PLECO: New Energy-Aware Programming Languages and Eco-Systems for the Internet of Things

Jon Robinson Kevin Lee
Department of Computing & Technology
Nottingham Trent University
Nottingham, UK
e-mail: jon.robinson@ntu.ac.uk
e-mail: kevin.lee@ntu.ac.uk

Kofi Appiah
Department of Computing
Sheffield Hallam University
Sheffield, UK
e-mail: k.e.appiah@shu.ac.uk

Abstract— **This paper outlines the aims of the Programming Language ECO-system (PLECO) to create new energy-aware programming languages and eco-systems for the Internet of Things (IoT). It builds upon the Lantern language and focuses on energy-awareness, security, resilience and communications for the large infrastructure underpinning the next generation of IoT. The paper outlines how IoT applications and deployments need to be developed in an energy-aware, secure and cost-effective manner using new secure, robust and energy-focused programming languages and the importance of taking such an approach.**

Keywords-energy-aware; Internet of Things; programming; distributed computing; security; Cyber-physical systems.

I. INTRODUCTION

It is projected that more than 50 billion Internet enabled devices will be online within the next 10 years [15]. This poses a problem for current ways of developing Internet of Things (IoT) and Cyber-Physical Systems (CPS) software as current practices do not consider the energy expenditure that these devices will introduce on existing power distribution networks. At present, developing applications for the IoT/CPS exposes devices to a number issues relating to energy use, security and reliability. IoT applications are currently developed using existing languages, frameworks and toolkits [16], which, in the case of programming languages, have not altered since their initial creation. Dynamic scripting languages like Python and JavaScript are being embraced by most IoT/CPS designers at the expense of high runtime cost due to the dynamic types and code optimisation techniques (e.g., Just-In-Time compilation) [19]. The applications they produce will tend to be less efficient and insecure [17] as the underlying development approach and programming language was initially designed without considering the core concepts of resiliency, energy-awareness and security. This leads to these concepts being added as an afterthought rather than as the primary focus of well-engineered software systems.

To engineer these applications appropriately requires the concepts of resiliency, energy-awareness and security to be central in the design and implementation of a system. To enable this, it is proposed that a new development approach, built on a language focused around these concepts be the way in which systems are written. This would ensure that software is secure, reliant (i.e., dealing with communication, distributed complexity and failover) and importantly, energy-aware from their inception by allowing developers to implement them using algorithms which promote these areas.

In Section II, the need for energy-awareness is discussed and current approaches to Internet of Things software development is introduced. In Section III, the proposed PLECO architecture is introduced and discussed. In Section IV, the experiences learnt from the initial Lantern energy-aware domain specific language is discussed. Finally, in Section V, a summary of the work is provided.

II. CURRENT PRACTICES

Energy-efficiency is a growing research focus in all areas of technology, including IoT. Energy-utilisation of hardware had been addressed widely in the embedded systems area where software can reduce the power usage of components of the underlying hardware. However, energy-awareness is still poorly represented when it comes to building large scale distributed systems and the algorithms used to implement them [18]. Additionally, languages suffer from providing developers with practices and language constructs which have been available in general purpose languages for many years. However, when these languages were initially designed, the computing and distributed landscape was significantly different than from what it is today. Concepts representing how to use energy-aware algorithms for efficient interacting distributed systems, coordinate, adapt, self-heal, secure and be resilient have not been fully considered in their design phases.

Energy-awareness has been a major focus within the embedded systems world where the conservation of energy is instrumental in the operation of a device. Incorporating energy efficiency within the design of the circuitry underlying the device has proven to be effective [1] and acts as another justification on the PLECO approach. Other static approaches to embedded design have been proposed [2][3].

Energy-awareness in software development has highlighted many challenges in the production of energy-aware software systems. For example, application-level approaches advocate applications being energy-aware and controlling their own energy use [4]. To support application development, tools

that monitor applications to provide energy use information [5][6] exist to aid in the process. In addition, by incorporating middleware to help with efficient energy usage in applications without them explicitly being aware of this focus [7][8] could be utilised. The need for a dedicated programming language with resource constraints to streamline usage was identified in [9] and has been a research trend mainly for security [10]. There are other approaches to IoT development which fall into cloud, Operating Systems, middleware or protocols (e.g., IF This Then That (IFTTT), Azure IoT or AWS IoT, Kontiki, Brillo), MQTT, Gaia, etc.). In [11], updates to the International Technology Roadmap for Semiconductors (ITRS) [12] are discussed. The ITRS provides a roadmap of hardware and software technologies in the design and development of silicon systems. The road map outlines the trends of future technologies to address challenges regarding the cost of design and power / energy use. Future trends within the ITRS show that power-aware systems are currently a challenge in the control of electronic devices. Thus, with the popularity of the uptake of IoT devices, programming them in an energy-aware manner is an important problem to address.

III. THE PLECO ARCHITECTURE

Current software development paradigms are not ideally suited to tackle the energy efficient and distributed nature of IoT and similar technologies. This is exacerbated by the lack of energy, security and reliability standards and frameworks for this domain. This opens up the need for alternative methods for developing, deploying and supporting software for IoT deployments and other related applications.

To solve this problem, a shift in software development which enables the efficient design, development, support systems and eco-system for these emerging distributed systems technologies. By focusing on the principles of low energy, security and reliability, this will best serve the needs of large-scale heterogeneous systems. We propose the development of a complete eco-system for modern software development, including development languages and support systems which enable the efficient design and use of system wide energy, and production of software systems which address the key challenges of modern Internet based systems.

The PLECO architecture aims to investigate new paradigms and languages for software development in large scale distributed systems by introducing a new energy-aware, reliable and security focused eco-system. This aligns with the ITRS future goal of energy-aware programming for IoT.

We advocate the there are three fundamental pillars underlying the design and development of a new eco-system for developing energy-aware systems. For each of these pillars, by directly integrating them into the development process, developers would plan, design, implement, test and deploy applications which satisfy these programming styles from the onset: Energy-Awareness, Security and Reliability.

The PLECO system builds on preliminary work into energy-aware Domain Specific programming languages and middlewares [13][14]. With the growing acceptance and use of IoT enabled devices and the lack of security with these devices, a new way in which to design, construct and implement solutions needs to be considered. At present, there are very few frameworks which directly address the robust production of IoT systems. However, security is only a secondary concern which leaves devices open to exploitation. Thusly, the adoption of a new way in which to design and build large scalable, secure, and robust distributed systems requires a new platform and language is required.

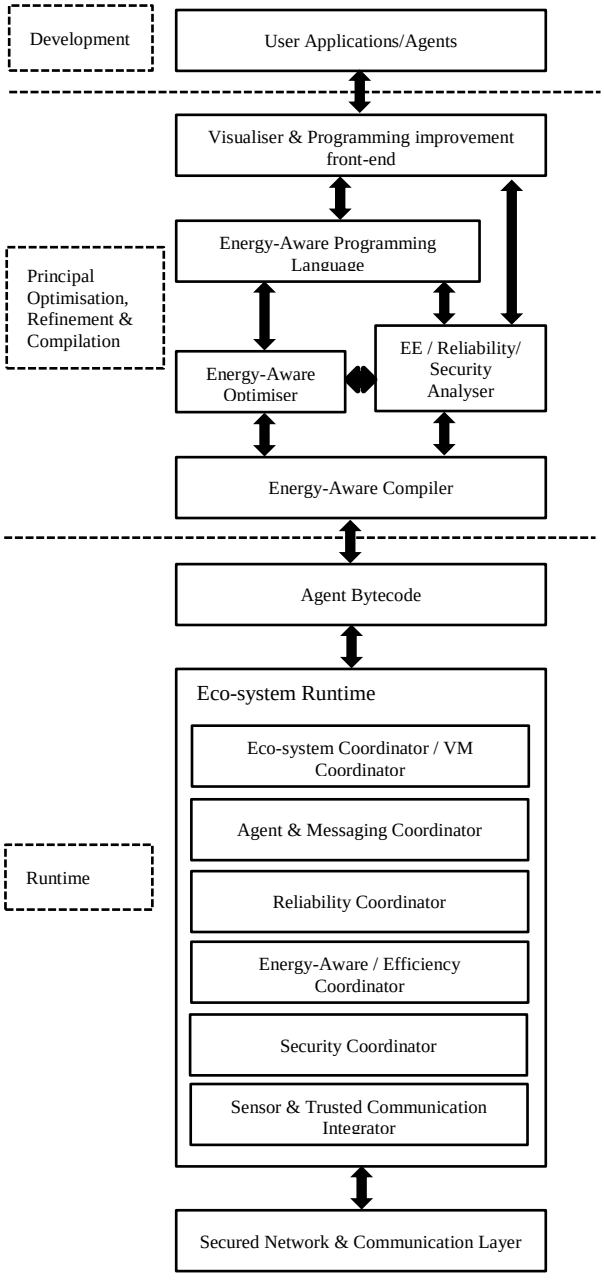


Figure 1. PLECO Eco-system architecture

The PLECO eco-system is presented in Figure 1. It outlines the proposed main components of the eco-system (middleware, compilers, language, optimisers and potential standards). This builds on previous work into service composition middlewares [13] and energy-aware domain specific languages (Lantern) [14].

The main contribution of this work can be summarised as: 1) fully complete energy-aware programming language for controlling and managing IoT devices; 2) inclusion of security into language design for secure software development as well as failover support; 3) distributed concepts for the management and communications in highly scalable IoT architectures; and, 4) infrastructures for supporting new methods. The approach taken is to consider the development and runtime in three distinct phases. Namely, *development*, *compilation and optimisations* and *runtime*. What follows provides the architectural breakdown of our proposed eco-system as well as the main components and what their expectations will be.

Layer 1: Development language

One of the contributions of this work is to provide a new programming language which enables users to develop energy-aware, secure and robust software systems which builds upon prior work. At present, the focus on developing systems is to use existing languages which forces developers down specific design routes which requires them to consider energy usage and security as secondary considerations. The Lantern [14] Domain Specific Language (DSL) has been previously developed, to provide developers with a language in which service agents can be constructed and was designed from the ground up to embody the notion of energy-awareness as a key concept (see Figure 2 for an example of Lantern code). The purpose of the Lantern system was to investigate how to provide an energy-aware domain specific language which was aimed at managing and controlling the energy consumption of disparate IoT devices. This was the first test iteration of a language to test ideas and confirm the viability of energy-aware languages. These agents interface with hardware-based devices as well as providing energy-aware adaptive abilities to monitor and adapt the power usage of devices within a home environment populated with IoT devices.

However, as it stands, Lantern acted as the first stage of investigating adaptive energy-aware DSL's and provides a language which allows the control of devices rather than a fully functional and semantically rich language for general purpose development. Its purpose was to adapt to the changing energy needs of a static location and allow devices within the environment to alter their energy use, thus saving energy. Other lessons that were learnt from this initial phase will be introduced into the second generation of the language. For example, a simplified notion of energy was represented within the language where the amount of energy used (Watts) was represented as values associated to power control structures. However, even though these were not strongly typed power values, the intention was that these would infer the amount of energy used. No other form of energy representation was included but the next step is to represent energy in a variety of more strongly typed language constructs which could represent Joules, or other energy-based representations (e.g., temperature).

Hence, the purpose of the next generation of the language is to provide developers with a new way of developing software programs for Internet of Things devices. It will be a semantically rich language, rather than a DSL which will provide them with the ability to provide more sophisticated software. By providing a new language, rather than using an API, requires them to consider the energy-awareness of the design and operation of the software, security and robustness which can have a direct impact of the energy use of the smart environment they are located within. This ensures that the considerations and requirements of writing software for today's highly distributed systems are considered from the outset of development rather than as a secondary consideration. For example, C and C++ languages have been in use for decades before large scale distributed networks of cooperating IoT devices where considered. Because of this, the underlying languages do not provide the concepts of security, energy-awareness and robust as central tenets of the language and, hence, at best are considered after the design and during the implementation of systems, and often not at all.

```
aliases {
  alias( heating_control ) -> device( heating, ERD204)
  alias( heating_temp ) <- device( heating, ERD204)
  alias( motion_control ) -> device( motion, ERD204)
  alias( lights_control ) -> device( lights_1, ERD204)
  alias( lights_power ) <- device( lights_1, ERD204)
  alias( PC_control ) -> device( pc1, ERD204)
  alias( PC_power ) <- device( pc1, ERD204)
}
environment(ERD):{
  location(ERD204) : {
    uses device( heating ) <- input(heating_temp)
    uses device( heating ) -> output(heating_control)
    uses device( motion ) <- input(motion_control)
    uses device( lights ) <- input(lights_power)
    uses device( lights ) -> output(lights_control)
    uses device( PC ) <- input(PC_power)
    uses device( PC ) -> output(PC_control)
  }
}
consumption(ERD):{
  override( ERD204 > 800 ) -> {
    condition( heating > high ) -> action( heating = off )
    condition( lights == on ) and condition( !movement ) -> action( lights =
off )
    condition( PC == on ) and condition( !movement ) -> action( PC = off )
  }
}
(identity:cmp3robinj):(location:ERD204) {
  condition( at(7:30) ) -> action( heating = on )
  condition( temperature < low ) -> action( heating = on )
  condition( temperature > high ) -> action( heating = off )
  condition( lights == off ) and condition ( !movement ) -> action( lights =
on)
  condition( at(20:00) ) and condition( !movement ) -> action( lights = off )
  condition( at(20:05) ) and condition( PC == on ) and condition(
!movement ) -> action( PC = off )
}
```

Figure 2. Example Lantern code

This will introduce new ways to represent the energy-awareness of systems through language constructs which enable devices and software systems to be actively aware and adapt their power utilisation. For instance, rather than focus on energy consumption within the hardware level, energy-aware constructs will allow software to be built which is both efficient and energy-aware by using algorithms and program design which facilitates in reducing the overall energy expenditure of the interacting system. The notion of security

is currently poorly represented in software design, so another key area of the language is to incorporate secure development and language constructs from the outset. This will enable developers to consider security related considerations in the design and implementation of systems by using algorithms and language constructs, which promote secure systems. The final component of the language is to incorporate constructs which allow resiliency (and robustness) within interactive systems. The complexities of distributed systems also will be addressed by providing distributed management constructs. This is to ensure that systems can adapt and reconfigure themselves if components of the larger system fail or are unavailable.

Layer 2: Optimising technologies

The purpose of this layer of the eco-system is to provide programming support to developers and users. A variety of sub-systems will be provided, which allow for the analysis and improvement of software by enabling support for code optimisation. A number of key components are required:

Energy-aware programming language: As has been previously discussed, the language will offer the concepts of energy-awareness (by allowing the monitoring and adaptation of energy use within an environment), resiliency/robustness through failover, distributed complexity, communications and management/control; and security through secure communication and language constructs. It will provide support on how to write adaptive systems which can react according to environmental stimuli to make best use of the resources on offer depending on the energy requirements of devices. A focus on the representing the interactions between devices and associating an energy cost to these interactions will enable algorithms to start to consider the economics between device interactions. The language will be extensible and introduce notions of *low carbon foot-printing, identity, distributed systems, agents, data generation, composition, mobility, reconfiguration, security, privacy, trust* built in which provides users with the means for writing effective systems.

Energy-aware optimiser: Optimisation of energy-aware systems requires the analysis of both the programming style and algorithms used within the software construction stage and the way in which agents will interact with each other to make best use of the resources that are on offer. This will primarily focus on inspecting the code written by the end-user to analyse whether there are more efficient ways of representing the code which can be made. It will not analyse how to make efficient use of the underlying hardware (e.g., turning off Wi-Fi, controlling processor state, etc) but instead will examine the algorithms that have been used to see if it is possible to increase their energy efficiency by modifying how they work and how they interact within a larger system.

Energy-Aware/Reliability/Security Analyser: This component provides analysis of the agent based on the notions of reliability and security. It will determine whether the best practices have been followed to ensure that the agent is secure. It will also analyse the agent to determine if it is reliable and robust (i.e. distributed complexity as well as securely constructed, including secure communication). Debugging information will be generated which allows higher-levels to visualise data based on how to improve the security and debugging of agents within the system.

Energy-Aware compiler: The energy aware compiler will produce byte-code which makes best use of the three main concepts behind the language to generate agents. Generated byte-code will be executed within a safe, secure and reliable environment provided by the eco-system runtime in line with the existing Lantern system. This is currently being developed and offers agents a distributed playpen in which to execute. The agents that are produced will automatically bind to the runtime which offers a controlled exposure to the underlying runtime properties. Performance and profiling information which includes programming specifications outlining the type of data produced, consumed and linked to, to help with the generation of mobile and location aware agents will be considered. The compiler would optimise applications based on the corpus of data generated by the user and how the user intends the agent to interact within the eco-system and how it consumes data and its reliance on other agents. The compiler will determine the appropriate hardware requirements and locality (i.e., closeness to work) where agents running within the runtime close to where tasks needs to be completed.

Visualisation & Programming style improvement system: Another key way in which languages need to be supported is through verification, validation and visualisation of software systems. This will provide a graphical front-end for improving agent design and development. A hints/help system will provide the user with ways in which to increase system efficiency by suggesting improvements to security and reliability considerations.

Layer 3: Eco-system runtime

The purpose of the runtime system is to provide a consistent environment for executing agents. It is comprised of many sub-components that provide control for: coordination; discovery; invocation; virtualisation; agent mobility; reliability (i.e., failover, tolerance); energy-awareness and adaptation; distributed complexity; and, trusted communication & security. This builds on the existing Lantern middleware as well as other areas that are currently being investigated. The ecosystem runtime layer will be formed out of the following coordinator sub-components:

Ecosystem coordinator / VM coordinator: This will provide executing agents a safe, protected, virtual environment to run within. Mobility of agents will be managed within this level so that they can make use of the resources within the environment. This was an issue with Lantern as the language and agent were based on statically located devices. For this iteration, agents will be mobile and be able to transport themselves around within the eco-system. This will mean that location dependency will ensure that agents are running and interfacing with the best set of devices depending on the whereabouts of users and deployed system. Exposure to adaptive and reconfigurable aspects of

the runtime will be provided to agents so that they are able to locate and adapt themselves to their surroundings. Agents will be exposed to the discovery and linkage to other agents provided by the agent and messaging coordinator. Therefore, this will provide agents with a distributed, reconfigurable, compositional and collaborative runtime for the coordination and secure control of systems. The runtime will also provide identity management and identity conflict resolution. This builds on the initial Lantern representation of identity which was weakly defined. For this iteration, to help with managing a number of identities, a group based approach will be taken which allows identities to inherit permissions and access control for different environments.

Agent & Messaging coordinator: This will maintain agents by providing them with resources within the runtime and provide them with mobility facilities. Control of processing resources, complexity, memory, storage, and message handling will be offered. The adaption and reconfigurable nature of agents will be handled by this coordinator so that they can adapt to conditions over time.

Reliability coordinator: This will ensure that agents can deal with situations where something goes wrong. This will be through a combination of approaches ranging from agent reconfiguration; failover control; re-incarnation; complexity; agent adaptation (where agents can self-heal); and, debugging information and mechanisms for diagnosing interaction and programming issues.

Energy-aware/efficiency coordinator: This will coordinate the most efficient use of devices and interactions / collaborations with other agents. Its primary purpose is to provide exposure to the energy consumption aspects of interfacing technology (i.e., actuators connected to devices). It will also control the re-configurability and binding of agents to devices based on power needs.

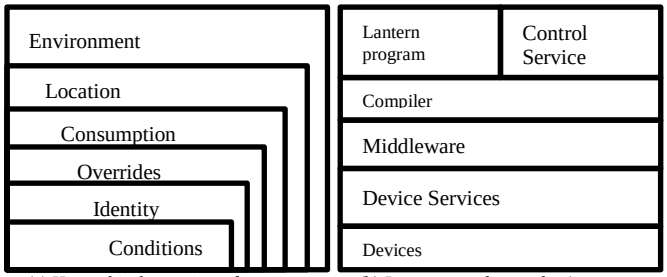
Security coordinator: This will provide the underlying security model for managing and coordinating agents and devices. Secure and trusted communication between devices will be provided for. It will offer protection from tampering of agents and devices from malicious entities (e.g., other users and systems) and provide cyber-security attack resistance.

Sensor & Trusted communication integrator: This will provide exposure to an "Internet of Trust" layer for which facilitates the trusted communication between agents to guarantee secure and private communication within the eco-system. Working in conjunction with the security coordinator, it will formulate trust-relationships between independent nodes within the eco-system. This is used in the building of an "Internet of Trust" between devices running the eco-system and sensors, and the brokering of collaborations and transmission of trusted information between trusted content producers and content consumers. It will also determine how trust can be integrated into communication protocols in a bid to aid in the routing of information between trusted parties. The sensor integration aspect to this layer will enable devices running the eco-system to securely interface with agents.

IV. PHASE 1: LANTERN LESSONS

The PLECO language will expand on the initial development phase of this work. The first phase, was the design and development of the Lantern Domain Specific Language (DSL) [14]. The key aim of this was to test ideas on how energy-awareness could be represented within programming language design for controlling Internet of Things based devices within a home environment. Based on these findings, several lessons were learnt which will be built upon for PLECO.

Figure 3a provides a hierarchical overview of the language constructs available within Lantern, while in Figure 3b shows the key components of the Lantern middleware.



(a) Hierarchical overview of language constructs

(b) Language and compiler / middleware stack

Figure 3. Lantern overview

An example of the Lantern language is given in Figure 2. The language took the approach of providing developers with a DSL which allowed environments to be represented within the language. These environments would in the real-world, translate into building representations. Any number of environments could be provided and allowed the definition of statically defined locations to be provided. These locations allowed *environments* to be divided up into smaller administrative boundaries to represent rooms, hallways and other types of locations and provided a means in which the grouping and control of devices located within these statically defined areas could be provided. Consumption rules provided the user with a way of specifying a number of rules in which to monitor the energy usage within these locations and provided a means in which the environment would adapt and use devices located within the location. The *consumption* construct acted as a container object where all rules which needed to override user specific energy utilisation rules would occur. The *override* construct would be used in tandem with the consumption definition to provide the fine grained adaptive nature of the environment. *Identities* were weakly defined within Lantern and represented the user based on their name or identifier. This meant that to allow the environment to be truly adaptive to various users, many identities would have to be defined. Because of this, a group-based membership approach will be used to coordinate this process. *Conditions* were used to represent the user defined rules governing the energy usage and adaptive nature of the system. An outline of the principal parts of a Lantern agent is shown in Figure 2.

As Lantern was to explore and experiment with languages to represent energy-awareness, it did not provide a platform for testing secure programming and communication styles or reliable software. However, several things were learnt from this initial phase which will inform PLECO. They are:

- using environments and sub locations proved to be quite effective in representing buildings and static locations. However, the mobility of devices and sensors was deemed to be insufficient to cope with environments which are dynamic and change over time. This will be investigated further so that the necessary coordination and control constructs are considered to allow situations where mobility is needed.
- telemetry constructs for allowing the flow of information from devices, as well as coordinating the control of such devices proved to be quite effective.
- an expanded set of strongly typed constructs for the representation of energy.
- a weak notion of identity was provided within Lantern and was not powerful enough to represent the number of users within an environment which resulted in scalability issues. To address this, a group-based membership/user approach to allow the inheritance of security principals as well as the management of group-based identities will be adopted.
- a security concern in the Lantern language showed that identities could be mapped to individuals due to the simplistic way in which identities were programmed (string based). However, this will be expanded upon to provide anonymous identities as well as obfuscation of user identities.
- condition rules allowed the introduction of the notion of time (i.e., at a point in time, do something). This will be expanded upon to provide users with more time-based constructs to deal with different length durations (e.g., to reduce the energy use for a specific amount of time in a day).
- conditions provided a clause in which an action would be performed once something had happened. This was found to be adequate but improved structures and language constructs for handling more complex interactions and reacting to non-time bound interactions will be.
- to support verification, other approaches are being looked at (for example contract based and assumed guarantee reasoning).

V. CONCLUSION AND NEXT STEPS

This paper has introduced the PLECO architecture and its objectives. The eco-system to support the next generation of languages and middlewares which have been designed from the ground up to incorporate the notions of energy-awareness, security and reliability rather than added them as a secondary consideration. By incorporating these notions in the design and development of systems will provide more robust and secure systems in which to control large scale distributed IoT devices. The lessons learnt from the first iteration of energy-aware languages have provided a foundation on which to provide the language aspect of the eco-system.

REFERENCES

[1] S. Mittal. “A survey of techniques for improving energy efficiency in embedded computing systems”, International Journal of Computer Aided Engineering and Technology, 6(4), pp. 440–459, 2014.

[2] P. Yang, P. Marchal, et al. “Managing dynamic concurrent tasks in embedded real-time multimedia systems”, Proceedings of the 15th international symposium on System Synthesis, pp. 112–119, ACM, 2002.

[3] Y. Ma, N. Sang, W. Jiang, and L. Zhang. “Feedback-controlled security-aware and energy-efficient scheduling for real-time embedded systems”, Embedded and Multimedia Computing Technology and Service, pp. 255–268, Springer, 2012.

[4] F. Alessi, P. Thoman, G. Georgakoudis, T. Fahringer, and D. S. Nikolopoulos. “Application-level energy awareness for openmp”, International Workshop on OpenMP, pp. 219–232, Springer, 2015.

[5] N. Amsel and B. Tomlinson. “Green tracker: a tool for estimating the energy consumption of software”, CHI’10 Extended Abstracts on Human Factors in Computing Systems, pp. 3337–3342, ACM, 2010.

[6] M. Sabharwal, A. Agrawal, and G. Metri. “Enabling green it through energy-aware software”, IT Professional, 15(1), pp. 19–27, 2013.

[7] N. Nikzad, O. Chipara, and W. G. Griswold. “Ape: an annotation language and middleware for energy-efficient mobile application development”, Proceedings of the 36th International Conference on Software Engineering, pp. 515–526, ACM, 2014.

[8] Y. Xiao, R. S. Kalyanaraman, and A. Ylä-Jääski. “Middleware for energy-awareness in mobile devices”, Proceedings of the Fourth International ICST Conference on COMmunication System softWAre and middlewaRE, p. 13, ACM, 2009.

[9] D. Aspinall, S. Gilmore, M. Hofmann, D. Sannella, and I. Stark. Mobile Resource Guarantees for Smart Devices, pp. 1–26, Springer Berlin Heidelberg, Berlin, Heidelberg, 2005, URL https://doi.org/10.1007/978-3- 540-30569-9_1.

[10] D. Franzen. “Quantitative bounds on the security-critical resource consumption of javascript apps”, PhD Thesis, University of Edinburgh, 2016.

[11] G. Smith. “Updates of the itrs design cost and power models”, 2014 IEEE 32nd International Conference on Computer Design (ICCD), pp. 161–165, 2014.

[12] Semiconductor Industry Association. “The international technology roadmap for semiconductors”, URL http://www.itrs2.net, 2013. [retrieved: January, 2018]

[13] J. Robinson, I. Wakeman, and D. Chalmers. “Composing software services in the pervasive computing environment: Languages or apis?”, Pervasive and Mobile Computing, 4(4), pp. 481–505, 2008.

[14] J. Robinson, K. Lee, and K. Appiah. “Lantern – A Smart-Home Enabled Domain Specific Language for Energy Awareness in Cyber-Physical Systems”, Under review - unpublished, 2017.

[15] L. Ericsson. "More than 50 billion connected devices." *White Paper*, 2011.

[16] A. Al-Fuqaha, M. Guizani, M. Mohammadi, M. Aledhari, and M. Ayyash. “Internet of things: A survey on enabling technologies, protocols, and applications”, IEEE Communications Surveys & Tutorials, 17(4), pp. 2347-2376. 2015.

[17] Z. Zhang, et al. "IoT security: ongoing challenges and research opportunities", Service-Oriented Computing and Applications (SOCA), IEEE 7th International Conference on. IEEE, pp. 230—234. 2014.

[18] A. Orgerie, M. de Assuncao, and L. Lefevre. "A survey on techniques for improving the energy efficiency of large-scale distributed systems." ACM Computing Surveys (CSUR) 46.4 47. 2014.

[19] C. Kim, et al. "Typed Architectures: Architectural Support for Lightweight Scripting." Proc. of the 22nd International Conference on Architectural Support for Programming Languages and Operating Systems, pp. 77-90, ACM, 2017.