Components Mobility for Energy Efficiency of Digital Home

Rémi Druilhe*†, Matthieu Anne*, Jacques Pulou*
Laurence Duchien†, Lionel Seinturier†

*Orange Labs
28 Chemin du Vieux Chêne
38420 Meylan, France
firstname.lastname@orange.com

†INRIA Lille-Nord Europe
LIFL (UMR CNRS 8022)
Université Lille 1, France
firstname.lastname@inria.fr

ABSTRACT

The number of connected devices in the home is growing dramatically, increasing the part of the Digital Home in the electric power demand. Reducing the overall energy consumption of the Digital Home becomes a concern in everyday life. Moving applications to the smaller set of devices enables to increase the number of devices that can be put into low power state, and thus reduce energy consumption. However, the application deployment constraints and the Digital Home heterogeneity limit the choice in deployment solutions onto available devices. We propose to consider distributed component-based applications to overcome this limitation. The distribution of applications constraints over its components improves their mobility, i.e., increasing the number of devices on which a component can be deployed. This approach is optimized to reduce the set of processed solutions. Moreover, the proposed architecture reacts continuously to relevant modifications in the Digital Home software architecture (connection and disconnection of devices, start and stop of applications) to always meet energy efficiency. The architecture is also designed to limit its own energy consumption impact. The feasibility of the approach is assessed with Digital Home applications and migration policies between devices.

Categories and Subject Descriptors

D.2.11 [**Software Engineering**]: [Software Architectures]; G.1.6 [**Optimization**]

General Terms

Design, Performances

Keywords

Energy Efficiency, Deployment Constraints, Components Mobility, Digital Home

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. To copy otherwise, to republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee.
CBSE'13, June 17–21, 2013, Vancouver, BC, Canada.
Copyright 2013 ACM 978-1-4503-2122-8/13/06 ...$15.00.

1. INTRODUCTION

The digitalization of the society introduces new devices and usages in the Digital Home (DH). However, despite existing mechanisms of self-optimization to reduce energy consumption, the growing presence of devices in the DH increases the overall energy consumption. Designing global management systems is becoming a necessity to overcome this problem.

Current global management systems assume that once a device is inactive, it can be put into low power state. These approaches reduce the energy consumption but they assume that applications are tied to the device. The applications distribution on the devices consumes a given energy. To increase inactive devices, the system has to wait until all the applications of a device stop to consider it unused.

However, if we consider that applications can move within the devices, new solutions become available to improve energy efficiency. It enables to modify the distribution plan to increase inactive devices. But, allowing a dynamic modification of the distribution plan requires to deal with the applications deployment constraints (e.g., hardware resources) on the heterogeneous devices of the DH [6].

Another property of the DH is its dynamicity, i.e., the unforeseeable appearance/disappearance of devices and applications. These events can jeopardize the energy efficiency of a distribution plan. Thus, the system must consider these events to always adapt the distribution plan consequently.

We advocate that using component-based distributed applications can improve the energy efficiency of the DH by increasing the set of possible distribution plans. However, a trade-off must be found between a large set of possible distribution plans and a small optimization process duration.

This approach requires to consider the following challenges:

- $c1$. Improve the mobility of an application to easily migrate it without modifying its execution;
- $c2$. Design an energy efficient architecture that adapts the distribution of applications to be as much as possible energy efficient.

Section 2 focuses on the model of the DH. Section 3 discusses the use of component-based distributed applications to improve energy efficiency such as the distribution of the constraints of the application. In Section 4, an architecture is defined to handle the dynamicity of the DH and to provide an energy efficient distribution plan. Section 5 provides the evaluation of our approach. Section 6 describes related work. Section 7 concludes and gives some perspectives.

2. BACKGROUND

The energy efficiency corresponds to the minimization of the energy used to provide a useful work [13]. In our case, it is a minimization of the energy used to provide a service.

We assume that an application can move to another device. Hence, minimizing energy is done with a better distribution of the applications on the devices. Unused devices are put into low power state. We do not address energy consumption of an application. Thus, a model is required to map the current distribution of the applications on the devices and to provide a way to modify this distribution plan. The model must also consider the dynamicity, its events and their impacts in the DH, because it can jeopardize the energy efficiency of the distribution plan by adding or removing devices or applications.

Finally, energy efficiency is achievable only if the moved application can provide the same service in all distribution plans. Degrading the service to reduce energy consumption does not lead to energy efficiency because we assume this is not the same service. Thus, we detail our definition of "service".

2.1 Modelling the DH

Finding an energy efficient distribution plan requires to consider the distribution of the applications on the devices at a given time t. To reach energy efficiency, we also have to model the actions to move applications within devices and the events that modify the size of the distribution plan.

The **distribution plan** (DP) maps the applications on the devices. It is represented by a matrix $O^t = (o^t_{da})_{1 \leq d \leq |D^t|, 1 \leq a \leq |A^t|}$ at time t. D^t is the non ordered set of devices available in the DH environment at time t and A^t is the non ordered set of applications at time t. Each time an application runs on a device, the corresponding cell in the matrix is set to 1, 0 otherwise. With this matrix, the number of possible DPs is $|D^t|^{|A^t|}$.

The **deployment plan** defines the actions required to go from one DP at the time t to another DP at the time $t+1$. It is a same size of matrix as the DP where a cell is set to 1 if an application is going to be deployed on a device, -1 if it is going to be removed from the device, 0 else.

The **dynamicity** is handled through events: devices and applications appear and disappear in an unforeseeable way. These events lead to a modification of the set of devices D^t or of the set of applications A^t by adding or removing an element. The resulting *updated* DP may not be energy efficient because the DP has changed. This triggers a deployment plan to get an *optimized* DP which is energy efficient.

2.2 Constraints of the application

A client, i.e., another application or a user, expects a service to satisfy its needs. Thus, these needs can be specified as constraints on the deployment of the application. Improving energy efficiency is equivalent to satisfy the constraints of the service with less energy on another device.

It first requires a preliminary work during the design phase: determining the constraints to satisfy to deploy an application on a device. These constraints are considered during the processing of a new DP. For example, an application can have constraints such as: `ROM = 20` or `UserPresence = true`.

The second step requires to look for the most energy efficient DP, considering the constraints of the applications and the dynamicity of the DH. In the case of the previous application, the system should get the position of the user, `UserPresence = true` on Device 1, and the remaining hardware resources for each device: `ROM = 200` on Device 1 and Device 2. Then the system should interrelate the data from the devices to the data from the application: user is on Device 1 (and not on Device 2) and there is enough hardware resources on both. The application must be deployed on Device 1.

3. DISTRIBUTION OF CONSTRAINTS

Current applications are usually monolithic. Lot of constraints must be satisfied to deploy them. It restricts the set of DPs, noted $\mathcal{S}_{ma}$, to host them in the DH.

However, if we consider component-based distributed applications, constraints of an application can be distributed among its components. Each component holds its own constraints, independently from the other components. Taken separately, the components can be deployed on a larger set of devices, enlarging the set of possible DPs, noted $\mathcal{S}_{da}$, where $\mathcal{S}_{ma} \subset \mathcal{S}_{da}$. This decomposition improves the mobility of a component, i.e., its ability to be deployed on a large number of devices, considering its deployment constraints.

Nevertheless, this decomposition increases the number of solutions to consider during the optimization process, and thus slows it down. Consequently, the number of possible DPs changes to $|D^t|^{|C^t|}$ (C^t is the set of components at the time t and $|C^t| \geq |A^t|$). Hence, it is important to reduce the number of entries considered by the optimization process.

We consider that a composite is a set of one or more components having the same constraints that do not reduce the mobility of the components. Using composites creates a new set of possible DPs, $\mathcal{S}_{ga}$, where : $\mathcal{S}_{ma} \subset \mathcal{S}_{ga} \subset \mathcal{S}_{da}$. We assume that this new set enables to look for an optimal solution which does not degrade the component mobility.

To find such composites, we divide the deployment constraints into two types:

- the "extensive constraint". This corresponds to an amount of a thing required by a component. For example, this is the case for components requiring an amount of hardware resources, e.g., `ROM = 20`.
- the "intensive constraint". This corresponds to an uncountable thing required by a component. For instance, components may require the user presence: `UserPresence = true` or to be deployed in a specific location: `Location = Kitchen`.

These two types of constraints illustrate how to distribute the constraints of an application to its components and how to group some of the components into a composite to reduce the set of possible solutions processed.

The decomposition increases the set of possible DPs by decomposing applications into components with their owns constraints. This step is made by the architect at the design phase, considering its knowledge about the application.

The decomposition assumes that an application comes with a set of deployment constraints and a set of components. First, the requirements of each component are defined (e.g., user presence, hardware resources). And then, the constraints of the application are distributed to the components, considering the requirements of the components. A constraint can be applied to many components and many different constraints can be applied to an unique component.

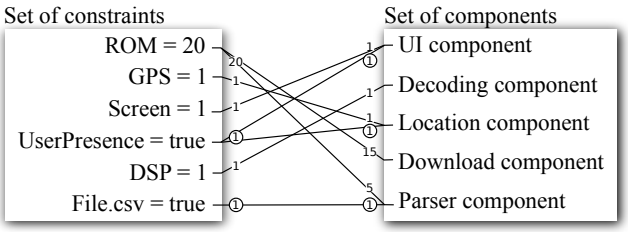


Figure 1: Constraints of the application distributed to components of the application. Circled numbers are intensive constraints. Non circled numbers are extensive constraints.

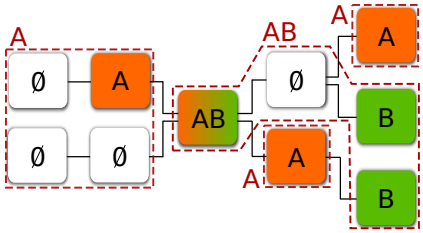


Figure 2: One decomposition of an application into components and one grouping of components into composite. Coloured components are components with deployment constraints. A is an extensive constraint. B is an intensive constraint. Red dotted lines are composites.

The distribution differs for both constraint types. The distribution of an intensive constraint to a set of components consists in duplicating the constraint and distributing it to the components. However, for extensive constraints, the constraint must be divided among the components requiring this constraint. Once divided, different amounts can be assigned to different components, but the sum of each one is equal to the constraint before its division (cf. Figure 1).

The distribution of the extensive constraints depends on the operational specifications (e.g., reliability, energy efficiency). On the one hand, the architect could assign more resources to a component because of reliability. On the other hand, the architect could also assign less resources to this same component because of another operational specification. Thus, the constraints distribution is not unique.

Despite the decomposition of an application into components and the distribution of the constraints, some components may be considered as "constraint-less". The constraint-less components require a small processor time and few RAM to run and thus could run on every devices of the DH without specifying their requirements.

After the decomposition, we obtain a new set of possibles DPs. But this set might be very large, increasing the optimization duration. Reducing the number of entries is done through the creation of composite based on constraints type.

The components having the same intensive constraints can be grouped. The resulting composite holds the same constraint as each of its components. However, the components having the same extensive constraints can not be grouped because the resulting composite has a new constraint. For example, if two components hold respectively a constraint of `ROM = 15` and `ROM = 5`, grouping them create a composite with a constraint of `ROM = 20`. This composite is more difficult to deploy than both components taken separately.

Finally, the constraint-less components are always deployed on the same device as components having constraints. Thus, instead of considering them as a new type of constrained components and grouping them together, they can be grouped with constrained components.

Figure 2 gives an example of group that preserves the mobility of the components. Other groups are possible in the example. We just present one of them.

Distributing the constraints among the components of the application addresses the challenge $c1$. The proposed approach improves the mobility of the application in the DH. Moreover, the given composites enable to limit the set of entries considered by the optimization process.

4. ARCHITECTURE

To provide an autonomic and always energy efficient DH, the system has to consider the current DP and the dynamicity of the DH. The resulting system must also not consume more energy than the gain from any optimizations of the DP. Thus, our architecture focuses on three aspects, described in detail in this section: (a) consider the DH environment through its events and its current state; (b) provide an always up to date and energy efficient DP; (c) have a low impact on energy consumption of the DH. We do not discuss about failures in this paper.

To define a system able to continuously adapt at runtime to the DH environment, the proposed architecture is based on the MAPE-K loop [3]. This loop enables to adapt continuously the DP in an energy efficient way.

The MAPE-K loop, illustrated in Figure 3, is implemented using two types of entity:

- the Global Coordinator (GC) has an overview of the environment, i.e., devices, applications. It is in charge of finding the most energy efficient DP.
- the Local Manager (LM) has a knowledge about the device.

4.1 The Local Manager

The MAPE-K loop manages the distribution of the composites on the set of devices. To manage this distribution, we introduce the Local Manager. The LM performs the Sensor and Effector sub-entities: (a) it senses events about the device (arrival or departure) and about the hosted composites (start or stop) and sends corresponding data to the GC and (b) it executes actions ordered by the GC (`start`, `stop` and `migrate` composites) to reach the energy efficient DP. Each device is represented by a LM. Hence, the MAPE-K loop must consider multiple sensors and multiple effectors.

To provide the most energy efficient DP, the GC requires data to modify the plan consequently. We consider that the LM is the most relevant entity to send these data because it is tied to an unique device. It knows the description of the device such as the components it is executing. These data are sent when an event happens. Hence, the LM acts as a **sensor sub-entity**, sensitive to the events from the device and the components.

The LM integrates an **effector sub-entity**. The LM applies the actions ordered by the GC. It can execute three actions: `start`, `stop` and `migrate`. These actions are a com-

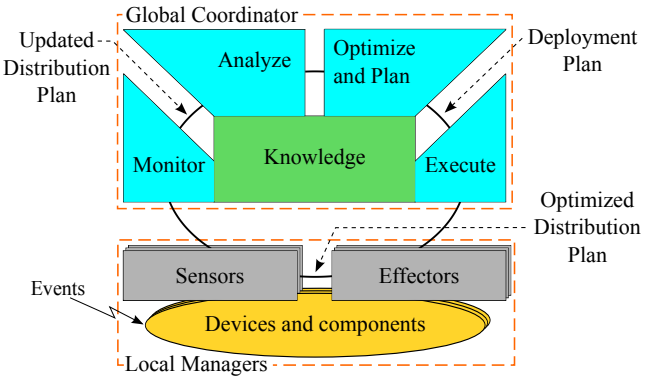


Figure 3: Distribution of the functionalities of the MAPE-K loop between the GC and the LMs.

position of deployment activities described in [9]: installation, activation, de-activation and de-installation. First, the `start` action installs a composite from the given URL and activates it. Second, the `stop` action allows the LM to de-activates a composite and then de-installs it. Finally, the `migrate` is a composition of the `stop` and `start` actions. The `migrate` action also enables the LM to wake up the remote device, assuming it can do it on its own.

From the adaptation point of view, i.e., adapting the set of composites on the set of devices, the LM performs only the Sensor and the Effector parts. However, the power management of the device is also handled by the LM. The LM knows if the device is inactive (no user, no component running). Thus, the LM has to decide if the device should be put into low power state, according to the operating system power management policies. The decision to suspend the device is taken when components are no longer running on the device.

4.2 The Global Coordinator

The system needs to be aware of each device in the environment. However, it is not possible if each of them is in low power state at the same time. Hence, one entity, named the Global Coordinator in the Figure 3, needs to be always active in the environment to be aware of events (e.g., device arrivals or departures, application started or stopped).

Our system is built in a centralized way. The DP is processed by an unique entity, to limit spending energy in the decision made in a consensus way. The GC is also designed as any other applications in the DH: it is a component-based distributed application. Thus, it also considers its composites during the optimization process and may be subject to migration, without loosing already collected data about the environment. In this solution, the GC is unavailable during its migration. Considered events could happen during this interval. However, they are not took in account and must be sent again once the GC migration is complete. As soon as the GC starts on a new device, it broadcasts its presence in the DH.

The GC is also the main part of the MAPE-K loop. It determines the most energy efficient DP by performing the following tasks: (1) monitor devices and composites events; (2) analyse the impact of the event on energy efficiency; (3) optimize DP; (4) plan a deployment plan; (5) send orders to LMs. On top of that, the GC must also keep data about the DH environment in the Knowledge sub-entity. Each sub-entity, detailed below, is defined into a composite.

To avoid optimizing continuously the DP, and thus consume energy, we assume that the DP is energy efficient until a new event happens. A new event is the appearance or disappearance of a device from the DH environment or the start or stop of an application. However, a device going into low power state is not an event because the device remains in the DH and can, at any moment, be woken up to host composites.

The **monitoring sub-entity** captures those events and updates the DP. The resulting *updated* DP may not be energy efficient instead of the previous DP, considered as *optimized*. Hence, the updated DP requires first to be analysed.

The **analysing sub-entity** calculates the power consumption of the updated DP. The calculation considers the power state of the devices in low power state and in active state. A device does not send its current energy consumption to the GC in real time. Instead, the energy consumption data is based on pre-determined values from the device, sent when the device appears in the DH. This value is a reference value to determine if a modification of the distribution plan is needed in the optimizer sub-entity.

The **optimizer sub-entity** considers the DP (cf. Section 2) and tries to reach the objective function, i.e., minimizing the energy consumption of the DH environment, considering the deployment constraints of the composites. This sub-entity implements the model described in [6].

The **planning sub-entity** creates a deployment plan which is the difference between the updated DP and the newly optimized DP. This plan details for each device the actions to apply, i.e., start a composite, stop a composite, migrate a composite to reach the optimized DP.

The **executing sub-entity** sends actions to the corresponding LM, once the deployment plan has been processed. These actions are performed by the effectors of the LMs.

The proposed architecture enables the system to be autonomic and energy efficient, answering to challenge $c2$. The system adapts to different events happening in the DH to find the most energy efficient DP, thus it handles the dynamicity of the DH.

5. VALIDATION

To validate our approach, we develop a prototype and discuss the impact of our choices over its implementation. We also provide its energetic impact on devices running it. Finally, we propose a use case to validate the overall approach.

The architecture[1] is developed using the iPOJO component framework [7]. This framework is also used to define the components of the mock applications. The distribution of the constraints and the grouping of components in a composite is carried out using the Felix OSGi framework [2]. Groups of components are embedded in a *bundle*.

The dynamicity is handled with Universal Plug and Play (UPnP) [3]. UPnP has been largely adopted by the consumers electronics manufacturers. It facilitates the discovery and the interaction between devices. Once a device appears or disappears from the DH, it broadcasts a message. The abil-

[1] `https://github.com/Orange-OpenSource/HomeNap`
[2] `http://felix.apache.org/`
[3] `http://4thline.org/projects/cling/`

Devices	Consumption (W)
Desktop computer (Dell Precision T3400) without screen	[80; 122]
Laptop 1 (Dell Latitude E6400)	[25; 48]
Laptop 2 (Dell Latitude D430)	[23; 33]
BeagleBoard with self-powered USB hub	[8; 10]

Table 1: Devices of the experiment. Power consumption of each devices is given in square brackets.

Devices	Duration (ms)	Power Mean (W)	Energy (Wh)
Desktop computer	700	109.00	0.0212
Laptop 1	1 239	43.53	0.0150
Laptop 2	2 373	26.73	0.0171
BeagleBoard	13 454	8.60	0.0323

Table 2: Optimization process consumption on various devices.

ity to send a message to the GC when an application is going to be started or stopped has been added.

Our prototype is deployed on x86 and ARM based home IT devices (cf. Table 1). The experiment describes a daily scenario in the DH and shows how our approach can reduce the energy consumption of the household. Measurements are made using a *PowerSpy* [4] connected to a computer, outside of the experiment. It measures the cumulated power value of the powerstrip where devices are plugged.

First, we measure how much energy is used to process a solution on every devices of the experiment (cf. Table 2). It shows that the Laptop 1 is the most suited device to perform this processing, because of a low energy consumption. Thus, the constraints on the composites of the GC should be chosen according to this result. Laptop 2 can also be considered as a good candidate to run the optimization.

To evaluate the overall approach, we define a scenario and play it. Figure 4 presents the global energy consumption of the scenario with and without our approach, respectively the green dotted line and the plain orange line. In both cases, the scenario is the same, except that the GC runs in the first case and not in the second case.

At the beginning, the set of devices includes the Laptop 2 and the Beagleboard: period A. They run a house surveillance application. The Beagleboard corresponds to a camera and runs an `Image Acquisition` composite. Laptop 2 holds an `Images Processing` composite, an `Alarm` composite and the entire GC composites.

During period B, a user wakes up the desktop computer. The LM is launched on this device during period C. The desktop computer appearance leads to an optimization. However nothing change because the desktop computer does not hold any components, and thus could be put into low power state if it stays inactive.

At the start of period D, the `GUI` composite is launched and must stay on the desktop computer, i.e., `Migrability = false`. This new composite leads to an optimization (peak

[4] `http://www.alciom.com/en/products/powerspy2.html`

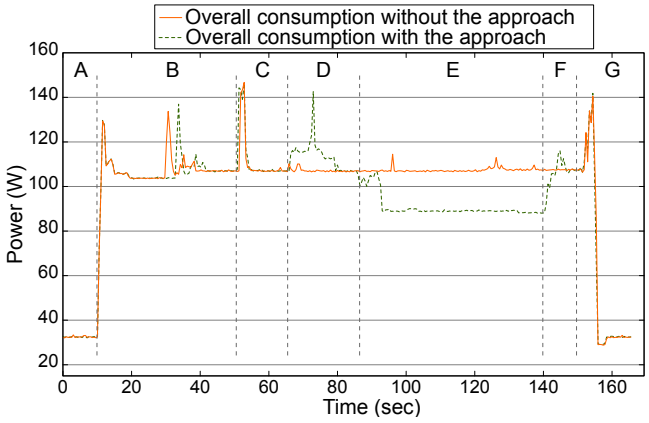


Figure 4: Comparison between the energy consumption with and without the solution.

in period D). Consequently, the GC migrates itself and both composites of the surveillance service from Laptop 2 to the desktop computer. In period E, the laptop goes to sleep because it does not hold any components.

`GUI` composite stops on period F. After Laptop 2 wakes up, the GC migrates itself and the both composites of the surveillance service to Laptop 2. Desktop computer goes into low power state during the period G.

The reduction of energy consumption is made during period E because Laptop 2 is stopped. Optimizations happen during periods C (1 sec), D (6 sec) and E(1 sec). Optimization consumption does not overcome the reduction of the energy consumption of the period E. This leads to a reduction of the energy consumption on the overall scenario. If the user had used the `GUI` composite for a longer time, the energetic gain would have increased.

This evaluation only considers a small set of devices and of composites. We believe that in a real DH, with more devices and applications, other and future usages can improve the reduction of energy consumption.

6. RELATED WORK

Finding a suitable distribution plan for constrained components may be a complex task. First because "suitable" is defined according to the operational specifications, e.g., performance, security, availability, energy efficiency. Secondly, because to process large sets, planners are usually based on Artificial Intelligent (AI) techniques: graphs colouring [10], evolution algorithms [1], constraint solvers [8, 5]. These planners consider the deployment of constrained components on various environments: network [8], datacenter [2], embedded systems [14] or even pervasive environments [12].

AI planners are usually implemented in a centralized way. The planner is deployed on a single device [8, 5, 1]. This device has to be always active to consider new events and process a new distribution plan consequently. The attachment of the planner to a single device makes this device consumes energy even when there is nothing to process.

An alternative processing of distribution plan is made in a consensus way [12]. Each device determines which components they can host considering their resources. However,

this approach requires for each device to be active at the same time, and thus consumes more energy.

In our approach, a DP is processed in a centralized way. But, the planner is not tied to a specific device. This separation limits its impact on the energy consumption of the DH. However, the duration of the optimization depends on the device hosting the planner.

Finally, our main operational specification is energy efficiency through the reduction of active devices. This approach has also been adopted in various environments. Especially, it has been successfully deployed in datacenters through the allocation of applications to the smaller set of servers [11, 2]. However, these works consider applications in a monolithic way, restricting further optimizations given with components-based distributed applications.

Minimizing active devices is also applied in office computing. Functionalities [15] or virtual desktop environments [4] can migrate to dedicated servers when the computers are not in use. During this idle phase, functionalities and network presence are performed by the dedicated servers. However, the use of dedicated servers does not take advantage of already running devices, restricting the energetic gain.

7. CONCLUSION

This paper focuses on an energy efficient distribution of component-based applications in the DH. Energy reduction consists in increasing unused devices and then putting them into low power state. Thanks to device heterogeneity, different distribution plans offer different energy consumptions. Then, energy optimization consists in choosing the most energy efficient distribution plan among the possible ones.

In response to challenge $c1$, the application decomposition increases the mobility of the application in the DH. This leads to a larger set of possible distribution plans for the same application set (component-based distribution plan). The paper also proposes a way to reduce the complexity of the optimization process by clustering components into composites without reducing the mobility of the components.

For the challenge $c2$, our architecture considers the dynamicity of the DH to adapt the distribution plan in an energy efficient way. The architecture is designed to minimize its energy consumption by moving the Global Coordinator, which implements the MAPE-K loop, to the device that is the most relevant to perform the optimization process.

A point of interest for future works concerns events. If two relevant events are close, reacting immediately to event occurrence is not energy efficient as the distribution evolution appears to be more costly than the expected energy gain. More annoying, this might induce double threshold pattern. To prevent this nasty effect, reaction to event occurrences could be delayed. Predicting relevant event occurrences by some self-learning "oracle" can also be considered in the future to prevent useless distribution plan adaptation.

Acknowledgement

This work is partially supported by the French FUI project EconHome and by the French Ministry of Higher Education and Research, Nord-Pas de Calais Regional Council and FEDER through the Contrat de Plan Etat Region Campus Intelligence Ambiante (CPER-CIA) 2007-2013.

8. REFERENCES

[1] A. Aleti, S. Bjornander, L. Grunske, and I. Meedeniya. Archeopterix: An extendable tool for architecture optimization of aadl models. In *Model-Based Methodologies for Pervasive and Embedded Software*, pages 61–71. IEEE, 2009.

[2] J. Anselmi, P. Cremonesi, and E. Amaldi. On the consolidation of data-centers with performance constraints. In *Conference on the Quality of Software Architectures: Architectures for Adaptive Software Systems*, pages 163–176. Springer-Verlag, 2009.

[3] A. Computing. An architectural blueprint for autonomic computing. *IBM White Paper*, 2003.

[4] T. Das, P. Padala, V. N Padmanabhan, R. Ramjee, and K.G. Shin. Litegreen: Saving energy in networked desktops using virtualization. In *USENIX annual technical conference*, pages 3–3, 2010.

[5] A. Dearle, G.N.C. Kirby, and A.J. McCarthy. A framework for constraint-based development and autonomic management of distributed applications. In *Autonomic Computing*, pages 300–301. IEEE, 2004.

[6] R. Druilhe, M. Anne, J. Pulou, L. Duchien, and L. Seinturier. Energy-driven Consolidation in Digital Home. In *Symposium on Applied Computing*, 2013.

[7] C. Escoffier, R.S. Hall, and P. Lalanda. ipojo: An extensible service-oriented component framework. In *Services Computing*, pages 474–481. IEEE, 2007.

[8] X. Fu, W. Shi, A. Akkerman, and V. Karamcheti. Cans: Composable, adaptive network services infrastructure. In *USENIX Symposium on Internet Technologies and Systems*, pages 135–146. Citeseer, 2001.

[9] R.S. Hall. *Agent-based software configuration and deployment*. PhD thesis, Citeseer, 1999.

[10] A.B. Hamida, F. Le Mouël, S. Frénot, M.B. Ahmed, et al. A graph-based approach for contextual service loading in pervasive environments. In *International Symposium on Distributed Objects and Applications*, volume 5331, pages 589–606, 2008.

[11] T. Heath, B. Diniz, E.V. Carrera, W. Meira Jr, and R. Bianchini. Energy conservation in heterogeneous server clusters. In *Symposium on Principles and practice of parallel programming*, pages 186–195. ACM, 2005.

[12] D. Hoareau and Y. Mahéo. Middleware support for the deployment of ubiquitous software components. *Personal and ubiquitous computing*, 12(2):167–178, 2008.

[13] M.G. Patterson. What is energy efficiency?: Concepts, indicators and methodological issues. *Energy policy*, 24(5):377–390, 1996.

[14] N. Shankaran, J. Balasubramanian, D. Schmidt, G. Biswas, P. Lardieri, E. Mulholland, and T. Damiano. A framework for (re) deploying components in distributed real-time and embedded systems. In *Proceedings of the 2006 ACM symposium on Applied computing*, pages 737–738. ACM, 2006.

[15] W. Zhong, G. Shi, Z. Zhao, and F. Xia. Parasite: A system for energy saving with performance improvement in networked desktops. In *Green Computing and Communications*, pages 79–84. IEEE, 2011.