A Multi-Agent Coordination Framework for Smart Building Energy Management

Thanos G. Stavropoulos[1,2], Emmanouil S. Rigas[2], Efstratios Kontopoulos[3], Nick Bassiliades[1,2], Ioannis Vlahavas[1,2]
[1] School of Science and Technology, International Hellenic University, Greece
[2] Department of Informatics, Aristotle University of Thessaloniki, 54124, Thessaloniki, Greece
{athstavr, erigas, nbassili, vlahavas}@csd.auth.gr
[3] Information Technologies Institute, Centre of Research & Technology - Hellas, 57001, Thessaloniki, Greece
skontopo@iti.gr

***Abstract*—This paper presents a novel energy management framework for multi-agent coordination in smart buildings. The framework builds on top of an existing Service-Oriented middleware for Ambient Intelligence, which offers sensor and actuator functions of wireless devices. The middleware also provides a semantics infrastructure that assists in authoring agent policies for reducing energy consumption and maximizing user comfort. Each agent within the framework is responsible for monitoring the environmental context and controlling the electrical appliances of a specific room. However, the collective behavior of the multi-agent system is controlled by a Coordinator Agent that approves or rejects the allocation of building resources in time, aiming at more "long-term" goals that are out of the reach and scope of the individual Room Agents. The agents' underlying logic is expressed via defeasible logics, a formalism offering intuitive knowledge representation and advanced conflict resolution mechanisms.**

Keywords—multi-agent systems; ambient intelligence; smart environments; web services; context-aware systems

I. Introduction

With dwindling fossil fuels, and the increasingly negative impact of climate change on society, several countries have instigated plans on a national level to reduce carbon emissions [1]. According to studies, the building sectors in the US represent more than 40% of primary energy consumption [2]. Considering that the majority of this energy is produced from non-renewable sources, it becomes clear that, in order for global CO_2 emissions to be reduced, the energy consumption of buildings has to be reduced as well. In this vein, advanced *Artificial Intelligence* (*AI*) techniques, such as *Ambient Intelligence* (*AmI*), can be deployed, in order to achieve efficient and real time management of energy-consuming activities in buildings.

AmI is one of the prominent computing paradigms amongst *Ubiquitous Computing* (*UbiComp*) and *Pervasive Computing* (*PerComp*). According to these approaches, ambient and non-intrusive devices are scattered around the environment, in order to sense and act appropriately to context. Consequently, such systems are also called *Context-Aware* and are mainly based on *Semantic Web* [3] constructs, e.g. *ontologies*, to model sensed context in a machine-interpretable way. So far, AmI systems have been largely based on Web Service technologies, such as *WSDL*[1], to model and exploit universal, platform-indepentent APIs for the various devices that need to be used in such settings. State-of-the-art AmI paradigms include applications of limited scale and targeting domains such as user comfort, office and home automation or even agriculture and healthcare (*Ambient Assisted Living – AAL*) [4].

This paper introduces a novel agent coordination framework for energy savings in buildings, based on an existing AmI infrastructure. More specifically, the existing deployment at the International Hellenic University[2] (IHU) consists of various wireless sensor and actuator networks, homogenized by a Web Service middleware. The semantic infrastructure includes an ontology and Semantic Web Service annotations, which aid domain experts in dynamically authoring energy-saving policies. The latter are expressed via human-intuitive defeasible logics and are distributed to individual Room Agents residing in each office or room. A single Coordinator Agent manages all underlying agents to achieve large-scale energy saving while ensuring comfort. Defeasible logics support rule superiority relationships, which, in turn, allow us to define three rule layers of ascending importance: preference, maintenance and emergency. Each Room Agent is able to submit its preference rules to the Coordinator Agent, which fuses them with maintenance and emergency policies for optimal long-term building management.

II. Smart Environment Integration

This section presents the middleware layer of the proposed architecture, along with descriptions for hardware integration. First, the nature of AmI applications is entailing a dynamic, real-time environment and heterogeneous devices. The *Service-Oriented Architecture* (*SOA*) has been proved to be extremely suitable for such environments and used extensively to provide the necessary abstractions for application development [5]. SOA provides a higher level of abstraction, agnostic of platform and communication protocol device programming. Additionally, it enhances extensibility, as new devices can be integrated via the authoring of new services under the same framework. All services comply with a Web-wide universally

The Smart IHU project was funded by the Operational Program Education and Lifelong Learning, OPS 200056 (International Hellenic University, Thessaloniki, Greece)

[1] Web Service Description Language: www.w3.org/TR/wsdl
[2] http://www.ihu.edu.gr/

established API, the W3C *WSDL* language, which resolves syntactic heterogeneity by type-defining service operations.

This work builds upon the *aWESoME* Web Service middleware for AmI introduced in [6]. The middleware itself entails three distinct layers, one for hardware integration, one that implements and provides (syntactically expressed) services and a semantic description layer on top. Our motivation for building a middleware from scratch is twofold: (a) to define semantics more efficiently, and, (b) to integrate hardware modules that existing middleware paradigms do not. However, whenever possible, open-source software has been properly deployed to interface with some parts of the hardware.

A. Hardware Integration Layer

This layer consists of the so-called *driver modules*, i.e. essentially plug-ins developed specifically for each device family of the proposed system that are briefly presented below.

Smart Plugs: Smart Plugs[3] are both sensors and actuators, which monitor an electrical appliance's status, power consumption and enable switching it on or off. They form wireless encrypted ZigBee networks managed by a single controller and are widely popular in both retail and research [6], [7]. The Smart Plug Driver module, implemented in Java, is invoked every time a 'get' or 'set' operation needs to be performed over the Smart Plug network and supports bidirectional communication.

Sensor Boards: Wireless networks of Sensor Boards were deployed to measure ambient environmental properties. Each chosen board[4] entails a multi-sensing platform for temperature, humidity and luminance and forms a mesh open ZigBee network. Unlike Smart Plug communication, Sensor Board communication is one-directional, as boards push notifications to the server Sensor Board Driver module.

Smart Clampers: The wireless Smart Clamper bundle[5] measures the total building and Data Center consumption. The Smart Clamper Driver is a Java module that collects clamper data to be retrieved by the Smart Clamper Service.

Z-Wave: This protocol unifies sensors and actuators from different manufacturers at communication layer level. The selected devices in our setting detect motion in each room and CO_2 concentration level at four main indoor locations. Being the largest and most diverse family of devices, the Z-Wave Driver module incorporates the open source Open Z-Wave library[6] to collect data from virtually any compatible device.

B. Web Service Layer

Building upon integration modules, the middleware offers a designated Web Service for each device family. In turn, each service offers multiple operations to support underlying functionality. The services follow the WSDL/SOAP description and communication standards and have been implemented using Java, JAX-WS[7]. In the context of this paper, the most significant operations are:

- *Smart Plug Service*: GetStatus, GetPower, SwitchOn, SwitchOff.
- *Sensor Board Service:* GetTemperature, GetHumidity, GetLuminance.
- *Smart Clamper Service:* GetPower.
- *Z-Wave Service:* GetMotion, GetCO_2.

C. Semantic Layer

The semantic layer of the architecture consists of two components: (a) the ontological infrastructure, and, (b) the semantically annotated service endpoints. The ontological infrastructure entails the *BOnSAI* (*Smart Building Ontology for Ambient Intelligence*) ontology [8], suitable for describing concepts relevant to Smart Buildings, Ambient Intelligence, Services and Sensor Networks. It also extends existing ontologies, such as the *OWL-S*[8] upper ontology for services, for interoperability outside the borders of our system. Meanwhile, the aWESoME web services are annotated following the SAWSDL lightweight standard, enhancing service discovery. In the context of this work some annotations are used during the dynamic authoring of rules by experts. Namely, *SensorOperation* or *ActuatorOperation* annotations on WSDL operations can be used in rule conditions and rule effects, respectively.

D. Application Layer

The application layer of the architecture makes use of both web service abstractions and semantic descriptions in order to provide high-level functionality to users or software agents. Human users can monitor and manage the infrastructure through a series of applications such as:

- A desktop administrative application to monitor all sensors and manually manage electrical appliances in real-time;
- A web application of aggregated consumption history, statistics, waste metrics and environmental data for visitors, students and staff to build energy-awareness;
- Intelligent agents who automatically manage the infrastructure, such as the multi-agent framework discussed in the next section.

III. Multi-Agent Coordination Framework

The proposed multi-agent framework integrates two types of agents: (a) *Room Agents* (*RAs*), each of which are assigned the energy management of a distinct room within the building,

[3] http://www.plugwise.com
[4] http://www.prismaelectronics.eu
[5] http://www.currentcost.com/
[6] https://code.google.com/p/open-zwave/
[7] Java API for XML Web Services: https://jax-ws.java.net/
[8] OWL-S: http://www.w3.org/Submission/OWL-S/

and, (b) a *Coordinator Agent* (*CA*) that is responsible for accepting or rejecting the allocation of building resources proposed by RAs. In practice, each RA ignores the existence of other RAs and is only aware of its own set of rules (forming a policy), which it submits to the CA for approval or rejection. The proposed application architecture is shown in Fig 1.

The agents' underlying logic is expressed via *defeasible logics*, a non-monotonic logics formalism that delivers intuitive knowledge representation and advanced conflict resolution mechanisms [9]. In defeasible logics there are three distinct types of rules:

- *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 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.

Additionally, the *superiority relationship* is used for resolving conflicts among defeasible rules. 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 > is acyclic.

The advantages of applying defeasible instead of classical logics are outlined as follows:

- Defeasible logics allow for reasoning with incomplete information; this is a critical trait in AmI settings, where perfect knowledge of the environment is very hard, if not impossible, to achieve;
- They introduce non-monotonicity, which leads to a more intuitive type of reasoning, much closer to human reasoning, where the emergence of new information can lead to abandoning (i.e. defeating) previously established conclusions and adopting new ones.

As mentioned previously, RAs have their own user-defined policies, each of which consisting of a set of rules expressing user preferences. These policies aim at maximizing user

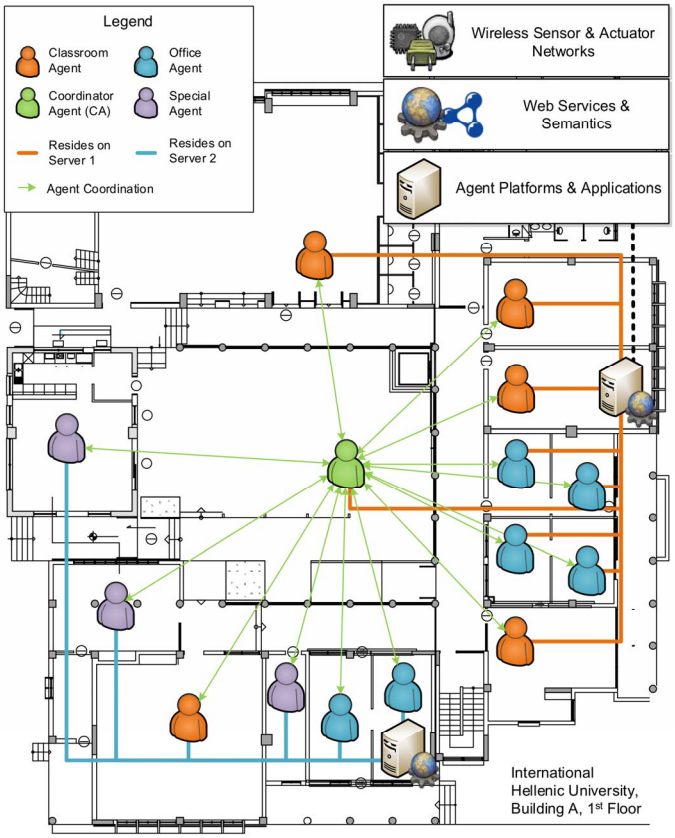

Fig 1. Framework architecture and Agents distribution at the IHU premises

comfort (e.g. the desired room temperature during the winter months) and optimizing energy consumption. Also, through a respective policy, the CA handles the power management scheme of the building (e.g. deactivation of certain appliances during late night hours), as well as potential emergency situations (e.g. activation of emergency lights in case of fire). Thus, within the framework, there are three distinct layers of rules with escalating priority: (a) *preference rules*, handled by RAs, (b) *maintenance rules*, and, (c) *emergency rules* – the two latter categories are handled by the CA. The notion of priority here refers to priority among defeasible rules, as discussed previously. The following sample use case illustrates the functionality of the proposed framework in more detail.

A. Sample Use Case – Rule Agents

Suppose that the following example involves two rooms, 'A1' and 'A2', within the smart building and that two RAs, RA_{A1} and RA_{A2}, are responsible for each room, respectively. Suppose, also, that the employees working in these rooms have defined the following preference rule sets, which have been assigned to the RAs[9]:

 # room 'A1' preference rules (*agent* RA_{A1})

[9] For authoring defeasible logic rule bases, one could download one of the available rule editors, like, e.g. *SPINdle's defeasible logic theory editor* (http://spin.nicta.org.au/spindle/tools.html), or the much more flexible *S*^{2}*DRREd* (*Syntactic-Semantic Defeasible Reasoning Rule Editor*) [10].

$p_{A1\text{-}01}$: time(X), X > 1600 → afternoon *#strict rule*
$p_{A1\text{-}02}$: time(X), X < 1600 → ¬afternoon *#strict rule*
$p_{A1\text{-}11}$: afternoon ⇒ switchOn(light) *#defeasible rule*
$p_{A1\text{-}12}$: ¬ afternoon ⇒ switchOff(light) *#defeasible rule*

 # room 'A2' preference rules (*agent* RA_{A2})
$p_{A2\text{-}01}$: time(X), X < 1700 → dayTime *#strict rule*
$p_{A2\text{-}02}$: time(X), X > 1700 → ¬dayTime *#strict rule*
$p_{A2\text{-}11}$: dayTime ⇒ switchOff(light) *#defeasible rule*
$p_{A2\text{-}12}$: ¬dayTime ⇒ switchOn(light) *#defeasible rule*

As already mentioned, preference rules primarily handle user comfort. According to the above preferences, the employee-user determines the time thresholds that designate – by personal standards – when the lights in each room should be turned on or off.

B. Sample Use Case – Coordinator Agent

The preference rule sets above are of no utility by themselves, since they have to be approved/rejected by the CA. The latter is already equipped with its own policy, containing maintenance and emergency rules:

 # maintenance rules
m_0: consumption(X, Y), Y > 2000 → savingMode(X)
m_1: savingMode(X) ⇒ switchOff(X, light)
m_2: isOn(X, light) ⇝ ¬switchOn(X, light) *#defeater*
m_3: isOff(X, light) ⇝¬switchOff(X, light) *#defeater*

 # emergency rules
e_0: smoke(X) → alert(X)
e_1: alert(X) ⇒ switchOn(X, light)

As can be observed, maintenance rules tend to enforce energy savings, often compromising the comfort requested by the user-defined preference rules. The specific rule set determines the conditions for having each room switch to 'saving mode', during which the system will attempt to minimize consumption. Also, the defeaters make sure that the light's actuator will not attempt to turn on an already operating light source and, conversely, will not attempt to turn off a light source that has already been switched off. Additionally, the emergency rules handle situations that imply unsafe conditions inside a room, like e.g. smoke. In this case, the system activates the room's alert, which then handles the operation of the appliances in the room accordingly (e.g. lights are turned on to help users find their way out of the room).

Whenever a room policy is updated, the respective RA immediately submits the revised rule set to the CA. In the running example, the following rules will be appended to the CA's policy (notice that RA rules undergo certain syntactic modifications when merged into the CA rule base):

$p_{A1\text{-}01}$: time(X), X > 1600 → afternoon
$p_{A1\text{-}02}$: time(X), X < 1600 → ¬afternoon
$p_{A1\text{-}11}$: afternoon ⇒ switchOn(A1, light)
$p_{A1\text{-}12}$: ¬ afternoon ⇒ switchOff(A1, light)
$p_{A2\text{-}01}$: time(X), X < 1700 → dayTime
$p_{A2\text{-}02}$: time(X), X > 1700 → ¬dayTime
$p_{A2\text{-}11}$: dayTime ⇒ switchOff(A2, light)
$p_{A2\text{-}12}$: ¬dayTime ⇒ switchOn(A2, light)

Conflicts are generated between pairs of rules: (a) with opposite conclusions, and, (b) where one rule concludes that a device should be turned on, while the other rule concludes that it should be turned off and vice-versa. The latter case is expressed via *conflicting literals* [11], which, however, are not further discussed in this work for brevity reasons. Conflicts can 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 preference rules), the superiority relationship has to be explicitly added to the knowledge base by the user. 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:

$m_1 > p_{A1\text{-}11}$
$m_1 > p_{A2\text{-}12}$
$e_1 > m_1$

After every update in its knowledge base, the CA executes the rule set through *SPINdle* [12], a state-of-the-art defeasible logic reasoning engine. The following two brief examples give a deeper insight into the reasoning process.

Example 1

Suppose that the following facts are inserted into the above rule base:

f_1: time(2300)
f_2: consumption(A1,2300)

Fact f_1 triggers rules $p_{A1\text{-}01}$ and $p_{A2\text{-}02}$ which, in turn, trigger rules $p_{A1\text{-}11}$ and $p_{A2\text{-}12}$ that conclude switchOn(A1, light) and switchOn(A2, light). On the other hand, fact f_2 triggers rule m_0 that, in turn, triggers rule m_1, which concludes switchOff(A1, light). Rules m_1 and $p_{A1\text{-}11}$ are conflicting and this conflict is resolved via the superiority relationship $m_1 > p_{A1\text{-}11}$, which eventually concludes that the light in room A1 should be turned off. The light in room A2, however, remains turned on.

Example 2

Suppose now that fact f_3: smoke(A1) is also added into the knowledge base. As a consequence, rule e_0 will be triggered, which, in turn, triggers rule e_1 that concludes switchOn(A1, light). A new conflict arises, this time between e_1 and m_1, but, since a respective superiority relationship has already been established ($e_1 > m_1$), the conflict is resolved, resulting in the lights being eventually turned on.

IV. STATE OF THE ART

As far as rule-based smart environments are concerned, one approach introduced in [13] is a meta-language defined over

JESS[10], to syntactically enhance the rule authoring process in ambient applications. However, this additional syntactic layer, called *Event-Control-Action* (*ECA*) model, is far less flexible and extensible than defeasible logic used in the proposed framework. Other similar approaches include *SESAME-S* [7], an all-in-a-box smart home prototype that uses ontologies and JESS reasoning to enforce rules. The main drawback is the lack of conflict resolution and scalability, while our work targets those aspects with defeasible logic. Regarding the proposed middleware approach, the SAWSDL bottom-up, lightweight standard is used, while most existing paradigms use more complex and, thus, less interoperable top-down standards, as in [10] [14], or custom ones, as in [15] [16].

Multi-agent systems have also been used for managing energy consumption of buildings. The approaches presented in [17] and [18] represent various entities in buildings as agents and target the reduction of the building's consumption, while comfort factors of the occupants are also taken into consideration. More specifically, [17] employs multi-objective *Markov Decision Problems*, and [18] uses *Particle Swarm Optimization* in order to optimally schedule energy consuming activities. On the contrary, our work provides agent coordination and defeasible logics for conflict resolution, while scheduling is encoded in policies authored by experts.

V. CONCLUSIONS AND FUTURE WORK

The proposed multi-agent coordination framework is able to optimize a building's appliances to achieve long-term energy savings while providing comfort. It does so, based on preference, maintenance and emergency policies authored by experts using human-intuitive defeasible logics and semantics. The framework is based on an existing Semantic Web Service and wireless sensor and actuator network deployment at a university building. Admittedly, the presented implementation is at a preliminary stage, but additional extensions and experimentation are well underway.

A crucial future task to accomplish is to implement the proposed framework on the target deployment, using existing components. Namely, communication and information exchange between existing agents needs to be established. Consequently, energy-saving results from such an application need to be measured, using the existing monitoring infrastructure, especially, during late spring and summer months when a power-demanding cooling system is activated.

Another interesting future direction would be to build our proposed model based on an advanced multi-agent framework like *EMERALD* [19]. The latter allows defining interoperating, knowledge-based intelligent agents, each of which is equipped with a reasoning engine forming a shared reasoning infrastructure. Although EMERALD currently features a few but widely diverse reasoning services, those services can easily be extended. A further advantage of the framework is that agents are knowledge-customizable, meaning that they are not confined in having their logics and strategies/policies hard-wired. Instead, they can be customized by altering the rule base, while the agent's knowledge and/or behavior will instantly be modified accordingly. This functionality fits our requirements and will potentially improve the flexibility of our proposed framework.

[10] http://herzberg.ca.sandia.gov/

ACKNOWLEDGMENT

The authors would like to thank Konstantinos Gottis, John Argyriou, George Pilikidis, Thodoris Tsompanidis and Andrea Dimitri for their work in various parts of system integration.

REFERENCES

[1] E. I. A. (US), *Annual Energy Review 2011*. Government Printing Office, 2012.

[2] U. S. DoE, "Buildings energy data book," *Energy Effic. Renew. Energy Dep.*, 2011.

[3] T. Berners-Lee, J. Hendler, and O. Lassila, "The semantic web," *Sci. Am.*, vol. 284, no. 5, pp. 28–37, 2001.

[4] D. J. Cook, J. C. Augusto, and V. R. Jakkula, "Ambient intelligence: Technologies, applications, and opportunities," *Pervasive Mob. Comput.*, vol. 5, no. 4, pp. 277–298, 2009.

[5] V. Issarny, M. Caporuscio, and N. Georgantas, "A perspective on the future of middleware-based software engineering," in *2007 Future of Software Engineering*, 2007, pp. 244–258.

[6] T. G. Stavropoulos, K. Gottis, D. Vrakas, and I. Vlahavas, "aWESoME: A web service middleware for ambient intelligence," *Expert Syst. Appl.*, vol. 40, no. 11, pp. 4380–4392, 2013.

[7] L. Daniele, P. D. Costa, and L. F. Pires, "Towards a rule-based approach for context-aware applications," in *Dependable and Adaptable Networks and Services*, Springer, 2007, pp. 33–43.

[8] T. G. Stavropoulos, D. Vrakas, D. Vlachava, and N. Bassiliades, "BOnSAI: a smart building ontology for ambient intelligence," in *Proceedings of the 2nd International Conference on Web Intelligence, Mining and Semantics*, 2012, p. 30.

[9] D. Nute, "Defeasible logic," *Handb. Log. Artif. Intell. Log. Program.*, vol. 3, pp. 353–395, 1994.

[10] E. Kontopoulos, T. Zetta, and N. Bassiliades, "Semantically-enhanced authoring of defeasible logic rule bases in the semantic web," in *Proceedings of the 2nd International Conference on Web Intelligence, Mining and Semantics*, 2012, p. 56.

[11] D. Billington, "Conflicting literals and defeasible logic," in *Proc. 2nd Australian Workshop on Commonsense Reasoning*, 1997, pp. 1–14.

[12] H.-P. Lam and G. Governatori, "The making of SPINdle," in *Rule Interchange and Applications*, Springer, 2009, pp. 315–322.

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

[14] G. Thomson, S. Bianco, S. B. Mokhtar, N. Georgantas, and V. Issarny, "Amigo aware services," in *Constructing Ambient Intelligence*, Springer, 2008, pp. 385–390.

[15] A. Fensel, S. Tomic, V. Kumar, M. Stefanovic, S. V. Aleshin, and D. O. Novikov, "Sesame-s: Semantic smart home system for energy efficiency," *Inform.-Spektrum*, vol. 36, no. 1, pp. 46–57, 2013.

[16] S. M. Iacob, J. P. A. Almeida, and M. E. Iacob, "Optimized dynamic semantic composition of services," in *Proceedings of the 2008 ACM symposium on Applied computing*, 2008, pp. 2286–2292.

[17] L. Klein, J. Kwak, G. Kavulya, F. Jazizadeh, B. Becerik-Gerber, P. Varakantham, and M. Tambe, "Coordinating occupant behavior for building energy and comfort management using multi-agent systems," *Autom. Constr.*, vol. 22, pp. 525–536, 2012.

[18] Z. Wang, R. Yang, and L. Wang, "Multi-agent control system with intelligent optimization for smart and energy-efficient buildings," in *IECON 2010-36th Annual Conference on IEEE Industrial Electronics Society*, 2010, pp. 1144–1149.

[19] K. Kravari, E. Kontopoulos, and N. Bassiliades, "EMERALD: a multi-agent system for knowledge-based reasoning interoperability in the semantic web," in *Artificial Intelligence: Theories, Models and Applications*, Springer, 2010, pp. 173–182.