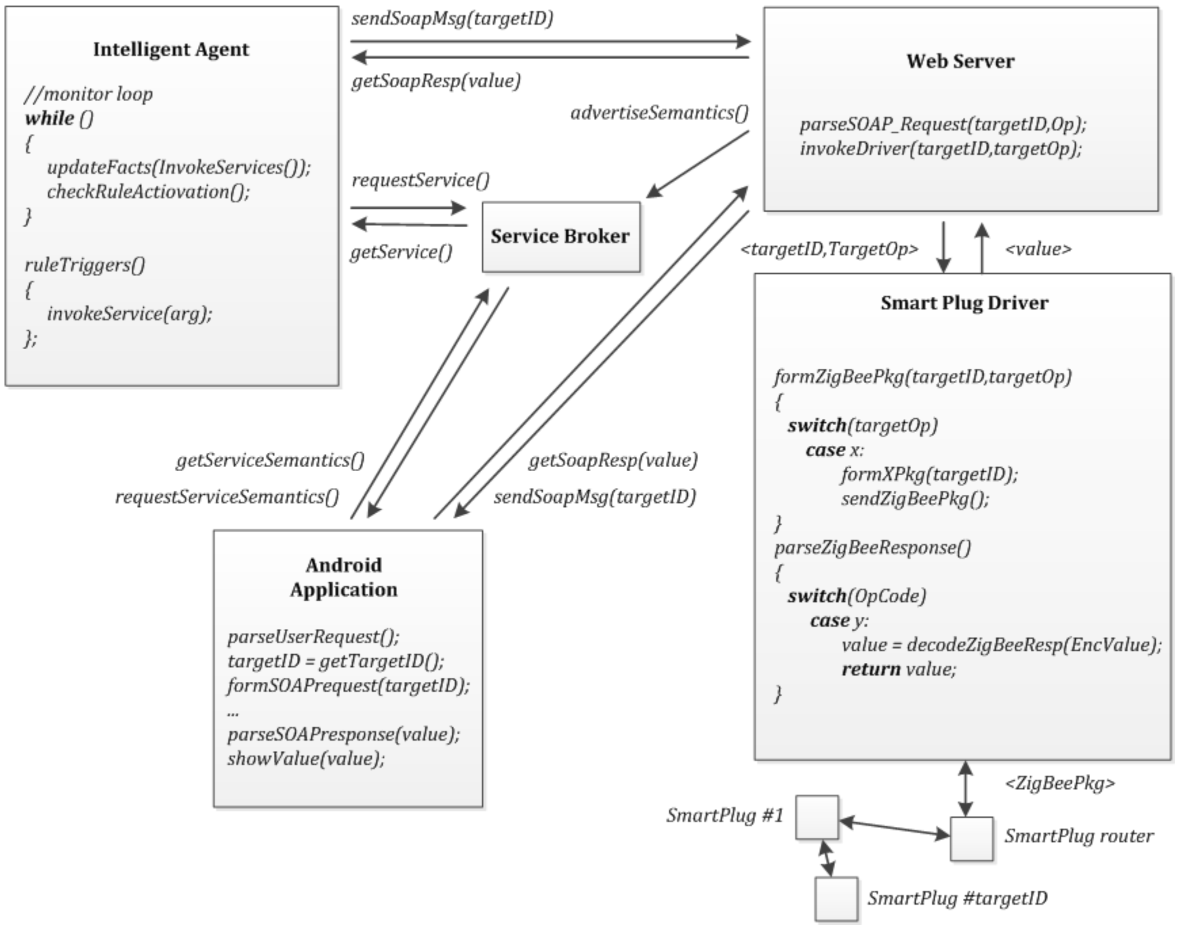


Fig. 1. Smart IHU three-layer architecture, focusing on hardware and middleware packages



```
//monitor loop
while ()
{
   updateFacts(InvokeServices());
   checkRuleActiovation();
}

ruleTriggers()
{
   invokeService(arg);
};
```

```
parseSOAP_Request(targetID,Op);
invokeDriver(targetID,targetOp);
```

```
formZigBeePkg(targetID,targetOp)
{
  switch(targetOp)
    case x:
        formXPkg(targetID);
        sendZigBeePkg();
}
parseZigBeeResponse()
{
  switch(OpCode)
    case y:
        value = decodeZigBeeResp(EncValue);
        return value;
}
```

```
parseUserRequest();
targetID = getTargetID();
formSOAPrequest(targetID);
...
parseSOAPresponse(value);
showValue(value);
```

Fig. 2. A use case scenario of the architecture by two applications, demonstrating distributed parts of algorithms for data collection, reasoning and presentation.

(Smart Building Ontology for Ambient Intelligence), suitable for describing Smart Building, Ambient Intelligence, Service and Sensor Network concepts has been designed [4]. An important aspect of ontology design is reuse. As the very aim of an ontology is to provide semantic interoperability outside the borders of a particular system implementation and across the web community, it must be universally adopted. This has indeed been an issue in Semantic Web technologies overall, since there has been no convergence to commonly-used models. Hence, BOnSAI extends leading existing ontologies found in each domain such as the Semantic Sensor Network (SSN) ontology and the OWL-S upper ontology for services.

Most existing approaches are using ontologies to insert all sensor measurements, usually referred to as context information, as instances of ontological concepts. This approach can be thought of as using the ontology itself as a database (as actual ontological DBMSes do exist) and performing reasoning on top of that data. On the contrary, while such an approach is feasible in the proposed infrastructure, we rather semantically describe services that return data instead of data itself. In detail, the ontology is used as a lexicon to describe terms and define relationships, rather than a database. Thus, when reasoning on real-time context data, information is retrieved much faster coming from the sensors themselves. The resulting SAWSDL files semantically describe input and output of services as well as the nature of operations themselves. All SAWSDL defined annotations (so-called model references) originate from the BOnSAI ontology and derive from the specific needs of applications which are then exploiting them (use-driven design).

Operations are annotated as either *SensorOperations* or *ActuatorOperations*. These annotations are currently used by rule-based software agents for both rule authoring and invocation. The safe assumption made here is that SensorOperations are suitable to serve as rule conditions (left-hand side), while ActuatorOperations are suitable for rule results (right-hand side). Hence, during the authoring of rules, available options are dynamically retrieved from available semantic service descriptions.

Inputs and Outputs are annotated as to their nature e.g. *sensor:ID*, *Temperature*, *Humidity*, *Power* etc. These annotations serve for a wider range of clients such as service discovery, matchmaking and composition. Namely, a software or human agent that is looking for a particular service can form semantic queries based on inputs and outputs. In the future this could lead to service composition which is outside the scope of this work, which focuses on automation and energy savings.

## III. APPLICATION LAYER

Based on the remote, universal web service API, the proposed architecture allows for multiple, diverse applications to coexist in the system. Given the high performance of today's

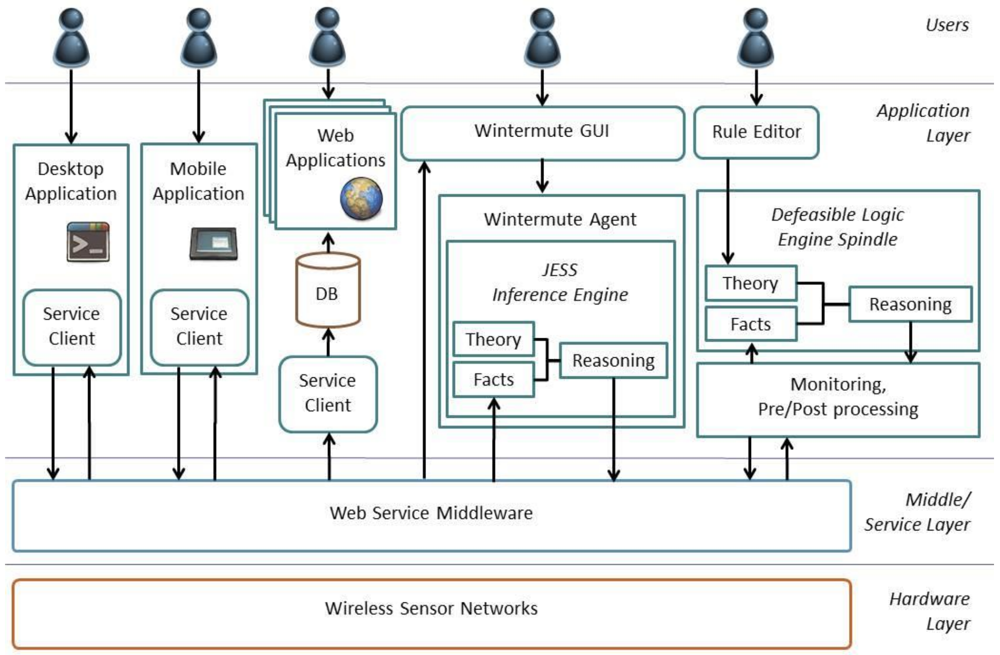


Fig. 3. Smart IHU architecture of multiple applications, ranging from desktop, mobile, web, intelligent agents to expert systems

computation and communication infrastructure, web services introduce an insignificant temporal delay for the applications to interface with hardware, in comparison to a hardcoded direct embedment of a single application occupying the hardware. The different applications developed so far can be categorized according to the purpose they serve in monitoring and administration, rules and automation, data center monitoring and energy analytics.

### A. *Monitoring and Administration*

Although automations and intelligent agents manipulate the infrastructure without any human intervention, monitoring and administrative application do play an important role in the system. Their purpose is twofold; for one, they allow human administrators to manually manipulate the devices but most importantly, to observe and analyze historical data of the building's behavior. Consequently, they are able to take decisions and author new optimized policies to be applied. Secondly, monitoring applications allow simple users such as the university's staff and students, visitors and web users globally to observe the infrastructure's behavior and status.

Monitoring applications target all popular platforms to ensure accessibility. iDEALISM is a desktop application for monitoring and management developed in Java. As shown on **Error! Reference source not found.**, a service client handles all service calls, manipulating sensors and actuators. The application is meant to be used by administrators only, such as the Smart IHU, IT and security staff, enabling them to monitor historical and real time data before taking action. Added-value functions allow them to group devices together and get aggregated values to better appreciate the actual environmental and consumption status of the building.

The mobile application counterpart, named PlugDroid, offers manipulation capabilities, plus some added-value facilities powered by the smartphone hardware. It has been implemented in the Java Android SDK, due to its openness and popularity combined, incorporating a Web Service client to connect to the middleware. Users are able to create, edit and maintain lists of sensors and actuators, invoke functions with one touch and view data. The abundance of integrated sensors present in smartphones provides an excellent basis for augmented-reality functions. The Smart IHU Android application currently takes advantage of the QR/barcode scanning function via the camera to implement such a feature. Specifically, mobile users can scan QR codes printed on various connected appliances or sensors in the building, adding them to their set of monitored devices. Thereafter, they are able to monitor and manage them remotely at any time over the web.

Finally, the main web application of Smart IHU[4], presents a view of historical sensor readings targeting students, guests and outside visitors abroad. The portal is one among many web applications built on top of a collective database of sensor data. Specifically, since a common need for multiple applications to access historical data was eminent, a (MySQL) database is being populated by a Service Client daemon, accessible by any remote application. The Smart IHU portal offers open access for any web user to select and view the most major of sensor data, such as power consumption of the building, data center and building appliances total (building minus data center), along with selected environmental measurements (e.g. $CO_2$ levels in the Labs). Another web application taking advantage of the same underlying infrastructure is an on-site flat TV-screen at the university's reception, which displays current, a seven-day average and a three-day line chart of power consumption, together with weather information and the course schedule timetable for the day.

[4] Smart IHU Portal http://smart.ihu.edu.gr/

## B. Rules and Automation

While simple client applications manually manipulate actuators and recover sensor data, intelligent applications allow the automatic smart administration of power consumption. One approach towards that end is the use of intelligent agents that incorporate knowledge, conduct decision making and manipulate the environment. Traditionally, these agents employ various forms of reasoning following different logic principles. For the purposes of energy saving in the proposed system, two different approaches have been followed.

An agent based on productive logic is responsible for reactive rules, where fast response is of the essence. The agent, named Wintermute and presented on Fig. 3, incorporates an instance of the JESS rule execution engine. Users are able to author and maintain automation policies using a GUI, implemented with JavaFx. The GUI is enhanced with semantic information dynamically acquired from descriptions of services online at the time. This is carried out based on the assumption that SensorOperations are suitable for rule condition-predicates and, likewise, ActuatorOperations are well-suited for action predicates. Hence, during rule authoring, users are presented with available option in a dropdown box. The Wintermute agent interfaces with semantic web services to periodically obtain facts of interest and update its knowledge of the world's state from the sensors. These facts along with authored policies in the agent's knowledge base are used for the reasoning process which, in turn, triggers and fires rules that invoke semantic web services to manipulate appliances. This agent, all in all, emphasizes in the straightforward approach of monitoring values and immediately firing up rules, at the expense of having to resolve conflicts during the rule authoring process.

For example a simple thermostat rule *(r1)*, in JESS syntax appears as follows:

```
(defrule ruleThermostat
   (call GetTemperature 9BA14E ?x0)
   (test (< ?x0 20) )
=>
   (call SwitchOn CA7712)
)
```

where `9BA14E, CA7712` are the temperature sensor ID and the heater's power actuator ID respectively. In this example, we go on to define a second rule *r2* that switches off the heater when no motion is detected. This rule indirectly conflicts with the previous one, as their results contradict. We would also like this rule to dominate over *r1*, hence, *r1* has to be redefined to include `(¬motion)` in its conditions. Proceeding with the rule set, a third energy-saving policy *r3* involves entering a so-called 'Saving Mode' where flexible, relatively excessive devices such as heating are switched off when crossing a power level threshold. Following this rule logic, for this rule to obtain maximum superiority, all subordinate rules *r1,r2* must explicitly enumerate the negation of all conflicting rules stated. Apparently, as the rule set grows, all subordinate rules' left hand side grows exponentially. For one, this issue introduces a huge hassle for the rule author to carefully review and maintain the rule base, which ultimately takes a non-intuitive form. Secondly, the method presumes that rule authors have complete access, knowledge and privileges over the rule base, which might not be the case in such an environment.

Therefore, a second agent paradigm is introduced, incorporating a defeasible logic rule engine (*SPINdle*[5]). This agent emphasizes on intuitive rule-authoring and much more conflict resolution in order to maintain a large and multi-purpose set of rules. This logic attaches a priority listing of rule superiority relationships to the standard fact and rule knowledge base. Hence, rules become short, intuitive declarations that are easy to maintain, while defining their priority handles conflict resolution. The rule author can use a variety of user interfaces such as *SPINdle's Defeasible Logic Theory Editor*[5], or the much more flexible *S$^2$DRREd* (*Syntactic-Semantic Defeasible Reasoning Rule Editor*) [6]. Finally, the reasoner superiority declarations were taken advantage of to define three different rule clusters: preferences, maintenance and emergency and provide different authorization levels to different users. Simple users only have access to the preference rule set, which has the lowest priority. Power users have access to maintenance and emergency rules of higher superiority, resolving the matter of privacy and security of the system. For demonstration purposes only, the above mentioned rule set in now transformed as follows:

```
# preference rules
p01: $@temp < 20$ -> tempLow
p1: tempLow => switchOn_heater
p2: - tempLow => switchOff_heater

# maintenance rules
m01: $@consumption_total > 2000$ -> savingMode
m1: -motion => switchOff_heater
m2: savingMode => switchOff_heater
```

where maintenance rules always prevail over preference rules. All in all, with both methodologies present, a hybrid approach is feasible, where the reactive agent handles fast-response simple rules and the defeasible agent handles decision making and deliberation over a large and more complex rule set.

## IV. State of the Art

The state of the art in the AmI field in general, includes many application domains such as health, Ambient Assisted Living, agriculture, multimedia and Smart Offices. Most of these approaches also employ the use of web services and semantics but follow the more complex top-down approaches of upper ontologies for services such as WSMO[6] or OWL-S[7] in [7] [8] or custom ones as in [3] [9]. On the contrary, the approach proposed in this work, employs the bottom-up lightweight and W3C recommended SAWSDL[8] standard for semantically annotating web service descriptions. It also implements a custom universal middleware based on SAWSDL to support a wide variety of affordable devices available in the market not found

[5] SPINdle Defeasible Logic Theory Editor: http://spin.nicta.org.au/spindle/tools.html

[6] WSMO W3C Submission: http://www.w3.org/Submission/WSMO/

[7] OWL-S W3C Submission: http://www.w3.org/Submission/OWL-S/

[8] SAWSDL W3C Recommendation: http://www.w3.org/TR/sawsdl/

in other approaches.

As far as rule-based smart environments are concerned, one approach introduced in [Daniele] is a meta-language defined over JESS, to syntactically enhance the rule authoring process in ambient applications. However, this additional syntactic layer, named the Event-Control-Action model, is far less flexible and extensible over defeasible logic used in Smart IHU. Other similar approaches include *SESAME-S* [2] is an all-in-a-box smart home prototype that uses ontologies and JESS reasoning to enforce rules. The approach in Yang also uses agents, web services and ontologies to store and reason on energy data. The main issue of both works is the lack of conflict resolution and rule set scalability enforcing them to apply all triggered rules, regardless. Additionally, Smart IHU does not store all information in ontological form but rather returns semantic information on-the-fly using SAWSDL and offering improved scalability, flexibility and extensibility to more clients.

## References

[1] Eisenhauer, M., Rosengren, P., & Antolin, P. (2009, June). A development platform for integrating wireless devices and sensors into ambient intelligence systems. *In Sensor, Mesh and Ad Hoc Communications and Networks Workshops, 2009. SECON Workshops' 09. 6th Annual IEEE Communications Society Conference on (pp. 1-3). IEEE.*

[2] Fensel, A., Tomic, S., Kumar, V., Stefanovic, M., Aleshin, S., Novikov, D. SESAME-S: Semantic Smart Home System for Energy Efficiency. *Informatik Spektrum*, 36(1), pp. 46-57, 2013.

[3] Issarny, V., Caporuscio, M., & Georgantas, N. (2007, May). A perspective on the future of middleware-based software engineering. *In 2007 Future of Software Engineering (pp. 244-258). IEEE Computer Society.*

[4] Stavropoulos, T. G., Vrakas, D., Vlachava, D., & Bassiliades, N. (2012, June). Bonsai: a smart building ontology for ambient intelligence. *In Proceedings of the 2nd International Conference on Web Intelligence, Mining and Semantics(p. 30). ACM.*

[5] Lam, H. P., & Governatori, G. Towards a Model of UAVs Navigation in Urban Canyon through Defeasible Logic. *Journal of Logic and Computation*, 2011.

[6] Kontopoulos, E., Zetta, T., & Bassiliades, N. Semantically-enhanced Authoring of Defeasible Logic Rule Bases in the Semantic Web. *Proc. 2nd Int. Conf. on Web Intelligence, Mining and Semantics (WIMS'12), ACM, Article 56, pp. 489-492, Craiova, Romania, June 13-15, 2012.*

[7] Thomson, G., Bianco, S., Mokhtar, S. B., Georgantas, N., & Issarny, V. (2008). Amigo aware services. *In Constructing Ambient Intelligence (pp. 385-390). Springer Berlin Heidelberg.*

[8] Iacob, S. M., Almeida, J. P. A., & Iacob, M. E. (2008, March). Optimized dynamic semantic composition of services. *In Proceedings of the 2008 ACM symposium on Applied computing (pp. 2286-2292). ACM.*

[9] Paz-Lopez, A., Varela, G., Becerra, J. A., Vazquez-Rodriguez, S., & Duro, R. J. (2012). Towards ubiquity in ambient intelligence: User-guided component mobility in the HI3 architecture. *Science of Computer Programming.*