RESOURCE SUBSTITUTION WITH COMPONENTS
Optimizing Energy Consumption

Christian Bunse and Hagen Höpfner
International University in Germany, School of IT, Campus 3, 76646 Bruchsal, Germany

Keywords: Resource Awareness, Software Engineering, Adaptability, Mobile Systems.

Abstract: Software development for mobile systems is becoming increasingly complex. Beneath enhanced functionality, resource scarcity of devices is a major reason. The relatively high energy requirements of such systems are a limiting factor due to reduced operating times. Reducing energy consumption of mobile devices in order to prolong their operation time has thus been an interesting research topic in past years. Interestingly the focus has mostly been on hardware optimization, energy profiles, or techniques such as "Micro-Energy Harvesting". Only recently, the impact of software on energy consumption by optimizing the use of resources has moved into the center of attention. Extensive wireless data transmissions, that are expensive, slow, and energy intensive can - for example - be reduced if mobile clients locally cache received data. Unfortunately, optimization at compile time is often inefficient since the optimal use of existing resources cannot really be foreseen. This paper discusses and applies novel strategies that allow systems to dynamically adapt at runtime. The focus is on resource substitution strategies that allow achieving a certain Quality-of-Service while sticking to a given energy limit.

1 INTRODUCTION

Mobile and small embedded systems usually have their own energy supply, an energy management system, internal logic, (wireless) communication interface, sensors and actors. Data is transmitted between single devices and/or servers by means of standardized communication protocols (e.g., ZigBee, Bluetooth, W-LAN, etc.). While such devices provide a growing range of functionality they are typically characterized by a scarcity of resources. The power/energy requirements of such systems are a limiting factor. To increase the quality and user acceptance it is therefore necessary to optimize the energy consumption for provide improved uptimes.

Research in the area of mobile phones and ad-hoc sensor networks has shown that especially communication (i.e., (wirelessly) transmitting data) is one of the largest cost factors (Domis 2006), (Stark et al 2002) (Zhang et al 2005). This makes communication an ideal candidate for optimizing a system's uptime. Energy optimization has to be aware of the tradeoffs between performance, energy consumption, and quality-of-service (QoS). In contrast to hardware optimizations, software systems are usually optimized at development time by specifying their energy characteristics and by adapting the implementation. However, this requires individual adaptations of each system variant, and often implies a negative impact on the performance or QoS of such systems. The challenge is to explore the relations among the various components and to understand the tradeoffs between performance, QoS and energy consumption (Zhang et al 2005). This enables the development of systems that achieve an optimal balance between performance, QoS, and energy consumption by adapting themselves at runtime (i.e., dynamic optimization).

Component-Based Software Development (CBSD) (Szyperski 2002) aims at reducing the complexity of software development and improving a system's maintenance by increasing reuse and independence. CBSE allows focusing on changes at the component level instead of changes at the implementation level. Adaptability is managed in terms of the creation/destruction of component instances and their links (i.e., dynamic reconfiguration or structural dynamism). But, runtime component replacement is not sufficient since the current state of a component is lost. Thus, self-adaptability of components is needed allowing updates at runtime while preserving the state.

Bunse C. and Höpfner H. (2008).
RESOURCE SUBSTITUTION WITH COMPONENTS - Optimizing Energy Consumption.
In *Proceedings of the Third International Conference on Software and Data Technologies - SE/GSDCA/MUSE*, pages 28-35
DOI: 10.5220/0001879000280035

This paper describes the component-based development of low-energy systems, based on the MARMOT/KobrA approach (Atkinson et al 2001). The central idea is the development of an energy-management component (EMC) that serves as a communication channel and makes use of resource substitution strategies (Höpfner & Bunse 2007) to improve energy consumption. The selection of strategies is based on an energy cost model. Typical strategies in this regard are caching, replication or hoarding. In addition, the paper reports on first findings obtained in a small series of case studies. Results show that systems that make use of wireless communication can profit from using the EMC. In contrast, systems that make use of cable-based communication do not show such significant improvements. This supports the assumption that wireless communication is a dominant factor concerning energy consumption.

The remainder of this paper is organized as follows. Section 2 provides an overview on resource substitution strategies in mobile information systems. Section 3 introduces the MARMOT approach for the component-based development of such systems, while Section 4 discusses static and dynamic optimization approaches and provides some insights to the EMC. Section 5 presents preliminary evaluation results and Section 6 provides a short summary and some conclusions.

2 RESOURCE SUBSTITIUTION IN MOBILE INFORMATION SYSTEMS

Mobile information systems are aimed at providing important data in real time at almost every place and anytime to assist decision makers (Gurun & Krintz 2003). In most cases mobile clients, like Java enabled mobile phones, connect to static and powerful information system servers. Unfortunately, lightweight mobile clients suffer from various limitations (reduced bandwidth, weak CPU, less memory, limited energy supply, etc.) that can be ease by a proper substitution strategy. We researched the possibilities and benefits of resource substitution for this application scenario.

The following subsections briefly describe the relevant resources and how they can be substituted by each other (see Höpfner & Bunse 2007) for more details).

2.1 Resources and their Usage

The client/server scenario used in our work implies that (1) mobile clients query data from a server, (2) compute and display them, and (3) synchronize changes with the server. Hence, *data communication* is extensively used in the first and the third phase. Another resource affected in both steps is the *energy supply*. Wireless data communication is energy intensive. Standard techniques for handling this problem include replication, hoarding, and caching. Replication stored explicitly defined data parts on the mobile device, caches keep queried data implicitly, and hoarding automatically extends explicitly or implicitly received data by additional data that might be useful for answering upcoming requests without communications. Hence, reducing data communication is done by storing data on the mobile client by using the *memory* resource. The *CPU* resource is not only used in the second phase for the "normal" computation and visualisation of the data but also necessary for supporting the local data management. If, for example, data is cached in a semantic manner (Ren & Dunham 2003) query processing becomes more *CPU* intensive since queries must be rewritten regarding the locally available data. In fact, some of these algorithms are known to be NP-hard. However, studies (Marwedel 2007) show that even complex calculations consume less energy than memory accesses.

2.2 Substitutability

We discussed the most relevant resources and how they are generally used in mobile information systems in Subsection 2.1 and also mentioned that extensive communications might be substituted by memory usage. However, there are more possible substitutes, such as:

- **CPU vs. Communication**
  In order to reduce CPU usage one can delegate complex calculations/conversions to a server or use web services (Ion, Caracaş & Höpfner 2007). So, communication is a substitute for CPU usage. On the other hand, reducing communication means either transmitting compressed data, to specify requested data more precisely or to maintain local copies. Hence, communication might be substituted by CPU usage.
- **CPU vs. Memory**
  If certain computations are frequent the results might be stored. In the database world this is

called view materialization. In other words, CPU usage is reduced by storing additional data; memory substitutes CPU usage. Saving memory can be done by compressing and decompressing data.

- **Memory vs. Communication**
  Caching, hoarding, or replications are well known techniques of a resource substitution. All create redundant data on the mobile device, and this data have to be stored. In order to substitute memory by communication one can think about storing all information the server only as it is supported by IMAP.
- **Energy vs. CPU, Memory, Communication**
  The resource energy interacts with all other resources. However, the intensity of energy consumptions is different for each other resource. Energy consumption associated with wireless data transmission is not negligible (Feeney & Nilsson 2001). Storing data locally requires less energy than wireless transmissions depending on data-size, storage time, and other factors. CPU usage needs comparatively less energy than memory storage (Marwedel 2007).

In order to implement software for mobile devices that fits the user's needs one has to consider the resources provided by the mobile device and the environmental infrastructure. So, the discussed substitutions are only some alternatives beside others. However, the examples illustrate the dynamics an implementation must support.

3 COMPONENT ENGINEERING

Reuse is a key success factor in industry today, and it can be seen as a major driving force in hardware and software development. Reuse is pushed forward mainly by the growing complexity of systems. This section introduces the MARMOT method (Bunse, 2006) that facilitates reuse in embedded systems development. MARMOT is an extension to the KobrA method (Atkinson 2001), a component-based development framework for information systems. MARMOT adds diagram types (UML and electronics related), model relationships, verification mechanisms, and transformation rules to KobrA that are specifically adapted to embedded systems.

Composition is a key activity in component-based development with MARMOT. A system is viewed as a hierarchy of components, in which the parent/child relation-ship represents composition, i.e., a super-ordinate component is composed out of its contained sub-ordinate components. Another established principle is the separation of interface and implementation that supports independent component development, and allows versions of a component to be exchanged. Following those principles, each component can be described through a suite of models (e.g., UML diagrams) as if it was a system in its own right (see Figure 1).

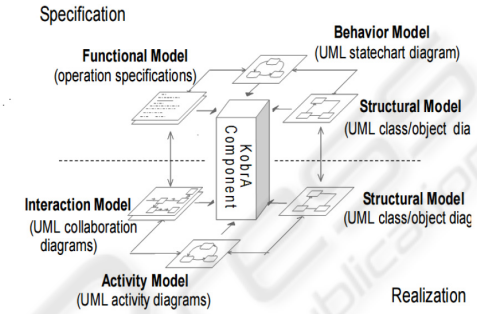

Figure 1: MARMOT Component Model.

3.1 MARMOT Process Model

The core principle of MARMOT is separation of concerns, with two basic development dimensions (Composition/Decomposition and Abstraction Concretization) that map to four basic activities (Decomposition, Embodiment, Composition, and Validation). These are depicted in Figure 2: 2 and described in the following list:

- *Decomposition*. A development project starts above the top left-hand side box in Figure 2: 2. It represents the entire system to be built. Before the specification of the box, the concepts of the domain in which the system is supposed to operate must be determined comprising descriptions of all entities relevant in the domain such as standard hardware components that will appear on the right-hand side towards concretization. These implementation specific entities determine the way in which a system is divided into smaller parts. Decomposition determines the shapes of identified individual components in an abstract and logical way.
- *Embodiment*. The systems, or its parts, are moved towards more concrete representations, mostly through reusing existing components (or custom development). There is no distinction between hard- and software components at this early phase since all components are treated in form of abstract models.

- *Composition*. The reused and implemented components are assembled according to the abstract model, and the subordinate boxes have to be coordinated with their respective super-ordinate boxes according to the MARMOT component model.
- Validation assesses to which extent the concrete composition of the embedded system corresponds to its abstract description.

3.2 MARMOT Properties

MARMOT follows the principles of encapsulation, modularity and unique identity, which lead to a number of obligatory properties:

- *Composability* is the primary property and it can be applied recursively: components make up components, which make up components, etc.
- *Reusability* is the second key property, separated into development for reuse, i.e., components are specified to be reusable, and development with reuse, dealing with the integration and adaptation of existing components in a new application.
- Having *unique identities* requires that a component may be uniquely identifiable within its development and runtime environment.
- *Modularity/encapsulation* refer to a component's scoping property as an assembly of services (also true for a hardware component), and as an assembly of common data (true for hardware and software components). The software represents an abstraction of the hardware.
- An additional important property is *communication* via interface contracts that becomes feasible in the embedded world through typical software abstractions. Here, additional hardware wrappers guarantee that the hardware communication protocol is translated into a component communication contract.

4 OPTIMIZATION

Research in the area of mobile- and small embedded systems (e.g., ad-hoc sensor networks) have shown that especially communication (i.e., wirelessly transmitting data) is one of the largest cost factors (Stark 2002, Zhang 2005). Therefore, we limit the focus of our approach towards optimizing a systems communication effort with respect to energy. MARMOT supports the optimization of energy consumption in two different ways: (1) Statically at development time (preferred means for optimizing "isolated" systems), and (2) dynamically at runtime. Static optimization has already been evaluated by (Domis, 2006). Therefore, the focus of this paper is on the dynamic optimization aspects.

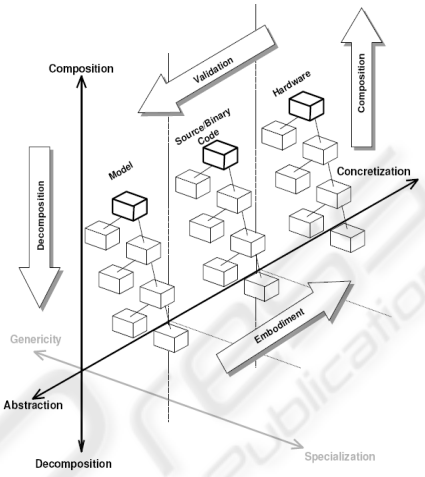

Figure 2: Development Dimensions.

4.1 Static

Static energy optimization in MARMOT is based on (Nieberg 2003), who follows a state-based approach by assigning a set S of all states $s_1,..., s_k$ with ($k \in N$ and $1 \leq k$) to all hardware component. This set is used to create a state transition matrix ST, whereby elements st_{ij} of ST specify the number of state transitions from state s_i to s_j ($i, j \in N$, $1 \leq i \leq k$ and $1 \leq j \leq k$). In addition each state s_i of S is combined with the power consumption P_i in this state, and each state transition st_{ij} becomes assigned to the energy consumption E_{ij} for the transition. By knowing the time t_i, in which the component is in state s_i, the energy consumption can be calculated by:

$$E = \sum_{j=1}^{k} t_j P_j + \sum_{\substack{i,j=1 \\ i \neq j}}^{k} st_{ij} E_{ij}$$

Within MARMOT this approach is used to model the energy related properties of a component (i.e., hardware node with embedded software components). Therefore, the component model (see Figure 1) was adapted at the specification- and at the realization level (see Figure 3:).

At the specification level (i.e., the component interface) the externally visible behavior of a component is already modeled by an abstract state machine. Following (Nieberg 2003) an additional power state machine is created that specifies the

power consumption for single states and the duration of transitions. It thus, depicts the externally visible energy consumption of a component. In addition the functional model, specifying the external visible operations of a component, is enhanced by an energy consumption attribute to specify the consumption of those operations that are not already covered by the power state machine.

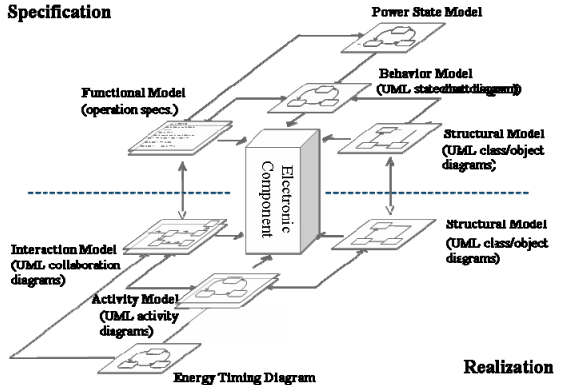

Figure 3: Energy Component Model (Domis 2006).

At the realization level a UML timing diagram is created for every operation, modeling the operation's control flow in relation to time. This allows (in cooperation with the specification models) to analyze the energy consumption of an operation and to manually optimize operations & control with a focus on communication routines (e.g., shortened communication cycles and messages, using sleep modes, reducing wake-up cycles, etc.).

4.2 Dynamic

In mobile systems, devices, services, environmental objects, and requirements are subject to change. Therefore, static optimization might not always be a suitable solution since the optimization of one property might have a negative impact onto other properties (e.g., improving energy consumption via communication optimization might deteriorate the system's quality-of-service). To avoid this, a system should be able to adapt its operation to the current requirements and the available resources.

Optimally, run-time adaptation requires that single components are analyzed at run-time (e.g., concerning their energy consumption) and are then dynamically exchanged. However, this would not only require implementation support (i.e., dynamic code exchange is not supported by Java), but also the provision of different component variants. When focussing on optimizing communication to reduce energy consumption, another solution is the development of a component (-wrapper) that is plugged into a system and that acts as a facilitator between the software and the hardware communication facilities (i.e., hardware wrapper).

Such a component (-wrapper) can be developed or specified using the MARMOT approach and then be "easily" integrated into existing systems. This is supported by MARMOT's inbuilt component reuse mechanisms that make use of conformance maps and wrappers (Atkinson 2001). The component then analyzes the communication requests to or from the system with the help of an internal (energy) cost model and then selects the "optimal" (substitution) concerning energy consumption.

The central element of such a component is the cost model. Within in MARMOT such a model can be developed by following the static optimization approach (i.e., specifying energy properties of a (hardware) component) as discussed in Section 4.1 whereby the model must be platform-specific. By having a platform specific power state machine as well as the energy-related costs for specific operations (functional and timing model) it is "easy" to calculate the differences between alternatives.

Examples of substitution strategies are on a lower level the accumulation of data, the decision to let the server perform specific operations, etc. On a more advanced level caching (i.e., the operation of storing information in the user's device after it has been sent from the server) and hoarding (i.e., the process of predicting the information that the user will request, for transferring it in advance to the client cache) might be used. As discussed in Section 2, caching and hoarding seem to be beneficial for systems with high communication loads (e.g., mobile information systems).

One obvious disadvantage of the substitution approach is that functionality has to exist at both the client and the server side (i.e., duplication). Therefore, substitution strategies, although in principle generally applicable, have to be specialized (refined) for a specific system. In addition, more memory space is required. However, first results (see Section 5) are quite promising.

4.3 The EMC

To dynamically adapt the energy consumption of mobile and embedded systems we developed a component, known as EMC (Energy Management Component), that manages the communication to/from the system and that therefore optimizes the

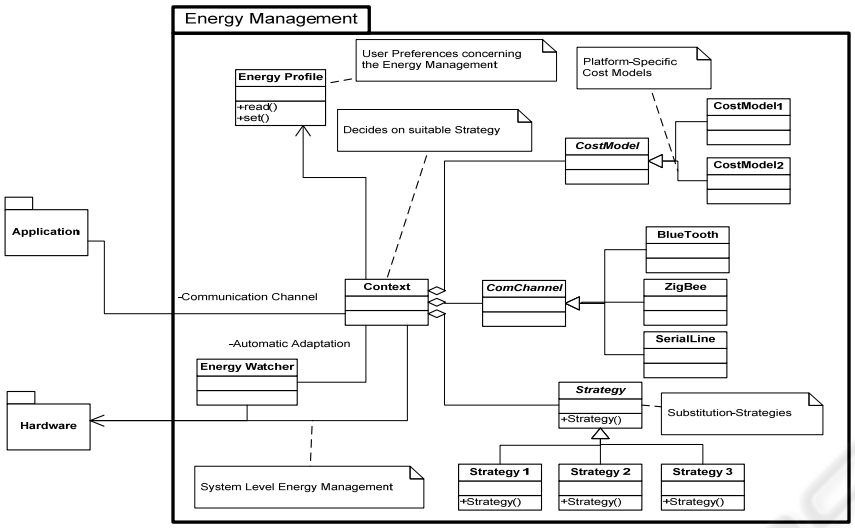

Figure 4: Architecture of the Energy Management Component.

system's energy consumption. In general, the EMC was developed by applying the MARMOT approach. On the one hand this eases reusing the EMC within different projects and also allowed to statically optimize the EMC regarding its own energy consumption. Technically, the EMC implements/realizes the resource substitution strategies outlined in section 4.2).

At its core the EMC defines a family of strategies, encapsulate each of these, and makes them interchangeable. This is nicely supported by the strategy pattern defined by (Gamma 1995). The strategy pattern supports the development of software systems as a loosely coupled collection of inter-changeable parts. The pattern decouples a strategy (algorithm) from its host, and encapsulates it into a separate class. It thus, supports the separation of an object and its behaviour by organizing them into two different classes. This allows switching to the strategy that is needed at a specific time.

There are several advantages of applying the strategy pattern for an adaptable software system. First, since the system has to choose the "best" substitution strategy it is simpler to keep track of them by implementing each strategy by a separate class instead of 'burying' it in the body of a method. Having separate classes allows simply adding, removing, or changing any of the strategies. Second, the use of the strategy pattern also provides an alternative to sub-classing an object. When sub-classing an object to change its behaviour, that behaviour is static. Changes therefore require the creation/instantiation of a different subclass and replacing that object with it. The strategy pattern allows switching the object's strategy, and it will immediately change how it behaves. Third, using the strategy pattern also eliminates the need for various conditional statements. When different strategies are implemented within one class, it is difficult to choose among them without resorting according to the conditional statements. The strategy pattern improves this situation since strategies are encapsulated as an object that is interchangeable at runtime. In the context of this paper we specified an Energy Management component (EMC) that makes use of strategy pattern variant, and that uses resource substitution strategies for optimizing energy consumption (see Figure 4:).

To select the most appropriate strategy the component has to 'know' the cost of operations. Therefore, a cost model is needed (see Section 4.2). Since these costs vary between target platforms the cost model entity provides a generic interface that has to be specialized prior to deployment.

Selecting the 'best' strategy concerning energy consumption might have a negative impact on other quality attributes (e.g., performance, memory, etc.). Thus, the best possible optimization might not be desirable from the user's point of view. The decision on suitable strategies cannot be made by the component on its own. Thus, the component has to be aware of user preferences. The EM component therefore provides a profile mechanism that allows users to specify their requirements concerning energy saving, performance, and quality-of-service. Furthermore, it allows to specify how the system should behave if the energy level gets low.

In addition to user profiling the EM component needs an interface to the underlying platform or hardware layer to monitor the actual battery status. In order to make the component deployable across different platforms the component uses a generic interface that is specialized according to the selected platform. In addition to monitoring battery levels this interface is also used to directly access hardware implemented energy saving procedures (e.g., reducing the transmitting power, graceful degradation, or dynamic voltage scaling).

Since the EMC itself was developed with MARMOT two facts have to be noted. First, due to the method's inbuilt reuse mechanisms the EMC can quickly and efficiently be adapted and reused within other projects systems (this is supported by the low reuse effort as reported in section 5.1.) Second, the EMC itself was optimized, regarding its energy consumption, by using the static optimization mechanisms outlined in section 4.1. The reason was to avoid that achieved savings are wasted by the EMC's own energy consumption. The results reported in section 5.1 support this assumption.

5 EVALUATION

Optimizing the communication behaviour is a promising means for reducing the energy consumption of mobile and embedded systems. The EMC component is based on the idea of resource substitution and can be plugged into other systems that have been developed using the MARMOT approach. In the following preliminary results of three, currently running, case studies are discussed.

Case studies were selected according to the used communication technology (e.g., ZigBee, Bluetooth and Cable-based). In addition, all case studies are micro-controller based. Systems that use an operating system (OS) do not allow to clearly investigating the energy relevant behaviour since it is not possible to differentiate between system and OS activities when measuring energy consumption.

All measurements are performed and recorded automatically using a digital oscilloscope that has a PC-communication module. During measurement three different versions of a system have to undergo the same, pre-specified action sequence that is repeated until the battery level reached a critical value. Measurements focus on energy consumption. In order to obtain valid measurement results the case studies are performed several times (multiple battery cycles), resulting in an overall measurement time of several days. Measurements currently continue in order to further confirm the initial findings.

5.1 Case Studies

As a first step towards evaluating the effected of using the EMC three case-studies were defined:

- A wireless sensor network that consists out of a controller, router- and end-nodes. Each node has a set of sensors and communicates with the other nodes using the ZigBee protocol. The system supports the surveillance of animals (i.e., cattle). Nodes integrated in a collar, measure life data (temperature, pulse …) and provide these together with localization information to the controller. The controller can then monitor movement, identify escaped, ill, or dead animals, can issue alarm calls, and create movement profiles (optimal land utilization).
- A fork-lifter system for handling transport within a warehouse. The system consists of a vehicle and a central control unit (i.e., a PC). The fork lifter is able to independently navigate through the warehouse and solve transportation tasks. The central control unit communicates with the vehicle via Bluetooth, provides new tasks, monitors its actual state and position, provides roadmaps and routes, and offers a manual control of all vehicle functions.
- A mirror control system, part of a larger door control unit, allows movement of a car mirror horizontally and vertically into the desired position whereby movement is indicated on a small LCD panel. Cars supporting driver profiles can store the mirror position and recall as soon as the profile is activated. Furthermore, the system provides a defreeze/defog function with a humidity sensor and a heater.

5.2 Initial Results and Discussion

Initial measurement within the case-studies shows that the EMC-related optimization results are close to those that can be obtained by static optimization. However, the perceived QoS is higher for dynamic optimization and the effort for integrating the EMC into a system is significantly lower than the required effort for static optimization. On average the adaptation and integration of the EMC required 10h. One reason for this is the use of MARMOT and its inbuilt reuse mechanisms. Interestingly, static optimization required nearly double this effort.

One finding is that using the EMC seems to be more effective when using the EMC within systems that make use of wireless communication (e.g., ZigBee/Bluetooth). Here the uptime was extended from ~13 to ~110 hours. Systems that make use of cable-based communication (e.g., UART) showed less significant results (i.e., uptime extension from 10 to 13 hours). This, indirectly, confirms that wireless communication is one of the largest cost factors regarding energy consumption.

Results reveal that the curves, formed by measurement-data, show some of the typical characteristics of battery-powered systems. If the energy consumption passes a maximum value they seem to suddenly lose their capacity, while, after phases with minimum energy consumption that follow short high-energy consumption phases they seem to re-charge themselves. This effect is known as Recovery Effect and depends on the charge history, age, and environment temperature of the battery. In addition, batteries are known to discharge themselves based on the environmental temperature and the actual use. Although, these effects make measurement and forecasts quite difficult, they did not have an impact onto the results of the case studies since every run used fresh batteries and followed the same scenario.

6 SUMMARY & CONCLUSIONS

Given the rising importance of mobile and small embedded devices, energy consumption becomes increasingly important. Currents estimates by EUROSTATS predict that in 2020 10-35 percent (depending on which devices are taken into account) of the global energy consumption is consumed by computers and that this value will likely rise. Therefore, means have to be found to save energy.

The focus of this paper is on resource substitution as a means for energy saving. Based on general substitution strategies (Höpfner & Bunse, 2007) we presented a general energy management component (EMC) that can be plugged into component based systems developed with MARMOT. The component acts as a mediator between system and communication facilities. All communication requests are analyzed concerning energy related cost and substitution strategies are used for optimization. Preliminary case-study results indicate that wireless communication is the major cost factor concerning energy consumption and that by using the EMC it is possible to significantly extend the uptime and to decrease the energy consumption of mobile & small embedded systems.

Our initial results are based on micro-controller systems. To systematically evaluate the effects of strategies such as caching or hoarding we currently prepare a case study for mobile information systems running on a PDA or Smartphone.

REFERENCES

Atkinson, C., Bayer, J., Bunse, C., et al (2001). *Component-Based Product-Line Engineering with UML*. Addison-Wesley, UK.

Bunse, C. (2006) *Developing µController-Systems with UML*. A MARMOT Case Study‘. Technical Report 111.06/E, Fraunhofer IESE, Germany.

Domis, D.J. (2006), *Component-based Energy-Modeling for Ambient Intelligence Systems*, in German. Master Thesis, Technical University Kaiserslautern.

Gurun, S. & Krintz, C. (2003), *Addressing the energy crisis in mobile computing with developing power aware software*, Technical Report 2003-15, University of California, Santa Barbara.

Feeney, L. M. & Nilsson, M. (2001), *Investigating the energy consumption of a wireless network interface in an ad hoc networking environment*, IEEE Conference on Computer Communications, Anchorage, USA.

Gamma, E., Helm, R., Johnson, R.E. (1995), *Design Patterns. Elements of Reusable Object-Oriented Software*, Addison-Wesley Longman.

Höpfner, H. & Bunse, C. (2007). *Resource Substitution for the Realization of Mobile Information Systems*. 2nd International Conference on Software and Data Technologies, Barcelona, Spain. pp. 283-289.

Ion, I., Caracaş, A. & Höpfner, H. (2007). *MTrainSchedule: Combining Web Services and Data Caching on Mobile Devices*, Datenbank-Spektrum 5(21), 51-53.

Marwedel, P. (2007), *Embedded System Design*, Springer.

Nieberg, T., Dulman, S., Havinga, P., van Hoesel, L., Wu, J. (2003), *Ambient Intelligence. Impact on Embedded System Design Ambient Intelligence: Collaborative Algorithms for Communication in Wireless Sensor Networks*. Kluwer.

Ren, Q. & Dunham, M. H. (2003), *Semantic caching and query processing*, Transactions on Knowledge and Data Engineering 15(1), 192–210.

Stark, W., Wang, H., Worthen, A. Lafortune, S. Teneketzis, D. (2002), *Low-energy wireless communication network design*. IEEE Wireless Communications, Vol 9(4), pp. 60-72.

Szyperski, C. (2002), *Component Software. Beyond Object-Oriented Programming*, Addison-Wesley.

Zhang, Y., Teng, X., Yu, H., Hu, H. (2005), *The Energy Cost Model of Clustering Wireless Sensor Network Architecture*. In Wu, Chen, Guo, Bu (Eds.): Embedded Software and Systems. Springer Verlag, pp 374-380.