Rule-based Approaches for Energy Savings in an Ambient Intelligence Environment

Thanos G. Stavropoulos[1,2], Efstratios Kontopoulos[2,3], Nick Bassiliades[1,2],
John Argyriou[1,4], Antonis Bikakis[5], Dimitris Vrakas[1] and Ioannis Vlahavas[1]

{athstavr, skontopo, iargyrio, dvrakas, nbassili, vlahavas}@csd.auth.gr, a.bikakis@ucl.ac.uk
[1] Dept. of Informatics, Aristotle University of Thessaloniki, Thessaloniki, Greece,
tel: +30 231099-8145, fax: +30 231099-8362
[2] School of Science and Technology, International Hellenic University, Thessaloniki, Greece
[3] Information Technologies Institute, Centre of Research & Technology - Hellas, Thessaloniki, Greece
[4] Department of Informatics, University of Sussex, UK
[5] Department of Information Studies, University College London, UK

Abstract. This paper presents a novel real-world application for energy savings in a Smart Building environment. The proposed system unifies heterogeneous wireless sensor networks under a Semantic Web Service middleware. Two complementary and mutually exclusive rule-based approaches for enforcing energy-saving policies are proposed: a reactive agent based on production rules and a deliberative agent based on defeasible logic. The system was deployed at a Greek University, showing promising experimental results (at least 4% daily savings). Although the percentage of energy savings may seem low, the greatest merit of the method is ensuring no energy is wasted by constantly enforcing the policies.

Keywords: defeasible logic, smart building, energy savings, ambient intelligence, semantic web services

1. Introduction

Contemporary information systems are rapidly undergoing various transformations, influenced by a multitude of emerging trends. One of these trends is *Green Computing* and the rise of the *Smart Grid*. Global warming, CO_2 emissions and energy waste have directed computing towards greener solutions on both hardware and software levels. At the same time, motivated by renewable energy sources and consumer-produced energy schemes, the idea of the Smart Grid refers to an infrastructure encouraging policies towards marketing and distribution of energy between producers, consumers, "prosumers" (i.e. producers and consumers) and brokers [1]. Conclusively, the concepts of sustainability and energy saving have penetrated all aspects of everyday life, including computing.

Meanwhile, progress in microcontrollers, board computers and – particularly – smartphones brings forward a novel computing paradigm, called *ubiquitous computing* (*ubiComp*), where users are being surrounded by powerful, portable computing devices [2]. To take things even further, in *pervasive computing* (*perComp*), multiple non-intrusive computing devices are scattered across the environment, attracting little or none of the user's attention. The vision of *Ambient Intelligence* (*AmI*) enriches perComp with *Artificial Intelligence* (*AI*) methodologies, aiming at increasing comfort and reducing intrusiveness for the user [2]. Towards this affair, intelligent behaviors, deriving from various areas of AI, such as Planning, Semantics and Reasoning, are typically employed for providing smart automations and intuitive human-computer interactions in such environments.

Unsurprisingly, AmI is heavily based on the widespread usage, performance and affordability of microcontrollers and wireless sensor networks, providing the essential environmental input data required by AI algorithms. Likewise, actions to be performed after the decision making process have to go through actuators in the environment in the form of motors, switches or other endpoints. Since both Green Computing and AmI are facilitated by the increased popularity and low cost of smart meters, environmental sensors, actuators and other devices, these features constitute the common grounds among the two disciplines. Also, both fields share similar goals, i.e. the vision of a comfortable but energy-efficient smart home.

Although AmI domains of application may vary widely, targeting energy savings has yet been only marginally explored. Existing service-oriented approaches in conjunction with smart home equipment have not practically considered intelligence or savings, but rather the issue of communication protocol unification [3]. Multi-agent systems have been employed to provide enhanced functionality or service composition, but certain issues on truly interoperable semantic descriptions still persist [4]. A categorization of AmI application domains in Smart Homes, Health, Transportation, Emergency, Education and Workplace, can be found in [5]. Apparently, most approaches have ignored the aspect of energy saving and are instead focused on multimedia technologies, providing context awareness in smart homes or offices based on UPnP/DLNA multimedia streaming [6].

This work presents a holistic approach to energy savings in a smart building belonging to a Greek Public University complex. The proposed approach lies in the intersection of many disciplines (AmI, Automated Reasoning, Semantic Web Services and Green Computing), facing challenges in each of the corresponding fields. More specifically, a set of state-of-the-art wireless sensors, actuators and smart meters has been deployed at the University premises. Heterogeneous sensor data and actuator functionality are both unified under a universal middleware based on the *Service Oriented Architecture* (*SOA*) [7] paradigm and the *Web Service Description Language*[1] (*WSDL*) for syntactic interoperability. The middleware also complies with some of the latest *Semantic Web* [8] technologies, like *Semantic Annotations for WSDL – SAWSDL* [9], to provide semantic interoperability on the application layer.

Towards achieving energy savings, two complementary and mutually exclusive rule-based approaches are proposed. Firstly, using production rules, domain experts are able to commit energy-saving policies maintained and applied by an autonomous reactive software agent. The second approach is, instead, based on a deliberative agent that incorporates a defeasible logic reasoner, greatly enhancing the expressiveness of the committed rule set. The application of defeasible reasoning [10] that offers a sophisticated conflict resolution mechanism is ideal in such environments with incomplete and contradictory information; in our approach three clusters of rules are defined: for preferences, maintenance and emergencies. Using the underlying infrastructure, the University's yearly consumption has been observed to identify energy behavioral patterns. Consequently, energy-saving and comfort-providing policies have been authored and applied accordingly, showing promising results. Experimental evaluation showed that, although daily savings of more than 4% may seem trivial in such a large infrastructure, the policies do ensure that no energy is wasted in any undesired manner.

The next section performs a thorough survey of state-of-the-art in the fields of context-aware applications, service-oriented middleware and the limited application of defeasible reasoning in AmI environments. Section 3 presents the basic principles of defeasible logics, its main characteristics, its syntax and operational semantics as well as the motivation for applying defeasible reasoning in AmI and Smart Building management. The next section presents the architecture of the proposed system in three parts: hardware, middleware and application layer. The two proposed software agents that reside on the application layer are presented subsequently, followed by a real-world application use case scenario, a proposed policy and its experimental results in energy savings. Critical discussion, limitations, future work and conclusions are listed in the final sections.

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

2. Related Work

The proposed system is built on a synergy of fields such as Ambient Intelligence, Smart Homes, Context-aware Systems, Green Computing, Semantics and Logic. Thus, this section attempts to present the most relevant of existing state-of-the-art, originating from any of the above viewpoints. It also tries to appoint the novelty of the proposed system to each respective work coming from one or more fields.

2.1. Rule-based Context-aware Systems in AmI

A common task amongst many AmI systems is to address context awareness, by providing automatic service composition on demand. A thorough survey of such work can be found in [11]. However, we consider the policy definition process as more fundamental. Automatic service composition offers the means to achieve a certain goal, but it does not define the goal itself or when to execute this series of actions. Hence, rules and triggers must come prior to composition, leaving the latter outside the scope of this work. Agents in state-of-the-art approaches frequently participate in the Web Service composition process, while in this work agents are self-contained entities that perceive and manage the environment.

On the other hand, certain paradigms indeed address context-awareness by following rule-based methodologies, such as the work in [12]. The authors have developed the *Event-Control-Action* (*ECA*) model, an architectural pattern for expressing context-awareness. According to this model, an "Event Module" is responsible for gathering context information i.e. facts, a "Control Module" formulates rules about these facts and an "Action Module" executes activated rules. To properly author ECA-rules, the proposed ECA domain language, *ECA-DL* [13], offers expressiveness (via the use of logical and arithmetical operands) and extensibility. To realize this higher level of expressing context information, ECA-DL integrates a UML-based representation model that defines a hierarchy of entities and relationships among them. The Java-based *JESS*[2] reasoner was chosen as the rule engine, while the authors proposed a mapping from ECA to the JESS syntax. There is a direct analogy between the two architectures: instances of context-information in ECA correspond to facts in the working memory of JESS, while ECA-DL rules directly correspond to JESS defrules. Additionally, the hierarchy and relationships between instances defined in the ECA model can be expressed in deftemplates, i.e. JESS structures for describing complex facts.

In comparison to our Smart IHU system, the ECA model can be seen as an optional rule-authoring meta-level. Both works address the concepts of context-awareness and, more generally, ubiquitous and pervasive computing. However, our proposed system focuses on energy savings and builds upon an AmI infrastructure of sensors, actuators and Semantic Web Services, while ECA-DL has not been deployed in such a practical manner. Additionally, the meta-level of ECA rule representation is considered rather redundant, since the JESS language itself provides sufficient expressiveness and extensibility. Moreover, ECA's representation model is fairly inferior to an ontology, which already provides querying mechanisms and significantly higher expressiveness. Instead of applying a UML-based model, our approach refers to our representation model, an OWL ontology, via semantic annotations on Web Service descriptions. Hence, instead of building a heterogeneous system that maps rules to ontological constructs, ontological entities are inserted into rules on-the-fly using *SAWSDL*. Finally, both works use JESS as an execution engine, but our approach offers higher expressiveness, integrating defeasible logic as well as the corresponding defeasible reasoner SPINdle (see Section 4.6).

Work by [14] employs a case-based reasoning (CBR) agent, Web Services and ontologies towards achieving energy savings. The CBR agent is placed within a multi-agent platform, along with a second agent responsible for data mining and a third one for invoking Web Services. The complete system is deployed over cloud services to enable remote reasoning. Data regarding temperature, humidity, luminance, CO_2 levels and consumption are collected over Web Services, parsed and placed within a domain ontology. The sensor deployment for data collection contains three ZigBee networks of forty devices in total. Then, each data instance is reasoned upon, to investigate similarities with previously known cases. Evaluations show that the architecture can effectively

[2] JESS rule engine: http://www.jessrules.com/jess/index.shtml

assimilate around 40% of cases, which then do not need to be handled by the backend server. The authors also claim that an energy saving percentage of approximately 22% is achieved.

Our proposed system demonstrates many similarities with the work by [14], but also numerous key differences. Both approaches deploy large wireless sensor networks that measure environmental values and energy consumption. Also both lines of work employ one or more agents capable of reasoning based on different logics, in order to achieve energy savings. Finally, both approaches employ the use of Web Services and SOA standards (WSDL/SOAP). On the other hand, our deployment is technically far richer, containing ZigBee, Z-Wave and RF (and potentially any other) communication protocols. These protocols are unified under the Web Service middleware, which is the main role of services in this work, placing it in an AmI ecosystem with other coexisting applications. For the same interoperability purposes, semantics are defining service operations instead of data. On the other hand, [14] basically employs Web Services to perform remote procedures, such as transforming syntactical data to semantic constructs and storing them. In other words, the main purpose of Web Services in [14] is the operation of the system over the cloud. As far as our work is concerned, the use of cloud services is merely a technical matter that resolves computing and storage resource issues, which in our case are irrelevant; outsourcing computation over the cloud does not affect the methodologies used, but rather the location of resources.

Moreover, in [14] it is unclear how decisions are applied to achieve energy savings and it is equally unclear how these energy savings are measured. We argue that savings should be measured relatively to carefully selected periods of time, since consumption is very context-sensitive, especially in large-scale. Major fluctuations occur between different seasons, times of day and daily events. We state the energy saving percentage compared to a daily average of yearly consumption and have determined exactly where these savings come from to make sure it's not incidental. Secondly, the proposed system guarantees that, given the policies, a variable degree of savings can be achieved by prohibiting unnecessary devices from turning on. The lower bound of the saving is achieved when the building already presents an energy-efficient behavior (minimum saving margin) and its upper bound is achieved when the building presents its maximum energy-wasting behavior (maximum saving margin).

Another analogous project, called *SESAME-S* [15], also presents many similarities to the presented work. SESAME-S is a partially commercial and partially research project with deliverables aiming at an all-in-a-box smart home platform. The system architecture revolves around a microcontroller that interfaces between wired smart devices, sensors and actuators and user endpoints. Deployed sensors measure temperature, humidity, luminance and energy consumption, while actuators control heating, cooling, water supply and power. Live data is collected over the LAN router and propagated over Ethernet or Wi-Fi to local and remote terminals, where it is stored in an *OWLIM* repository [16] in the form of RDF triples. Additionally, the system stores policies as SWRL/ SQWRL rules [17], performs reasoning and executes them using JESS. Rules are committed by both home and power users through a Web application. Home users are able to enter any rule or choose from pre-built recommended rules, based on past test-bed experiments. Power users, i.e. administrators and domain-experts, are able to view the raw, semantic rule format and the rules they insert have a higher significance, being executed after any home automation rules, in order to enforce the desired final state of appliances. To support the above functionality, SESAME-S entails the use of three custom ontologies: (a) the *Automation Ontology* models policies, activities, appliances, sensors, locations and thresholds, (b) the *Meter Data Ontology* publishes meter data in compliance to the DLMS/COSEM specification, and, (c) the *Pricing Ontology* models usage tariffs and selection criteria for the system to allocate appliances to proper time slots, in the spirit of Smart Grids. The final test-bed embeds all hardware in a metal suitcase and has experimentally been used in a few buildings. Finally, it has been estimated that the adoption of such a system may lead to approx. 20% reduction in energy consumption.

In contrast to SESAME-S, our work presents many developments on all levels. At the infrastructure level, the Smart IHU architecture supports many wireless device platforms, unified under the aWESoME-S web service middleware, like Plugwise, Prisma sensor boards, CurrentCost and various other manufacturers under the Z-Wave alliance (see Section 4.2 for further information). Additionally, due to the nature of the semantic service infrastructure, devices can be scattered across buildings in a fully distributed deployment, instead of embedding everything in a box. Another major difference is SESAME-S's lack of storing all data in an ontology for reasoning. Instead, Smart IHU attaches semantics to Web Services using the SAWSDL standard, allowing any Web client, human or software, to access the semantics of sensor data and actuator functions. Furthermore, the use of

defeasible reasoning in Smart IHU allows for a more user-intuitive design of rules and an inherent prioritization and conflict resolution between them. This method avoids the "brute-force" application of all rules in priority order that occurs in SESAME-S, improving the user experience and saving computing resources. Finally, due to SESAME-S's current experimental and constrained state, energy saving results constitute merely user predictions, while the Smart IHU platform already presents actual savings on its large-scale building-wide deployment.

2.2. Middleware for AmI

Implementing a middleware has been indeed one of the leading approaches to decouple AmI applications from the physical layer. Examples of existing middleware for enabling AmI applications include the work in [18], where a middleware provides higher level functionalities, such as user presence detection and user profiling. Sensor fusion and analysis are also performed within the middleware to provide presence detection and deliberate about energy management by actuators. Contrasted, the middleware utilized in this work focuses on integrating a variety of devices deployed at a larger scale. It focuses on basic functions and real-time enablement of AmI, leaving analysis and deliberation for the higher level components. It also defines semantics for higher level interoperability. Other approaches, such as [14], use middleware to obtain access and benefit from cloud services. The work in [19] also uses cloud services as well as a middleware for generic communication between devices. The cloud does provide easy access to large computational and memory resources for online analysis. However, in this work, the middleware is oriented towards operating the actual AmI environment in a semantically well-defined manner. Naturally, all state-of-the-art middleware is unique supporting different sets of devices, e.g. ZigBee, Z-Wave etc., while earlier approaches were more focused on X10, UPnP, Lonworks etc. [20]. Our middleware is also tailored to our specific set of devices from a technical point of view. A much more detailed survey of existing middleware can be found in [21].

Much work has also been invested in the adoption of semantic annotations on the middleware level. The majority of previous existing work has been focused on complex top-down approaches such as the OWL-S and WSMO ontologies, extending them to describe a service. A survey on such work can be found in [11]. The current tendency, however, is to switch to lightweight, bottom-up approaches, such as SAWSDL and WSMO-Lite. Although, convergence to a single standard has yet to occur, SAWSDL is recommended by W3C and offers many advantages. The annotations in SAWSDL are placed into the WSDL itself, which is a functional API, very popular among research and industry. Additionally, despite its low complexity, it has been found sufficient to serve as a basis for selection, matching, discovery and composition of services [22]. Hence, the proposed system makes use of the SAWSDL standard for defining and exploiting semantic interoperability on services.

Other approaches include the AmIi middleware in [23], which builds an additional layer of abstraction on top of WSDL, OWL-S and SAWSDL, translating service descriptions to its own format. Likewise, the *Hydra* middleware for AmI [24] uses an internal, similar representation to SAWSDL. On the contrary, services presented in this work solely utilize the universal standard of SAWSDL, in order to retain maximum and direct interoperability across the Web Service community and outside the borders of the system.

2.3. Defeasible Logic in AmI

To the best of our knowledge, only the work by [25] investigates the deployment of defeasible logics in an AmI setting. In their work, the authors propose a distributed reasoning approach based on the representation of context knowledge shared by the ambient agents in the environment. Taking into consideration the highly dynamic nature of the setting, defeasible logic is proposed as the basis for representing the context knowledge possessed by each agent (i.e. the agent's local rule base). Additionally, defeasible logic is also applied for resolving the potential conflicts that arise from the information exchange between the agents. More specifically, the authors introduce an extension to defeasible logics, called *Contextual Defeasible Logic* (*CDL*) [26]. The latter serves as a contextual reasoning model that leverages the challenges posed by the dynamic nature of the AmI environment using the concepts of context, mappings and contextual preferences. Compared to the work presented above, our proposed Smart IHU approach demonstrates apparent similarities: both approaches are designed for AmI environments and both of them apply defeasible reasoning in real-life, practical applications.

On the other hand, there are three important differences between the two works. We present a holistic approach, which includes deployment of the proposed reasoning model in a real environment, while [25] focuses only on the properties of the reasoning model and does not present its implementation. A second difference lies in the diverse areas – within each system – where defeasible reasoning is applied. More specifically, [25] applies defeasible reasoning as a means for handling conflicts caused by the integration of data from different sources. Contrasted, our approach investigates the application of Defeasible Logic as a tool for supporting the end-user in representing his/her energy management scheme more effectively and improving the expressiveness of the authored policies. Finally, the Smart IHU approach follows a centralized architecture, where a central server is responsible for collecting the data from the sensor network and for performing the reasoning. On the other hand, [25] presents a fully distributed architecture, arguing that such a model better suits environments where context changes may be frequent, device connection can prove troublesome and wireless communications are unreliable and restricted by the range of the transmitters. The different application domains (ambient-assisted living vs. energy consumption monitoring) and the different ways in which priority information is represented (preferences on data sources in [25] vs. priorities on rules in our work) constitute two secondary differences of the two works.

3. Defeasible Logics

This section describes the basic characteristics of defeasible logic, its syntax and operational semantics as well as the underlying motivation for applying defeasible reasoning in the context of Ambient Intelligence and Smart Building management.

3.1. Main Characteristics

Defeasible logics [10] stem from research in knowledge representation and inheritance networks in particular. In this context, defeasible logics can be seen as inheritance networks, expressed in a logical rule language.

Belonging to the family of non-monotonic logics, defeasible logics deal with conflicts and inconsistencies among knowledge items. Conflicts between rules are triggered by a conflict among their conclusions. The simplest case is that one conclusion is the negation of the other. However, more complex cases may arise, when conclusions are declared as *mutually exclusive*, a very convenient representation feature in practical applications that will be further examined later on.

Defeasible logics are sceptical in the sense that conflicting rules do not fire, preserving the consistency of the drawn conclusions [27]. Conflicts among rules are typically resolved via rule priorities, which constitute an additional representational feature of defeasible logics.

The main advantage of defeasible logics is their low computational complexity [28], which is achieved through:

- the local nature of conflicts;
- the fact that the logics are sceptical, as mentioned above;
- the absence of disjunction;
- the use of unidirectional rules;
- the local nature of priorities – only priorities between conflicting rules are used.

Undoubtedly, these restrictions come at the price of reduced expressive power, compared to other non-monotonic reasoning approaches. However, despite its conceptual simplicity, defeasible logic appears effective in the representation and execution of a variety of practical problems, like, e.g. recommender systems [29], UAV navigation [30] and elevator control [31].

3.2. The Language

A knowledge base in defeasible logic is called a *defeasible theory* (let's represent it by $\mathcal{D}$) and consists of three basic elements: a set of *facts* ($\mathcal{F}$), a set of *rules* ($\mathcal{R}$) and a *superiority relationship* (>). Therefore, $\mathcal{D}$ can be represented by the triple ($\mathcal{F}$, $\mathcal{R}$, >).

In defeasible logic, there are three distinct types of rules: *strict rules*, *defeasible rules* and *defeaters*.

- *Strict rules* are denoted by $A \rightarrow p$ and are interpreted in the typical sense: whenever the premises are indisputable, then so is the conclusion. An example of a strict rule is: r_1: summerTime → tempHigh ("*the temperature is high during the summer time*").
- *Defeasible rules* are denoted by $A \Rightarrow p$ and, contrary to strict rules, they can be defeated by contrary evidence. Two defeater examples are: r_2: tempHigh ⇒ switchOnCooler ("*the cooler is typically turned on when the temperature is high*") and r_3: highConsumption ⇒ ¬switchOnCooler ("*the cooler should not be turned on whenever there is high energy consumption*").
- *Defeaters* are denoted by $A \rightsquigarrow p$ and they do not actively support conclusions, but can only prevent deriving some of them. In other words, they are used to defeat respective defeasible conclusions, by producing evidence to the contrary. A defeater example is: r_2': isOnCooler ⇝ ¬switchOnCooler ("*when the cooler is on, there is no need to turn it on again*"), which can defeat e.g. rule r_2 mentioned previously.

The *superiority relationship* is used for resolving conflicts among rules belonging to the rule set $\mathcal{R}$. For example, given the defeasible rules r_2 and r_3 above, no conclusive decision can be made about whether the cooler should be turned on or not. But, if the superiority relationship $r_3 > r_2$ is introduced, then r_3 overrides r_2 and we can eventually conclude that the cooler will not be turned on. In this case rule r_3 is called *superior* to r_2 and r_2 *inferior* to r_3. Note that the relation > on $\mathcal{R}$ is acyclic, meaning that the transitive closure of > is irreflexive.

Finally, another important element of defeasible reasoning is the notion of *conflicting literals* [32], a feature that plays a significant role in our framework as well. $\mathcal{C}$ is defined as a set of conflicting literals if-f at most one literal of the set should be derived each time. Obviously, if p is a proposition then $\{p, \neg p\}$ is such a set of conflicting literals. However, in real-life applications there are various other kinds of sets of conflicting literals, which cannot be represented merely by a proposition and its negation. Such an example might be the distinct values a variable could take, like, e.g. {*on*, *off*}, {*hot*, *warm*, *cold*} or {*high*, *medium*, *low*}. Whenever there are three or more values, representation by just propositions and their negations becomes impractical and counterintuitive. Thus, a sample conflicting literal set might be:

$$\mathcal{C} = \{\texttt{switchOnCooler}, \texttt{switchOffCooler}\}$$

According to set $\mathcal{C}$ above, the following two rules will produce a conflict, which is resolved via the superiority relationship $r_4 > r_2$ and, thus, the conclusion of the latter rule will override the conclusion of the former one.

r_2: tempHigh ⇒ switchOnCooler ("*the cooler is typically turned on when the temperature is high*")

r_4: ¬motion ⇒ switchOffCooler ("*the cooler should be turned off if there is no motion inside the room*")

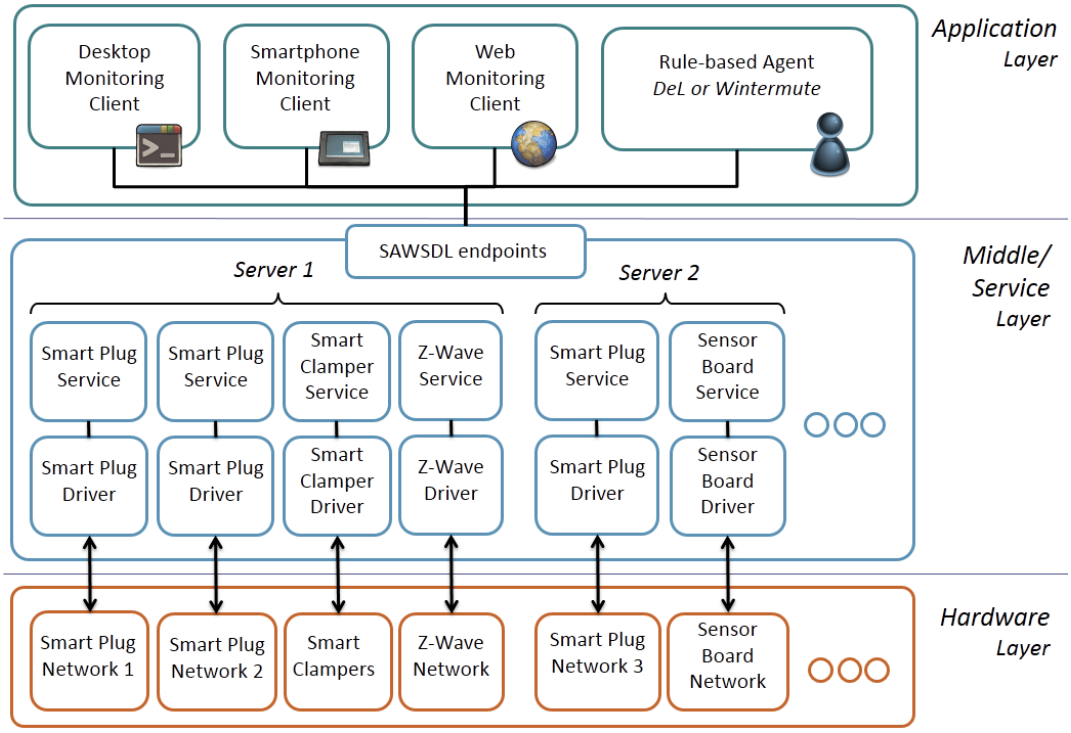

Figure 1. The Smart IHU overall architecture, showing two server instances and all applications.

3.3. Motivation for using Defeasible Logics

Besides its evident advantage of low computational complexity (see Section 3.1), the motivation behind applying defeasible logic instead of classical logic involves certain additional incentives. Firstly, classical reasoning approaches, like e.g. First Order Logic, are based on the assumption of perfect knowledge of the environment, which is very difficult, if not impossible, to achieve in AmI settings. Reasoning with incomplete information regarding the context is significantly more convenient with non-monotonic approaches [33].

Additionally, defeasible logic is much closer to human reasoning and is, therefore, more intuitive. The use of classical logic would require that the truth status of all pieces of information is known, before commencing the reasoning process. But, as human knowledge is quite often incomplete or even contradictory, this requirement cannot be always satisfied. Defeasible logic introduces non-monotonicity, which leads to a more intuitive type of reasoning, where the emergence of new information can lead to abandoning (i.e. defeating) previously established conclusions and adopting new ones.

A further motivating factor for using defeasible logic is the conciseness of its representation. Defeasible logic does not promise to do more than other logics, but, instead, as Covington states, "[...] *the difference is in how it does it*" [34]. Consequently, while classical logic would probably need extensive representations for expressing all possible exceptions to a rule, defeasible logic can simply add a more specific rule that overrides the former rule and expresses exactly the same information.

4. Implementation

The framework proposed in this work is applied at a Greek State University, namely Building A of the International Hellenic University (IHU). The rule-based system is a component of a wider AmI system, aimed to

provide automations, comfort and savings, named the Smart IHU project[3]. This section presents the overall architecture of the proposed system within Smart IHU and describes each of its components in detail.

4.1. Architecture

The generic architecture of the Smart IHU environment follows a three-layer approach (see Figure 1). The bottom layer hosts the hardware, namely, the wireless sensor and actuator networks deployed within the building. A service-oriented middleware is responsible for handling platform and data heterogeneity present at the hardware layer. The devices are also distributed in the environment, so that each cluster of co-located devices is allocated to a server running an instance of the Web Service middleware. Finally, client applications can receive data and operate on the infrastructure via one or more instances of the middleware on the various servers. Where necessary, applications provide a graphical interface to communicate with human users. Services additionally provide semantically annotated descriptions in SAWSDL and a broker for effective and automatic discovery.

In the specific case of rule-based systems introduced in this work, the application layer hosts intelligent agents that automatically monitor and manage the infrastructure (see Section 4.4). Each agent integrates a knowledge base and a reasoning engine. The knowledge base is filled with facts about the world, i.e. measured values of the environment, but also contains policies in the form of rules, entered by expert human users. After collecting the facts at each cycle, the agent performs reasoning, in order to decide on a proper set of actions, according to the policies. The agents are also accompanied by graphical user interfaces (GUIs).

4.2. Hardware Layer – Deployment of Wireless Sensors and Actuators

The hardware layer of the proposed system mainly hosts a wide range of sensors for perceiving and actuators for manipulating the environment. A first requirement for selecting a set of sensors was to examine which parameters would be suitable for composing and enforcing decision-making policies. Such parameters should include environmental qualities, power measurements and user-activity metrics. Additionally, the target large-scale deployment calls for devices that can form wireless networks of sufficient range and are affordable in large quantities. Consequently, a further substantial requirement was to build the platform over devices that are inexpensive and widely-available in the market.

The set of chosen devices is believed to satisfy the trade-off between cost and efficiency. The first set of deployed devices is a sensor/actuator network, referred to as *Smart Plugs*[4]. Each Smart Plug is plugged in-between a wall socket and any electronic appliance and can measure its power consumption, return its power state (on or off) and switch it on or off. A special kind of Smart Plugs, intended for non-pluggable appliances, e.g. air conditioning, can intersect power supply cables to carry out the same operations. From now on, both kinds will be referred to as Smart Plugs to preserve generality. Each network of maximum thirty Smart Plugs, interconnected over encrypted ZigBee, has a single coordinator that gives access to the network's data and functions. The current deployment consists of forty-five plugs, divided into three plug networks.

While Smart Plugs cover small-scale, per appliance power measurements, *Smart Clampers*[5] are designated to measure large-scale consumption, i.e. whole departments or buildings. The Clampers are attached to main power supply cables and measure consumption via induction. In this deployment, two three-phase clampers are placed at the building's main power supply and one at the data center, the University's server room. Hence, the sum of the first two clampers measures the total building consumption (including the data center), while the third clamper measures the data center's consumption. All clampers transmit their measurements to a single monitor/receiver with a PC-interface, over simple radio frequency (433MHz SRD band).

[3] The Smart IHU Project: http://rad.ihu.edu.gr/smartihu/
[4] Plugwise: http://www.plugwise.com
[5] Current Cost: http://www.currentcost.com/

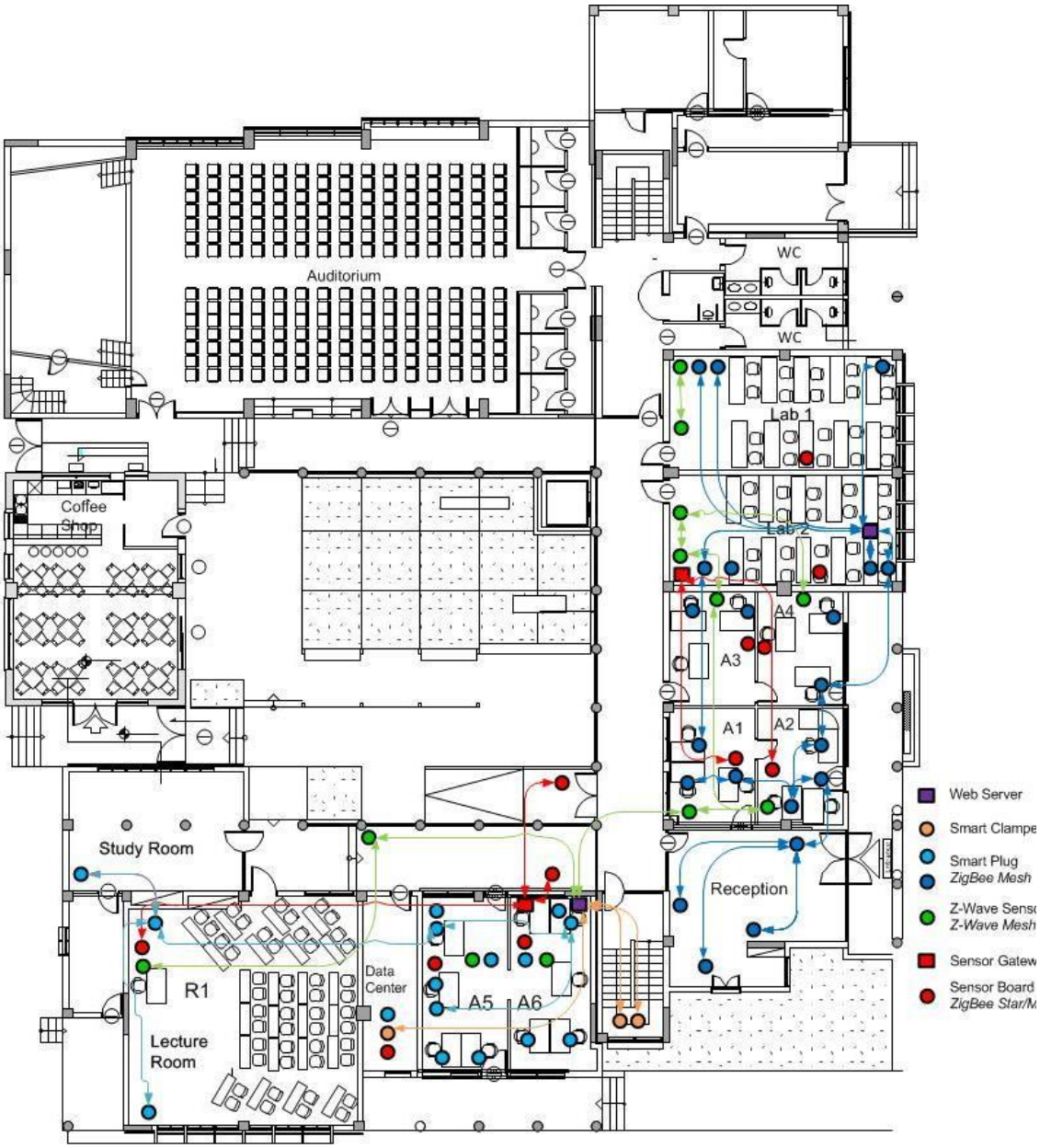

Figure 2. Topology of Wireless Sensor Networks on the ground floor of the building.

The third set of devices enables indoor and outdoor environmental monitoring. Twenty *Sensor Boards*[6] form another ZigBee mesh network and report temperature, luminance and humidity measurements to two designated ZigBee-to-TCP/IP Gateways. The Sensor Boards are fully customizable via coding their microcontroller. To maximize battery life, the period for measurement transmission has been set to ten minutes. Additionally, they have been configured not to act as routers, transforming the network topology from mesh to star. Thus, they are able to enter sleep mode in-between transmissions, which is critical, again, for extending battery life.

The final set of devices complies with a different wireless communication protocol, especially designed for smart spaces, the Z-Wave protocol. The Z-Wave alliance supports extensive interoperability between all

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

compliant devices. In fact, almost every device in a network comes from another manufacturer. A USB Stick is set as the network coordinator and PC interface. The mesh network of sensors comprises of fourteen motion detection sensors, four smoke alarms, a multi-sensor (for temperature, luminance, humidity and motion) and air quality sensors that measure CO_2 levels. More devices are instantly supported at communication protocol level and could possibly be added in the future, e.g. door or window sensors, to further enrich automation scenarios.

Figure 2 presents the exact placement of sensors, actuators and web servers in the building. To satisfy wireless range restrictions, two clusters of networks mainly exist in the building, each administrated by a collector server. The first cluster is found at the southern part and includes Rooms A5, A6, the data center, Lecture room 1, the Hall and the Study Room. The second cluster is visible at the eastern part of the building and includes the Reception, Rooms A1, A2, A3, A4 and the two Computer Labs. Each cluster has its own Sensor Board star network and a corresponding gateway collector. The first cluster is appliance-rich, so it contains two overlapping Smart Plug networks, while the second cluster only contains one. The Z-Wave network features fewer nodes and longer range so it spans across both clusters. The building's power supply is close to the first cluster, so the Smart Clamper receiver is assigned to it. Sensor Board data can also be received by both clusters over IP, so the second server was chosen due to its lower load. Each cluster has a web server to collect and expose data and functions, as described in the next section.

4.3. Middle Layer – aWESoME-S Web Service Middleware

Possibly the key challenge for AmI applications is to find the common grounds for high-level programming of different tasks and activities. In other words, low-level hardware programming of a diverse variety of sensors and actuators needs to be transparent on the upper (application) layer. Wireless sensor network management, topology and properties need to be transparent as well. Devices enforce various constraints, such as data format heterogeneity and platform-specific APIs. Applications, on the other hand, require universal, platform-agnostic access to device functions and data. In a large, distributed environment, such as a Smart Building, universal and remote access also needs to be ensured. Additionally, in such environments, devices dynamically enter and leave the space and this is a non-trivial parameter that has to be seriously taken into consideration.

With the rise of Service-Oriented Architecture and Service-Oriented Computing, ambient applications found a suitable paradigm that tackles all the aforementioned requirements. Service-orientation follows the principle of "services instead of data", hiding data and functions behind a universal API. Thus, internal application logic is transparent to the service client. In this case, hardware or software related services in the environment provide the necessary abstractions from device topology and function. Additionally, services allow for universal service-clients on any platform and in any programming language. Platform-dependent device operation is performed, regardless, on the server-side.

To realize the Smart IHU platform, we used the *aWESoME middleware* which we introduced in one of our previous works [21]. *aWESoME* is a service-oriented middleware that follows well-defined Web standards, providing access over WSDL services to every device operation on the hardware layer. Since WSDL provides syntactic interoperability, services are usable even outside the borders of this system. This work presents the *aWESoME-S* enhancement over the existing middleware that features SAWSDL semantic annotations to support the required higher-level intelligence. The so-called Semantic Web Services expressed in SAWSDL are machine interpretable and can be utilized for automatic discovery, selection, matching and composition. Table 1 provides a comprehensive list of service operations available, along with their semantic annotations (SAWSDL Model References). The Smart Building concepts, environmental parameters and devices featured in the *BOnSAI* ontology [35] were used for the annotation. The semantic description of device capabilities constitutes an added-value addition that further enhances dynamicity and interoperability.

Apparently, different services host similar operations, as different sensors provide the same environmental measurements. Semantic annotations help applications handle these operations similarly, e.g. a temperature reading is provided by both SensorBoardService and ZWaveService. Also, sensor operations are distinguished from actuator operations, which bear an object property of 'hasEffect' on the status of devices. Finally, note that

different instances of the same services are hosted by more than one server in the environment, as detailed previously in our sensor network deployment.

The communication and information exchange paradigm of aWESoME-S is essentially pull-based instead of push, for two major reasons. First of all, Web Services, which constitute the middleware's API, are inherently pull-based, i.e. they have to be invoked to respond. The middleware has to unify and aggregate three different communication paradigms under a common one. Z-Wave devices do provide push (i.e. publish/subscribe or event-based) communication, while SmartPlugs are pull-based and SensorBoards push data on a periodic (not event-based) basis. Consequently, the common paradigm emerging here is indeed pull-based for all devices, by gathering data from push-based devices. As a trade-off, loss of timing precision occurs here as pulling clients are not immediately informed for data changes, like e.g. motion detection. If push communication was a requirement, an additional layer on top of aWESoME-S could virtually transform pull-based to publish/subscribe communications, but this would not alleviate the overhead of actual frequent internal pulling. Secondly, pushing does introduce its own overhead of e.g. maintaining a long, dynamic list of clients [36]. Apparently in our case, pulling introduces a smaller overhead. To investigate even further, clients handling emergencies could be hosted right on top of devices that do support pushing. However, such clients would bypass the middleware and miss out on its benefits, i.e. communication with the rest of the system. All in all, we chose to minimize overhead by unifying all communications as pull-based and handle loss of timing precision by frequently pulling data when needed. Response times will be further discussed in the experimental section. However, it should be noted that the nature of our application is energy-saving and not emergency alerts, so responding rapidly to sensor readings is not a key requirement.

Another aspect of aWESoME-S is supporting error handling and quality of information. Firstly, aWESoME-S services support the service fault protocol, generating custom faults (much like exceptions) to be handled on the client-side. This way, the clients become aware of hardware or software failures on the middleware/hardware side. Furthermore, all services actually offer additional variants of the operations (not listed here for brevity reasons) to ensure that observations are up-to-date. Specifically, each operation variant returns either (a) just the observations, (b) the observations together with timestamps, or, (3) the observations that satisfy a certain freshness threshold. The agents presented in this work mainly utilize the latter method, in order to guarantee that observations are up-to-date. All in all, the middleware supports the necessary Quality of Context attributes for this setup, as defined in [37], such as up-to-dateness. Meanwhile, hardware failures that induce false readings have yet to occur in our

Table 1. List of service operations available in the environment.

Service	Operation	Output Semantics	Operation Semantics
SmartPlugService	SwitchOn	hasEffect:Status	ActuatorOperation
SmartPlugService	SwitchOff	hasEffect:Status	ActuatorOperation
SmartPlugService	GetPower	Power	SensorOperation
SmartPlugService	GetStatus	Status	SensorOperation
SmartClamperService	GetPower	Power	SensorOperation
SensorBoardService	GetTemperature	Temperature	SensorOperation
SensorBoardService	GetHumidity	Humidity	SensorOperation
SensorBoardService	GetLuminance	Luminance	SensorOperation
SensorBoardService	GetBatteryLevel	BatteryLevel	SensorOperation
ZWaveService	GetMotion	Motion	SensorOperation
ZWaveService	GetCO2	CO_2 level	SensorOperation
ZWaveService	GetTemperature	Temperature	SensorOperation
ZWaveService	GetHumidity	Humidity	SensorOperation
ZWaveService	GetLuminance	Luminance	SensorOperation
ZWaveService	GetSmoke	Smoke Alarm	SensorOperation
ZWaveService	GetBatteryLevel	BatteryLevel	SensorOperation

setting. Hence, the respective Quality of Context aspects, such as probability of correctness and trust-worthiness, were not considered (i.e. sources are 100% reliable). Fuzzy reasoning under uncertainty, which could handle such issues, extends beyond the scope of this work.

4.4. Application Layer

The syntactic and semantic interoperability provided by the middle layer (see previous section) supports a variety of platform-independent applications on the topmost layer of the infrastructure. Applications developed so far cover all major platforms and serve many different purposes. A desktop application enables comprehensive monitoring and management of the complete infrastructure. System administrators are able to monitor all real-time and historic data in graphs. Additionally, they can invoke actuator operations and manipulate the building's energy consumption, thus enabling savings through decision making [38]. An Android mobile application enables the same administrative functionality. Various other applications enable querying the service registry using semantic criteria, i.e. service selection, discovery and matching. History-monitoring clients mentioned above are not presented here, since they are outside the scope of this work. However, their monitoring capabilities have been used to observe and design the energy-saving policies applied by the rule-based systems.

This work focuses specifically on presenting two complementary and mutually exclusive rule-based agent approaches for automatic and effective energy management, both of which are designed to act autonomously based on an initial rule base and input from sensor readings. The first agent, called *Wintermute*, represents an initial attempt to using production rules, while the second agent, called *DeL*, evolves the approach one step further, using defeasible logics (see Section 3). Each is presented in the corresponding sections that follow.

4.5. Reactive Agent (Wintermute)

Wintermute represents the first agent implemented and integrated into the architecture. The initial requirement analysis is aiming at an intelligent agent system that incorporates user knowledge and, from then on, acts autonomously to achieve the energy-saving objective. The following functional requirements have been set:

a. The agent must be able to maintain a knowledge base, which will serve as its own internal representation of the world's state. This information will have to be acquired through the middleware's Semantic Web Service interface.
b. The agent must be able to perform reasoning on the world state and apply the derived decisions towards modifying it accordingly, by invoking the Semantic Web Services.
c. The agent must be able to communicate with human users and/or other software agents to exchange information or negotiate actions.

Additional non-functional requirements to be considered include reliability, fast performance, satisfactory response time, extensibility, usability and interoperation with other already active client applications.

Different AI methodologies, such as learning, can be employed to automate the decision making process of such a system. In this approach, we chose to embed a rule-based expert system into an intelligent agent residing on a multi-agent platform. Expert systems traditionally comprise of two parts: (a) the knowledge base, which is a dynamically changing repository of known facts and rules, and, (b) the inference mechanism that performs reasoning on the knowledge base using principles of a formal logic.

Facing the above challenges, a multi-component architecture (see Figure 3) consisting of closely connected modules has been adopted to manipulate the Smart IHU environment. Like every other application on the Smart IHU platform, Wintermute is built on top of the already established middleware and hardware layers. The aWESoME-S middleware exposes all data and functions through Web Services, providing the necessary abstractions for Wintermute to manipulate the environment. The multi-agent platform hosting Wintermute is *JADE*[7]. It is, hence, possible for JADE to host multiple Wintermute instances running simultaneously, each of

[7] Java Agent Development framework – JADE: http://jade.tilab.com/

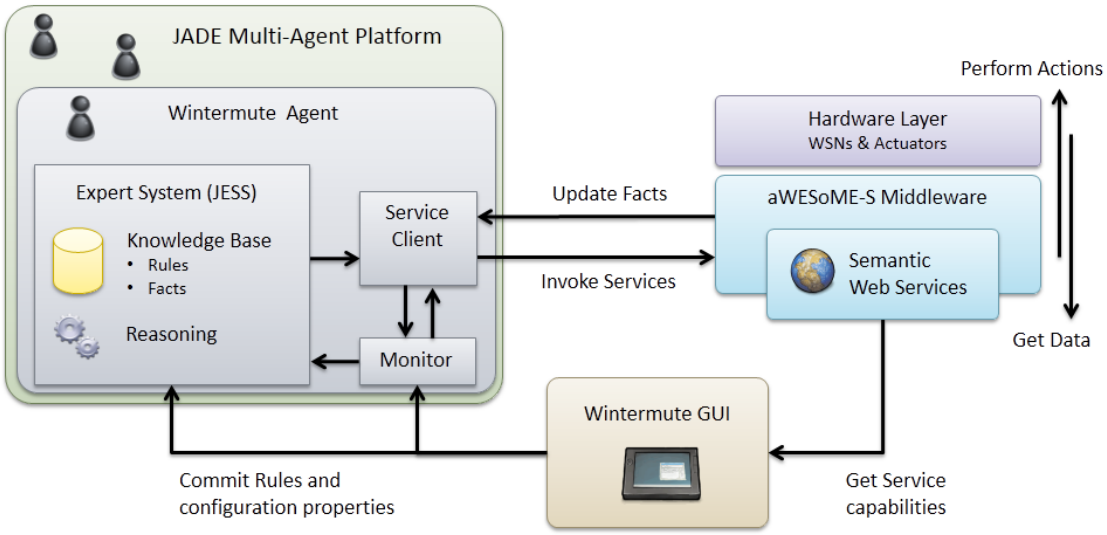

Figure 3. Wintermute agent architecture.

which can bear a different portion of the knowledge base and be assigned a different subset of tasks. For example, each Wintermute instance could be assigned the energy management policies for a different room. However, this work does not yet involve agent negotiation and collaboration techniques and, thus, from now on we consider a single instance of Wintermute responsible for all aspects of energy saving within the building.

Wintermute's GUI component provides a connection between the system and human user administrators. The GUI is semantically-enabled, using information from semantic service descriptions to help users author policies, which are later committed to the Wintermute agent's knowledge base. The application is developed in Java and the JavaFX UI platform. A primary purpose is to allow and facilitate users to author rules that represent user-defined energy-saving policies. Towards this end, the tool parses Semantic Web Service descriptions (SAWSDL files) of the services available in the Smart IHU environment. These machine interpretable semantic descriptions enhance the effectiveness, scalability and ease-of-use of the application. Specifically, each service operation is either annotated as ActuatorOperation or SensoryOperation (Table 1), both subclasses of the Operation ontology class. Operations annotated with SensoryOperation belong to sensors and retrieve sensory output. Hence, a direct convention can be made, that all sensory operations are suitable to be used as rule condition predicates (LHS), but are completely unfit for rule actions (RHS). Similarly, service operations annotated with ActuatorOperation are fit for rule action predicates. As a result, the application is aware, at any given time, about which operations should be presented during the condition (rule body) or action (rule head) authoring process of a rule. This also enhances the extensibility of the application, when adding completely new services, and its adaptability, as service providers interleave the ambient environments. To implement the service description parser, we used the *easyWSDL* toolbox[8].

Figure 4 shows the main window of Wintermute's GUI. The right panel shows a collection of rules loaded in the workspace, while the left pane allows authoring or editing an existing rule. Semantic properties of services are loaded in a dropdown box in order for the user to select available conditions or actions dynamically. Additionally to rules, the GUI also provides the agent with configuration properties, such as the time interval for the monitoring loop. Users can set a monitoring period in real time according to the urgency of policies. When a high sampling frequency is selected, the agent is fast and responsive, while a low frequency gives room to other service client applications running concurrently.

[8] The easyWSDL toolbox: http://easywsdl.ow2.org/

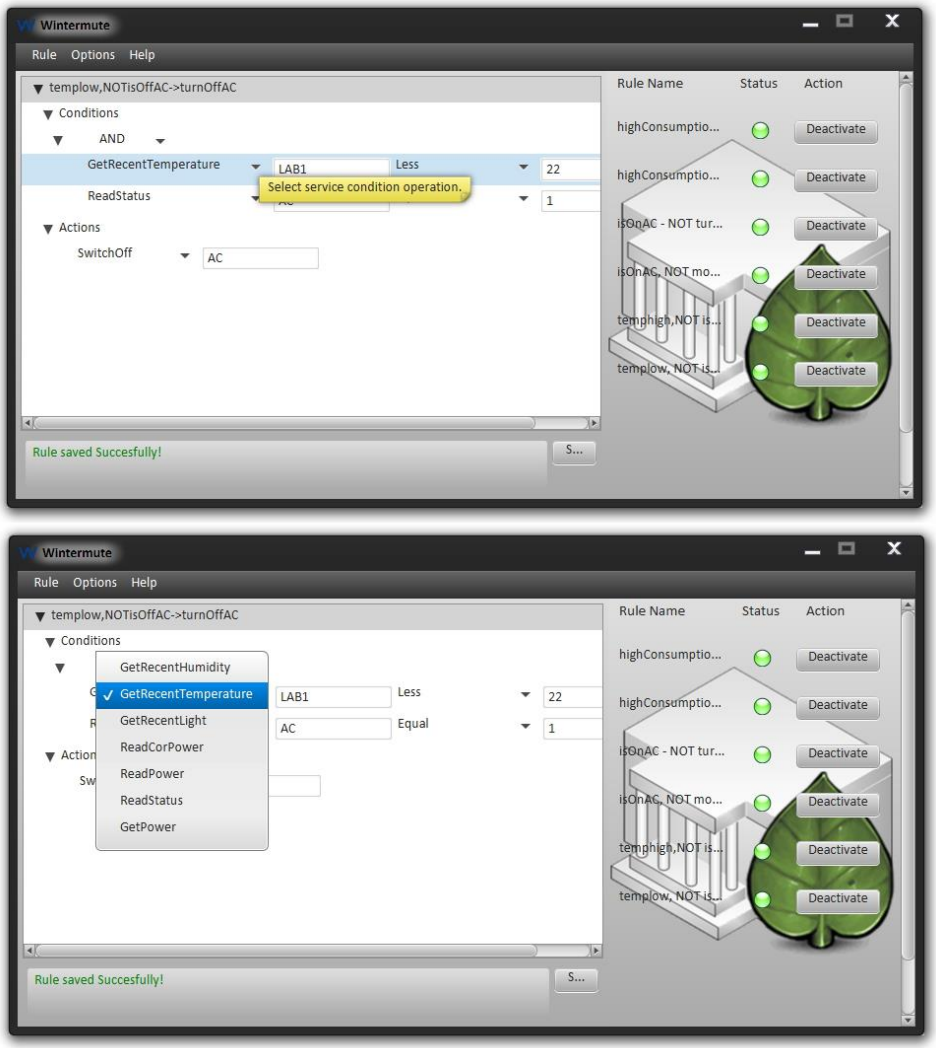

Figure 4. Both images illustrate the Wintermute GUI and its rule editing capabilities: rule conditions and actions can be manipulated accordingly in a user-friendly manner (top) while the user is able to select conditions and actions from a dropdown list, dynamically retrieved from semantic web service descriptions (bottom).

The rule inference engine in the core of Wintermute is JESS. The agent's knowledge base is filled with facts and rules about the state of the building, environmental-, power- and energy-wise. Rules are authored by the user over the GUI and represent energy saving or user comfort policies. The agent semantically parses these rules and monitors aspects of interest that they may contain. A rule's conditions correspond to specific measurements that translate to particular sensors, e.g. a room's temperature sensor. The Monitor module periodically invokes the corresponding Semantic Web Service operations, through the Service Client, according to the sampling frequency set in the GUI. Responses are inserted as facts to the Knowledge Base. This way, the agent does not have to constantly monitor the entire infrastructure, thus, saving traffic load on the Web Server. The monitored values take the form of deffact constructs in the knowledge base that represent a model of the current knowledge about the world's state, while rules are represented as JESS defrule constructs. For instance, a rule for switching off the cooler when the temperature is below a given threshold would be expressed as follows:

```
(defrule ruleTemp
   (status 99678C on)
   (temperature 8C3C0A ?x0)
```

```
    (test (< ?x0 25))
=>
    (call SwitchOff 99678C)
)
```

where `8C3C0A` and `99678C` are the IDs for the temperature sensor and cooler actuator, respectively.

At the same time, the JESS inference engine checks for activated rules by matching patterns. Upon activation, actuator web services are invoked to apply the appropriate actions. Since, even with numerous actuators on various appliances, the platform is still much less flexible than a human being, email alerts are in some cases issued to the system administrator. The latter can then physically act himself or simply take notice on needed changes. In the future, a wider variety of more sophisticated actuators, perhaps even robotic agents, could take over the workload from humans.

Notice that in case multiple rules fire, the system is not able to automatically handle conflicts, e.g. two rules invoke contradictory actions. When such behavior is observed, manual re-authoring of the rule base is needed. This drawback of reactive rules has been alleviated by using defeasible logic as an alternative approach, as shown in the next section. Section 5.3 elaborates on the comparison between the two rule paradigms. Additionally, checking a device's status on every rule's condition, as in the above example, dramatically decreases the possibility of firing the same rules in consecutive cycles. Even if that happens, the system prevents failures by handling this on middleware level i.e. the middleware ignores an on (or off) command if a device is already on (or off). A final precaution in the context of the reactive approach, before moving on to defeasible logic, would be to alert the system administrator when such loops of consecutive rules occur, as this would mean that the system is not able to achieve the desired goals. Such alerts are, however, a purely technical matter and, thus, not examined in this work.

4.6. Deliberative Defeasible Logic Agent (DeL)

Building on top of Smart IHU's middleware layer, the defeasible logic (DeL) agent is an autonomous software tool responsible for maintaining user comfort and improving energy consumption patterns. Its purpose is more or less identical to that of Wintermute; both agents aim to provide an overall user-friendly interface for system administrators and energy experts to formulate energy-saving and preservation policies. The agents are then responsible for autonomously carrying out these policies, manipulating the environment via the actuators. However, DeL features a comparatively more "deliberative" behavior, entailing at the same time a different software architecture. More specifically, DeL offers more flexibility and user-friendliness in the representation of the policies, providing the advantages outlined previously in this paper (see Sections 3.1 and 3.3).

4.6.1. Defeasible Logic Rule Base

Based on the superiority relationship feature of defeasible logics (see Section 3.2), DeL presents the user with the definition of three distinct rule sets $\mathcal{P}$, $\mathcal{M}$ and $\mathcal{E}$:

- *Preferences rules* ($\mathcal{P}$): These are rules that deal with the comfort of the user and his/her personal preferences, like e.g. the desired room temperature during the winter months. A sample preference rule is r_2 from Section 3.2: `tempHigh` $\Rightarrow$ `switchOnCooler`.
- *Maintenance rules* ($\mathcal{M}$): Namely, the rules that formulate the power management scheme of the building, like e.g. the deactivation of certain devices or appliances during late night hours. A maintenance rule paradigm is: `highConsumption` $\Rightarrow$ `switchOffCooler`.
- *Emergency rules* ($\mathcal{E}$): These are rules that are triggered in case of an emergency, like e.g. the activation of emergency lights in the case of fire, which could be represented by the following rule: `fire` $\Rightarrow$ `switchOnLights`.

If $\mathcal{R}$ is the set of all rules in the framework, then for the three rule sets: $\mathcal{P} \cup \mathcal{M} \cup \mathcal{E} \equiv \mathcal{R}$. Hence, in our approach rules can belong to preference, maintenance or emergency rule sets, adding an intuitive and effective classification practice for energy experts. Additionally, this approach is a refined version of the rules deployed in the Wintermute agent (see Section 4.5).

Regarding the scope of each rule set, $\mathcal{P}$ is restricted to isolated rooms, since it contains rules that deal with the user-defined "micro-management" of the conditions inside a room. The scopes of the other two rule sets are broader, covering the whole building where the framework is deployed, including the rooms as well as the rest of the other shared areas and facilities, like corridors and storage rooms. The threef rule sets demonstrate escalating priority and, more specifically: $\forall\, p \in \mathcal{P}, \forall\, m \in \mathcal{M}, \forall\, e \in \mathcal{E} \rightarrow p < m < e$. Notice that rules belonging to any of the three sets must – in essence – be defeasible, in order for the priority relationship to be effective.

4.6.2. Defeasible Reasoning

For the reasoning process, DeL incorporates an instance of *SPINdle* [39], a Java-based state-of-the-art defeasible reasoning engine. Although SPINdle lacks a theory grounding mechanism (i.e. no variables can be used in the predicates), which would be greatly beneficial for the purposes of this work, the engine's advantages (fast, reliable, highly integrated and easily-deployable) constitute the incentive for preferring the specific reasoner. The lack of grounding mechanism is solved in our approach via replacing atoms containing arguments and variables with appropriately composed sets of synthetic predicates and appending them to the knowledge base. For example, atom `switchOn(cooler)` becomes `switchOn_cooler`, while atom `switchOn(X)` is replaced by a set of synthetic predicates {`switchOn_device_1`, `switchOn_device_2`, …} for all the registered devices. In case SPINDLE supports first-order theories in the future, this pre-processing phase will not be necessary any more.

Figure 5 depicts the software architecture of the DeL agent and the information flow between its components. At the topmost level, human administrators and domain experts use an interface to author policies for the agent to enforce. Contrary to Wintermute, DeL does not feature an integrated GUI for rule authoring, but one could separately download one of the available defeasible logic rule editors, like, e.g. *SPINdle's Defeasible Logic Theory Editor*[9], or the much more flexible *S*^{2}*DRREd* (*Syntactic-Semantic Defeasible Reasoning Rule Editor*) [40].

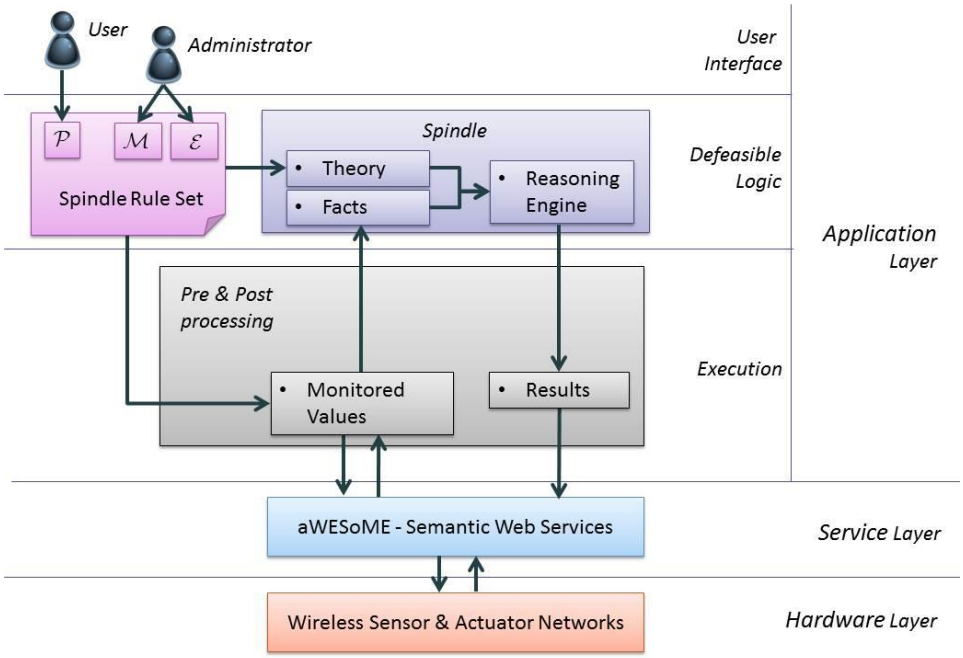

Figure 5. Architecture of the DeL agent in the application layer.

[9]

The latter adds a supplementary level of semantic assistance during rule base development, creating meta-models of the main defeasible logic theory notions and, thus, offering the flexibility of authoring rule bases of various syntaxes.

Figure 6 displays the main window of the tool as well as its *STM* (*Semantic Tag Mapping*) window that provides a meta-modeling facility for generating schemas over various language versions. A meta-model provides a schema for semantic data, specifying what elements may be contained in the model and how they relate to one another. In essence, each meta-model is a specification of a domain-specific modeling language.

Notice that each type of system user is granted authoring rights for different rule subsets (see previous subsection). Either manually or via the help of a specialized GUI, a part of the authored policies is translated into SPINdle-compliant syntax and formulates the initial rule base of the system, while the remainder needs to undergo a pre- and post-processing phase at the execution layer. The purpose of this layer is to handle rules with conditions containing predicates that require the invocation of sensors via the middleware layer of the architecture. Thus, the literals of the rule conditions have to be transformed into a suitable syntax for invoking the respective Web Services. For example, the `motion` predicate is replaced by a '*ZWaveService.GetMotion*()' service operation call. This preprocessing transformation is currently performed via a lookup table, but future plans involve integrating this association in the back-end ontology of the system. Finally, instead of serially examining all conditions, each service operation is only invoked once, as duplicate conditions may exist within the rule set.

Transcending from the Middleware (or Service) layer to the hardware layer, values are retrieved from the distributed wireless sensor networks across the building. Upon completion, all predicates are replaced with numeric values forming the actual facts to be appended to the knowledge base of the SPINdle reasoner, again with the help of the lookup service. This is the reverse process (i.e. post-processing) that has to take place when the readings of the sensors are retrieved. Notice that, rule conditions typically involve sensor readings and rule conclusions usually point to actuator activities. However, contrary to Wintermute's purely reactive rules, DeL's defeasible logic allows for more complex reasoning, where the conclusions of a rule may serve as conditions for further rules. Although our rule bases do not currently include such cases, this feature is already handled by the aforementioned lookup service that clearly distinguishes sensor readings and actuator activities.

The reasoning process is performed once in the beginning, when the knowledge base is complete after the pre- and post-processing phases, and afterwards it is repeated every time the knowledge base is updated with new facts. The reasoning results from each cycle have to go through the same processing steps and then to the invocation of the respective services (if any). Following this pattern, e.g. the `switchOnLight` derived predicate would point to the '*SmartPlugService.SwitchOn*(*Light*)' service operation call. Regarding the rest of the predicates, they are all expressed in direct correspondence to service operation names on Table 1 via the lookup table. Through service invocation, wireless actuator networks in the building are used to apply the desired effects. In other cases, services simply perform software operations such as sending an email alert.

5. Use Case: International Hellenic University Building A

As already stated, the proposed framework is applied at the International Hellenic University (IHU) Building A; this section demonstrates the deployment of the policies and their impact on the overall energy consumption. The specific building is a large, appliance-rich building that bears a restrictive and wasteful infrastructure. Thus, it can serve as a stimulating test-bed of our methodology, which can then be applied to other buildings as well.

The percentage of energy savings accomplished by any method is only subject to a building's prior state of efficient usage, i.e. the greater the waste the greater the savings are. Thus, prior to formulating the energy management scheme policies, the building's current behavioral patterns had to be observed and identified, at least as far as energy consumption is concerned.

5.1. Current Scheme

As presented in previous attempts [21], the hardware, Web Service middleware and appropriate client applications have already provided a basis for comprehensive monitoring and management of energy consumption along with environmental sensor readings. Apart from desktop and mobile administration software, a Web application[10] makes energy monitoring publicly accessible to any external viewer as well.

Observing the vast energy consumption of the building in total, one has to initially disaggregate it to smaller clusters of appliances and/or activities. Attaching a sensor on the data center's power supply allows us to directly isolate its power consumption from the rest of the building. Unfortunately, the data center consisting of servers and cooling units is already configured to its energy-consumption minimum for adequate cooling, meaning that its consumption (~ 50% of total daily consumption) has to be taken as granted. The remaining consumption refers to appliances in the building and comprises mainly of computers, lighting, heating and cooling units and a few projectors, peripherals, coffee makers and kitchen appliances. Most of this equipment is monitored separately using individual meters/actuators. Interesting observations include many appliances left constantly on, especially the high consuming photocopiers. Lighting during the night is already working on schedule and is responsible for the nightly consumption along with the stand-by consumption of all appliances.

However, taking action on heating and cooling is moderately restricted due to the nature of the building's existing infrastructure. Heating and cooling both take place in a central boiler/cooler and are distributed to the building through fan coil endpoints, typically one per room. During the winter, the central heating system consumes oil, which translates to less power usage. The cooling system, on the other hand, uses a considerably large amount of energy per day to centrally freeze water. During the winter, fan coils just let heat pass through, even while powered off. When powered on, the fans accelerate heat dissemination. Conversely, during the summer, fan coils provide utility only while they are on, while having the fans off actually saves up on the central cooling system consumption, as it has to freeze less water over time. However, most of the fan coils in the building would have to coordinate for these savings to take effect. Overall, the fans themselves hold an insignificant amount of consumption even as a whole. Observing the yearly course of the seasons, consumption of appliances was thus found to be much lower, since oil is used and fan coils are mostly powered off. Additionally, the university employees do keep the fan coil usage to a minimum. Combined with their very small consumption, their daily energy needs go unnoticed in total. All the above, result in much less energy spent for heating compared to cooling and, thus, much less saving potential.

A final observation was made during the early morning hours. Every morning, from 6am to 8am, the total power consumption increases by 5kW, resulting in 10kWh of energy per day, in average, measured over the last year (higher during working days and lower during weekends). Inspecting the appliances, this amount comes from the building lighting being switched on by the early morning cleaning services. Since the cleaning crew does not need constant lighting to the whole university for this course of two hours, there is definitely some space for savings here.

5.2. Applying Defeasible Logics – Sample Policies

This subsection features a simplified example demonstrating the deployment of the policies rule base and the improved flexibility offered by the different rule sets (see Section 4.6.1). Suppose that the following example explicitly involves room 'A6' of IHU's Building A and revolves around the operation of the room's cooler during the summer months. Towards this affair, we can formulate the three distinct rule sets $\mathcal{P}$, $\mathcal{M}$ and $\mathcal{E}$ that, as already stated, represent the policies for preference, maintenance and emergency, respectively.

To begin with, let's suppose that the employee working in room 'A6' defines the following preferences policy (rule set $\mathcal{P}$) regarding the operation of the room's cooler:

p_{01}: `temp(a6,X), X < 18` → `tempLow(a6)`

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

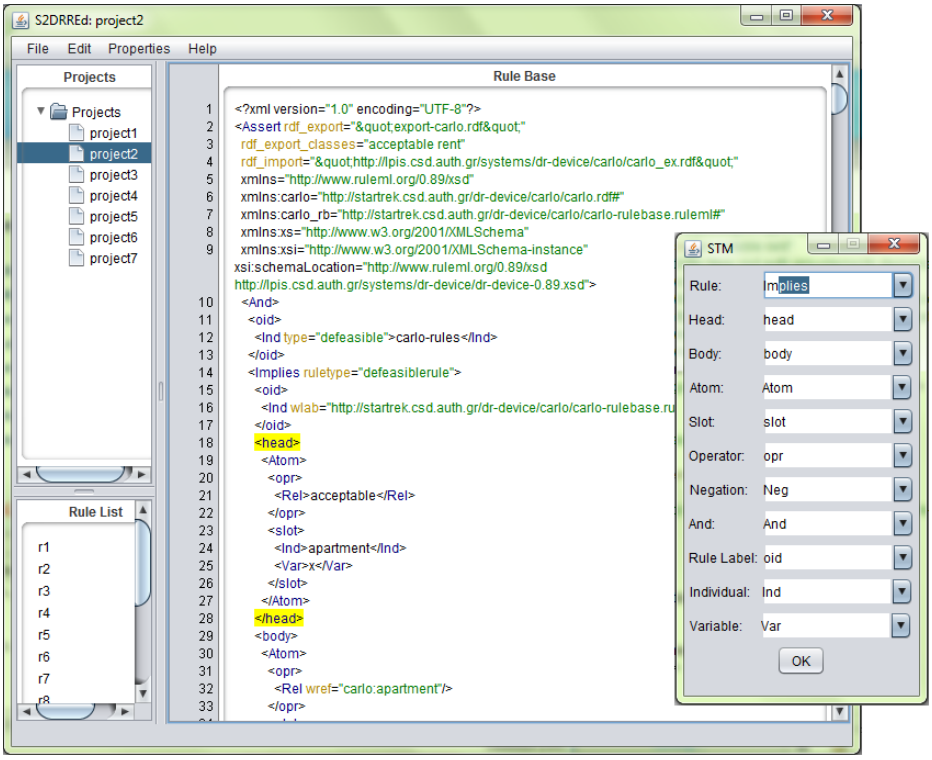

Figure 6. S[2]DRREd Main and STM windows.

p_{02}: temp(a6,X), X > 28 → tempHigh(a6)

p_1: tempHigh(a6) ⇒ switchOn(a6,cooler)

p_2: ¬tempHigh(a6) ⇒ switchOff(a6,cooler)

As mentioned before, the preference rules primarily handle user comfort. Thus, according to the above preferences, the employee-user determines the thresholds that define – by personal standards – the 'low' and 'high' temperatures in the room and designates when the cooler should be turned on or off in accordance to these thresholds. Preference policies like these allow for a periodic use of fan coils, instead of the non-stop, manually scheduled usage in the past. Usage history has shown that employees usually do not mind crossing the temperature thresholds and leave the appliances on, even after leaving the room they are working in. The periodic usage of coolers has a positive impact on the building daily usage, removing the fan coil footprint itself, but most importantly by saving up on the central cooling system consumption, less water has to freeze over time.

Moving on to the system's administrator, a maintenance policy (rule set $\mathcal{M}$) has to be defined, in order to optimize the overall operation of the building and its respective power consumption. Maintenance rules tend to enforce further savings, often compromising the comfort requested by the user-defined preference rules. The scope of the following rules lies again in the operation of coolers in rooms:

m_{01}: consumption(X,Y), Y > 2000 → savingMode(X)

m_{02}: time(X,Y), Y > 2200 → savingMode(X)

m_1: ¬motion(X) ⇒ switchOff(X,cooler)

m_2: motion(X) ⇒ ¬switchOff(X,cooler)

m_3: functioning(X,pc) ⇒ ¬switchOff(X,cooler)

m_4: savingMode(X) $\Rightarrow$ switchOff(X,cooler)

m_{03}: isOn(X,cooler) $\rightsquigarrow \neg$switchOn(X,cooler)

m_{04}: isOff(X,cooler) $\rightsquigarrow \neg$switchOff(X,cooler)

$m_4 > m_2$

$m_4 > m_3$

$m_1 > m_3$

As can be observed, the administrator starts by determining the conditions for having each room switch to 'saving mode', during which the system will attempt to minimize consumption. More specifically, rule m_{01} deals with the total energy consumption of each room, handling flexible loads to ensure a total consumption per room is not exceeded in any given moment. Rule m_{02} handles the operation of the room's appliances during late night hours[11]. The following four rules, m_1 to m_4, deal with user context: they determine when the cooler will switch off, depending on the presence of people in the room (rules m_1 and m_2) and whether an employee works on his/her PC (rule m_3) and they also handle the activation of the 'saving mode' (rule m_4). Also, the last two defeaters make sure that the cooler's actuator will not attempt to turn on an already operating cooler and, conversely, will not attempt to turn off a cooler that has already been switched off. Finally, the policy includes a set of binary superiority relationships among conflicting rules in set $\mathcal{M}$.

Eventually, the administrator has to formulate the policy for emergency situations (rule set $\mathcal{E}$):

e_{01}: smoke(X) $\rightarrow$ alert(X)

e_{02}: highCO2(X) $\rightarrow$ alert(X)

e_1: alert(X) $\Rightarrow$ switchOn(X,alarm)

e_2: alert(X) $\Rightarrow$ switchOff(X,cooler)

When a sensor reading is received that implies unsafe conditions inside a room, like e.g. smoke or too high CO_2 concentration (or potentially other measured values exceeding safety limits), the system activates the room's alert (rules e_{01} and e_{02}). The alert then handles the operation of the appliances in the room accordingly (rules e_1 and e_2).

Concluding the example, the following set of conflicting literals (see Section 3.2) has to be inserted into the rule base:

$\mathcal{C}$ = {switchOn(X,Y), switchOff(X,Y)}

This way, a conflict is automatically generated between pairs of rules, where one rule concludes that a device (e.g. cooler, alarm etc.) should be turned on and the other one concludes that it should be turned off and vice-versa. These conflicts will then have to be resolved via appropriate superiority relationships among each pair of conflicting rules. In the cases of conflicts between rules belonging to the same rule set (e.g. both rules are maintenance rules), the superiority relationship has to be explicitly added to the knowledge base by the user with the appropriate access permissions (in the case of maintenance rules this would be the administrator). On the other hand, the cases of conflicts between rules belonging to different rule sets is handled automatically by the system, via rewriting the theory and adding the respective superiority relationships, according to the scope of each rule set. Thus, for the specific rule set presented in this subsection, the following superiority relationships are appended to the rule base:

[11] Other appliances that could potentially benefit from this operation are the printing devices that typically remain on during the night and are responsible for some major energy waste due to periodic calibrations. Having the system automatically switch off the printers according to timely schedule would provide a solution. During the past year, though, many students have visited the computer lab after late hours, wanting to print out lecture notes. Thus, an appropriate rule should also turn on the photocopiers in case of detected motion. Of course, this functionality can easily be integrated into the system, but is omitted here due to simplicity.

$m_1 > p_1$ $\quad$ $e_2 > p_1$

$m_2 > p_2$ $\quad$ $e_2 > m_2$

$m_3 > p_2$ $\quad$ $e_2 > m_3$

$m_4 > p_1$

Understandably, the overall sample rule base presented here is rather simplistic, but hopefully demonstrates the expressiveness and user-friendliness of the defeasible logic representation. Moreover, the flexibility of the conflict resolution mechanisms of defeasible reasoning combined with the three distinct rule sets of escalating priority proposed in this approach, produce a versatile Smart Building management system. To give a deeper insight, we present the following brief examples.

Example 1

Suppose that the following facts, generated by respective sensor readings in room A6, are inserted into the above rule base:

f_1: `temp(a6,30)`

f_2: `time(2300)`

f_3: `motion(a6)`

Fact f_1 triggers rule p_{02}, whose derived conclusion matches the condition of rule p_1. At the same time, fact f_2 triggers rule m_{02} and the derived conclusion of the latter matches the condition of rule m_4. Finally, fact f_3 triggers rule m_2. Thus, a conflict is initially observed among rules m_2 and m_4 within rule set $\mathcal{M}$, because the conclusion of the former rule is the negation of the conclusion of the latter. The conflict is resolved via the superiority relationship $m_4 > m_2$ and it is eventually concluded that `switchOff(a6,cooler)`. Furthermore, by observing rules p_1 and m_4 one can detect that the two conclusions `switchOn(a6,cooler)` and `switchOff(a6,cooler)` contradict each other, because of the set of conflicting literals $\mathcal{C}$. Thus, it is not clear whether the cooler should ultimately be turned on or off. However, since all maintenance rules have a higher priority than preference rules (see previously), the system automatically infers that $p_1 < m_4$ and the maintenance rule eventually prevails, switching off the cooler.

Example 2

Suppose that at a certain point in time an additional fact is inserted into the above rule base, indicating a high level of CO_2 in the room:

f_4: `highCO2(a6).`

The rule engine will execute the rules once again. Fact f_4 triggers rule e_{02}, whose derived conclusion matches the conditions of rules e_1 and e_2. Rule e_1 is not attacked by any other rule, and as a result, the alarm in room A6 is activated. However, the conclusion of rule e_2 contradicts the conclusions of rules m_2 and p_1. But, as already stated, all emergency rules have top priority when attacked by preference or maintenance rules. Therefore, it is automatically inferred that $p_1 < e_2$ and $m_2 < e_2$ and it is concluded that the cooler should be switched off. However, since the device is already switched off from the previous rule execution cycle, defeater m_{04} prohibits the cooler's actuator from re-issuing the command. Demonstrating the dynamicity of the system, if CO_2 levels fall back to normal later on, the fact base will be updated resulting to a new execution of the rule set and a new set of derived conclusions.

As already mentioned before, SPINdle does not support theory grounding and, additionally, features like conflicting literals are substituted by appropriate sets of conventional defeasible rules and superiority relationships before being submitted to the reasoner. Appendix A illustrates the above theory, together with the facts used in the two previous examples, in the SPINdle-specific syntax. The transformation from the initial syntax, or any other

input rule syntax like e.g. *defeasible RuleML*[12], to the target syntax constitutes a task assigned to the end-user/administrator and can be achieved via appropriate automatized functionality of a specialized GUI used for authoring the rule base (see Section 4.6.2). Since DeL does not yet feature such a specialized GUI, this automatic transformation functionality is not yet integrated into the system.

5.3. Comparison of the Two Rule Paradigms

As demonstrated in the previous subsection, the representation of policies via defeasible logic is highly intuitive and flexible. In order to demonstrate these advantages, we present the reader with rule p_1 from the above rule base, expressed, however, as a reactive rule instead of a defeasible one:

```
tempHigh(a6), motion(a6), ¬savingMode(a6), ¬isOn(a6,cooler), ¬alert(a6)
    → switchOn(a6,cooler)
```

One can easily realize that the latter version of the rule is required to include all possible exceptions (e.g. `¬savingMode`) in its body. It should be mentioned here that negation in the reactive rule setting is not strong negation, but negation-as-failure, i.e. a negated atom is true, when it is not present in the current content of the knowledge base. These extensive representations for expressing all possible exceptions to a rule are counterintuitive, since in cases of hundreds of rules the resulting representations would be greatly impractical.

Additionally, this peculiarity makes the whole system less flexible, as it would require the end-user to have access to (and knowledge of) parts of the internal policies, namely, more than his preferences, which does not comply with the end user's access levels. Likewise, maintenance rules would need to have access to predicates belonging to emergency policies, ultimately rendering useless the initial distinction in three separate classes (preferences, maintenance, and emergency).

Finally, as already mentioned in Section 4.5, the production system cannot easily handle rule conflicts that arise when two rules invoke contradictory actions. Such conflicts could be resolved by rewriting (parts of) the rule base, which is a counterintuitive approach. On the other hand, defeasible logic offers more intuitive conflict resolution mechanisms, via rule superiorities and sets of conflicting literals.

5.4. Experimental Evaluation

Before deploying the agent policies presented above, we carefully examined the building's consumption patterns, and tried to correlate them as much as possible to known parameters such as activities and weather. Using the existing infrastructure, one is able to disaggregate total building consumption to: (a) Data center consumption, which is nearly 50%, and, (b) the rest of the appliances, including office and classroom equipment, lighting, cooling and heating. After that, the most apparent energy loss in daily patterns can be observed during early morning hours between 6am and 8am.

Figure 7shows a typical daily pattern of the past year, up to 15/6/2013. During these two hours, the only significant human activity taking place in the university is cleaning, which causes the unaccounted for peak in energy. The cleaning crew is used to leave the lights on while continuously interleaving rooms. As lights are the most power-demanding single appliance in the building, they had to be kept off as long as possible according to frequent non-motion detection. Hence, the experimental evaluation proposed here employs a composite method: each of the two proposed agents is assigned a different, distinct, time window within the day, according to each agent's suitability for the different tasks. That way, the agents complement each other rather than compete; the reactive agent handles the (straightforward) morning task that requires responsiveness, while the deliberative agent handles more complex tasks occurring during the rest of the day.

The reactive agent (Wintermute), equipped with an appropriately confined rule-set, is perfectly suitable for frequently pulling and responding to a few observations. As already mentioned, both hardware and middleware are essentially pull-based. To minimize overhead, the agent iteratively pulls the necessary data only, and as rarely

[12] Schema Specification of Defeasible RuleML Version 1.0: http://ruleml.org/1.0/defeasible/defeasible.html

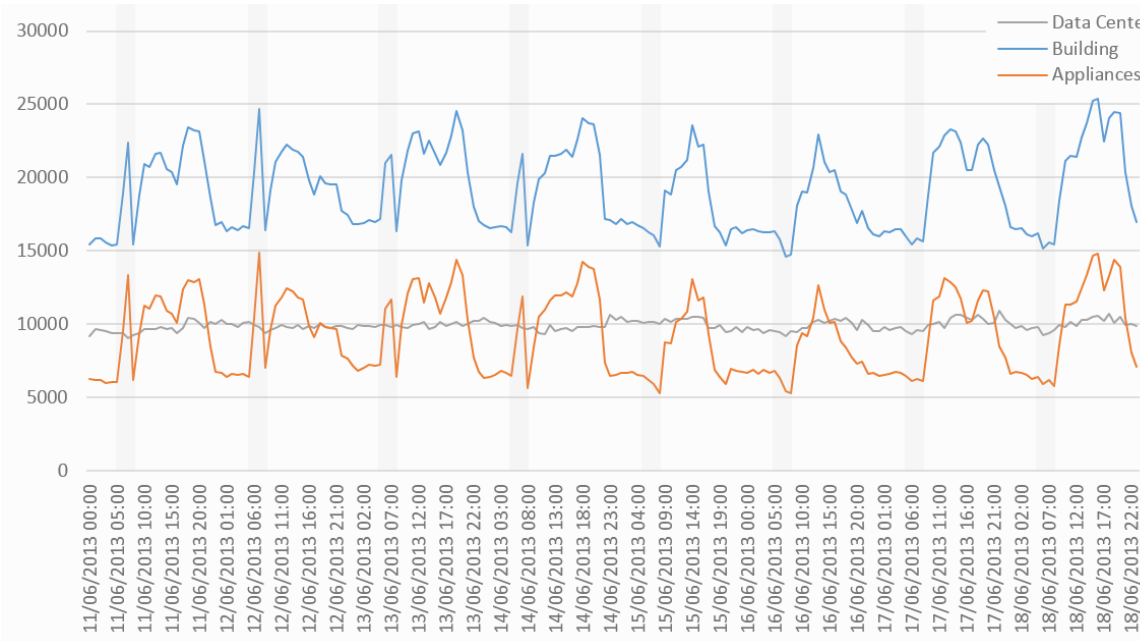

Figure 7. Building, data center and appliance consumption during a portion of the experiment. The installation of the system on 15th July results in the elimination of early morning energy waste during the responsive time window, which is highlighted.

as possible, so that resources are kept free for other agents potentially co-existing in the environment. Data is not lost, but rather received later than the time of observation. However, as already stated in Section 4.3, the nature of our application is energy-saving and not emergency alerts, so responding rapidly to sensor readings is not a key requirement.

On the other hand, the deliberative agent (DeL) was assigned the rest of the day (0am – 6am and 8am – 12pm), for the same time period, in order to ensure minimum power waste originating from misuse of office equipment and cooling. DeL was considered suitable for this task, since handling the lights and other equipment should be made after considerable observations, so that conditions have been stabilized, e.g. to be sure that someone has left the office.

The entire experiment was carried out from 15/6/2013 to 26/6/2013, while an equal time interval of the previous 10 days (5/6/2013 to 14/6/2013) serves as a baseline for the comparison to ensure maximum similarity in weather and activity conditions. During this period there were no significant changes in climate (even sunrise, which greatly affects 6am-8am behavior) and curricular activity. As shown in Figure 7, the morning peak during the reactive time window was completely eliminated during the experiment, guaranteeing 41% savings during that time. This amount of daily energy savings averages at 8688.4Wh, which accounts for 4% of past average daily consumption (2222051.2Wh).

However, experimental results are less optimistic during DeL's operational time window, where no apparent savings are noted. Although the agent did take decisions and switched on and off appliances, their cumulative effect was not enough to notice on such a large scale. As mentioned before, no activity changes were noted, which means that all employees were at their desks during working hours and using lights and office equipment consistently. Hence, no savings were observed, but, on the other hand, the agent ensured that no energy was wasted. Additionally, we concluded that stricter monitoring, e.g. accounting people present at a time, was needed to detect minor fluctuations in activities that apparently reflect on higher energy consumption. This is due to the fact that various events (workshops and lectures) taking place at the university premises, which occur at a non-periodic fashion, result in small changes that are harder to detect.

Figure 8 shows the daily hourly average consumption showing both time windows, and 24h total in order to put them into perspective. Again, the elimination of a large amount of energy during the reactive window is apparent, showing its impact also on the 24h total daily consumption. Interestingly, the savings originating from the smaller, two-hour, window have a noticeable impact on the daily consumption. Please note here, that the presented approach is a composite one, consisting of both agents acting in a mutually-exclusive way in time. In other words, presenting either agent perform outside the time window for which they were especially designed seems both irrelevant and uninteresting. Aspects of both agents discussed throughout the text emphasize the fact that they would naturally not reduce consumption in unsuited tasks.

All in all, the experimental integration of the agents in the Smart IHU architecture guaranteed at least 4% daily energy savings over the past week (also 4% measured over the past year) or 2% including the data center, which is a significant percentage considering the large scale of energy consumed by a university building. To put it in perspective, the university consumes around 438.2kWh per day (average of past year), around half of which belongs to the data center (221.5kWh). This amount could easily be matched to a monthly consumption of a small household, while the amount of 2% could equal the total daily consumption of that household.

As a final remark, the amount and percentage of the system's savings are only relevant to how much energy was being wasted in the first place. All in all, the system can only guarantee to enforce all energy-saving behavior and ensure that energy-wasting behavior is kept to a minimum.

On improving savings, we believe that major savings in this setting can occur when collectively shutting down cooling units. Specifically, asynchronously switching off single cooling units on the ground floor during the experiment saved insignificant amounts of energy. Instead, simultaneously turning off the majority of cooling units, also on the whole building (where the system has yet to be deployed), will result in major savings as it would hibernate the main cooling supplier unit. To explore this direction, we plan to extend the system on the whole building and employ agent cooperation and negotiation between rooms to timely and collectively shut down the main cooling supplier.

6. Critical Discussion and Future Work

This subsection discusses the most significant limitations of the proposed framework and suggests possible ways to tackle each one of them, giving at the same time appealing insights into directions for future work.

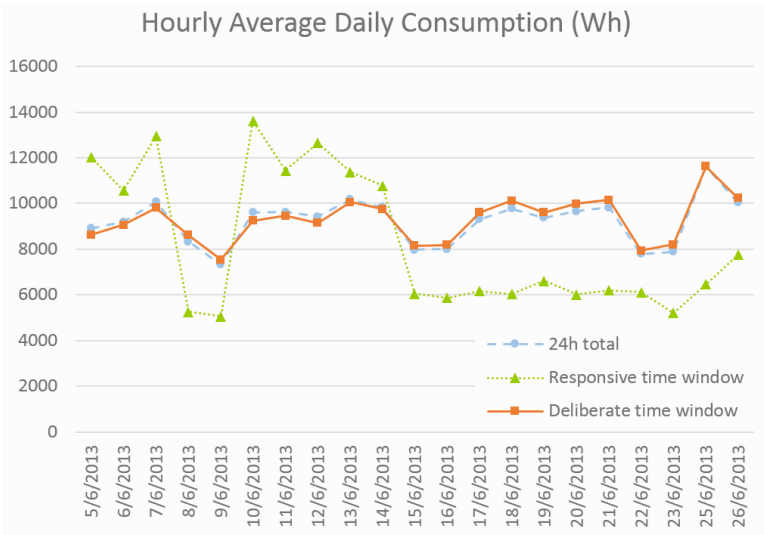

Figure 8. Hourly consumption daily average (in Wh), showing 24 total, deliberative time window and reactive time window.

The previous sections demonstrated the features of the two distinct agent types (reactive and deliberative), presented their respective pros and cons and discussed their role inside the Smart IHU AmI architecture. To recap, the reactive agent is based on production rules and naturally excels at instantly invoking actuators, thus, managing the reactive operation of devices in cases when response time is critical. On the other hand, the deliberative agent offers higher expressiveness and more intuitive conflict resolution mechanisms, permitting a clear distinction of the different rule sets and access levels. A further advantage of this agent type is the fact that it can reason on more complex rule bases, where the conclusions of a rule may serve as conditions for other rules etc. This capability provides an internal state to the agent as well as a more complete modeling of the environment.

Although both agent types bring different elements to the table, it is not necessary to choose one over the other. Instead, a *hybrid* agent type has been proposed in literature that is concurrently capable of reactive and proactive behavior [41]. Hybrid agents typically integrate – amongst others – two distinct layers in their architecture that deal respectively with the two types of behaviors (reactive and proactive). The specific architecture of the agent, nevertheless, depends on the layering as well as the information and control flows within the layers. Though stimulating as an idea, the hybrid agent type is out of the scope of this paper and is included in plans for future development.

A limitation in the current implementation of the framework lies in the somewhat simplistic expressiveness of the policies. It would definitely be interesting to investigate incorporating more complex requirements, such as temporal reasoning. Recent advances in defeasible logics research have led to *temporal defeasible logics* (*TDL*) [42], which efficiently deal with temporalized durative facts and with potential delays between rule antecedents and consequences and overall capture the temporal aspect in our framework. Specifically the work presented in [43] proposes algorithms for computing extensions of TDL. Although SPINdle does not support TDL reasoning yet, we investigating the integration of temporal defeasible logics in our framework through the TDL-specific reasoner presented in [44].

Another limitation is that currently our system does not take into consideration potential *rule dependencies*. The most basic level of rule dependency suggests that rule B is considered dependent on rule A, if the former directly uses terms asserted by the latter. More advanced (e.g. semantic) types of dependencies may be determined in cases when terms appearing in a set of rules (conditions or actions) are somehow related to each other (e.g. semantically equivalent or subsumed). Therefore, even simple rule sets can result in a vast variety of rule interdependencies. However, these dependencies offer a very clear overview of the underlying logical layers present in a rule base, especially when the rules involved are at successive levels of abstraction, i.e. rules of upper layers using the results of inference produced by rules of the lower layers. Since we do not currently consider more than one logical layer of rule inference, there was no need in investigating that aspect of rule dependencies. However, as the system unavoidably evolves and becomes more and more complex, it would be interesting to investigate methodologies for automatically determining rule dependencies (e.g. see [45], where rule dependencies among SWRL rule sets are detected, taking into consideration the underlying OWL semantics, as well) and rule dependency visualization ([45], [46]).

Regarding the rest of the ideas for future improvements, besides investigating the integration of a hybrid agent type into the proposed architecture (see above), our future plans involve extending the scale and invasiveness of the experiments. Rules can grow in numbers, raising the load of operations to perform throughout the building. More interfering rules, such as killing the stand-by mode of PCs, will carefully be introduced. Such rules have been avoided thus far, due to their high intrusiveness and difficulty to detect when that device will be needed again (the placement of the appropriate hardware can be experimented with). Towards improving the tools accompanying the framework, a Web UI is being developed, in order to enhance user experience and convenience, but will obviously have no impact on the observed results.

Furthermore, the platform can be extended and enhanced on every layer. Regarding the service layer, the agents can benefit from automatic service discovery, for dynamically discovering or substituting services that interleave the environment in real time. This way, new portable and redundant service providers could be introduced into the system. On the user and rule layers, a machine learning component can be employed to discover regular consumption patterns and suggest respective energy-saving rules.

On the agent platform level, we will investigate integrating more instances of same or different agent types (reactive, deliberative or hybrid) to serve different roles. Each agent will be assigned a sub-task or a region within the building and negotiate with the others towards achieving a certain goal (e.g. comfort and/or energy savings). Agent negotiation will be needed especially for resolving potential conflicts among agents, e.g. when two or more agents are assigned the operation of the same device(s).

7. Conclusions

This work presents a building-wide real-world rule-based Ambient Intelligence application that aims for user comfort and energy savings. Multiple wireless sensor and actuator networks that comply with different product families and communication protocols have been deployed to collect environmental and energy data across a State University building. The infrastructure integrates a middleware providing service-orientation by unifying all underlying distributed and heterogeneous wireless sensor and actuator platforms. Two complementary and mutually exclusive rule-based approaches for managing energy saving schemes have been proposed: Wintermute and DeL.

The former involves a reactive rule-based agent, which enables administrators to author and enforce energy saving policies, as sets of purely reactive rules that run using a production rule engine. The latter, more advanced, approach involves a defeasible logic reasoner that delivers more intuitive policy authoring, offering sophisticated conflict representation and resolution mechanisms. A priority ordering for each rule type is defined and, via this capability, a priority scheme is proposed that categorizes rules in clusters of preference, maintenance and emergency policies.

For the experimental part, a careful analysis and monitoring of year-long consumption data has been performed and suitable policies were authored. A considerable amount of savings was achieved, but most importantly, the policies guarantee to keep any potential energy waste to a minimum. Another observation was that such real time systems require the use of more than one (type of) software agent to especially ensure instant response and reactiveness for the most important of policies.

8. Acknowledgements

The Smart IHU project is funded by Operational Program Education and Lifelong Learning, OPS200056 (International Hellenic University, Thessaloniki, Greece). The authors would like to thank the anonymous reviewers for their valuable comments and suggestions to improve the quality of this paper.

9. References

[1] Farhangi, H. The Path of the Smart Grid. *Power and Energy Magazine*, vol.8, no.1, pp. 18-28, IEEE, January-February 2010.

[2] Weiser, M. Some Computer Science Issues in Ubiquitous Computing. *Communications of the ACM*, 36 (7), pp. 74-84, 1993.

[3] Bottaro, A., & Gérodolle, A. (2008, July). Home SOA – facing protocol heterogeneity in pervasive applications. *Proc. Int. Conf. Pervasive Services* (pp. 73-80). ACM.

[4] Vallée, M., Ramparany, F., & Vercouter, L. (2005, May). A multi-agent system for dynamic service composition in ambient intelligence environments. *3rd Int. Conf. Pervasive Computing (PERVASIVE 2005)*, pp. 165-171.

[5] Cook, D. J., Augusto, J. C., & Jakkula, V. R. (2009). Ambient intelligence: Technologies, applications, and opportunities. *Pervasive and Mobile Computing*, 5(4), 277-298.

[6] Davidyuk, O., Georgantas, N., Issarny, V., & Riekki, J. (2011). MEDUSA: Middleware for End-User Composition. *Handbook of Research on Ambient Intelligence and Smart Environments: Trends and Perspectives* (1 Volume), 197.

[7] Papazoglou, M. P. Service-oriented Computing: Concepts, Characteristics and Directions. *Proc. Fourth Int. Conf. on Web Information Systems Engineering (WISE 2003)*, pp. 3-12, 10-12 Dec. 2003.

[8] Berners-Lee, T., Hendler, J., & Lassila, O. The Semantic Web. *Scientific American*, May 2001, p. 29-37.

[9] Kopecky, J., Vitvar, T., Bournez, C., & Farrell, J. SAWSDL: Semantic Annotations for WSDL and XML Schema. *Internet Computing*, 11(6), pp. 60-67, IEEE, Nov.-Dec. 2007.

[10] Nute, D. Defeasible Logic. *Handbook of Logic in Artificial Intelligence and Logic Programming*, Vol. 3: Non-monotonic Reasoning and Uncertain Reasoning, p.p. 353-395, Oxford University Press, 1994.

[11] Stavropoulos, T. G., Vrakas, D., & Vlahavas, I. A Survey of Service Composition in Ambient Intelligence Environments. *Artificial Intelligence Review*, pp. 1-24, 2011.

[12] Daniele, L., Costa, P. D., & Pires, L. F. Towards a Rule-based Approach for Context-aware Applications. *Dependable and Adaptable Networks and Services*, pp. 33-43, Springer Berlin Heidelberg, 2007.

[13] Etter, R., Dockhorn Costa, P., Broens, T. A Rule-based Approach towards Context-aware User Notification Services. *IEEE Int. Conf. on Pervasive Services 2006*, pp. 281-284, IEEE Computer Society Press, Los Alamitos, California, 2006.

[14] Yang, S. Y. Developing an Energy-saving and Case-based Reasoning Information Agent with Web Service and Ontology Techniques. *Expert Systems with Applications*, 40(9), pp. 3351-3369, July 2013.

[15] 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.

[16] Kiryakov, A., Ognyanov, D., & Manov, D. OWLIM - A Pragmatic Semantic Repository for OWL. *Proc. Int. Conf. on Web Information Systems Engineering (WISE'05)*, Dean, M., Guo, Y., Jun, W., Kaschek, R., & Krishnaswamy, S. (Eds.), pp. 182-192, Springer-Verlag, Berlin, Heidelberg, 2005.

[17] W3C, "SWRL: A Semantic Web Rule Language - W3C Member Submission," 21 May 2004. [Online]. Available: http://www.w3.org/Submission/SWRL/. [Accessed: February 2013].

[18] De Paola, A., Gaglio, S., Lo Re, G., & Ortolani, M. (2012). Sensor9k: A testbed for designing and experimenting with WSN-based ambient intelligence applications. *Pervasive and Mobile Computing*, *8*(3), 448-466.

[19] Yeh, H. W., Lu, C. H., Huang, Y. C., Yang, T. H., & Fu, L. C. (2011, December). Cloud-Enabled Adaptive Activity-Aware Energy-Saving System in a Dynamic Environment. In *Dependable, Autonomic and Secure Computing (DASC), 2011 IEEE Ninth International Conference on* (pp. 690-696). IEEE.

[20] Capone, A., Barros, M., Hrasnica, H., & Tompros, S. (2009, March). A new architecture for reduction of energy consumption of home appliances. In*TOWARDS eENVIRONMENT, European conference of the Czech Presidency of the Council of the EU* (pp. 1-8).

[21] Stavropoulos, T. G., Gottis, K., Vrakas, D., & Vlahavas, I. aWESoME: A web service middleware for ambient intelligence. *Expert Systems with Applications*, 40 (11), pp. 4380-4392, Elsevier, 2013.

[22] Hobold, G. C., & Siqueira, F. Discovery of Semantic Web Services Compositions based on SAWSDL Annotations. *Proc. 19th Int. Conf. on Web Services* (*ICWS'12*), pp. 280-287, IEEE, 2012.

[23] 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. In *Handbook of Ambient Intelligence and Smart Environments* (pp. 1139-1169). Springer US.

[24] Eisenhauer, M., Rosengren, P., & Antolin, P. (2010). Hydra: A development platform for integrating wireless devices and sensors into ambient intelligence systems. In *The Internet of Things* (pp. 367-373). Springer New York.

[25] Bikakis, A., Hassapis, P., & Antoniou, G.: Strategies for contextual reasoning with conflicts in ambient intelligence. *Knowledge and Information Systems*, 27(1), pp. 45-84, 2011.

[26] Bikakis, A., & Antoniou, G.: Defeasible Contextual Reasoning with Arguments in Ambient Intelligence. *IEEE Trans. on Knowledge and Data Engineering*, 22(11), pp. 1492-1506, 2010.

[27] Antoniou, G., Maher, M. J., Billington, D., & Governatori, G. A Comparison of Sceptical NAF-Free Logic Programming Approaches. *Proc. 5th Int. Conf. on Logic Programming and Nonmonotonic Reasoning* (*LPNMR '99*), Michael Gelfond, Nicola Leone, and Gerald Pfeifer (Eds.). Springer-Verlag, London, UK, p.p. 347-356, 1999.

[28] Maher M. J. Propositional Defeasible Logic has Linear Complexity. *Theory and Practice of Logic Programming*, 1(6), pp. 691-711, 2001.

[29] Chesñevar, C., Maguitman, A. ARGUNET: An Argument-based Recommender System for Solving Web Search Queries. *Proc. 2nd IEEE Int. IS-2004 Conf.*, Varna, Bulgaria, June 2004, pp. 282-287.

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

[31] Covington, M. A. Logical Control of an Elevator with Defeasible Logic. *IEEE Transactions on Automatic Control*, 45(7):1347-1349, 2000.

[32] Billington, D. Conflicting Literals and Defeasible Logic", In: Nayak, A., Pagnucco, M. (Eds.) *Proc. 2nd Australian Workshop Commonsense Reasoning*, December 1, pp. 1–15, Australian Artificial Intelligence Institute, Australia, 1997.

[33] Moawad, A., Bikakis, A., Caire, P., Nain, G., & Traon, Y. L. A Rule-Based Contextual Reasoning Platform for Ambient Intelligence Environments. *Proc. 7th Int. Symposium RuleML 2013*, pp. 158-172, Seattle, WA, USA, July 11-13, 2013.

[34] Covington, M. A. Defeasible Logic on an Embedded Microcontroller. *Applied Intelligence*, Vol. 13, pp. 259–264, Kluwer Academic Publishers, The Netherlands, 2000.

[35] Stavropoulos, T. G., Vrakas, D., Vlachava, D., & Bassiliades, N. Bonsai: A Smart Building Ontology for Ambient Intelligence. *Proc. 2nd Int. Conf. on Web Intelligence, Mining and Semantics (WIMS '12)*, Article 30, 12 pages, ACM, New York, NY, USA, 2012.

[36] Traugott, S., & Huddleston, J. (1998, December). Bootstrapping an Infrastructure. In *LISA* (pp. 181-196).

[37] Buchholz, T., Küpper, A., & Schiffers, M. (2003, July). Quality of context: What it is and why we need it. In *Proceedings of the workshop of the HP OpenView University Association* (Vol. 2003).

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

[39] Lam, H. P., & Governatori, G. The Making of SPINdle. *Proc. of 2009 Int. Symposium on Rule Interchange and Applications* (*RuleML '09*), Governatori, G., Hall, J., & Paschke, A. (Eds.), Springer-Verlag, Berlin, Heidelberg, p.p. 315-322, 2009.

[40] 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.

[41] Wooldridge, M. *Introduction to Multiagent Systems*. John Wiley & Sons, Inc., New York, NY, USA, 2001.

[42] Governatori, G., & Terenziani, P. Temporal Extensions to Defeasible Logic. *Proc. 20th Australian joint conference on Advances in artificial intelligence* (AI'07), Mehmet A. Orgun and John Thornton (Eds.). Springer-Verlag, Berlin, Heidelberg, pp. 476-485, 2007.

[43] Governatori, G., & Rotolo, A. Computing Temporal Defeasible Logic. *Proc. 7th Int. Symposium RuleML 2013*, LNCS 8035, pp 114-128, Springer Berlin Heidelberg, Seattle, WA, USA, July 11-13, 2013.

[44] Rubino, R., & Rotolo, A. A Java Implementation of Temporal Defeasible Logic. *Proc. Int. Symposium RuleML 2009*, LNCS 5858, pp 297-304, Springer Berlin Heidelberg, Las Vegas, Nevada, USA, November 5-7, 2009.

[45] Hassanpour, S., O'Connor, M.J., & Das, A.K. Visualizing Logical Dependencies in SWRL Rule Bases. *Proc. Int. Symposium RuleML 2010*, LNCS 6403, pp 259-272, Springer Berlin Heidelberg, Washington, DC, USA, October 21-23, 2010.

[46] Kontopoulos, E., Bassiliades, N., Antoniou, G., & Seridou A. Visual Modeling of Defeasible Logic Rules with DR-VisMo. *International Journal of Artificial Intelligence Tools (IJAIT)*, World Scientific, 17(5), pp. 903-924, 2008.

Appendix A – SPINdle Syntax

The following fragment illustrates the SPINdle-specific syntax for the sample rule base presented in Section 5.2. Notice that conflicting literals are represented by specific rules added to the rule base, accompanied by respective superiority relationships.

```
set @temp_a6 = 30
set @time = 2300
>> motion_a6
>> highCO2_a6

# preference rules for room a6
p01: $@temp_a6 < 18$ -> tempLow_a6
p02: $@temp_a6 > 28$ -> tempHigh_a6
p1: tempHigh_a6 => switchOn_a6_cooler
p2: -tempHigh_a6 => switchOff_a6_cooler

# maintenance rules for room a6
m01: $@consumption_a6 > 2000$ -> savingMode_a6
m02: $@time > 2200$ -> savingMode_a6
m1: -motion_a6 => switchOff_a6_cooler
m2: motion_a6 => -switchOff_a6_cooler
m3: functioning_a6_pc => -switchOff_a6_cooler
m4: savingMode_a6 => switchOff_a6_cooler
m03: isOn_a6_cooler ~> -switchOn_a6_cooler
m04: isOff_a6_cooler ~> -switchOff_a6_cooler

# emergency rules for room a6
e01: smoke_a6 -> alert_a6
e02: highCO2_a6 -> alert_a6
e1: alert_a6 => switchOn_a6_alarm
e2: alert_a6 => switchOff_a6_cooler

# rules for handling conflicting literals
c01: switchOff_a6_cooler => -switchOn_a6_cooler
c02: switchOn_a6_cooler => -switchOff_a6_cooler
m1 > c02
m4 > c02
e2 > c02
c01 > p1
```