Supporting Energy-driven Adaptations in Distributed Environments

Adel Noureddine, Romain Rouvoy, Lionel Seinturier

Supporting Energy-driven Adaptations in Distributed Environments

Adel Noureddine, Romain Rouvoy and Lionel Seinturier
INRIA Lille – Nord Europe, Project-team ADAM
University Lille 1 - LIFL CNRS UMR 8022, France
firstname.lastname@inria.fr

ABSTRACT

The rise of the usage of digital devices and software services contribute to the increase of energy consumption of IT infrastructures. However, energy is still largely produced by limited resources. Therefore, optimizing and reducing its consumption is an economic and human necessity. Related works addressing energy optimization in computer science are widespread, but at the middleware layer, existing approaches are limited in their scope, adaptability and autonomous functioning. In this paper, we propose the foundations of a middleware architecture capable of handling various types of energy techniques and in different contexts. The distributed nature of our approach fits in a ubiquitous environment and covers the energy dimensions of both devices and software services. We also present the experimental results of a prototype implementing a subset of our proposed architecture. These results illustrate the potential of our energy-aware and autonomous approach.

Categories and Subject Descriptors

D.2.11 [**Software Engineering**]: Software Architectures—*Service-oriented architecture (SOA)*

General Terms

Middleware, Energy awareness, Distributed Environments

1. INTRODUCTION

With the increase in the usage of digital devices (smartphones, tablets, televisions), and software services *in the cloud* that require important processing and memory resources, energy consumption of these devices and services is starting to become unsustainable. While raw materials needed for energy production are limited, their impact is rising and reaching a peak of consumption [4].

Research to improve the efficiency of devices and reduce the power consumption of services, or *Green IT*, has and is still carried out to address these challenges. Reducing the energy consumption of connected devices and computers requires a comprehensive view of the different layers of the system. The middleware layer is, thus, a good candidate for hosting energy-aware approaches and solutions. However, current proposed solutions are limited in their scope or adaptability (cf. Section 5).

In this paper, we present an energy-aware adaptive middleware architecture that tackles some of the challenges of energy-aware middleware. In particular, we propose an architecture capable of handling various types of energy optimization techniques that can be build on top of an adaptive middleware. Its distributed nature and its reliance on distributed architectural styles, such as REST, could allow it to scale from managing the energy consumption of smart homes up to data centers and large-scale systems.

The remainder of this paper is organized as follows. Section 2 introduces our motivation scenario. Section 3 reports on the reference architecture of our framework. In Section 4, implementation details of a prototype of a subset of our architecture, as well as experimentation results on the motivation scenario are presented and discussed. Finally, we discuss related work and conclude in Sections 5 and 6, respectively.

2. MOTIVATION SCENARIO

Watching streaming videos on a tablet PC

Tom wishes to watch a movie on his new tablet PC and prefers watching it while lying on his bed. Tom cannot connect his tablet to an electrical outlet, thus his tablet is limited by its battery capacity. The movie is stored on a NAS server and can be streamed over WiFi. When Tom starts playing the movie, the home energy management system detects this event and starts collecting energy and context data from both the tablet (including the video player software) and the NAS server. During the movie playback, the system monitors the evolution of the energy consumption and adapts the system components in order to satisfy Tom's main objective (watching the whole movie before the battery runs out), while still respecting Tom's pre-defined preferences (*e.g.*, minimum video quality). The adaptation varies from degrading the quality of the video playback to lowering sound volume or screen brightness.

This motivation scenario illustrates a use case example of our contribution: managing and optimizing the energy consumption of digital devices and components in an autonomous manner. The energy management system adapts the video and tablet in order to satisfy Tom's wishes to watch the entire movie without interrupting the playback or running out of energy.

3. MIDDLEWARE ARCHITECTURE

In this section, we overview the main principles of our architecture, and then we detail a prototype architecture for our decision engine.

3.1 Overview

We propose a middleware architecture (cf. Figure 1) built on the REST architectural style [6] and following the MAPE-K autonomic control loop reference model [7]. The architecture is therefore distributed and can be scaled to be deployed in different environments, from smart homes up to large-scale systems.

REST (*REpresentational State Transfer*) is a resource-oriented software architecture style for building distributed applications [6]. The resources are uniquely addressable using an universal syntax (*e.g.*, URL in HTTP), and share an uniform interface for the transfer of application states between client and server (*e.g.*, GET, POST, PUT, DELETE in HTTP). Sensors and actuators in our architecture are reified as REST resources, each having a dedicated URI.

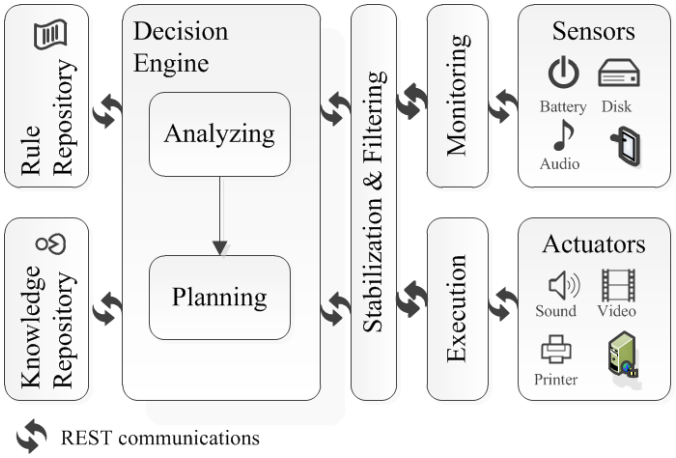

Figure 1: A General Overview of our Reference Architecture.

Our architecture is built through different components of the MAPE-K autonomic control loop. The *monitoring component* collects energy information and observes the environment for any context changes. These collected information are then used by the *analyzing component* for processing and identifying *symptoms* that require an adaptation. When a symptom is detected, the *planning component* selects efficient adaptation *actions* from the rule repository, and forwards them to the *executing component* to be executed.

A stabilization and filtering layer is used to filter outdated measures or insignificant changes considering the *Quality of Context* (QoC) [5] of the energy measures. The layer filters also adaptation actions that are considered as obsolete after the decision process.

Adapting the system requires energy information to be monitored from the environment. However, in addition to energy measures, additional contextual information is also needed. In particular, we identified component characteristics, communication protocols characteristics, and network quality as key information for energy-aware adaptations.

In addition to four parameters of QoC classification [9] that are relevant to energy-awareness, we also extend the classification with an additional QoC dimension: *Energy Source Type*. This parameter defines the type of the energy source used to power one or a group of components, and its value is taken from a list of keywords (*e.g.*, battery, AC).

3.2 Autonomous Decision Engine

Two main components form our decision engine: an analyzing component (cf. Figure 2) and a planning component (cf. Figure 3).

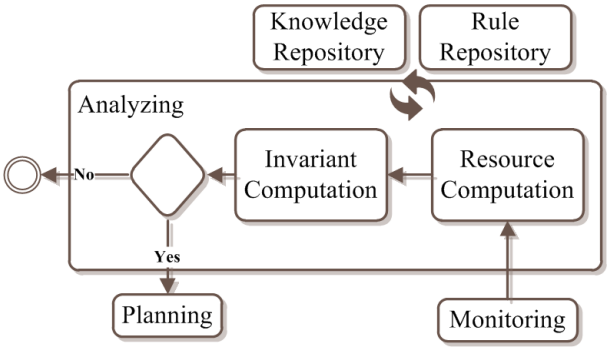

Figure 2: The Analyzing Component.

The analyzing component receives monitored information from the *monitoring component*, processes this information and decides if the energy context needs adaptation. It uses information from the *knowledge repository* and *rule repository* in order to help in the decision. Information such as users preferences and objectives, or stored component characteristics are therefore used in the computation phases of the analysis. In particular, the *analyzing component* calculates resource data (*e.g.*, available power in the battery, or time left of the video playback). The calculated resources are not limited to direct energy computation. Temperature, budget or user location resources could also be calculated in this phase and used in the decision process. For our prototype architecture, we propose two resource parameters: *required resource* and *available resource*. Available resource defines how much resources the components are allowed to consume (*e.g.*, power left in the battery). Required resource defines the amount of resources the components need in order to carry out the user objective (*e.g.*, watch the video entirely). The *analyzing component* calculates also system invariants, such as the minimum or maximum values of a component parameter (*e.g.*, minimum video resolution as stated by the user preferences). Finally, it compares the calculated resources and determines if the system needs an adaptation. In the latter case, it notify the *planning component*.

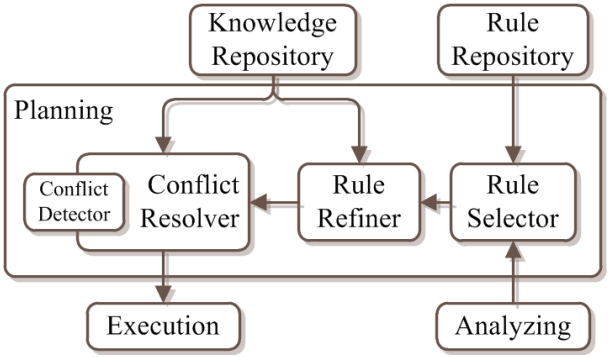

Figure 3: The Planning Component.

The planning component (cf. Figure 3) is notified when the system requires energy-aware adaptations. It first selects relevant rules from the *rule repository* based on their conditions, then refines the selected rules using information from the knowledge repository (*e.g.*, component characteristics, user preferences). Rules follow the *conditions* $\rightarrow$ *actions* principle, with conditions being based on the calculated resource parameters (*e.g.*, energy, time, budget). Rules can be complex, but we decided to use basic rules in our prototype for simplicity, as described in Listing 1. In this example, the *screen brightness* will be set to 70%, the *sound volume* to 85% and the *video resolution* to 1024×780 if the *available battery power* is less than 80% of its total capacity.

Rule 1 Scarce Battery

```
// energy becomes low on the tablet PC
if analyzing.Available < 80% × anlyzing.Required then
  // lower the quality of the video playback
  executing.Screen.Brightness ← 70%
  executing.Sound.Volume ← 85%
  executing.Player.Video.Resolution ← 1024 × 780
end if
```

Next, the *planning component* detects and resolves conflicts between the selected actions and makes sure that these actions will not damage the system integrity or have a negative impact on energy consumption.

For that, the resolver needs information from the knowledge repository, and performs its verifications in three steps:

1. First, it considers a *list of known conflicts*, which is stored in the *knowledge repository* and is maintained and updated whenever a new conflict is detected. The list contains conflicts, such as incompatibility between several actions, or combinations of actions that could lead to damaging the system integrity or security, or to substantial energy losses. The conflict detection is based on a *conflict detector* component that uses context learning techniques in order to detect conflicts between system components. These techniques are based on energy consumption history and an association of previously applied actions with the variation of energy consumption afterward. The conflict detection approaches are still an ongoing research and are planned for future works.
2. If no conflicts are detected, and several actions apply to the same parameter of a component, a *priority system* is then used to differentiate rules and actions. If a priority value is assigned to a rule, then all of its actions inherit the same priority. However, actions, indiscriminately of their rules, can be assigned a priority value that will be used in the selection process instead of its inherited priority.
3. Finally, in the case of actions having the same priority, the resolver will *estimate the saved energy* for each action, and selects the one that maximizes energy saving.

Finally, selected actions are deployed on the system through the *executing component*. The latter uses actuator interfaces to forward adaptation actions using REST principles.

4. SIMULATION AND EMPIRICAL VALIDATION

Our experimentation is based on the video streaming scenario described in Section 2. In particular, we first deployed the application in a simulated environment to check the coverage and the efficiency of rules, and then we tested the application in a real environment to observe potential side-effects of energy-driven adaptations.

Software Prototype

We used *Service Component Architecture* (SCA) [2] architectural styles to build our prototype. SCA is a set of specifications for constructing and combining distributed applications, based on the principles of SOA (*Service Oriented Architecture*) and CBSE (*Component Based Software Engineering*). SCA applications are created from basic construction blocks—*i.e.*, *components*. They have *services* (provided interfaces), *references* (required interfaces), and expose *properties*. SCA is designed to be language-agnostic, which is an advantage for building adaptable and scalable energy-driven architectures.

We built a subset of our architecture on top of the FRASCATI [16] middleware platform. FRASCATI is a flexible and extensible platform that allows the execution of SCA applications. Interestingly, it offers runtime adaptation and manageability properties to SCA applications. FRASCATI also offers reflective capabilities with introspection, intercession, monitoring, and dynamic reconfiguration. These properties and capabilities meet the requirements of our architecture. The reflective capabilities allows dynamic reconfiguration of the architecture itself, such as *on-the-fly* updates of the planning component.

SCA Components

Our prototype monitors four system components (battery, CPU, screen, and sound) and the video stream. It can send adaptation actions to three components (screen, sound, and video). Both monitoring and sending adaptations are based on REST principles—*i.e.*, components expose their data as REST resources. The video component uses MPlayer[1] to stream and control the video. We used an existing Java class[2] to encapsulate MPlayer as an SCA video component (cf. Figure 4). The experimentation has been performed on a single computer (to reduce the noise induced by the network infrastructure), but it can be easily reproduced in a distributed system.

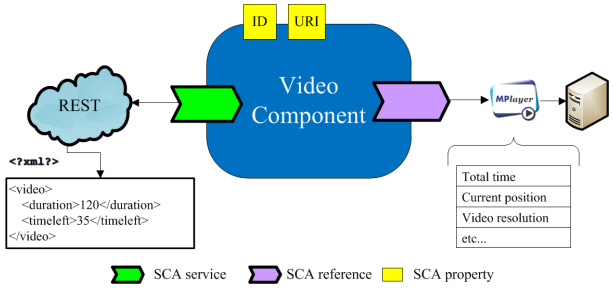

Figure 4: SCA Video Component used in the Experimentation.

[1] http://www.mplayerhq.hu
[2] http://beradrian.users.sf.net/articles/JMPlayer.java

4.1 Simulation

We first simulated the hardware and software components (actuators and sensors) and run our prototype using four simulated SCA components: CPU, battery, video, and screen. The hardware and software access of these SCA components are simulated using mathematical formulas (*e.g.*, battery draining is simulated by calculating the power consumed per minute by the CPU and the screen). Thanks to this simulation, we can also send adaptation actions to the CPU component without having to manage the operating system's power management of the CPU using DVFS [8]. The simulation allows us to monitor the adaptation process in a controlled environment in order to tune our decision rules. It also shows the flexibility of our architecture as the calculated resources types differ between the simulation and the experimentation.

Required and available resources are defined as required energy (energy needed to achieve the user objective) and available energy (in the battery). The required energy is calculated based on the following formula, using the CPU *Thermal Design Power* (TDP):

$$E_{required} = Timeleft_{video} \times TDP \quad (1)$$

with:

$$TDP = TDP_{max} \times (\frac{frequency}{frequency_{max}}) \times (\frac{voltage}{voltage_{max}})^2 \quad (2)$$

The simulated video has a total length of 30 minutes with 4 resolutions (1280x720, 1027x780, 640x480, and 320x240). The processor supports five different frequencies and voltages: 1.5GHz for a 1.2V, 1.2GHz-1.1V, 1GHz-1V, 800MHz-0.9V and 500MHz-0.8V. The video needs at least 800MHz to run correctly without glitches with the lowest 320x240 resolution. The screen consumes 1Watt for a 100% brightness and 0.2Watt for 0% brightness. The battery has a full charge capacity of 1400mAh with 3.7V. In this context, the user wishes to have a minimum resolution of 640x480 and a minimum screen brightness of 50%.

We tested our prototype with these simulation parameters using 3 different rule sets (named *eco1*, *eco2*, and *eco3*). These 3 rule sets define the same actions on the managed components (CPU, video and screen), but use different comparison percentages in their rule conditions. This difference in the percentage value means that actions in eco3 rules are triggered earlier in the process—*i.e.*, when available resource value is closer to the required resource value. Using three different rule sets allows us to analyze the impact of changing rules on both the energy consumption and the quality of service of the video playback experience. We defined the 3 rule sets so that eco3 should, in theory, allow better energy economy than eco2 and eco1, and eco2 allows better energy economy than eco1. The results we obtained are consistent with our initial theory and are reported in Figure 5 (we stopped the simulation after 31 minutes), while Figure 6 depicts the evolution of both required and available energy variables (calculated in the analyzing component) with rule set eco1.

Discussion

Simulation results show a high gain starting at 30% for rule set eco1 comparing to the base scenario with no adaptation, growing to 42% for eco2, and more for eco3. The evolution of the required and available energy variables in Figure 6 shows clearly the steps where adaptation actions were executed (t = 8, 18, and

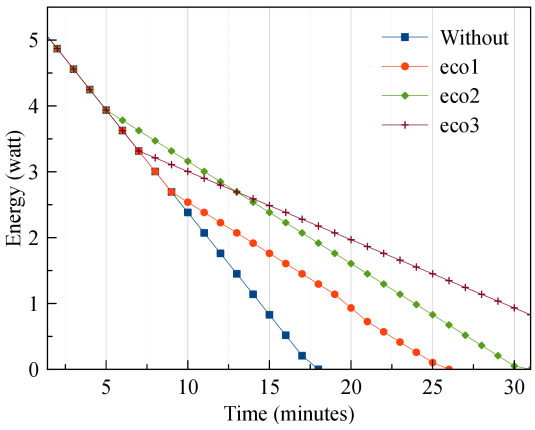

Figure 5: Device Battery Energy Evolution (Rule Sets *eco1*, *eco2*, and *eco3*).

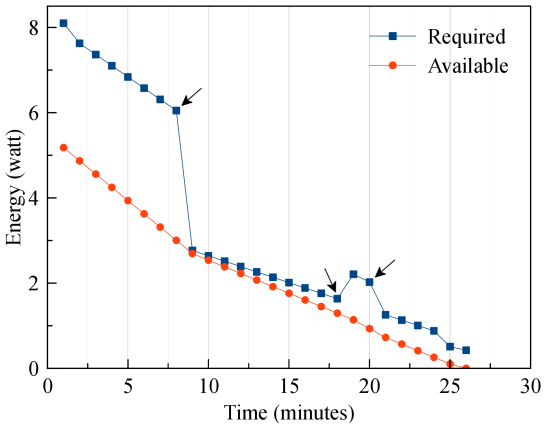

Figure 6: Required and Available Energy Evolution (Rule Set *eco1*), with highlights on the moments when adaptations were applied.

20 minutes). Rules in the simulation were defined so that adaptation actions are executed only when the calculated available energy is lower than the calculated required energy. Our prototype tries to close the gap between both calculated resources. When this gap is smaller, the context has already changed and other rules verify the new conditions, and new actions are executed (*e.g.*, t = 18 minutes).

4.2 Experimentation

We have also fully implemented the hardware and software components and carried out experimentation on real hardware. Our experimentation is deployed on a Dell Latitude D410 laptop (2 GHz Intel Pentium M processor with 1 GB of RAM and a 12.1" screen) running Ubuntu 10.04 (GNOME 2.30.2 and Linux 2.6.32-24 kernel). The video used for the scenario has a total duration of 39:06 minutes, and is encoded in five resolutions (1280x720, 1024x780, 640x480, 480x270, and 320x240). Required and available resources are defined as *video time left* and *battery estimated time left*, respectively. The former is extracted from the video playback using the MPlayer slave properties, while the latter is ex-

tracted using the ACPI (*Advanced Configuration and Power Interface*) command.

We tested our prototype with 2 rule sets, similar to those used in our simulation. Actions in these rules apply on the video, screen, and sound components. We named the rule sets *eco5* and *eco6*, with *eco6* allowing better energy economy than *eco5*.

Results with rule sets *eco5* and *eco6* are depicted in Figure 7. The battery has 12.2Wh and a percentage of 38.1%. The value of the battery time left is taken from the ACPI program. It is an estimated value and depends on the system activity on the time of the call. This explains the difference in the values in the first two minutes between *eco5* and *eco6*.

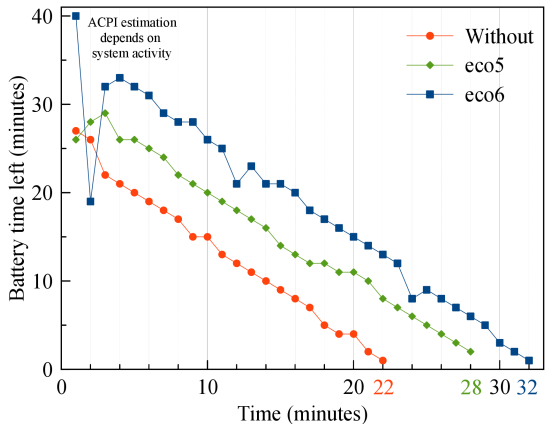

Figure 7: Device Battery Lifespan Evolution with Rule Set *eco5* and *eco6*.

Discussion

The results show clearly the advantage of using our energy-driven adaptation architecture, since the use of *eco6* rules demonstrated a gain of 31.25% in the playback lifespan compared to the base scenario without adaptation. In our experiments, the battery time left is way shorter than the length of the video ($\approx$ 26 min for a 39 min video). We managed to gain 6 minutes with rule set *eco5*, and 10 minutes with *eco6* in comparison to results without any adaptations. Our experimentation shows that, although the gain is smaller in comparison to the simulation, we still get major gains ranging from 10 to 30% depending on rule sets used.

4.3 Discussions

Simulation and experimentation

The main difference in the experimentation is that the sensor and actuator components had access to the hardware and software, unlike the simulation where this access is simulated using mathematical formulas. The energy consumption follows a straight evolution in the simulation, and results show a optimal efficiency of our adaptations. In the experimentation, the results are less predictable, but still shows an average gain of 20%, with a peak over 30%. This difference in energy gains comes from the experimental conditions, which are closer to the reality then the simulation ones.

Importance of rules

The results confirmed the impact of rule definitions on adaptation. Choosing strict rules allows the system to save more energy or time, and thus enabling it to achieve the user objective. However, and in order to achieve this energy saving, stricter rules impact the quality of service and the user experience for a longer time (playing with the lowest resolution version of the video, or reducing the brightness of the screen earlier in the playback). The way rules are defined impacts the time of the adaptation outbreak and this impact influences the adaptation process and the results. User preferences also influence the results due to the fact that they act as rule filters (Cf. Simulation in Section 4.1). Finally, the environment context in which the application evolves also influences decisions made by our architecture (*e.g.*, executing other applications while playing the video).

5. RELATED WORKS

Existing middleware energy solutions currently focus on one or few optimization techniques and system levels at a time.

Energy modeling

In [12], Petre presents an energy model using the middleware language MIDAS [13] that classifies energy as data, code, or computation unit resources. Computation units are distinguished into 3 types: *software*, *hardware*, and *electrical sockets*. Energy is defined as a quantity that is consumed by the hardware (and indirectly by the software) components. In our architecture, we list the information to be gathered and needed for the energy analysis. We also extend the definition of quality of context to add the *energy source type*, which covers the types listed by the author, and allows the consideration of other or new types (such as renewable energy).

Mobile Computing and Networks

The rules we used in our prototype are similar to the Transhumance [11] adaptation policies, but our rule model focuses on the user objective and is defined according to that goal. Energy management in Transhumance focuses mostly on adapting the middleware modules. Applications adaptations is achieved if the middleware adaptations affect the applications normal functioning. Our approach, however, targets software and hardware components regardless of they being part of the applications or the middleware. Unlike Transhumance, we also incorporate conflict detection and resolving management in our architecture.

In [17], the authors propose an energy-aware middleware for mobile devices that is based on application classifications and power estimations to accomplish application-specific energy optimizations. Applications are first classified based on their activity and a power estimator is used for estimating energy consumption of hardware components and applications. Then, a processing engine manages events and power consumption and uses a policy manager to chose adaptive policies to apply on the system. In comparison, we use a rule-based decision engine to chose adaptation actions, which is similar to the general approach proposed by the authors. However, our approach is based on general components and resources computation while they base their energy management on application specific power estimation and classification. We also propose some mechanisms (*e.g.*, conflict resolver) to maintain system integrity and adaptation coherence, and we do not rely solely on the correct definitions of adaptations rules. The authors, however, plan to add the functionality of conflict management in their future works.

In [10], the authors proposed a middleware framework, PARM, for low-power devices. PARM dynamically reconfigures component distribution on these devices and migrate components to proxy

servers in order to save energy on the mobile client. PARM uses an algorithm to determine what components to migrate, and uses policies to determine when and how often the algorithm is executed. However, PARM is limited to environments with proxies and servers, while we propose a distributed architecture that doesn't require specific devices. We also apply various adaptations while PARM is limited to one technique: component migration from mobile clients to proxy servers.

Sensor Networks

In [14, 15], SANDMAN is presented as an energy efficient middleware built upon the BASE middleware [1]. It saves energy through 3 concepts: *i)* selecting the most efficient communication protocol, *ii)* switching idle devices to low power mode (race to sleep), and *iii)* allowing clients to choose services based on their energy efficiency. In our approach, we take into account the communication protocols efficiency, but extend the knowledge to components characteristics. Our rule-based decision engine is agnostic to specific optimization techniques. It can apply the *race to sleep* strategy granted proper rules (*e.g.*, for turning off devices) are defined in the rule repository. Our approach also respects the user preferences, such as limiting his services choices to energy efficient components. But also, the architecture can evolve and manage different or new techniques, and allows an easy approach to users preferences evolution.

Data centers

In [3], the authors propose a dispatch algorithm for data centers that uses the *race to sleep* strategy in conjunction with the *proxy* strategy. In particular, the system consolidates services in a minimum number of servers and shuts down the unused ones to save energy. Their approach consolidates services while respecting the system service level agreements. Our architecture allows the consideration of both *race to sleep* and *proxy* strategies. It includes different optimization strategies stored in the rule repository. The advantage of our architecture is that it is not limited to a single optimization technique and is open to various strategies.

6. CONCLUSION AND FUTURE WORKS

In this paper, we presented our energy-driven middleware architecture. It allows system-wide energy adaptations in an efficient and coherent way. It handles different types of energy optimizations and chooses adaptation actions based on the environment context. The architectural proposal can be built on top of an adaptive middleware and communicates using distributed technologies.

We demonstrated the feasibility of our reference architecture based on the MAPE-K autonomic model by reporting on an implementation based on the FRASCATI middleware platform. The prototype is an implementation of a rather small subset of our architecture and has been used in a simulated and then a realistic environment to demonstrate the benefit of our approach on energy savings. Although, the experimentations we reported show the potential of our solution, some work needs to be done to:

- measure the energy overhead of our architecture in a digital home environment,
- integrate mechanisms for helping users in defining energy rules. These mechanisms could be fully automated or can accompany users. Machine learning approaches could also be used to limit users intervention in the adaptation process and tune the rules according to user objectives,
- develop methods to translate business rules and objectives into efficient components energy adaptation rules,
- propose detailed models and meta-models of the architecture, its computational algorithms and interaction diagrams.

Acknowledgments

This work was supported by Ministry of Higher Education and Research, Nord-Pas de Calais Regional Council and FEDER through the "*Contrat de Projets Etat Region (CPER) 2007-2013*".

References

[1] C. Becker, G. Schiele, H. Gubbels, and K. Rothermel. Base - a micro-broker-based middleware for pervasive computing. In *Pervasive Computing and Communications, 2003. (PerCom 2003). Proceedings of the First IEEE International Conference on*, pages 443–451, 2003.

[2] M. Beisiegel and et al et al. Service Component Architecture. www.osoa.org, 2007.

[3] W. Binder and N. Suri. Green Computing: Energy Consumption Optimized Service Hosting. In *SOFSEM '09: Proceedings of the 35th Conference on Current Trends in Theory and Practice of Computer Science*, pages 117–128, Berlin, Heidelberg, 2009. Springer-Verlag.

[4] BP. Statistical Review of World Energy 2010. www.bp.com/statisticalreview, 2010.

[5] T. Buchholz, A. Küpper, and M. Schiffers. Quality of Context Information: What it is and why we need it. In *In Proceedings of the 10th HPOVUA Workshop, 2003, Geneva, Switzerland*, 2003.

[6] R. T. Fielding. *Architectural styles and the design of network-based software architectures*. PhD thesis, University of California, Irvine, 2000. Chair-Taylor, Richard N.

[7] J. O. Kephart and D. M. Chess. The Vision of Autonomic Computing. *IEEE Computer*, 36(1):41–50, 2003.

[8] E. Le Sueur and G. Heiser. Dynamic voltage and frequency scaling: The laws of diminishing returns. In *Proceedings of the 2010 Workshop on Power Aware Computing and Systems (HotPower'10)*, Vancouver, Canada, Oct 2010.

[9] A. Manzoor, H.-L. Truong, and S. Dustdar. On the Evaluation of Quality of Context. In *Proceedings of the 3rd European Conference on Smart Sensing and Context (EuroSSC)*, pages 140–153. Springer, 2008.

[10] S. Mohapatra and N. Venkatasubramanian. Parm: Power aware reconfigurable middleware. In *ICDCS '03: Proceedings of the 23rd International Conference on Distributed Computing Systems*, page 312, Washington, DC, USA, 2003. IEEE Computer Society.

[11] G. Paroux, lsabelle Demeure, and L. Reynaud. A Power-aware Middleware for Mobile Ad-hoc Networks. In *Proceedings of the 8th International Conference on New Technologies in Distributed Systems (NOTERE)*, pages 1–7. ACM, 2008.

[12] L. Petre. Energy-Aware Middleware. In *Proceedings of the 15th Annual International Conference and Workshop on the Engineering of Computer Based Systems (ECBS)*, pages 326–334. IEEE, 2008.

[13] L. Petre, K. Sere, and M. Waldén. A language for modeling network availability. In Z. Liu and J. He, editors, *Formal Methods and Software Engineering*, volume 4260 of *Lecture Notes in Computer Science*, pages 639–659. Springer Berlin, Heidelberg, 2006.

[14] G. Schiele and C. Becker. SANDMAN: an Energy-Efficient Middleware for Pervasive Computing. Technical report, University of Mannheim, Karlsruhe, Germany, october 2007.

[15] G. Schiele, M. Handte, and C. Becker. Experiences in designing an energy-aware middleware for pervasive computing. In *Proceedings of the 2008 Sixth Annual IEEE International Conference on Pervasive Computing and Communications*, pages 504–508, Washington, DC, USA, 2008. IEEE Computer Society.

[16] L. Seinturier, P. Merle, D. Fournier, N. Dolet, V. Schiavoni, and J.-B. Stefani. Reconfigurable SCA Applications with the FRASCATI Platform. In *Proceedings of the International Conference on Services Computing (SCC)*, pages 268–275. IEEE, 2009.

[17] Y. Xiao, R. S. Kalyanaraman, and A. Ylä-Jääski. Middleware for energy-awareness in mobile devices. In *COMSWARE '09: Proceedings of the Fourth International ICST Conference on COMmunication System softWAre and middlewaRE*, pages 1–6, New York, NY, USA, 2009. ACM.