Energy-Conscious Deployment Optimization for Component-Based Cyber-Foraging Systems

Proposal for Master’s Thesis of

Sebastian Dieter Krach

At the Department of Informatics
Institute for Program Structures and Data Organization (IPD)
and
Carnegie Mellon Software Engineering Institute (SEI)

Reviewer: Prof. Dr. Ralf H. Reussner
Second reviewer: Jun.-Prof. Dr.-Ing. Anne Koziolek
Advisor: Grace Lewis (SEI)
Second advisor: Christian Stier (KIT)
Third advisor: Dr.-Ing. Klaus Krogmann (KIT)

Contents

1 Introduction ........................................... 1
   1.1 Goal of the Proposed Thesis ................. 1
   1.2 Structure of this Proposal .............. 2

2 Foundations ......................................... 3
   2.1 Cloud-Based Mobile Augmentation .......... 3
       2.1.1 Cloud Offloading .................. 3
       2.1.2 Cyber-Foraging ................. 4
   2.2 The Palladio Approach .................... 4
       2.2.1 The Palladio Component Model ....... 5
       2.2.2 Resource Consumption in the PCM ... 6
       2.2.3 Simulation, Analysis and Prediction . 6
       2.2.4 Automatic Optimization of Software Architectures . 7
   2.3 Energy Consumption and Power Models .... 7
       2.3.1 Energy Consumption Models for Mobile Devices ... 7
       2.3.2 Benchmarking and Profiling .......... 9

3 Related Work ....................................... 11
   3.1 Energy Consumption of Mobile Devices .... 11
   3.2 Partitioning of Cyber-Foraging Applications . . 12

4 Conception and Implementation ................... 15
   4.1 Optimizing Deployment Allocation ....... 15
   4.2 Modeling Device Characteristics .......... 16
   4.3 Extending PerOpteryx .................... 17
   4.4 Modeling Context-Dependent Behavior .... 17
   4.5 Validation ................................ 17

5 Organizational Matters ............................. 19
   5.1 Advisement ................................ 19
   5.2 Artifacts .................................. 19
   5.3 Environment ................................ 20

6 Time Scheduling .................................... 21
   6.1 Work Packages ................................ 21
   6.2 Risk Management ........................... 22
   6.3 Detailed Scheduling ...................... 23

Bibliography ............................................ 27
1. Introduction

While CPU performance of modern handheld devices has continued to grow significantly, an increase of the battery capacity in the same order of magnitude is undiscernible. Cloud-based mobile augmentation describes a concept, where cloud computing infrastructure is leveraged to "increase, enhance and optimize computing capabilities" [ASA+14]. Apple’s Siri [App14] is one of the most commonly used examples for cloud-offloading applications. The phone records your voice and transmits the record to a more powerful cloud server, which in turn executes language processing. Recent work [CBC+10, CIM+11, KG08, KL10, HLSS11] analyzes offloading of resource-intensive tasks into the cloud in order to increase battery run-time.

Satyanarayanan et al. [SBCD09] propose cyber-foraging as trade-off between increased latency and dependence on the stability of the internet connection on the one hand, and increased performance through more sophisticated hardware on the other. The authors introduce surrogate hosts or cloudlets, micro data centers located in one-hop proximity to the mobile device. A significant increase in performance and decrease in latency can be achieved through offloading application components to cloudlet instead of a distant cloud. In particular for crisis situations poor connectivity and limited resource availability present serious obstacles, e.g. for first responders. Cloudlet-based cyber-foraging can increase the efficiency of tactical edge applications, e.g. face recognition, helping to overcome the aforementioned obstacles [LSN+13].

Application execution efficiency on mobile devices can benefit from cyber-foraging in various ways: increased performance through more sophisticated CPUs, decreased dependence on large-scale mobile networks through relying on local one-hop connections, and increased battery run-time through offloading of computations with high resource demand. Nevertheless, offloading application components from mobile device to a cloudlet does not necessarily lead to the desired decrease in energy consumption. Generally speaking, the execution of a component on a cloudlet can be advantageous if the energy that would be consumed by local execution exceeds the energy necessary for data transmission and waiting for the cloudlet to finish processing.

1.1 Goal of the Proposed Thesis

The hardware and software platforms of mobile devices and cloudlets usually differ significantly. While cloudlets mostly employ powerful hardware built for desktop or server use,
mobile devices rely on energy-efficient variants. Although the platforms are usually incompatible, there exist frameworks which support the offloading of application logic to the cloud(let). CloudClone [CIM+11] and MAUI [CBC+10] are two representatives which rely on application virtualization (DalvikVM and .Net-CLR, respectively). Nevertheless, both approaches are dependent on the underlying virtualization platform. Furthermore, using platform-specific services often allows a more efficient implementation of the application logic.

The goal of the proposed master’s thesis is to enable application developers to determine which components are suitable to offloading, already during design time. Using a model-based simulation we want to derive the optimal allocation of components with regard to the energy consumption. With PerOpteryx there exists already an approach which determines configurations for a Palladio Component Model (PCM) instance which are Pareto-optimal with respect to certain quality criteria. Currently, PerOpteryx does not support energy consumption as a quality measure.

The main contribution of the proposed thesis is an automatic partitioning optimization for component-based applications. The optimization is determined depending on a model-based simulation of the application’s resource consumption using energy consumption characteristics for the simulated mobile device.

1.2 Structure of this Proposal

This proposal is structured as follows. Chapter 2 will provide the foundations which the thesis will build on. In Chapter 3 related work will be presented and distinguished from the proposed approach. Chapter 4 will describe the concepts we propose to achieve the envisioned goal. Organizational matters are explained in chapter 5. A more detailed overview of the work packages, a risk analysis and detailed time planning is presented in chapter 6.
2. Foundations

This chapter provides an overview of scientific work which form the basis for the proposed master’s thesis.

2.1 Cloud-Based Mobile Augmentation

Mobile devices present significant restrictions, particularly with respect to battery capacity. Cloud-based mobile augmentation schemes help to overcome mobile device resource restrictions [BBV09]. Mobile applications can leverage more powerful infrastructures accessible using a (wireless) network connection.

2.1.1 Cloud Offloading

Offloading of application parts is a very common augmentation scheme. It refers to the execution of certain application parts on a server instead of the mobile device. The reduced computational work on the mobile device is payed for by increased network transmissions and additional coordination effort. Kumar et al. [KL10] identified computation and communication costs as the two determining factors for offloading decisions. Furthermore, different restrictions for offloading schemes exist, e.g. application parts which are dependent on mobile-device specific hardware (e.g. camera, sensors) can only be executed locally.

Application partitioning or offloading mechanisms can be categorized as either static or dynamic. Static approaches answer the question which application parts are supposed to be offloaded before the application is executed and once for the entire run-time. Dynamic approaches rely on run-time profiling techniques to determine the optimal partitioning while the application is executing. While dynamic approaches are able to react on changes to the environment (e.g. network congestions), they increase the application complexity and often limit application developers design possibilities.

Offloading mechanisms differ with respect to their granularity. Lewis et al. [LLP14] identify five levels of granularity: Process, Function, Component, Service and Application.

**Application Level** Entire application logic is offloaded. The mobile device client is kept as small as possible.

**Service Level** Offloading of self-contained artefacts, accessible through specified interfaces, which encapsulate coarse-grained application capabilities.
Component Level Software artefacts, mostly determined to execute in specific container, are offloaded. The artefacts are most-likely self-contained but often show tight dependencies to the execution platform on the mobile device.

Function Level Functions, methods or operations are offloaded. Global states have to be synchronized. Offloading candidates are often manually annotated.

Process Level A clone of the mobile device runs virtualized on the offloading target. Process states are exchanged. Process level granularity allows the most fine-grained offloading schemes.

The proposed thesis will focus component-based software architectures. A more restrictive definition of components will be applied in comparison with aforementioned component-level offloading granularity (see Section 2.2 for details).

2.1.2 Cyber-Foraging

Cloud-based mobile augmentation schemes can significantly increase mobile application capabilities. However, offloading of application parts to a distant cloud infrastructure depends on a good network quality. The necessary availability and accessibility of cloud resources can not be guaranteed in certain hostile environments. Ha et al. [HLSS11] define these hostile environments as dominantly characterized by a "short-term large-magnitude uncertainty". Therefore, mobile application design for personnel operating at the tactical edge, e.g. first responders in crisis areas, is constrained by environmental characteristics. The characteristics include resource-restrictions (e.g. limited battery capacity), unreliable connection to traditional infrastructure, potentially large amounts of data and unpredictable environmental changes. Furthermore, the personnel has to operate under stress and high cognitive loads [LSN13].

Ha et al. [HLSS11] employ cyber-foraging mechanisms to improve mobile device usage in hostile environments. Cyber-foraging refers to enhancing performance of mobile clients through the usage of opportunistically discovered servers in the environment [BFS02]. Ha et al. propose the introduction of cloudlets as surrogate hosts in single-hop proximity of the mobile device. Cloudlets are stateless servers located in direct proximity of mobile devices which are discovered and afterward initialized dynamically upon usage [SBCD09]. Simanta et al. [SLM12] present a reference architecture (see Figure 2.1) for hostile environments leveraging the concept of Ha et al. [HLSS11].

Simanta et al. [SLM12] use VM synthesis as core technique to provision discovered cloudlets fast. VM synthesis requires a base VM image containing the underlying operating system to be deployed to all potential cloudlets. After discovery, the mobile device transmits an application overlay to the cloudlet. The overlay together with the base image forms an executable VM containing the offloaded application logic, ready to interact with the mobile device.

2.2 The Palladio Approach

Palladio is a model-based approach allowing to predict non-functional quality criteria for component-based software architectures. The understanding of software components in Palladio is in accordance with the definition of Szyperski [SGM02]:

"A software component is a unit of composition with contractually specified interfaces and explicit context dependencies only. A software component can be deployed independently and is subject to composition by third parties."
With respect to Palladio two terms have to be distinguished. The Palladio Component Model (PCM) is a domain-specific language that allows the description of component-based software architectures. Palladio Bench summarizes a set of Eclipse plugins to analyze PCM model instances or perform simulations. In this proposal Palladio is used to refer to the overall approach, and when no distinction between model and simulation tools is necessary. Otherwise, we will refer to the PCM and to Palladio Bench directly.

2.2.1 The Palladio Component Model

The Palladio Component Model (PCM) is a domain specific language to describe component-based software architectures. Usually, for large software systems there are multiple people responsible for different development or design tasks. A component-based development process as presented by Cheesman and Daniels [CD00], and later extended with Quality of Service (QoS) capabilities by Koziolek and Happe [KH06], identifies four distinct roles in the development process: component developer, system architect, system deployer and domain expert.

Component developers are responsible for the specification of components and their implementation. Every component has to have at least one Provides Interface, specifying services offered by this component. In turn, a component can rely on an arbitrary number of Requires Interfaces, specifying services which have to be provided by other components in order for this component to work. Component and interface specifications in the context of the PCM are part of the Repository Model.

System architects assemble complete systems from single components. Every required interface of every component has to be connected with a compatible Provides Interface of a different component. The assembling details are described by a System Model based on the component and interface descriptions of a Repository Model.

System deployers determine the execution context for the components of a system. The execution context specifies the infrastructure the system will run on. It describes computing nodes, their resources (e.g. CPU, HDD), execution containers (e.g. application servers), and network connections. The allocation of components to specific nodes is described by
the Allocation Model referencing infrastructure and available resources described in the Resource Environment Model.

Domain experts provide statistical data to describe the behavior of the target user group. In Usage Models they specify usage patterns to describe which system services are requested and in what manner. The patterns parametrize the PCM model to allow evaluations of the software architecture with respect to non-functional quality characteristics.

### 2.2.2 Resource Consumption in the PCM

**Service Effect Specifications (SEFF)** are abstract descriptions of the component’s internal behavior for every service a component specifies in one of its provided interfaces. A SEFF provides a representation of the control flow in the component. In particular, it models the calls to services provided by other components. Services can specify multiple parameters and at most one result value. Parameters or result can be specified using stochastic expressions and by referencing parameters passed to the service or returned by called services. Stochastic expressions allow to describe values using statistical distributions.

**Resource Demanding Service Effect Specifications (RDSEFFs)** extend SEFFs with the support to model demand for resources (e.g. CPU) used by the component [BKR07]. Hauck et al. [HKKR09] extended RDSEFFs with explicit modeling of resource requests through Resource Interfaces. Actual demand for a resource is modeled as requests to a service provided through the resource’s Resource Interface. Explicit modeling of resource demand allows a component to formally specify requirements for its execution environment.

Resource demand can be specified directly through stochastic expressions or referencing parameters passed to the service. Stochastic expressions can be used to model statistic distributions of resource demand. Hauck et al. [HKKR09] further introduced Resource Containers to allow explicit modeling of layered software architectures. Resource containers distinguish four different types of layers. The uppermost layer, Business Components, includes the application components. Infrastructure Components, located below, allow for modeling of components with are required by application components and provided as part of the execution environment (e.g. application servers, frameworks, JVM). Controllers located in layers below infrastructure components provide at least one resource interface for upper layers. In turn they request at least one service interface which has to be provided by entities located in layers below. The bottommost layer contains Resources which only provide resource interfaces.

### 2.2.3 Simulation, Analysis and Prediction

Palladio Bench is a set of Eclipse plugins which support prediction and analysis of non-functional characteristics for software architectures modeled using the PCM. Different simulation frameworks are developed to work with Palladio Bench. SimuCom is the default framework and has been developed the longest. The basic concepts of SimuCom are outlined by Becker et al. [BKR09]. The following section presents a short summary of the concepts.

SimuCom relies on code generation to transform PCM models into Java code using the openArchitectureWare (oAW) framework. A Java class is generated for each component in the PCM instance. The behavior of the class is built based on the component’s RDSEFF. Calls to services of components through the Required Interfaces are translated into method invocations of the respective generated Java class. Explicitly modeled resource consumption is processed by scheduler implementations specific to the demanded resource. A simulation controller executes the generated simulation code and records measurements either in memory or in a relational database.
2.2.4 Automatic Optimization of Software Architectures

Martens et al. [MBR09] propose the concept PerOpteryx to optimize given software architectures by automatically adjusting model parameters. PerOpteryx takes as input a fully specified PCM instance, a description of variation possibilities and specification of the optimization goals. Per default, it uses an evolutionary algorithm to determine Pareto-optimal configurations for the specified attributes. Starting with the initial configuration as candidate, it systematically generates new candidates using different evolvement techniques. After each evolution the candidates are evaluated with respect to the desired quality criteria. Currently, PerOpteryx supports three criteria: performance, reliability and cost. Performance measuring relies on distinct SimuCom simulation runs. For each candidate the initial PCM is adjusted, a SimuCom simulation is run, and PerOpteryx evaluates the generated results.

Objectives and constraints for the optimization are specified using a model representation of the Quality of Service Modeling Language (QML). A short summary of the concepts, as described by Frølund and Koistinen [FK98], is given in this paragraph. QML distinguishes Contract Types, Contracts and Profiles. Contract Types bundles different quality dimensions (e.g. response time), and describes the ordering for each dimension (e.g. lower values are better). Contracts instantiate Contract types by allowing to set constraints for the quality dimensions. Profiles connect service interfaces with specifications of the QoS parameters of that interface. PerOpteryx requires the user to define a Profile each the usage scenario which is supposed to be optimized. The Profile references the required Contracts which in turn specify the quality dimensions subject to optimization.

2.3 Energy Consumption and Power Models

Energy consumption of microelectronic devices is subject to intensive scientific analysis. In times of rapidly increasing costs for power energy-efficiency becomes a non-negligible factor in system design. According to Rivoire et al. [RSR+07], modeling of energy consumption behavior is particularly important for energy-efficient component design.

Energy consumption models, or power models, are abstractions of the real world energy consumption behavior of the regarded system. The models describe relations between resource consumption and the amount of power drawn by the system. According to Rivoire et al. [RSR+07] an ideal power model has to have several characteristics. The most notable are:

- Accuracy
- Portability across different hardware designs
- Applicability for different workloads

Accuracy and portability of a power model is highly dependent on its degree of abstraction. The more hardware details are taken into account, the more accurately a prediction can become. On the other hand, every hardware detail additionally taken into account reduces the portability, since only platforms which comply to the model can be analyzed.

2.3.1 Energy Consumption Models for Mobile Devices

There are basically two general approaches to derive energy consumption from system behavior: leveraging system-metrics (e.g. CPU utilization) as proxy, or assigning absolute amounts to resource requests (e.g. a CPU cycle). Both types offer advantages and disadvantages, respectively. Further, granularity of the model is distinguished, from modeling
the consumption of the entire system to modeling single hardware components. Equations 2.2, 2.3 and 2.4 are taken from [JKY+12].

The overall power consumption of a system can be modeled as the sum of its components’ consumption:

$$P_{\text{system}}(t) = \sum_{i=1}^{n} P_{\text{component}_i}(t)$$ (2.1)

Many power models proposed for mobile devices rely on Linear Regression (LR), which assumes a linear relationship between system metrics and the energy consumption of a single component [SSM09, CH10, ZTD+10, DZ11, MKC12, JKY+12]. In particular, for the CPU most models use utilization as predictor for energy consumption. Modern mobile processors employ Dynamic Voltage Frequency Scaling (DVFS) to reduce energy consumption, when not executing under full load. Therefore, most models account for different states ($freq_i$ in Equation 2.2).

$$P_{\text{CPU}}(t) = \sum_{i=1}^{n} \theta_{freq_i}(t) * \beta_{freq_i} * u(t), \quad \theta_{freq_i} = \begin{cases} 1, & \text{freq}_i \text{ active} \\ 0, & \text{else} \end{cases}$$ (2.2)

Energy consumption of WiFi and cellular network devices can be modeled using distinct power states [ZTD+10]. The 3G power states IDLE, FACH and DCH are identified by the Radio Resource Control (RRC) protocol state machine [3GP14]. The actual consumption of the cellular network is entirely determined by the active power state.

$$P_{\text{Cellular}}(t) = \beta_{rcc}, \quad rcc \in \{IDLE, FACH, DCH\}$$ (2.3)

Zhang et al. [ZTD+10] identified four power states for the WiFi interface. The state machine is shown in Figure 2.2. The energy consumption of the interface is determined entirely by the consumption of the active state if the state does not support sending of data (low, high). Outgoing traffic is additionally accounted for on a linear basis [ZTD+10, JKY+12].

$$P_{\text{WiFi}}(t) = \begin{cases} \beta_{LT} * p(t) + \beta_{LT \text{ base}}, & \text{if } p \leq \text{Threshold} \\ \beta_{HT} * p(t) + \beta_{HT \text{ base}}, & \text{if } p > \text{Threshold} \end{cases}$$ (2.4)

There exist models for other components, e.g. display, GPS or audio [SSM09, ZTD+10, MKC12, JKY+12]. A more detailed description of these models is omitted because the energy consumption is not directly relevant to the decision whether to offload a component or not.
2.3.2 Benchmarking and Profiling

Power model parameters (e.g. the $\beta$s in Section 2.3) are device-specific abstractions of the concrete device hardware characteristics. The parameters can be determined using detailed information of the hardware platform. Therefore, many scientific approaches consider it the responsibility of device manufacturers to provide the models \cite{MKC12, JKY12}.

Alternatively, power models can be created through benchmarking and profiling. Benchmarking refers to the execution of a specific representative workload. Usually, \textit{micro} and \textit{macro} benchmarks are distinguished. Micro benchmarks test one isolated workload in particular (e.g. calculate $\pi$). Macro benchmarks usually combine different micro benchmarks to generate a more general and more realistic workload. Profiling refers to measuring a devices behavior for a certain reference workload. In case of power profiling the energy consumption behavior is measured.

Xu et al. \cite{XLLZ13} describe the model generation as a \textit{training phase}. Micro benchmarks stress specific hardware components while the energy consumption is profiled. The energy consumption measures can either be taken from the integrated \textit{battery management unit} \cite{JKY12}, or more accurately determined by an external power monitor \cite{ZTD10, MKC12}. Linear regression or more complex techniques calculate the model parameters from the measures obtained through profiling.

Zhang et al. \cite{ZTD10} propose the Android application PowerBooter. It runs several micro-benchmarks while using external measuring hardware to profile the energy consumption. Using a linear regression, PowerBooter generates device-specific power models. Similar to PowerBooter, DevScope presented by Jung et al \cite{JKY12} automatically derives a power model through micro-benchmarking. Instead of using external measurement equipment, they rely on integrated power management APIs. Both applications are not openly available but show the feasibility of the micro-benchmarking and profiling concept.
3. Related Work

This chapter provides an overview of related work and distinguishes the respective approach from the proposed master’s thesis.

3.1 Energy Consumption of Mobile Devices

Energy consumption for mobile devices has been studied intensively over the last years. In this section the different approaches to model the energy consumption of mobile devices are presented.

Energy Consumption Prediction Using Palladio

In his bachelor’s thesis Rosenthal [Ros14] extended Palladio with basic support to predict energy consumption for the Galaxy Nexus mobile device based on model-driven simulation. He extended the PCM with capabilities to model batteries and hardware sensors, e.g. camera or GPS sensor. While Rosenthal’s solution requires code extensions to support additional mobile devices, we envision a modeling of the devices’ energy consumption behavior. In contrast to his work we only want to predict the energy consumption for hardware parts which directly influence the offloading decision. Furthermore, we plan to implement more sophisticated energy models to increase the prediction accuracy.

Mobile Energy Consumption Measurement

Rivoire et al. [RRK08] compared different power models which link energy consumption to CPU utilization and other system-metrics (I/O rate for disk, parallelism). Their work did not focus mobile devices. They showed that based only on CPU and disk utilization, energy consumption can be predicted with an error smaller than 10% on average.

Rice et al. [RH10] implemented a system to automatically run energy measurements on Android mobile devices. They also implemented a test framework which synchronizes the execution of test cases with the energy profiling using external hardware. Their goal was to analyze measure energy consumption on a fine-grained level to identify the influence of specific phone and network activity.

Carroll et al. [CH10] conducted fine-grained measurements on a Neo Freerunner mobile device. Using freely available circuit schematics, they measured the energy consumption of single hardware components while executing different benchmarks. Since necessary circuit schematics are not available for commercial mobile devices, the approach is not viable for a broader application.
Power Estimation and Prediction

Zhang et al. [ZTD+10] proposed PowerTutor which estimates the power consumption for each application and hardware component while running on the mobile device. PowerTutor uses the energy consumption models created with the authors’ PowerBooster approach (see Section 2.3.2) and the total energy consumption derived from the internal battery management interface. Similarly, Jung et al. [JKY+12] proposed AppScope, which is similar to PowerTutor, is supposed to estimate energy consumption per application using energy models created beforehand using their DevScope concept (see Section 2.3.2). In contrast to our model-based prediction, both approaches determine energy consumption only for existing applications running on the mobile device.

Kjærgaard and Blunck [KB12] proposed PowerProf to generate power models using a genetic algorithm. They execute different training measurements, micro-benchmarks, and used the internal power management APIs to profile the consumption. The measures are fed to a genetic algorithm which determines the parameters for the power model.

Prediction Model Creation

Wilke et al. [WRG+12] proposed an approach to compare the energy consumption of different applications in the same application domain. Therefore, abstract test-cases would have to be created for each application domain, which represent typical user interaction. Every application developer would have to implement the test-case. The energy consumption can then be measured and compared using their energy testing approach JouleUnit [WGR13]. JouleUnit extends JUnit with the capabilities to profile energy consumption using external power meters and logging of significant events using the Android Debugging Bridge (ADB).

Hao et al. [HLHG13] measured the amount of energy consumed for every byte-code instruction of the Dalvik Virtual Machine. Using their per-instruction energy model, they can provide power consumption estimates at source-code level with a variable level of granularity (e.g. per-method, per-line). In contrast to the model-based approach using the PCM, eLens requires to distinguish between different VM instructions.

Pathak et al. [PHZ+11] traced system calls of the application. They represent device energy consumption through finite state machine and use system-calls as transition triggers. The finite state machine concept allows to model so-called tail power states, which they claim to be very important for wireless network modeling. Pathak et al. [PHZ+11] showed that their approach can achieve better results compared to system-metric based approaches (e.g. [ZTD+10, MKCT2, JKY+12]). Nevertheless, tracing system calls to generate the power model requires complex benchmarks to profile the consumption of every system call. Furthermore, the benchmarking and profiling process becomes highly dependent on the operating system. Therefore, every new version of the operating system that introduces system call changes, requires the benchmarking process to be altered.

Like our approach, Mittal et al. [MKCT2] aimed to enable energy consumption prediction during application development. They generated a power model similar to Zhang et al. [ZTD+10] using micro-benchmarks. Their approach extends the development platform emulator with the capabilities to predict energy consumption. In contrast to our model-based approach, they require applications to be implemented to run on the emulator.

3.2 Partitioning of Cyber-Foraging Applications

The following section presents different scientific approaches which support energy consumption minimizing application partitioning. As mentioned in Section 2.1.1 static and
dynamic concepts are distinguished. Dynamic approaches determine the application partitioning scheme during run-time using profiling techniques. Static approaches make the offloading decision before the application is executed.

Chen et al.\cite{CKK04} propose a concept to dynamically offload parts of mobile java applications to cloud servers. The offloading decision is made based on information on the energy complexity which has to be provided through annotations. The applied energy model only regards CPU consumption.

SmartDiet \cite{SSX11} supports developers to identify parts of existing applications which are potential offloading candidates. Therefore, it executes the application and collects run-time information. SmartDiet requires the application developer to actually make offloading decisions.

MAUI \cite{CBC10} is framework which enables dynamic offloading for .net-based applications. Developers are required to annotate methods which are suitable candidates for offloading. MAUI determines the most energy-efficient partitioning scheme at run-time.

CloudClone \cite{CIM11} runs a virtualized clone of the mobile device on the offloading target. The optimal partitioning scheme is determined before run-time through a combination of static code analysis and profiling of application test-runs. Based on the test results CloudClone realizes fine-grained static offloading schemes for existing, non-altered applications.

SmartDiet and CloudClone are both static approaches which focus on enabling offloading mechanisms for existing applications. All of the presented offloading approaches require the application to be implemented already. Our approach aims at determining partitioning schemes before the implementation phase.
4. Conception and Implementation

Offloading of application components can achieve significant energy consumption savings. Nevertheless, offloading every possible application component does not necessarily achieve the energy consumption optimum for the mobile device. In contrast to dynamic approaches (e.g. MAUI \cite{CBC10} or CloudClone \cite{CIM11}), which determine the optimal application partitioning based on profiling results during run-time, we propose a static model-based analysis. Static approaches suffer from the difficulties of modeling unforeseeable environmental influences which are characteristic for cyber-foraging applications, e.g. unreliable internet connection. Although a dynamic approach might determine a better deployment, profiling and run-time measurements are resource consuming. Furthermore, the dynamic offloading frameworks are almost always platform dependent. Our model-based analysis does not require the application to exist, but instead provides feedback already during application design.

4.1 Optimizing Deployment Allocation

The primary goal of this thesis is to enable deployment allocation optimization. The Palladio-Bench add-on PerOpteryx (see Section 2.2.4) provides a feature-rich automatic optimization framework for PCM models. Per default, it uses evolutionary algorithms to determine new model candidates. For each candidate a SimuCom simulation run can be executed. PerOpteryx provides a set of quality criteria which evaluate the candidate based on the simulation results. Currently, PerOpteryx supports only performance, reliability and costs as criteria. After the evaluation promising candidates are kept, less promising are discarded until either a certain quality threshold is achieved or the maximum number of evolutionary iterations has been reached.

We envision the following application scenario for our goal: The application which we want to optimize is provided as a PCM instance. Available application components and possible alternatives are specified in the repository model. The resource environment model describes the execution environment, in particular, the mobile device and the cloudlet. The remaining models (system, allocation) describe an exemplary assembly and deployment which is used as initial candidate for the subsequent optimization. Additionally, we have to specify a QMLDeclarations model defining the optimization objectives and optional constraints (e.g. optimize energy consumption and define upper limits for the response time). Based on the configuration PerOpteryx will generate possible design candidates and execute a simulation run for each possible deployment scheme. The simulation results
are evaluated in accordance with the objective specification to determine Pareto-optimal configuration.

Four major work packets can be identified:

- Extension of the PCM and SimuCom with prediction capabilities for mobile device energy consumption
- Extension of PerOpteryx to evaluate energy consumption as additional quality criterion
- Extension of the PCM and SimuCom to allow modeling of execution-context dependent behavior
- Validation of the extensions

The following sections will provide a more detailed description to each of these work packages.

4.2 Modeling Device Characteristics

Applications are rarely developed for one particular device. Furthermore, energy consumption of a device is subject to change under different operating systems. Different versions of the same operating system can present different characteristics on the same mobile device. Therefore, the analysis of software architectures requires ways to easily adapt simulation parameters to reflect different device characteristics. Currently, the energy consumption behavior for the entire device is encapsulated by the battery specification. Changes to device parameters have to made directly to the simulation code. Although the simulation allows for extension of new Battery specifications, the implementation requires more complex changes to the simulation framework.

Therefore, we propose an extension of the PCM to explicitly model energy consumption behavior. Since execution- or deployment-context is specified in the resource environment model, this model is the most suitable to capture energy consumption characteristics. In accordance with Section 2.3, the energy consumption of a device equals the sum of the consumption of all device components. Therefore we propose to annotate resources with their respective energy consumption model.

The current implementation requires absolute energy amounts to be derived for each resource request independently. This constitutes an obstacle particularly with respect to CPU consumption. Accurate predictions are hard to make based on the absolute amount of cycles requested only. Therefore, we want to support more elaborate models using certain execution-time characteristics, e.g. resource utilization. An important task is to identify efficient and correct ways the represent energy model annotations in the PCM and find the right trade-off between fine-grained energy consumption prediction and the information available at architecture level granularity during design time.

The model parameters which best reflect the characteristics presented by the mobile device have to be determined. Zhang et al. [ZTD+10] and Mittal et al. [MKC12] presented approaches to derive these parameters from profiling measurements of mobile devices. We would like to build upon their work if possible. Otherwise, we plan to employ a similar technique and therefore propose implementing a mobile device application. The application will run several micro-benchmarks to stress the hardware components while the energy consumption will be measured. A decision to be made is whether to use internal battery management APIs or to rely on external power measurement utilities. Ideally, the profiling application will prompt the user with the ready-to-use model parameter values.
4.3 Extending PerOpteryx

At the moment, the QMLDeclarations model does not support energy consumption as objective. Therefore, we propose to extend the corresponding meta-model accordingly.

PerOpteryx has to generate PCM instances for each newly identified candidate. In order to achieve our goal, we require PerOpteryx to be able to change the allocation of components to different resource containers. Components which can be run on the mobile device and can be offloaded to the cloudlet have to be evaluated for both configurations. Since PerOpteryx already supports allocation scheme optimization, it provides sufficient capabilities to adapt the model parameters in order to generate new candidates for the simulation.

Each design candidate is evaluated based on the simulation measurements. Quality criteria determine how far a design candidate is promising with respect to the optimization objectives. At the moment, PerOpteryx does not evaluate energy consumption as a quality criterion. Therefore, we propose the extension with capabilities to use the predicted energy consumption values generated during simulation. The extension to PerOpteryx is not necessarily dependent on mobile device context and could be utilized for different contexts provided that there exist compatible energy consumption prediction capabilities.

4.4 Modeling Context-Dependent Behavior

The run-time behavior of application components is highly dependent on the execution environment. Running the same component implementation on a server can show a different resource consumption behavior than on a mobile device.

At the moment, the PCM cannot properly model influences of the execution environment when specifying the resource demand of a component. Memory management is a very good example: Android terminates applications currently not running in the foreground when the system has run out of memory. Desktop operating systems (e.g. Windows or Linux) usually use the hard-disk to swap out applications. Furthermore component implementations can actually behave differently, e.g. because they were compiled differently or they explicitly check the execution environment.

Currently, we are required to model a distinct component implementation type which captures the platform dependent behavior in the RDSEFF. Hence, the model is missing the necessary semantics that the expressed behavior is only valid for the respective platform. Another approach is to introduce middleware components which abstract the platform differences for the component. The introduced middleware components would be dependent on the application components for which the platform differences should be abstracted. Other application components would require other middleware components to reflect the different behavior, which is a contradiction to the independent middleware concept.

To enable modeling of execution-environment-conscious component behavior we propose an extension to the PCM to support multiple RDSEFFs per component. There should be a distinct RDSEFF, which describes the the consumption behavior, for every execution environment. The extension enables us to represent all run-time influences and therefore, correctly model components which behave differently contingent on their deployment context.

4.5 Validation

The overall goal of the proposed thesis is to optimize the deployment of a component-based cyber-foraging application automatically. In our validation phase we plan to verify
the predicted optimal deployment scheme through actual deployment onto the test infrastructure and measure the energy consumption. Therefore, we will model the selected application using our extended version of the PCM and execute a PerOpteryx optimization run. We deploy the application according to the identified Pareto-optimal solution set and other non-optimal candidates. In order to validate our automatic optimization approach, we measure the energy consumption for each candidate.

We envision conducting a case study for the validation of our extensions to Palladio. Therefore, a suitable component-based application, preferably from a tactical edge scenario, has to be identified. As a first step of our validation phase we will formally define the requirements for the test application. Based on the requirements, we will try to find a suitable component-based application which can be leveraged for our case. The application has to consist of multiple components that can be executed on the mobile device as well as offloaded to the cloudlet.

In preliminary literature research we were unable to identify a suitable application for our study. In case the condition persists, we will fall back on a prototype implementation. Using a prototype we can induce artificial load on the different hardware resources. Eventually, the Palladio add-on ProtoCom can be used to generate executable component prototypes. The generated implementations will probably have to be adapted before they can be executed on mobile devices.
5. Organizational Matters

5.1 Advisement

The proposed thesis will be written in the context of a cooperation of the Institute of Program Structures and Data Organization (IPD) at the Karlsruhe Institute of Technology (KIT) and the Software Engineering Institute (SEI) at the Carnegie Mellon University (CMU) in Pittsburgh.

The thesis will be reviewed by Prof. Dr. Ralf Reussner of the chair of Software Design and Quality (SDQ). The second reviewer will be Jun.-Prof. Dr.-Ing. Anne Koziolek. Grace Lewis will act as advisor at the SEI. She will be supporting, in particular with respect to questions arising from the cyber-foraging aspect. At the KIT, Christian Stier will oversee the entire process and help in particular with questions concerning the Palladio environment. Dr. Klaus Krogmann (KIT) will be the third advisor.

5.2 Artifacts

- Model extensions to the PCM
  - Support for device characteristics
  - Support for execution-context dependent behavior
- Code extensions to SimuCom
  - Support for device characteristics
  - Support for execution-context dependent behavior
- Profiling application for Android devices
- Code extension to PerOpteryx
  - Support for new quality criterion energy consumption
- Validation artifacts
  - Enhanced model of tactical edge application (prototype)
  - Executable (prototype) with implementations for mobile device and cloudlet
5.3 Environment

The thesis will be written in \LaTeX using TeXnicCenter. UML Models are created using Microsoft Visio 2013. The profiling application will be developed for mobile devices using the most recent version of Android Studio. For the implementation of non-mobile applications, e.g. extensions to Palladio, Eclipse IDE will be used as default environment. Quality assurance of the created code artefacts will be supported by JUnit as framework for unit testing and Checkstyle to check the compliance with SDQ development standards.

The generation of PCM models for existing applications can be supported using SoMoX for reverse engineering. The inverse direction, generation of implementations for PCM models, can be supported by ProtoCom.

Fine-grained measuring of the energy consumption of mobile devices is possible using a Monsoon Power Monitor [Mon].
6. Time Scheduling

6.1 Work Packages

The following work packages will group the tasks to implement the concepts presented in Section 4. Every work package consists of several tasks and is concluded by a milestone.

Work Package 1: Energy Consumption Model:

**Extend PCM** Extend the resource environment model to allow for device characteristics to be specified. Eventually, additional resource types have to be implemented to support the different types of demand.

**Develop Profiling Application** Either extend an existing approach (see Section 2.3.2) or develop an Android application which executes the micro-benchmarks while measuring the energy consumption. In this task it will be determined whether to use the in-built power management interface or to leverage external measuring hardware. The latter case will probably require the implementation of an application to evaluate the measured power consumption.

**Extend Simulation** Determine and implement the necessary extensions to the simulation framework.

**Milestone M1** Palladio Simulator can predict energy consumption given device characteristics and a PCM instance of the application.

Work Package 2: PerOpteryx Extension:

**Implement Quality Criterion** Implement the necessary extension to PerOpteryx to evaluate the energy consumption values generated through simulation runs.

**Milestone M2** PerOpteryx evaluates a given PCM instance and determines optimal deployment schemes.

Work Package 3: Execution Context Dependent Behavior:

**Define Requirements** Formalize which behavioral-specifics have to be reflected.

**Identify Design Alternatives** Identify alternatives of how the PCM can be extended to allow modeling of execution-context dependent behavior which fulfill the previously determined requirements.
Implement Extension  Implement the extensions to the PCM and the simulation.

Milestone M3  Component behavior can be specified individually for different execution contexts.

Work Package 4: Validation:

Identify Validation Scenario  Search for suitable validation application (scenarios).

Prepare Application/Prototype  Assure the availability of implementations for multiple platforms for components which are suitable for offloading.

Evaluate & Optimize  Deploy applications to test infrastructure and measure energy consumption for different deployment schemes. Analyze deviations and refine energy consumption model.

Milestone M4  Optimization finished and evaluation conducted. No more refinement from this point on.

Work Package 5: Documentation:

Milestone M5  First complete version of thesis for review is finished.

Milestone M6  Final version of thesis is finished.

6.2 Risk Management

At the time of writing this proposal the following risks are identified. For every risk a more detailed explanation is provided and measures to mitigate the risk are presented.

Risk R1  Definition of Validation Scenario very complex / time-consuming

Explanation  The identification of a suitable validation scenario is crucial for the success of the proposed thesis. The scenario should stem from the context of tactical edge applications and consist of multiple components. Otherwise automatic deployment optimization cannot be reasonably argued.

Mitigation  The respective work package will be addressed first. In case the budgeted amount of time will not suffice, the amount of time spent on optimization tasks can be reduced. If the amount of time exceeds the budgeted significantly milestones M3 can be declared as optional.

Risk R2  (Partial) implementation of test application components necessary

Explanation  A meaningful validation requires all application components which are feasible for offloading to be available for the server platform as well as for the mobile device.

Mitigation  See mitigation of R1. If no real application which can be leveraged for our purposes can be found we will fall back on the implementation of an artificial prototype.

Risk R3  Underestimated Benchmark Complexity

Explanation  The lack of reusable scientific benchmarks readily usable for profiling the energy consumption of mobile devices requires a non-negligible amount of time spent on implementing one. Dependent on the device characteristics identified in M2, developing and implementing benchmark routines can become very complex and time-consuming.
**Mitigation** The work package will be addressed as early as possible. Refinement of the benchmark routines will be done iteratively during the optimization phase. If the amount of time exceeds the budget significantly milestones M3 can be declared as optional.

**Risk R4** Obstacles through unknown work environment

**Explanation** The thesis will be written in Pittsburgh. Obstacles arising in the settling period cannot be precluded, e.g. in finding accommodation.

**Mitigation** The risk cannot be entirely mitigated. Arising questions or problems will be dealt with as early as possible.

### 6.3 Detailed Scheduling

The following gantt chart visualizes the more detailed schedule starting Oct 20th, 2014 until the official termination Apr 30th, 2015. Milestones are only explicitly visualized if they do not equate to the end of the respective work package.

<table>
<thead>
<tr>
<th>2014</th>
<th>2015</th>
</tr>
</thead>
<tbody>
<tr>
<td>Oct</td>
<td>Nov</td>
</tr>
<tr>
<td>43</td>
<td>44</td>
</tr>
</tbody>
</table>

- **Energy Consumption Model**
  - Extend PCM
  - Develop profiling application
- **Extend PerOpteryx**
  - Implement Criterion
- **Execution Dependent Behavior**
  - Define Requirements
  - Identify Design Alternatives
  - Implement Extension
- **Validation**
  - Identify Validation Scenario
  - Prepare Application/Prototype
  - Evaluate & Optimize
- **Documentation**
  - Write Documentation
  - Focus on Writing
- **Environmental**
  - Settle in Pittsburgh
  - Finalize and Print

*Milestone M5 ♦*
Acronyms

CMU Carnegie Mellon University
IPD Institute of Program Structures and Data Organization
KIT Karlsruhe Institute of Technology
oAW openArchitectureWare
PCM Palladio Component Model
QML Quality of Service Modeling Language
SDQ Software Design and Quality
SEI Software Engineering Institute
Bibliography


