Canevas de programmation pour gérer l'héterogénéité et la consommation d'énergie des mobiles dans un environnement ubiquitaire

Hongyu Guan

UNIVERSITÉ DE BORDEAUX

Département de formation doctorale en informatique

École doctorale EDMI Bordeaux
N° d'ordre : 4526

Canevas de programmation pour gérer l'hétérogénéité et la consommation d'énergie des mobiles dans un environnement ubiquitaire

THÈSE

soutenue le 01 juin 2012

pour l'obtention du

Doctorat de l'Université de Bordeaux 1

(spécialité informatique)

par

Hongyu GUAN

Jury

Président :	Ye-Qiong Song,	Professeur à l'Université de Lorraine - ENSEM
Rapporteurs :	Philippe Lalanda,	Professeur à l'Université Joseph Fourier de Grenoble
	Philippe Roose,	Maître de Conférences (HdR) à l'Université de Pau
Encadrants :	Charles Consel,	Professeur à l'Institut Polytechnique de Bordeaux
	Patrice Kadionik,	Maître de Conférences à l'Institut Polytechnique de Bordeaux

LaBRI
Unité Mixte de Recherche CNRS (UMR 5800),
351 cours de la libération,
F-33405 Talence cedex

INRIA Bordeaux Sud-Ouest
Bâtiment A 29,
351 cours de la libération,
F-33405 Talence Cedex

Université de Bordeaux 1

Première édition, juin 2012.

ABSTRACT

MANAGING HETEROGENEITY AND ENERGY VIA HIGH-LEVEL PROGRAMMING FRAMEWORK

The topics of heterogeneity and energy are two fundamental considerations for pervasive computing environments. In this thesis, we describe our approach to manage heterogeneity and to handle energy concerns via a high-level programming framework.

To manage heterogeneity, we describe a methodology and a programming support that use the SIP protocol as a universal communication bus in pervasive computing environments. Our work enables homogeneous communications between heterogeneous distributed entities. In doing so, we integrate the SIP communication bus into our programming framework. We rely on a declarative language named DiaSpec to describe the architecture of pervasive applications. This description is passed to a generator for producing a Java programming framework dedicated to the application area. We leverage the generated framework with SIP adaptations to raise the abstraction level of SIP operations. We then present a classification of a wide variety of entities in terms of features, capabilities and network connectors. Based on this classification, a methodology and a programming support are described for connecting entities on the SIP communication bus. This work has been validated by applications using the SIP communication bus to coordinate widely varying entities, including serial-based sensors (RS232, 1-Wire), ZigBee devices, X10 devices, PDA, native SIP entities, and software components.

Regarding the energy concerns, we describe a methodology that uses two strategies, namely computation offloading and data compression, to minimize energy cost of mobile applications. In doing so, we present an execution and transfer model for a task of a mobile application and define its five different stubs for three program execution and data transfer modes. Based on this model and our two strategies, we construct a strategy scheme to determine the most efficient stub in terms of energy consumption. We then design the OffDeci tool, using this strategy scheme, to provide energy feedback for the developer and to analyze the balance between local and remote computing with consideration of data compression. Our experimental study demonstrates the feasibility of the strategy scheme of our approach. Finally, we extend DiaSpec with declarations dedicated to manage energy concerns during the application design phase. We sketched the

integration of this energy-handling declaration and OffDeci into our high-level programming framework. This integration permits to determine the best stub of a declared DiaSpec component in terms of its energy cost.

KEYWORDS: Architecture Description Language, Domain-Specific Language, SIP, Heterogeneity, Energy consumption, Mobile application, Computation offloading, Data compression

RÉSUMÉ

L'hétérogénéité des objets communicants et la consommation d'énergie sont deux considérations fondamentales pour les environnements informatiques ubiquitaires. Dans cette thèse, nous présentons notre approche pour gérer l'hétérogénéité et pour économiser l'énergie via des canevas de programmation dédiés. Pour gérer l'hétérogénéité, nous proposons une méthodologie et un support de programmation qui visent à faire communiquer les différents objets communicants de l'environnement ubiquitaire, et ce, en utilisant le protocole SIP considéré alors comme un bus de communication universel. Nous avons intégré ce bus SIP dans les canevas de programmation générés à partir des descriptions d'applications d'informatique ubiquitaire, écrites dans le langage DiaSpec. Concernant la consommation d'énergie, nous proposons une méthodologie qui utilise les techniques d'offloading et de compression de données pour minimiser la consommation d'énergie des applications mobiles. Nous avons ainsi construit une stratégie d'aide à la conception au travers d'un outil qui permet de déterminer le meilleur mode d'exécution pour une tâche donnée. Nous proposons l'intégration de cette stratégie dans le langage de description DiaSpec.

MOTS CLÉS: Architecture Logicielle, Langage Dédié, SIP, Hétérogénéité, Consommation d'énergie, Applications mobiles, Calcul déporté, Compression de données

LISTE DES PUBLICATIONS

Les travaux discutés dans cette thèse ont été présentés précédemment.

CONFÉRENCES INTERNATIONALES

- "SIP as a Universal Communication Bus: A Methodology and an Experimental Study," dans *ICC'10 : Proceedings of the International Conference on Communications*, 2010, Cape Town, South Africa, Bertran Benjamin, Consel Charles, Jouve Wilfried, Guan Hongyu et Kadionik Patrice

REMERCIEMENTS

Cette thèse, qui est constituée de moments de joie, de courage et de doute, n'aurait jamais pu arriver à son terme sans l'aide et le soutien de nombreuses personnes. Je remercie tout d'abord mes deux encadrants. Je n'aurais jamais pu commencer et effectuer cette thèse dans de si bonnes conditions sans mon directeur de thèse Professeur Charles Consel. Il a su me faire confiance et Il m'a poussé afin de donner le meilleur de moi-même pour soumettre dans les meilleures conférences internationales. Aussi, cette thèse n'aurait pas non plus vu le jour sans le support inconditionnel de Maître de Conférences Patrice Kadionik. Je n'ai jamais vu personne d'autre se donner autant pour ses étudiants: il m'a introduit dans une nouvelle communauté de logiciel libre, il m'a permit de m'ouvrir l'esprit et m'a invité à participer aux Rencontres Mondiales du Logiciel Libre.

Je remercie les membres de mon jury :

- Je remercie Philippe Lalanda, professeur à l'Université Joseph Fourier de Grenoble, et Philippe Roose, Maître de Conférences (HdR) à l'Université de Pau, pour avoir accepté la charge de rapporteur.

- Je remercie Ye-Qiong Song, professeur à l'Université de Lorraine, qui a présidé ce jury et ma soutenance.

Je remercie tous les membres de l'équipe Phoenix. En particulier, je voudrais donc remercier Benjamin Bertran pour ton soutien lors de l'écriture des articles et pour tes conseils, Benjamin a su être assez courageux pour relire et corriger mes premiers brouillons lors des écritures des articles. Je le remercie aussi pour son aide précieuse dans la concrétisation de l'approche SIP. Je voudrais également remercier Julien B. et Henner pour leurs conseils concernant ma thèse. Je voudrais aussi remercier Luc et Ghislain car vous m'avez poussé à faire toujours mieux et pour toutes nos discussions que ce soit sur le domaine tecnique ou encore sur les cultures françaises et chinoises. Je voudrais particulièrement remercier Pengfei Liu pour toutes nos discussions sur nos travaux de thèses, sur la technologie de l'électronique et de l'informatique, la politique, la vie des étrangers en France, la cusine et la culture chinoise et pour finir l'actualité de la Chine. Merci à Quentin pour tes conseils sur mes travaux et merci à Stéphanie et Zoé pour leurs bonnes humeurs. Merci aussi à Chrystel et Sylvie d'avoir si bien organisé et tant aidé en ce qui concerne mes tâches administratives durant ma thèse et merci à Aurélien, Christophe, Florent et Jérome d'avoir résolu tous mes

problèmes techniques concernant le système informatique de l'INRIA.

Enfin, je remercie tous les membres de ma famille. Je remercie en particulier ma mère (Mme Shuqin LIU), mon père (M. Xingchang GUAN), mon frère (M. Hongfeng GUAN), ma belle-mère (Mme Shuhua DU), mon beau-père (M. Zhiyi GUO), ma belle soeur(Mme Qingli WANG), ma nièce (Mlle Enya GUAN), mon grand père (M. Shanyi LIU). Ils m'ont fourni un cadre idéal pour mes études et sans le soutient de leurs parts je n'aurais pas effectué ma thèse. Ces derniers mots sont pour ma femme (Mlle Zhuosha GUO) dont l'amour m'a été indispensable durant ma thèse et me le sera encore. Je lui consacre mes remerciements les plus sincères. Sans elle, cette thèse n'aurait jamais pu commencer. Elle m'a apporté le meilleur soutien durant ma thèse.

Je dédie cette thèse à tous les membres de ma famille car c'est aussi la leurs.

CONTENTS

LIST OF FIGURES

LIST OF TABLES

LISTINGS

1 INTRODUCTION

Pervasive computing will be a fertile source of challenging research problems in computer systems for many years to come. Solving these problems will require us to broaden our discourse on some topics, and to revisit long-standing design assumptions in others.

— Satyanarayanan, M. [61]

In 1991, *Mark Weiser*, chief technology officer for Xerox's Palo Alto Research Centre, wrote: *"The most profound technologies are those that disappear. They weave themselves into the fabric of everyday life until they are indistinguishable from it."* This paper [67] described his vision for 21st century computing that countered the ubiquity of personal computers. This vision corresponds to ubiquitous computing, now also called pervasive computing. The essence of that vision was the creation and the development of environments saturated by computing and communication capabilities, yet gracefully integrated with human users. At that time, this vision looks like science fiction. Today, different technical trends seem to tell us that this vision is going to become a reality.

Firstly with the emergence of networking PC became connected. Computers evolved to distribute computing. They can share capabilities over the network. Distributed computing marked the next step toward pervasive computing by introducing seamless access to remote information resources and communications with fault tolerance, high availability, and security [61].

Secondly mobile computing began from the integration of cellular technology with the Web [60]. The miniaturization technology of equipment has made great progress in recent years. At the same time, computation, storage and communication capacity have increased significantly. Consumption energy of these devices has also greatly reduced. This progress allows that both the size and price of mobile devices are falling everyday and could eventually support *Weiser's* vision of ubiquitous inch-scale computing devices readily available to users in any human environment.

The technological advances in distributed computing and mobile computing prepare the way to achieve ubiquitous or perva-

sive computing's goal and build a pervasive computing environments.

Potential applications in the field of pervasive environments are numerous ranging from building management to healthcare. They present real economic, social and technical issues. New applications in the field tend to share capabilities of several devices. Despite recent technological advances in hardware and networking, ubiquitous and pervasive applications are relatively few and far from the vision of *Weiser*. This is mainly due to the lack of abstraction layer for system infrastructure to facilitate the creation of pervasive applications. An abstraction layer for system infrastructure is directly dependent on the challenges of pervasive computing system. Pervasive computing subsumes the research challenges of distributed computing and mobile computing while going much further and opening up new issues and demands. Specifically the challenges and requirements of pervasive computing environments are identified: scalability, heterogeneity, invisibility, energy consumption, smart spaces, fault tolerance and security, *etc* [59], [61]. The challenges of pervasive computing provide many opportunities for research and require more than a single thesis. Therefore we narrow the scope of this thesis and focus on two aspects: heterogeneity and energy consumption.

As device-rich networked environments are becoming increasingly prevalent in different areas, heterogeneity increases in such environments. These pervasive computing environments consist of a variety of entities that are heterogeneous in many aspects: (1) they are either hardware (e.g., camera and telephone) or software (e.g., agenda, news and location server); (2) they rely on different network layers generally dedicated to pervasive environments (e.g., X10, 1-Wire, ZigBee, and IP); (3) they interact using various modes of communication (e.g., events and streams); and, (4) they exchange various kinds of data (e.g., temperature measurements and video streams). Such environments are also highly dynamic with entities appearing and disappearing over time (e.g., a telephone is switched on/off). Moreover, software systems managing these entities need to be open-ended to keep pace with a constant flow of technological advances.

As heterogeneity increases in pervasive computing environments, developing applications in such environments become exceedingly difficult. Managing heterogeneity in an ad-hoc way leads to make the integration of all various platforms almost impossible. We need an abstraction layer to hide heterogeneity and to facilitate the application creation. Several general-purpose middlewares (e.g., CORBA, DCOM) have been studied in this domain. While existing software engineering approaches support the de-

velopers for the most error-prone tasks and ad hoc techniques [6], they still lack a platform that relies on industrial standards to tackle the heterogeneity, using a high-level abstraction layer for all kind of entities.

Today's smart phones are programmable and come with a growing set of cheap powerful computation and communication capabilities. These technological advances meet the requirements of pervasive computing for the seamless interoperation of diverse devices. Especially with these increasingly pervasive smart phones, more and more services which used to be available only by personal computer are expected to be provided by mobile device. As there are more and more various and complex services on mobile, the required performance of mobile devices increases. Thus power consumption by mobile devices is growing rapidly. However, battery capacity has shown slower growth. Therefore, reducing power consumption effectively is a crucial issue in mobile devices. Moreover, battery consumption is clearly important for users when buying mobile phones and choosing applications and services to use. Software systems must integrate energy consumption concerns of the pervasive smart phones into pervasive computing systems.

Because smart phones offer more and more pervasive services and applications, they play a major role in this domain. Energy consumption impacts all resources of a smart phone. Integrating energy consumption concerns in an ad-hoc way leads to making maintenance and upgrading of the system virtually impossible. Bass et al. [13] propose to consider them through design, implementation, and deployment phases of software systems. To support and facilitate the work for the developer, we need tools that can give energy consumption feedback during the application development and debugging cycle.

1.1 APPROACH

To manage heterogeneity and energy concerns in pervasive environments, our approach leverages DiaSuite[1], a toolkit which provides a development environment dedicated to pervasive computing systems. The core of DiaSuite is a domain-specific design language for pervasive computing systems named DiaSpec. A dedicated programming framework is generated from a DiaSpec specification, which supports the implementation of the functionality, testing, and deployment of applications. The programming framework is able to provide the programmer with high-level programming mechanisms, raising the abstraction level of interac-

1 http://diasuite.inria.fr/

tion modes between entities. Thanks to this support, developers manipulate high-level concepts (e.g., service discovery) in Java.

Based on this observation, this thesis proposes an approach to managing heterogeneity and energy concerns of pervasive computing environments and to going beyond the limits of existing approaches. Our approach is composed in two parts. First, we present a methodology and programming support to use SIP as a universal communication bus for pervasive computing environments. Second, we construct a program execution and data transfer strategy scheme that uses computation offloading and data compression technologies. We then apply this scheme to help the developer to determine whether and how to offload computation of a given program with or without compression data transfer.

In the first part, our research aims to address the heterogeneity of pervasive computing environments by generalizing SIP [55] (Session Initiation Protocol) to a software communication bus. SIP is now widely deployed in both local and global telecommunication infrastructures, as well as at home with the Internet gateway (ADSL box). This industry standard for Internet telephony provides a basis to address the challenges of pervasive computing environments. For example, dynamicity can be addressed by leveraging SIP's mechanism for user mobility. The heterogeneous modes of communications between entities can leverage SIP's general-purpose forms of communications, namely multimedia sessions, events and instant messaging. We proposed a Java programming framework to develop such services.

In the second part, our research aims to address how to take energy consumption into consideration in the development of applications and services on pervasive smart mobiles for the programmer. It is well known that reducing energy consumption should be based on accurate energy consumption estimation technique since it is possible to over- or under-constrain design without accurate estimation. To handle this problem we explore two technologies to reduce the energy consumption on smartphones: computation offloading to minimize local calculations and data compression to reduce the volume of data to transfer. We then construct a program execution and data transfer strategy scheme, using these two technologies. We integrated this scheme into a tool named "OffDeci" that permits the programmer to estimate more accurately energy consumption of wireless communication and local processing for a given program. Through comparison between local computation and computation offloading, this tool decides the trade-off point. We sketch the integration of this tool into our high-level programming framework.

By using an integrated programming framework, our approach facilitates the application creation. In the application design phase, our approach hides the heterogeneity in pervasive computing environments and takes energy consumption concerns into account. It allows the programmer to concentrate on high-level design issues rather than on implementation details (e.g., communication layers and interaction modes). In doing so, both parts of our approach leverage DiaSuite. To facilitate the development, we provide the programmer with DiaSpec on top of the SIP-based support framework that is presented in the first part of our approach. A Java dedicated programming framework is generated from a DiaSpec specifications, which supports the implementation, the testing and the deployment of application. The programmer writes pervasive computing applications using high-level abstraction methods that are mapped into SIP-compliant operations. For the second part of our approach we extend existing DiaSpec specifications to generate coherent configurations for each relevant DiaSpec component. The implementation of these configurations are passed to OffDeci tool. The most efficient energy consumption execution mode of a DiaSpec component is automatically deployed in the application execution time.

1.2 THESIS CONTRIBUTION

This work implements and validates an approach to providing an abstraction layer for heterogeneity of pervasive computing and to integrating energy consumption concerns into pervasive computing systems. The specific contributions of this thesis are described below.

Main contributions of the first part of this thesis are as follows:

A UNIVERSAL COMMUNICATION BUS BASED ON SIP We use a domain specific design language approach to generate dedicated SIP-based programming frameworks, raising the abstraction level of SIP-native operations and introducing a uniform mechanism to invoke non-SIP resources.

A CLASSIFICATION OF A WIDE VARIETY OF ENTITIES This classification facilitates the integration of these entities in the SIP communication bus and a methodology and programming support that make each class of entities SIP compliant.

AN EXPERIMENTAL STUDY This study validates the SIP approach as a communication bus for pervasive computing envi-

ronments and comprises numerous entities with vastly varying features and capabilities.

Main contributions of the second part of this thesis are as follows:

PROGRAMMING SUPPORT FOR TRADE-OFF POINT OF COMPUTATION OFFLOADING Our approach allows selecting one execution and transfer mode for the programmer to minimize the energy consumed by a given task with a fixed amount of input and output data. For a variable amount of input and output data, our approach allows the programmer to choose one execution and transfer mode with help of prediction estimation rules. Our tool can verify the correctness of the choice for a specific task.

DATA COMPRESSION SUPPORT Data compression has large impact on the cost energy of data transfer. We consider compressed data transfer in our program execution and transfer strategy scheme.

ENERGY ESTIMATION WITHOUT NEED FOR EXTERNAL EQUIPMENT In our tool, we design a software stopwatch-benchmarking tool to measure execution time for a task, and we use the Sesame [20] tool to estimate power consumption. Our tool calculates then energy estimation for a given task in three execution and transfer modes. There is no need for external equipment to calculate energy consumption.

VALIDATION We have applied our approach in the development of two applications. We measure precisely the energy consumption of the three execution and transfer modes of these applications with external equipment hardware for power measurement.

ENERGY-HANDLING DECLARATION We have extended a domain-specific design language to declare energy concerns at a design level. A declarative approach is introduced to define components of an mobile application that may be offloaded. Such declaration allow managing energy cost of components via high-level programming framework.

1.3 ROADMAP

This thesis is split into three parts: The first part represents a review of the state of the art that introduces the context and

the challenge of pervasive computing and then describes our approach that targets heterogeneity of the pervasive computing environment; the second part presents our approach that addresses energy consumption of the mobile in this type of environment; and the third part concludes the thesis and outlines future work.

1.3.1 *Heterogeneity*

Chapter 2 sketches three pervasive computing scenarios. It also outlines the requirements for pervasive computing. We restrict and highlight the research question of this thesis and we then introduce our design language dedicated to pervasive computing, DiaSpec, with which we leverage to manage heterogeneity and energy concerns in pervasive environments. Chapter 3 describes our approach design, gives some backgrounds on the SIP protocol and presents its benefits in a pervasive computing context. We detail the general structure of needed SIP adaptions and adapters, connecting entities to the SIP universal communication bus in this chapter. We also propose a SIP back-end for DiaSpec to develop SIP-based applications. Chapter 4 introduces our experimental platform and examines our experiment results. We present evaluation and related works relevant to heterogeneity in the pervasive computing environment. Chapter 5 summarizes our approach to manage heterogeneity using SIP communication bus in the first part of this thesis.

1.3.2 *Energy consumption*

Chapter 6 gives a concise statement of the challenge about energy consumption on smartphons. It then sketches two applications and outlines the requirements for energy consumption issue on smartphones. Chapter 7 presents an overview of our approach to minimize energy cost of mobile applications. It introduces overview of three execution and transfer modes and their strategy scheme. We also present the overall design of our tool with the integration of our program partition strategy scheme. Chapter 8 reports experimental results, which are collected from a HTC Hero smart phone. We then discuss the prediction rules and also present related work relevant to energy cost concerns on smartphones. We leverage DiaSpec with our OffDeci tool in the Chapter 9. Chapter 10 summarizes our approach to handle energy concerns for mobile application using computation offloading and data compression in the second part of this thesis.

1.3.3 *Conclusions*

The third part summarizes our approach to manage heterogeneity and energy concerns in pervasive environments. Chapter 11 draws overall conclusions while Chapter 12 points out remaining problems and outlines present avenues for future work.

Part I

SIP AS A UNIVERSAL COMMUNICATION BUS: A METHODOLOGY AND AN EXPERIMENTAL STUDY

2 BACKGROUND

Conversion from one domain to another is integral to computing and communication. Assuming that uniform and compatible implementations of smart environments are not achievable, pervasive computing must find ways to mask this heterogeneity – or uneven conditioning, as it has been called – from users.

— Saha, D. and Mukherjee, A. [59]

This Chapter sketches three pervasive computing scenarios and outlines the requirements for pervasive computing. It then introduces DiaSpec that addresses the application-level requirements. We present how to leverage DiaSpec approach to handle the heterogeneity and the energy concerns in the Chapters ahead.

2.1 REQUIREMENTS OF PERVASIVE COMPUTING SYSTEMS

This Section presents several scenarios for pervasive computing environments and identify their needs, limitations, and risks. Additionally, we describe a typical pervasive computing environments. Finally, general analysis illustrates a number of concerns specific to the pervasive computing environments.

2.1.1 *Motivating examples*

Let us examine three scenarios from three areas: home automation, surveillance and security, and healthcare. These scenarios involve the use of a variety of devices including alarms, video cameras, phones and televisions. These devices are connected to a home/office network infrastructure. Besides devices, our working scenarios also consist of external software services such as agendas and an SMS gateway.

ADVANCED INTERCOM In this scenario, when someone uses the home intercom, it calls every phone in the house. After a period of time, if the call has not been answered, it is redirected to the mobile phone of one of the home owners. Whoever gets

the call can talk to the visitor, as well as remotely open the door using the keypad of his/her phone.

This application not only illustrates the use of existing SIP features, such as audio streams and DTMF [27], but it also exhibits the need to leverage the telephony infrastructure to enable home equipments (e. g. doors, lights, alarms) to be controlled remotely.

INTRUSION DETECTION This scenario is dedicated to house security. When an intrusion is detected, the application sends an SMS and an email to the house owner, both with a video. It also calls the police, providing specific information (e. g. number of detected intruders and area of intrusion) using a text-to-speech component.

This scenario demonstrates the need for advanced event mechanisms to bring rich information to applications (for instance, the location of the intrusion). It also points out the need to attach non-functional information to distinguish and discover entities. For example, if each video camera has an attribute specifying its location, then it is possible to dynamically and selectively record a video of the intrusion scene. Finally, this scenario shows home components interacting with external services (e. g. SMS).

TALKING TO THE PATIENT : MEDIA CONTENT INFORMATION From her office, the nurse in charge of the treatment establishes an audio call with the patient in his home. To facilitate her explanations (e. g. a new dosage), she can remotely push information like graphs or test results on the patient's TV screen from her smartphone. From her office, the nurse in charge of the treatment establishes an audio call with the patient. To facilitate her explanations (e. g. a new dosage), she can remotely push informations like graphs or test results on the patient's TV screen from her smartphone.

This scenario underlines the need for combining home and office equipments to perform a number of tasks. It also introduces the need for session mechanisms to bring rich information (e. g. media content) to applications.

2.1.2 *The environment*

After analyzing our working scenarios, we draw an overview of a typical pervasive computing environment and provide a preliminary assessment of its capabilities.

SYSTEM INFRASTRUCTURE The system infrastructure of a pervasive computing environment is populated with numerous de-

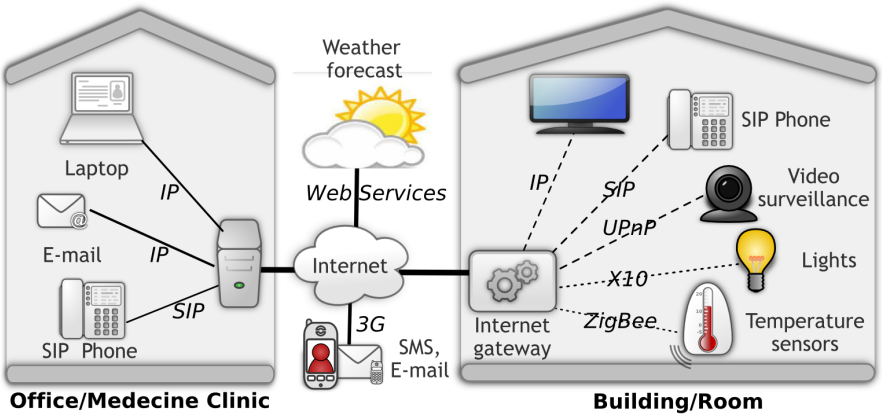


Figure 1: Pervasive computing environment

vices in Figure 1. Each of them provides services accessible through specific interaction modes and communication technologies (e. g. Web Services, UPnP, X10, ZigBee, and SIP). This system infrastructure assumes that an Internet gateway exists for each building. The gateway makes a bridge between the building and Internet, giving an access to online services (e. g. calendar, news, e-mail, and weather forecasts). The gateway contains an entire system with storage, and computing capabilities (e. g. hard drive, general-purpose CPU, sizable memory, USB ports, *etc.*).

APPLICATIONS In our study, an application controls, manages, and coordinates several entities, whether hardware or software, realizing a scenario. The orchestration of these entities leads the applications to interoperate with various devices and software components coming from heterogeneous domains: surveillance (e. g. video cameras, alarms), multimedia (e. g. media server, TV), or home automation (e. g. heaters, lights). In a pervasive computing environment, applications are hosted in the home network (e. g. in the Internet gateway itself, or in a dedicated device) or in the core network (e. g. cloud computing).

2.1.3 *Requirements*

Beyond the specific requirements illustrated in our working scenarios, the system infrastructure and the creation of applications entail the common requirements of pervasive computing environments. These requirements can thus be split into two levels. To fulfill these requirements, the middleware is a necessary element. The term middleware refers to create an abstraction layer for system infrastructure with interaction modes that enables heterogeneous entities to communicate each other in both local

area network and wide area network and provide the support for application creation through a programming framework or an API. Thus, pervasive computing environments are staged in three categories: system infrastructure, middleware, and applications (Figure 2).

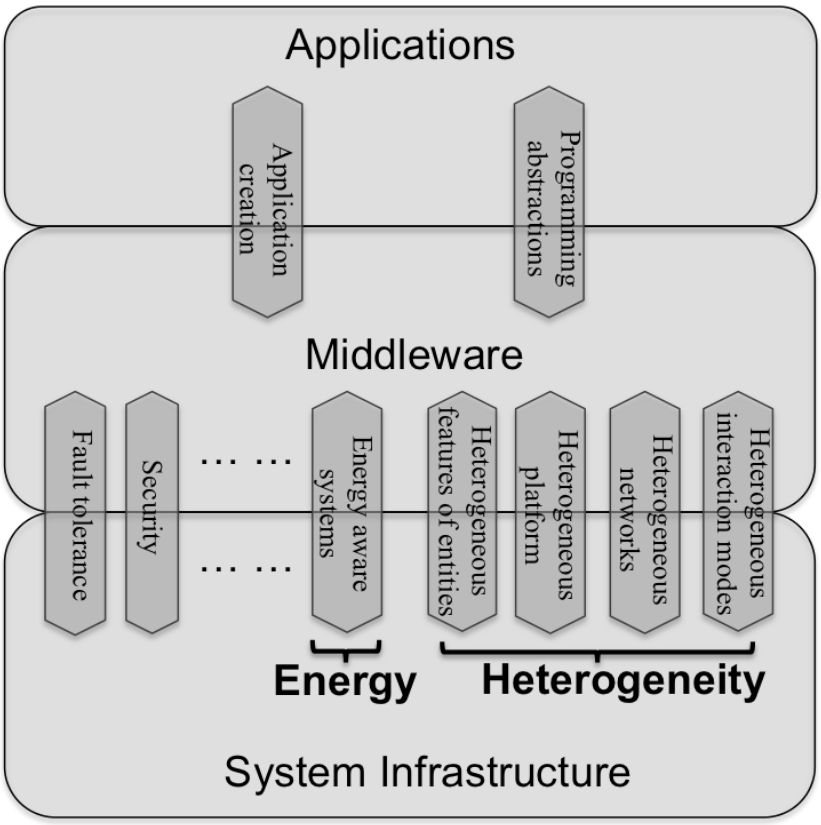


Figure 2: Special requirements of pervasive computing environments

The requirements implied by the system infrastructure are various (e. g. heterogeneous networks, heterogeneous platform, energy concern, security) as shown in Figure 2. These requirements provide many opportunities in different research areas. This thesis focus on two main aspects: heterogeneity and energy concerns. Let us examine different characteristics of these two aspects.

- Heterogeneous Interaction modes - The study of a wide range of entities suggests that three interaction modes are needed: commands, events, and sessions. Commands are used to perform actions on devices (e. g. operating a light or triggering an alarm). Events enable entities to react to situations by publishing information (e. g. motion detection and temperature change). A session mechanism allows to configure a communication channel to exchange data over a period of time (e. g. video streaming from a video camera).

- Heterogeneous networks - Existing devices and services rely on heterogeneous communication technologies (e. g. UPnP, X10, or Web Services), constituting heterogenous networks. The management and coordination of this heterogeneity are necessary.
- Heterogeneous platforms - Numerous platforms are widely deployed in existing pervasive computing environments (e. g. telephony over IP infrastructures or dedicated home automation platforms). The convergence of these platforms can have significant integration cost. The platform has to seamlessly integrate these existing infrastructures.
- Heterogeneous features of entities - The system infrastructure contains numerous entities (hardware and software), providing different features. Technologies advances make these environments in constant evolution, with new devices and features. The extensibility is needed to cope with the integration of entities and their features.
- Energy aware system - Mobile devices are widely used in pervasive computing environments, especially smartphones are rapidly becoming the central computation and communication device. More and more services and applications are available on them. However, smartphones are still energy-limited device if complex signal processing and machine learning algorithms are required. We need to take the energy concerns into account during the application design phase.

Let us examine also the application-level requirements.

- Programming abstractions - Heterogeneity of entities requires an approach for abstracting away entity features that are not relevant to applications (e. g. protocol, model, and firmware version). This approach would allow entities that share common functionalities to be uniformly manipulated.
- Application creation - The development of pervasive computing applications needs to be supported in many aspects: structuring, insurance of innocuousness, tests, evolution, and deployment.

2.2 THE DIASPEC APPROACH

In this Section, we examine the aspects that make a programming framework as an ideal choice to fulfill the application-level requirements (highlighted in Figure 3). We then present the DiaSuite approach that generates a programming framework from the DiaSpec architecture description. This approach covers the entire development lifecycle and provides guidance and support to implementing the application logic.

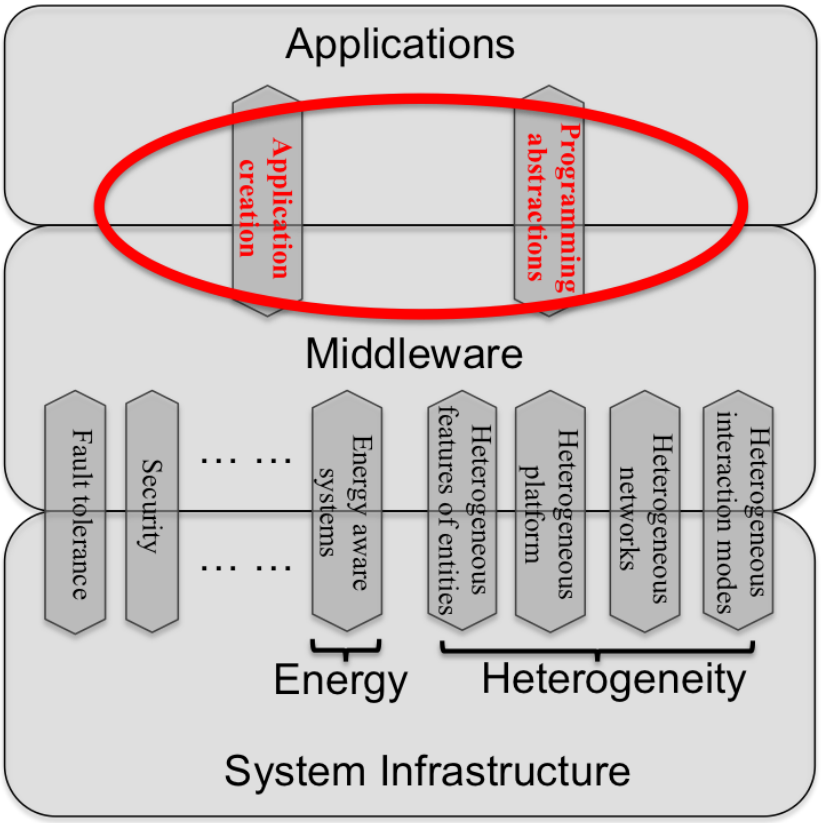


Figure 3: Special requirements of pervasive computing environments

Programming abstractions

As mentioned earlier, a programming framework can factorize the boilerplate code to map the programming support to a communication back-end (e. g. RMI, SIP or local). The programming framework is able to provide the programmer with high-level programming mechanisms, raising the abstraction level of communication operations.

Application creation

An Architecture Description Language [40] (ADL) dedicated to the pervasive computing domain can facilitate the creation of new applications. We can use this ADL to generate a dedicated programming framework that abstracts over the underlying communication layer (e. g. RMI, SIP or local). The dedicated programming framework allows us to quickly develop, adapt, and check our applications. The programming framework can allow the logic for the application to be developed, without requiring the knowledge about the underlying communication layer.

DiaSuite toolkit

After this analysis, DiaSuite as a toolkit can fulfill the application-level requirements. DiaSuite provides a development environment dedicated to pervasive computing systems. The core of DiaSuite is a domain-specific design language for pervasive computing systems named DiaSpec.

DiaSpec [17, 31] is a lightweight Architecture Description Language [40] dedicated to the pervasive computing domain. From a DiaSpec specification, the DiaSpec compiler generates a dedicated programming framework. The generated support provides the developer with high-level programming mechanisms, abstracting over the underlying communication layer (e. g. RMI, SIP or local). This makes it possible to write applications and device wrappers, without knowing about this protocol. Figure 4 illustrates the DiaSpec approach.

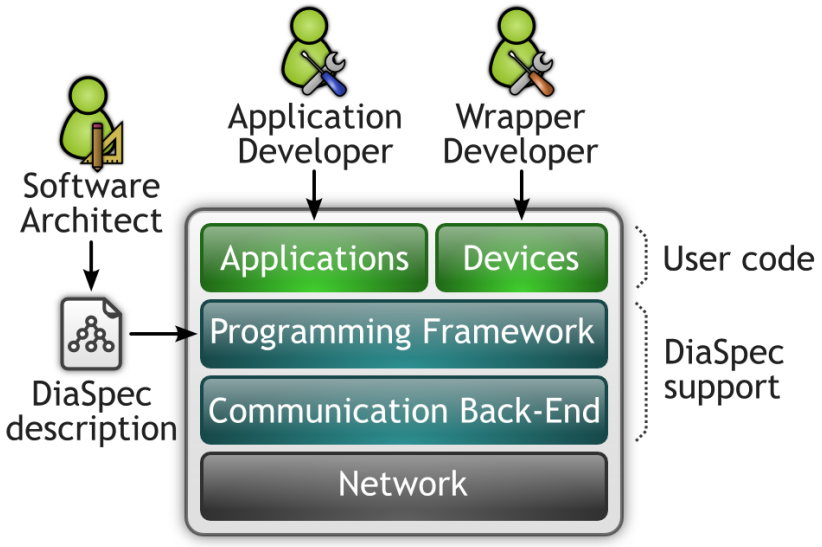


Figure 4: The DiaSpec software structure

2.2.1 *The DiaSpec Language*

A DiaSpec specification defines a taxonomy of entities dedicated to the target application area. It consists of declarations of classes of entities, each declaration gathers entities that share commonalities; their differences are expressed by attribute declarations; and, three connector declarations are used to define their interactions with other devices, namely, events, commands, and sessions. Attributes can represent a constant property (*e.g.,* a colour or a range) or a dynamic state, such as the current location of a mobile object.

Listing 1 describes the architecture of a light regulation application. The environment is composed of lights, light sensors, and a controller able to receive information from sensors and to trigger operations on lights. This architecture illustrates both the command and event interaction modes. As can be noticed, each declared connector is optionally refined with the class of entities it may interact with. For example, the `LightSensor` component provides the `Luminosity` event *to* `LightController` entities. The `LightController` component requires the `Variation` command *from* the `Light` entities.

```
1  component Device(String building, String room) { }
2
3  component Light extends Device {
4    provides command Variation to LightController;
5  }
6
7  component LightSensor extends Device {
8    provides event Luminosity to LightController;
9  }
10
11 component LightController  {
12   requires command Variation from Light;
13   requires event Luminosity from LightSensor;
14 }
15
16 icommand Variation {
17   void increase();
18   void decrease();
19   void setLevel(int value);
20 }
```

Listing 1: A DiaSpec specification

2.2.2 *Programming Support*

From a DiaSpec description, the DiaSpec compiler generates a dedicated programming framework. Conforming to the specified environment, it provides the developer with a Java programming

support, facilitating creation of applications and wrappers. The generated support is independent of a given communication technology. Developers only manipulate high-level distributed programming concepts (e. g. registration, discovery and remote calls).

The generated programming framework is highly customized with respect to a given DiaSpec description. Listing 2 shows the use of this generated support. Implementation of `MyLightController` constructor looks for `LightSensors` in the building `A29`, then subscribes to the `Luminosity` event. The notify method is called if the `LightController` receives a notification of a `Luminosity` event. In this implementation, lights of the room are dimmed or brightened depending on the luminosity event value.

```
1  public class MyLightController extends LightController {
2
3    public MyLightController() {
4      LightSensorComposite sensors =
5        select(lightSensorsWhere().building("A29"));
6      sensors.subscribeLuminosity();
7    }
8
9    @Override
10   public void notify(Proxy servicePublisher, Luminosity event) {
11     LightComposite lights =
12       select(lightsWhere().
13       building(event.building).room(event.room));
14     if (event.luminosityValue < 5000)
15       lights.increase();
16     else if (event.luminosityValue > 6000)
17       lights.decrease();
18   }
19 }
```

Listing 2: A `LightController` implementation using supplied framework

This code fragment illustrates discoveries, subscriptions and commands. Every object used in this implementation comes from the dedicated support, generated from the DiaSpec description shown earlier (Listing 1). This support hides the underlying communication technology. Each technology is addressed by its own back-end.

2.3 SUMMARY

This Section presented DiaSpec, our design language for pervasive computing systems. We illustrated how the logic of applications are designed and implemented throughout the different development phases. We described how DiaSpec provides a

programming framework facilitating the creation of applications and raises the abstraction level of underlying communication bus. But, as we mentioned in the previous Section, application-level requirements are not sufficient, heterogeneity and energy concerns have to be addressed during the development process as well. The next Chapters present our approach to manage heterogeneity and energy concerns into the existing development process of DiaSpec.

3

SIP FOR HETEROGENEITY HANDLING

This Chapter presents the benefits of the SIP communication bus in a pervasive computing context and describes our design approach. We then detail SIP adapters and adaptions needed, connecting entities to the SIP communication bus.

Our approach aims to leverage DiaSuite and a SIP-based infrastructure to developing pervasive computing applications. To do so, we develop a SIP Back-End as the communication back-end of DiaSpec. Several SIP adapters and SIP adaptations are needed to fulfill the pervasive computing requirements. These SIP adapters make each non-SIP entity SIP compliant. These SIP adaptations must be in conformance with the SIP protocol and reuse existing extensions whenever as possible. These two constraints are critical for a seamless integration of our approach in already-deployed SIP infrastructures.

Before presenting SIP adapters and SIP adaptations of our approach, let us examine how SIP fulfills the requirements (heterogeneity) needed by the system infrastructure as mentioned in previous chapter (highlighted in Figure 5).

We elaborate respectively SIP adapters and SIP adaptations to enable SIP as a universal communication bus in Section 3.2 and Section 3.3. We describe the integration of the SIP-based support framework into DiaSpec in Section 3.4.

3.1 A CASE FOR SIP AS A UNIVERSAL COMMUNICATION BUS

Let us examine the aspects that make SIP an ideal basis to form a universal communication bus.

Heterogeneous Interaction modes

Originally designed to deal with sessions, SIP has the potential to provide general-purpose communication forms, namely, commands (RPC-like based on instant messaging [8, 16]), events [50], and sessions of data streams [57]. These forms of communications cover what is required by an application to coordinate entities in a pervasive computing environment. More specifically,

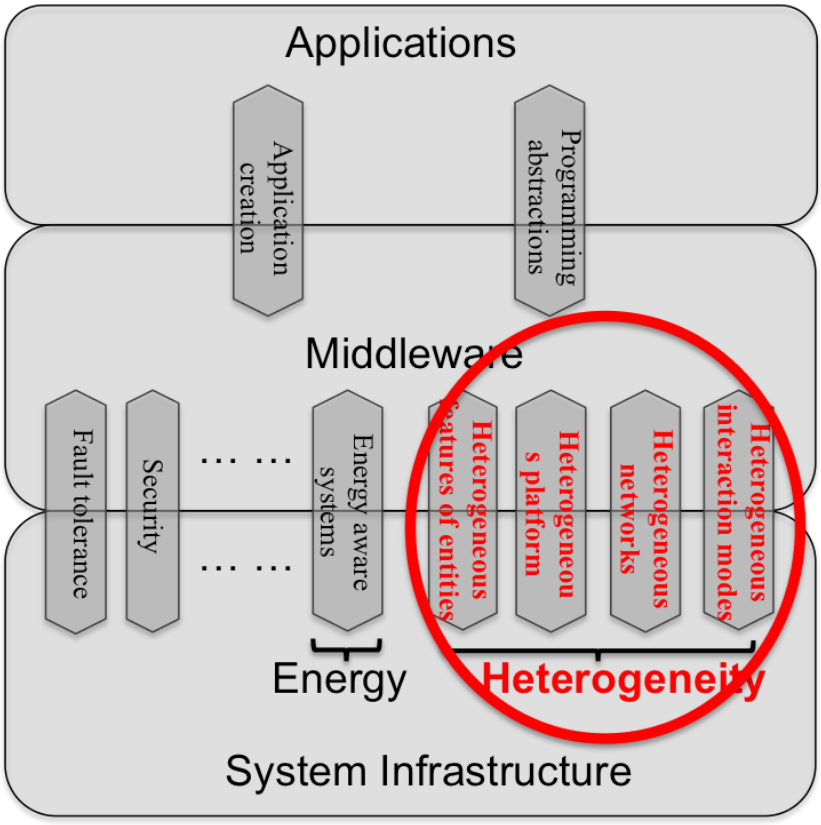


Figure 5: Special requirements of pervasive computing environments

instant messaging is a one-to-one interaction mode; it can be used, for example, to query a temperature measurement from a sensor. To do so, the instant message payload can be interpreted by the recipient to execute actions; and if needed, a return value can be included in the response payload. Event is a one-to-many interaction mode; it is the preferred mechanism to propagate information such as the presence status. Finally, session is a one-to-one interaction mode with data exchanged over a period of time; it is typically used to set up a multimedia stream between two entities, but it can be generalized to a stream of arbitrary data. For example, a GPS device produces a stream of Cartesian coordinates.

Heterogeneous networks

In pervasive computing systems, heterogeneous networks (e. g. UPnP, X10, ZigBee, and SIP) that connect real services provide different mechanisms and interaction modes (e. g. command, event and session). SIP can provide these different mechanisms. Moreover, the extensibility of SIP facilitates the integration to wrap several proprietary protocols into SIP, including X10, to control binary and dimming devices, ZigBee, to take advantage

of various sensors (e. g. temperature, light), HTTP, to control video cameras (e. g. motion, zoom, snapshot), and Web Services, to get agenda information or TV programs.

Heterogeneous platforms

Because it is a *de facto* standard for IP telephony, SIP platforms are already widely deployed in various forms, including dedicated IP telephony systems, ADSL modems, and set-top boxes. Importantly, SIP infrastructures are more open-ended than past proprietary infrastructures. They often offer extended interfaces to develop applications. Pervasive computing applications can thus leverage these platforms, expanding their original scope. Moreover the increasingly prevalent nature of SIP makes it a converging point for many technologies. Beyond SIP phones (whether hardware or software), other SIP-compliant entities are starting to become available (*e.g.*, video camera[1]). In fact, SIP is likely embedded in an increasing number of devices and software systems, representing a convergence point of a number of technologies and areas.

Heterogeneous features of entities

As illustrated in the previous chapter, the extensibility is needed to cope with the integration of entities and their features. SIP is an HTTP-like request/response protocol, text-based and transport-independent. Like HTTP, SIP is extensible in terms of methods, headers, and message payload. This allows the protocol to be completed with numerous standardized extensions matching specific needs, namely, instant messaging [8, 16], and events [50]. Message payload is format-independent, enabling SIP to embed any kind of data (e. g. SDP [24], presence information [65], and SOAP [69]).

Moreover, SIP provides another important feature that can handling the environment dynamicity. Dynamicity is an inherent feature of a pervasive environment. SIP provides a mechanism that deals with a form of dynamicity, namely user mobility. To address this issue, SIP relies on the use of Uniform Resource Identifiers (URIs) to refer to agents, abstracting over the terminal network address. This mechanism can be used to define functional entities in a pervasive environment, abstracting over concrete entities whose availability may vary over time. As a result, the use of SIP URI shields the application code from runtime configuration changes in the environment.

1 Mobotix SIP Cameras, `http://www.abptech.com/products/Mobotix/`.

3.2 BUILDING SIP ADAPTERS

We have motivated the use of SIP as a universal communication bus between heterogeneous distributed entities. Let us now examine how entities need to be adapted to connect them to the SIP communication bus. This adaptation process is driven by criteria, classifying entities.

3.2.1 *Entity classification*

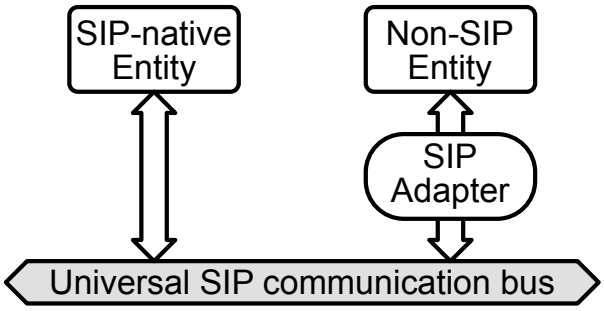


Figure 6: Adapting entities to SIP

Our entity classification uses three criteria. This classification builds on our study of a large panel of entities and factorizes our experience in developing entity-specific adapters to the SIP communication bus. The first criterion is whether or not an entity is SIP native. As shown in Figure 6, a SIP-native entity is directly connected to the SIP communication bus; such entity is referred to as *type 1*. In contrast, a non-SIP entity needs an adapter. To address a non-SIP entity, a second criterion identifies whether it is IP-enabled. If so, a third criterion determines whether the entity is programmable, making it possible to introduce a SIP stack; this class of entities is of *type 2*. *Type 3* is a non-SIP, non-programmable entity; as such, it requires the use of a SIP gateway. *Type 4* is a non-SIP entity without IP capability, requiring an extended gateway. This classification of entities is summarized in Figure 7. Examples are listed in Table 1. From this classification, solutions are proposed to create SIP adapters.

3.2.2 *Functional architecture of a SIP adapter*

We now present the layers required to adapt each class of entities to the SIP communication bus, omitting entities of type 1 that support SIP natively. To be SIP compliant, an entity must provide access to its functionalities via SIP-compliant mechanisms. To do so, access to entity functionalities are defined in terms of the three

Type	Examples	Gateway
1	SIP video camera, SIP phone, SIP softphone	No
2	PDA, Greenphone, Calendar, Monitoring entities	No
3	IP video camera, Printer	Yes
4	X10 or 1-Wire devices, Temperature sensors	Yes

Table 1: Entity examples

interaction modes available in SIP: commands (*i.e.,* status query and entity control), events (*i.e.,* event publishing and subscription) and sessions (*i.e.,* invitation to a session of data stream). Yet, these interaction modes need to pass and receive data that may have different formats: command-parameter values (*e.g.,* using SOAP), event values (*e.g.,* using an XML-based format [52]) and session-capability descriptions (*e.g.,* using plain text SDP).

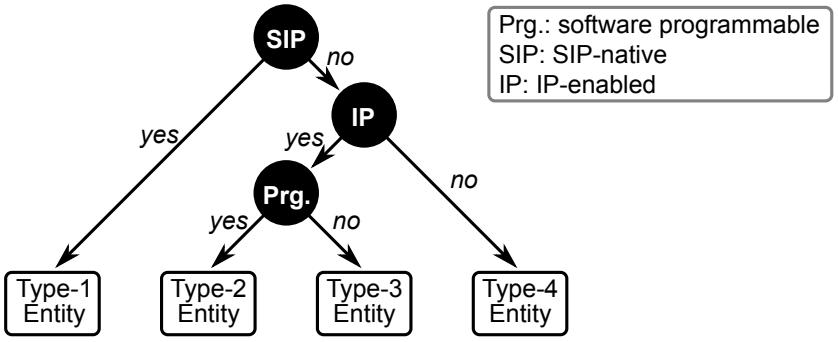


Figure 7: Entity Classification

Entities of type 2

Despite SIP's rich forms of communications, the SIP communication bus needs careful parameterization to cope with a constant flow of new non-SIP entities, introducing ever changing functionalities and data formats. To address this situation, SIP adapters wrap entity functionalities with an *interpreter*. For a given SIP method, this layer extracts from the payload of a SIP message, the constituent parts of the corresponding interaction mode (*i.e.,* command, event or session). For example, a SIP request with a `MESSAGE` method corresponds to a command interaction. The payload interpreter then extracts from the request payload a SOAP message, indicating the command name (*e.g.,* `getTemperature`) and the parameter values (*e.g,* a measurement unit). The *payload interpreter* then calls the *invocation layer* of the corresponding inter-

action mode (*i.e.,* command, event or session) with its constituent parts. This layer is responsible to invoke the target functionality in the entity (*e.g.,* an operation to measure a temperature, given a measurement unit). Figure 8 depicts the layers involved in adapting a non-SIP, programmable entity to the SIP communication bus. Because a type-2 entity is programmable, its SIP adapter can reside on the entity, making it self-contained.

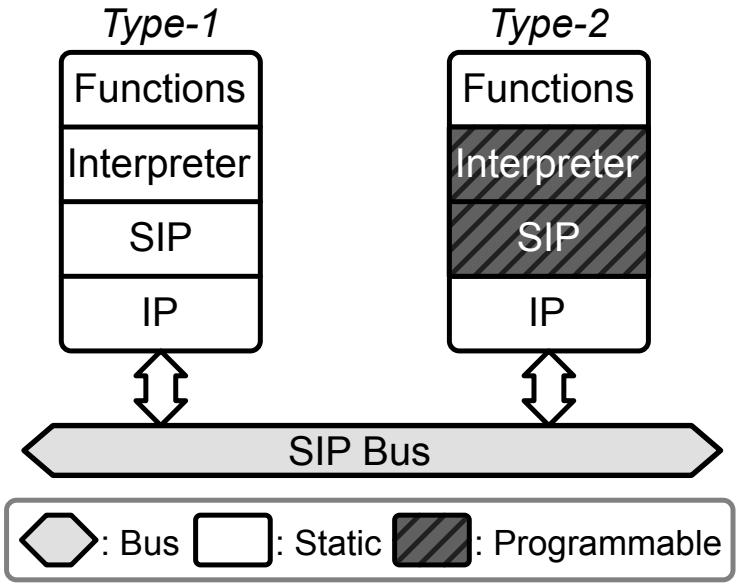


Figure 8: SIP adapter for entities of types 1 and 2

Entities of types 3 and 4

When a non-SIP entity is not programmable, the SIP adapter is implemented as a hardware gateway. Note that a hardware gateway can also be used for a type-2 entity to reduce energy consumption or increase performance. A hardware gateway is mandatory for a type-4 entity to enable IP and SIP capabilities.

As illustrated in Figure 9, functionalities of entities of types 3 and 4 are accessed through ad hoc communication buses consisting of a software communication bus and an associated hardware communication bus. The hardware communication bus can be proprietary. It may simply be the processor bus of the device. Requested data can be directly accessed via registers mapped in memory. The hardware communication bus can also implement an industry standard such as X10 [9], for power line-based communication, and ZigBee [10], for wireless communication. There are low-level devices that use serial communication buses such as RS232, I2C, or 1-Wire bus [38]. In our approach, these devices are hidden behind a SIP-compliant component that directly accesses their functionalities. In fact, each time a hardware communication

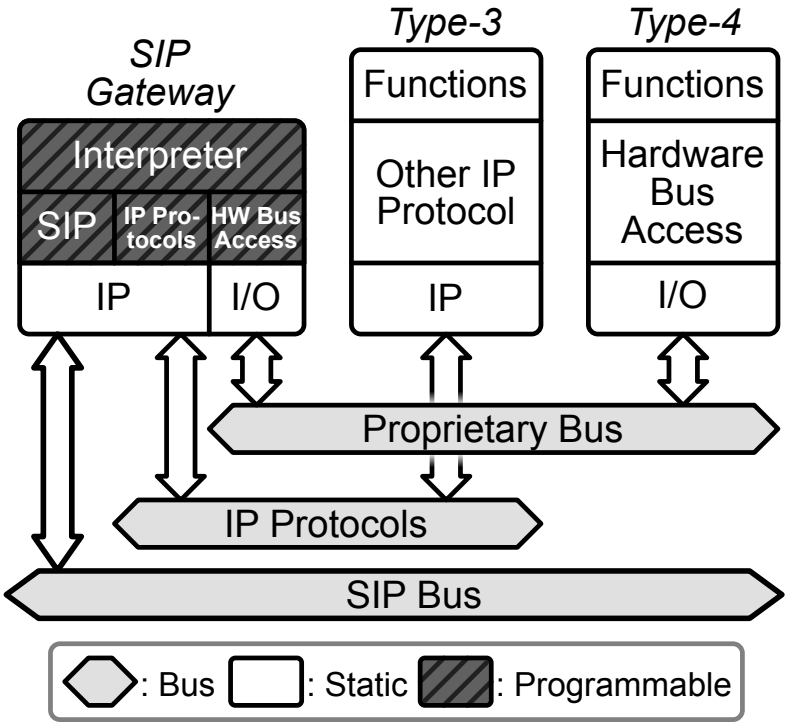


Figure 9: Entities of types 3 and 4 and gateway architecture

bus is used, the corresponding specific software communication bus must be created for hiding underlying hardware specificities.

In practice, our four classes of entities and our methodology have been successful in adapting all the devices and software components that we have encountered in developing a variety of pervasive computing applications.

3.3 ENABLING SIP COMMUNICATION

Because SIP is HTTP-inspired, it is highly versatile and extensible. These features make this protocol amenable to be used in a range of extended forms of communications such as instant messaging [16, 33], presence [52], as well as converged applications combining telephony with software systems [32]. Furthermore, SIP enables to transport a range of data formats by leveraging such protocols as SDP [24], PIDF [65], and SOAP. More generally, SIP is widely used in ToIP infrastructures, ranging from small domestic environments to country-wide telephony networks.

Our approach aims to leverage a SIP-based infrastructure and its key benefits to developing home automation applications. To do so, several adaptations are needed to fulfill the home automation requirements. These adaptations must be in conformance with the SIP protocol and reuse existing extensions whenever possible. These two constraints are critical for a seamless integration of our approach in already-deployed SIP infrastructures. These adaptations are described in the next section.

To facilitate the use of our approach by developers, we provide them with a complete programming framework on top of SIP and our adaptations. Thanks to this support, developers manipulate high-level concepts (e. g. service discovery) in Java. As well, a developer does not have to forge SIP messages nor manage SIP transactions, preventing him from writing boilerplate code.

Because this programming support is generated with respect to a description of the home environment, it guides the development of applications with typed operations and methods to be implemented.

This section presents how some requirements can be directly mapped into SIP and what adaptations are needed to address the remaining ones. Our proposed adaptations revolve around the use of SOAP to transfer arbitrarily rich data. SOAP messages are embedded in SIP message payloads. These adaptations are designed to be in conformance with SIP platforms and SIP-native agents.

3.3.1 *Registration and discovery*

Dynamicity is an inherent feature of home automation. New entities appear and disappear over time (e. g. SIP phones are switched on/off). SIP provides a mechanism that deals with a form of dynamicity, namely user mobility. To address this issue, SIP relies the use of Uniform Resource Identifiers (URIs) to refer to agents, abstracting over the terminal network address. We can

use this mechanism to identify an entity, whether hardware or software, by its name expressed as a URI. Entities can thus be viewed as SIP agents that can be found via their URI locally as well as throughout the Internet.

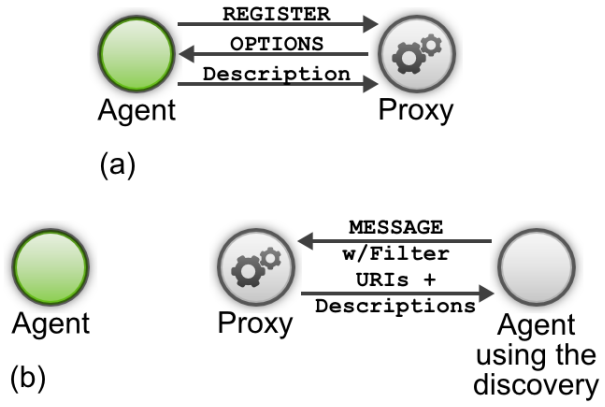


Figure 10: Registration (a) and discovery (b) processes

For existing SIP services

To achieve service discovery, SIP entities register their SIP URI with the registration server. This server associates SIP URIs and network addresses of the entities (e. g. IP address and port). When communicating, SIP entities have their network address looked up from their URI by the proxy server. The SIP URI is a useful building block to provide service discovery. However, it does not take into account attributes that are needed to refine entity selection.

For non-SIP compliant services

Although rudimentary, this SIP mechanism can be used to achieve entity discovery. We propose to extend the registration process of SIP agents with a semantic description of entities. To do so, we decompose this process into two steps. The first step is the normal registration procedure; it uses the `REGISTER` message, like native SIP agents. This allows entities to register and provide their URI and network information as mentioned earlier. Unfortunately, a URI does not give any information about the nature of its associated entity (e. g. device type and location). To obtain additional information from an entity, we add a second step to registration: we query an entity to handle the `OPTIONS` message. If the entity belongs to our approach, it returns an enriched response describing itself in terms of attributes and its type name. An attribute is property-value pair characterizing the

entity. The type name denotes a set of entities that shares the same functionalities (e. g. a light, a fan, and an alarm), allowing the application to manipulate them uniformly. The type name of an entity and its attributes are used by the application to discover entities in a given home environment. A query for entity discovery takes the form of a `MESSAGE` message including the search criteria. The response message contains a list of matching entities. This exchange is built as a command invocation, described below. Figure 10 summarizes the registration process and the entity discovery.

3.3.2 *Commands*

Home automation devices often make their functionalities accessible via an RPC-like command, which is a one-to-one operation between two services. A command invocation consists of a name and argument values; it produces a return value. We implement this mechanism in SIP using extensions for instant messaging [16, 33]. A command invocation is encoded using SOAP. The caller builds a `MESSAGE` message with a SOAP payload. The targeted service decodes the SOAP message and executes the command code. When the execution completes successfully, the result is encoded into a 200 `MESSAGE` response. Otherwise, an error message is returned.

3.3.3 *Events*

Events are based on the publish/subscribe paradigm [50, 56]. In this model, an entity publishes its events to an *event notifier*, which in turn notifies the subscribing entities. This is a one-to-many interaction, where the publisher does not know subscribers (illustrated in Figure 11). A subscriber targets a particular event coming from a specific entity (*i.e.*, a URI). Several existing SIP extensions address this interaction mode [57] but none allows arbitrary payloads for event subscription and publication.

We lift this limitation by introducing SOAP over SIP event messages. `PUBLISH` request body consists of the SOAP-encoded event name and event value. Similarly, `SUBSCRIBE` message consists of the SOAP-encoded URI of the publisher and the event name. In addition to the contents of a `PUBLISH` payload, a `NOTIFY` payload includes the URI of the event publisher to pass this information to the event subscribers.

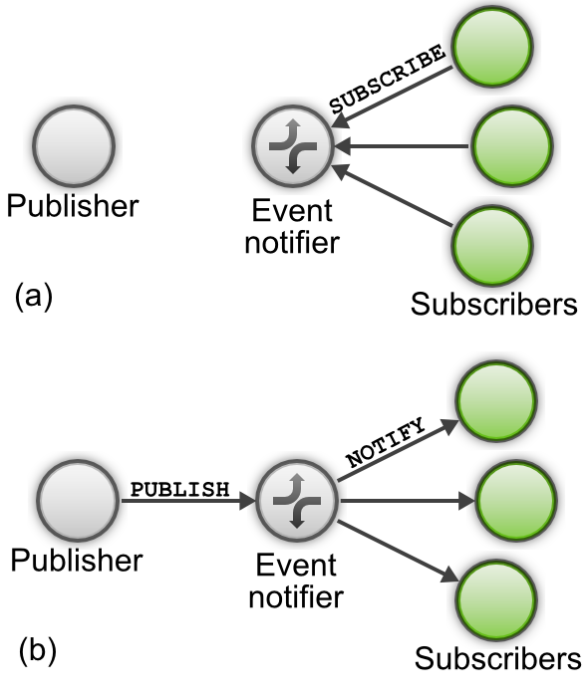


Figure 11: Event interaction mode: (a) subscription (b) publication

3.3.4 *Sessions*

A SIP session supports multimedia sessions that are described by using SDP. Negotiation of SDP session parameters is based on the offer/answer model [53]. Once negotiated, the session is started and a data stream is transmitted. For delivering multimedia streams, the Real-time Transport Protocol (RTP) is widely used [62]. Today, combination of SDP with RTP is widely used to deal with multimedia sessions from negotiation to streaming. However, this combination is limited when addressing other kinds of streams. Yet, emerging applications combine telephony with sensors for tracking people's location, detecting intrusion, monitoring devices, *etc.*

To address these new situations, our approach consists of generalizing SDP to any Java data type, while re-using its negotiation model. Java data types are used by DiaGen [17] to generate a codec that takes the form of a serializer, allowing RTP to transmit streams of any data type. In doing so, our approach leverages existing technologies and APIs such as JMF [42].

3.4 SIP BACK-END

We have implemented a SIP back-end for DiaSpec(illustrated in Figure 12). It is responsible for mapping DiaSpec concepts into the ones of SIP. To do so, our back-end makes use of the SIP adaptations described earlier.

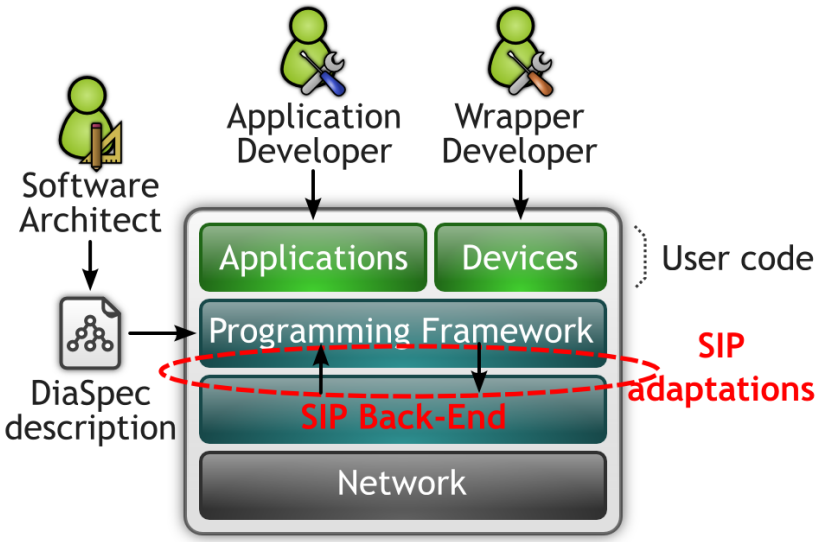


Figure 12: The DiaSpec software structure

3.4.1 *SIP message support*

The SIP back-end is mainly responsible of the management of SIP transactions et dialogues. As a consequence, the back-end builds, sends, receives, and processes SIP messages. It also has to deal with transaction states, and thus, with timeouts, message loss, request re-emission, *etc.* This task usually involve a large amount of boilerplate and error-prone code. In opposition to ad-hoc application development, the code is decomposed and factorized for each concept, namely service discovery, session, event, and command. Our implementation of the back-end leverages the Jain-SIP library for message parsing and generation.

3.4.2 *Message bodies*

The SIP back-end is also in charge of encoding and decoding the payload of messages, serializing and deserializing Java objects. Depending on their nature, data payloads are parsed using the correct library (e. g. kSOAP for SOAP payloads, or Jain-SDP for session negotiation data). Figure 3 illustrates a command `increase` sent to a `Light`, resulting from a call to the instruction

`increase()` (Figure 2, line 5). The choice of the payload format depends on the Java data structure and the interaction mode (e. g. SDP is chosen for audio session negotiation).

```
1  MESSAGE sip:Light.Kitchen@home.com SIP/2.0
2  From: <sip:MyLightController@home.com>;tag=cefd113d
3  To: <sip:Light.Kitchen@home.com>
4  Call-ID: 2ad17cb28971a961a669411c6acc2c64@home.com
5  CSeq: 11 MESSAGE
6  [...]
7  User-Agent: DiaSpec v1.1
8  Content-Type: application/soap+xml
9  Content-Length: 327
10
11 <v:Envelope
12  xmlns:i="http://www.w3.org/2001/XMLSchema-instance"
13  xmlns:d="http://www.w3.org/2001/XMLSchema"
14  xmlns:c="http://www.w3.org/2001/12/soap-encoding"
15  xmlns:v="http://www.w3.org/2001/12/soap-envelope">
16  <v:Header />
17  <v:Body>
18   <n0:increase id="o0" c:root="1" />
19  </v:Body>
20 </v:Envelope>
```

Listing 3: A SIP message generated by the back-end

3.4.3 *Service discovery*

The discovery service allows to register and look up entities. SIP provides a basis to deal with the dynamic pervasive computing environments via its support for user mobility. Specifically, SIP entities send a SIP `REGISTER` request to register their SIP URI with the registration server; this server associates entity SIP URIs with network addresses. In addition, our approach consists of using the SIP `OPTIONS` request to complete the registration process with a description of the registering entity.

Once registered, an entity can be looked up by querying the registration server. To do so, a lookup request is sent in a SIP `MESSAGE` request, containing a description of the required entity or entities. The registration server returns all registered entities matching the request.

For example, our approach for service discovery provides the programmer with an abstraction layer over SIP URIs. As illustrated in Figure 2 line 5, a discovery query contains the target device class (e. g. `Light`) and the attribute values the device must match (e. g. building number is the A29). Matching entities are returned as Java objects, containing agents' URI (hidden to the developer). However, the developer has access to attribute values and declared operations.

3.4.4 *Interaction modes*

The notification service allows entities to subscribe and publish events. It improves the scalability of the overall platform by decoupling producers and consumers of events. The notification service receives SIP `PUBLISH` requests containing events from publishers and sends SIP `NOTIFY` requests to all entities that subscribed to the related type of events (*e.g.,* calendar event) using the SIP `SUBSCRIBE` request.

In addition to the event interaction mode, the DiaGen middleware allows entities to interact via the command and session interaction modes. In the command interaction mode, an entity sends a SIP `MESSAGE` request to operate another entity. In the session interaction mode, an entity sends a SIP `INVITE` request to negotiate session parameters and to establish a session with another entity.

Exchanged data are serialized in the SOAP format using kSOAP [5] and transported via SIP request and response bodies. kSOAP is a SOAP Web service library for resource-constrained Java environments such as J2ME applications. The SOAP format is standardized, XML-based and widely used. Although verbose, SOAP is increasingly supported by embedded systems. Furthermore, some implementations are now written in C, providing high performance and low memory footprint (*e.g.,* XPath offers an API for manipulating XML in embedded systems).

3.5 SUMMARY

This chapter presented the benefits of using SIP as the universal communication bus in a pervasive computing context. We presented a classification of a wide variety of entities in terms of features, capabilities and network connectors. Based on this classification, a methodology and programming support are described for connecting entities on the SIP communication bus. We developed SIP adapters for non-SIP entities to make them SIP compliant. The SIP adaptation layer is developed to enable distributed entities to exchange any media using various interaction modes. Also, we described the integration of the SIP-based support framework into DiaSpec. We are now ready to evaluate our approach with an experimental study; this work is described in the next chapter.

4

EVALUATION OF THE PLATFORM

To assess the feasibility of our approach to heterogeneity handling, we applied it to our experimental platform. This experimental study allow us to evaluate the performances of our SIP communication bus for each type of entities (discussed in the previous Chapter). We also evaluate DiaSpec with a SIP back-end in the application development for home automation. We focus on three aspects: flexibility, accessibility and expressiveness. We then discuss on the related works.

4.1 EXPERIMENTAL PLATFORM

Our universal communication bus has been developed in the context of a home automation project. The goal of this project is to design and implement a home automation platform based on SIP. Experiments have been made in a real environment, depicted in Figure 13. This environment was populated by various home automation entities, ranging from telephony equipments to home appliances.

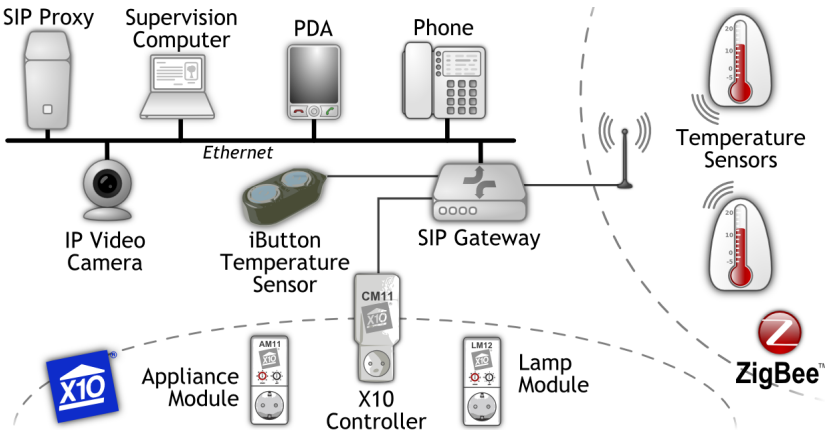


Figure 13: Experimental platform

This platform serves as a vehicle to experiment with various scenarios. For example, we have developed a surveillance application that involves IP video cameras, X10 alarms, SIP phones and PDAs. Another example is an application displaying various information of interest on a screen, including appointments and weather conditions.

4.2 EXPERIMENTAL STUDY

In this section, we validate our use of SIP as a universal communication bus. This validation is done in the context of our experimental platform, equipped with entities belonging to all the types discussed earlier. First, we examine the adaptation work required for each type of entities. Then, we present and analyze performance measurements.

4.2.1 *Entity adaptations*

Let us examine the development work required to make each entity type SIP compatible.

Type-1 entity

By design, the DiaGen middleware is fully compatible with SIP-native entities. Application code developed with the DiaGen middleware can thus directly interact with these entities. This situation allows to leverage existing SIP infrastructures (*e.g.,* OpenSER server) and entities (*e.g.,* SIP video cameras, SIP phones and softphones).

Type-2 entity

There exists a wide variety of existing entities with programming capabilities, ranging from PDAs to software calendars. Our approach consists of providing the developers with a Java programming framework to create invocation layers and to connect entity functionalities to the SIP communication bus. Developers rely on high-level operations to (1) register and lookup entities and (2) implement and invoke entity functionalities. Our Java programming framework abstracts over the intricacies of the underlying technologies and prevents developers from writing boilerplate code, *e.g.,* SIP method creation, payload marshalling/unmarshalling and concurrency handling.

Type-3 entity

The type-3 category consists of non-programmable entities, supporting IP protocols (*e.g.,* HTTP and RTSP for IP video cameras). Making these entities SIP compliant amounts to develop adapters mapping their protocol into SIP. Such adapters form a SIP gateway. Our programming framework provides support for the developers to build such gateway. We propose two approaches

to implement a gateway. The first approach is based on Java and requires adequate resources in the platform. The second approach is less resource-demanding: it relies on a C version of our programming framework. We chose this second approach and embedded a C-based gateway into a small Single-Board Computer (SBC) (*e.g.,* an ARM-based board [2]). The functional architecture of a SIP gateway is shown in Figure 14.

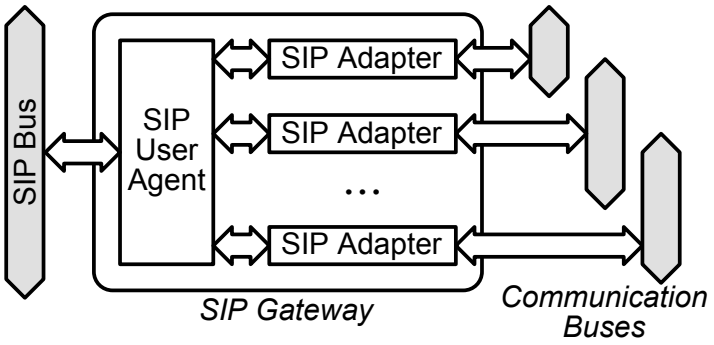


Figure 14: Software architecture of a SIP gateway

To develop our SIP gateway, we have first ported Linux [6] 2.6 with its own root file system to the SBC board. The GNU oSIP library [7] has then been ported to the SBC board, and a SIP user agent has been developed on top of this library. When deployed, the SIP user agent registers each entity it serves.

To illustrate the use of our SIP gateway, consider the IP video camera. A surveillance entity sends an `INVITE` request to the SIP gateway of the IP video camera to establish a session of video stream with a PDA. The SIP gateway extracts appropriate information from the SIP message and sends an RTSP request to the camera. Once the communication is established, the SIP gateway is no longer involved and the video is streamed directly from the camera to the SIP client of the PDA using the RTP protocol.

Type-4 entity

The type-4 entities represent the majority of devices deployed in a typical home environment. This type consists of entities that are non-programmable and communicate with a non-IP protocol. An adapter needs to be developed for every entity relying on a new protocol. This type of adapter is difficult to write because it involves low-level communication operations. We gather the adapters for non-IP protocols into a SIP gateway (Figure 14). This gateway resides in another SBC board (shown in Figure 15) with specific interfaces (*e.g.,* ZigBee and X10). For ZigBee entities, we have developed our own specific ZigBee SIP adapter to get the temperature measurement from the ZigBee temperature sensor

via the serial ZigBee base connected to the SBC board (shown in Figure 16).

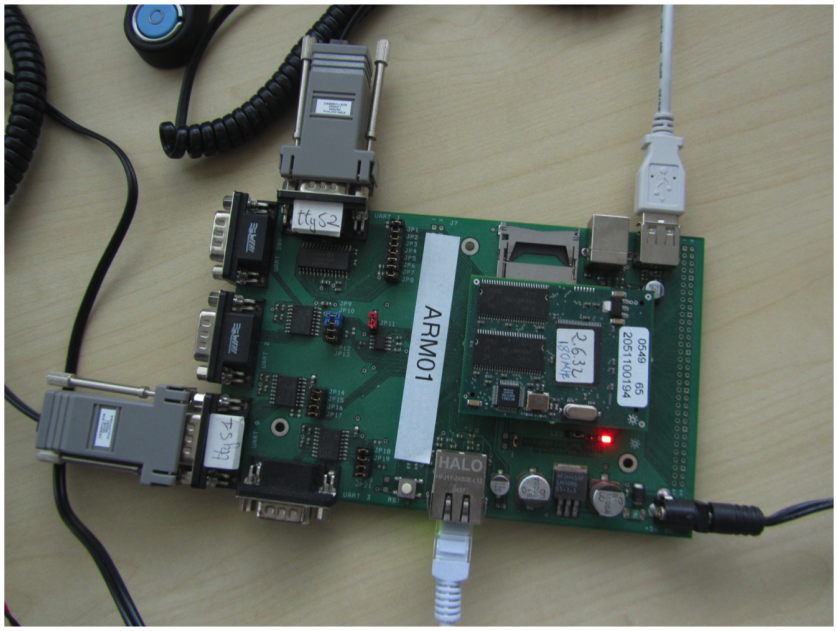


Figure 15: Hardware platform of the SIP gateway

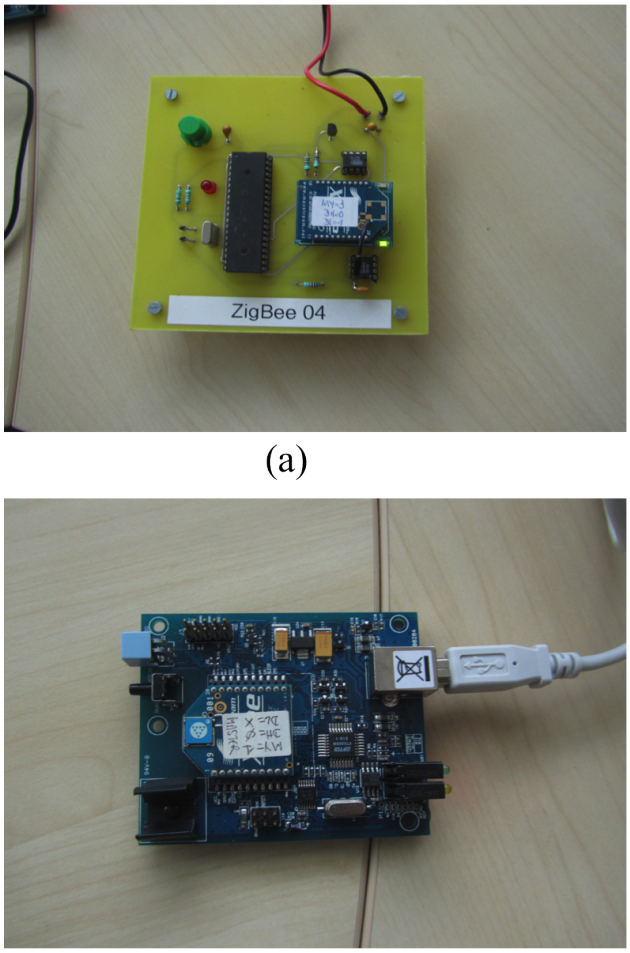
For X10 entities in Figure 17, we have ported the Heyu open source project [3] to the SBC board. A specific SIP adapter has been written. A USB CM11 module, which handles several X10 devices, is connected to the SIP adapter. It receives X10 commands from the adapter and sends them to X10 entities connected to the power line network.

For iButton temperature sensor entities in Figure 18, we have modified an open source library developed by Dallas Semiconductors [4] and ported it on Linux. A specific iButton SIP adapter has been written.

4.2.2 *Results and Discussion*

This section assesses the validity of our approach. To do so, we have conducted experimental studies to measure the performances of our platform and its scalability. We omit the analysis of type-1 entities because they are SIP native and provide the required performances by design. In practice, all type-2 entities we encountered offer enough computing power to map functionalities into the operations supported by an entity. As a result, this category of entities incurs negligible overhead.

Type-3 and type-4 entities both require a SIP gateway. However, type-4 entities are the most demanding in terms of computing power because they translate a high-level protocol, namely SIP,



(a)

(b)

Figure 16: ZigBee module: (a) ZigBee base (b) Our own ZigBee sensor

(a)

(b)

Figure 17: X10 module: (a) CM11 (b) LM11

into a low-level one, such as ZigBee or iButton. Moreover, the type-4 entities represent the vast majority of the devices deployed in a typical home environment. Consequently, our experimental study concentrates on the type-4 entities.

Figure 18: iButton device

	SIP adapters			SIP user agent	SIP Gateway
	ZigBee	iButton	X10		
Memory footprint	113 KB	107 KB	326 KB	350 KB	518 KB

Table 2: SIP gateway memory footprint

Our experimental platform includes a SIP gateway that adapts two ZigBee temperature sensors, two X10 entities and an iButton temperature sensor to the SIP communication bus. For the implementation, we used a 180 MHz ARM9 processor running Linux 2.6.32 with 32 MB SDRAM and 8 MB flash memory.

First, we measure the memory footprint of the run-time support of our implementation, using the Exmap-console tool[1].

This measurement was performed on the adapters and the user agent of the SIP gateway. Their sizes are shown in Table 2. On our resource-constrained platform, the memory footprint of the entire SIP gateway is 518 KB, representing less than 2% of the total available memory of the SBC board (32 MB SDRAM). Note that this SIP gateway comprises three SIP adapters. These figures demonstrate that adapters for non-IP protocols incur minimal overhead, making our approach realistic to resource-constrained platforms.

In our implementation, a command or an event is encoded in SOAP. Like SIP, SOAP uses textual representation. As a result, message processing is much more computation intensive than binary encoding such as BER [1]. However, SOAP deals with complex data structures, facilitates interoperability and enables extensibility.

1 Exmap-console tool, `http://labs.o-hand.com/exmap-console`.

Run Time	Mode	User agent	Adapters	Total
ZigBee	IM	15.3 ms	175 ms	190.3 ms
read	PUB	6.4 ms	175 ms	181.4 ms
iButton	IM	15.3 ms	557 ms	572.3 ms
read	PUB	6.4 ms	557 ms	563.4 ms
X10 write	IM	15.3 ms	373 ms	388.3 ms

Table 3: SIP gateway run-time overhead
IM: Instant messaging for command; PUB: Publish for event

Table 3 reports run time of our SIP gateway. The first column lists read and write operations on ZigBee, iButton and X10 devices. The second column gives the mode of the read/write operation, which can either be implemented as an instant message or an event publication. The remaining columns provide the execution time of the implementation mode, the adapter and the total time, respectively.

We observe that the SIP user agent executes an event (less than 7 ms) twice as fast as a command (less than 16 ms). This is due to the fact that a command produces a full-fledged return value, whereas an event returns a status. Examining the measurements of the adapters, we note that their run times vary widely. This variation depends on the nature of the non-IP protocols. Specifically, the iButton sensor uses a 1-Wire bus through a serial interface that is much slower than the other communication buses. This results in making the iButton adapter a bottleneck (more than 550 ms), compared the processing of SIP messages performed by the user agent (less than 16 ms).

In fact, one can notice that the processing time of the user agent is 10 to 90 times faster than the adapters. This observation leads us to introduce a multithreaded SIP gateway to optimize the SBC board resources. We used POSIX threads to cache values of sensors. It allows to increase scalability of our SIP gateway. Our implementation deals with more than 60 commands (1308 bytes per command on average) or 150 events (1346 bytes per event on average) per second. Based on the interactions we had with our industrial partners in the telecommunication domain, this performance fulfills the requirements of realistic home environments.

To evaluate our SIP gateway, we also measured the run time of our implementation, varying the processor frequency from 180 MHz down to 80 MHz. The variation of the performances is shown in Figure 19. We observed that decreasing the processor frequency logically increases the run time to handle a command or event, almost inversely proportionally, as shown in Figure 19-(a). However, the execution time to read or write

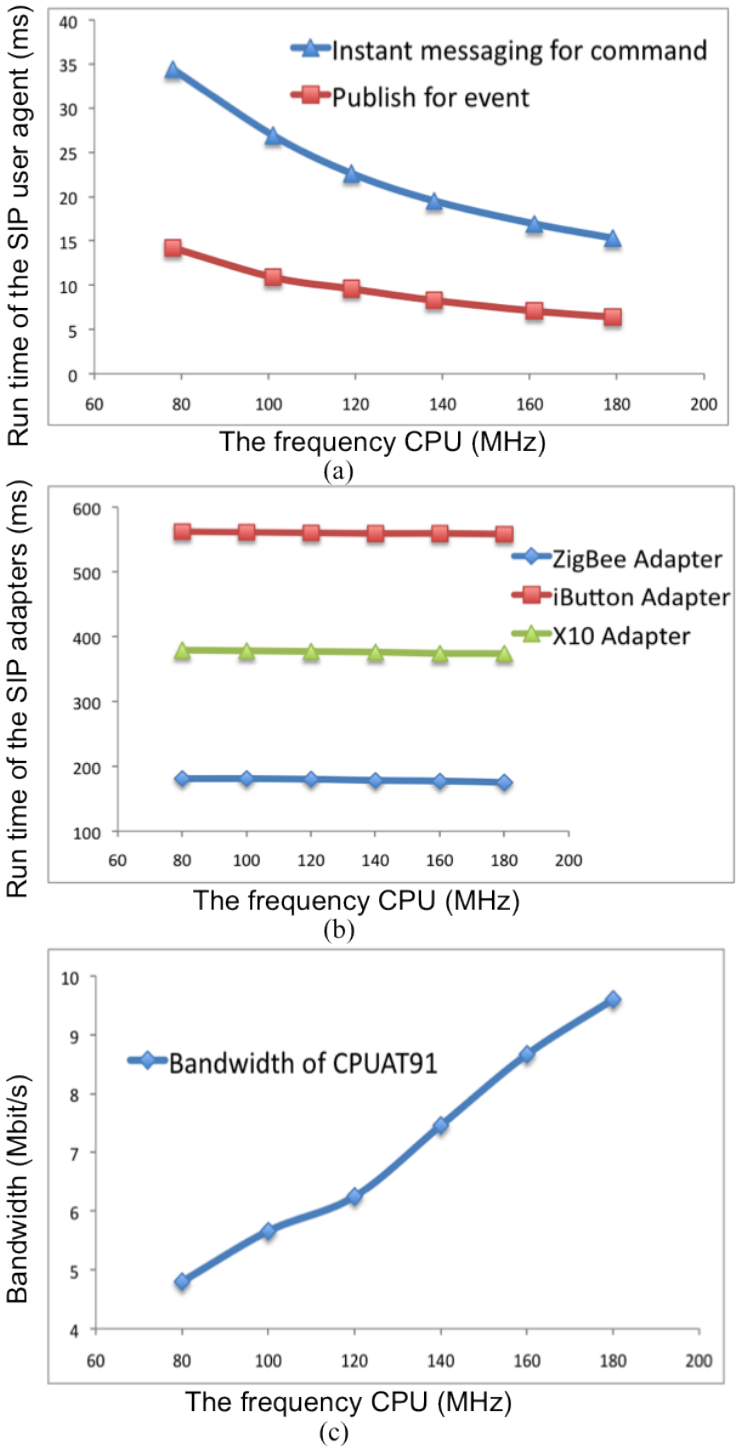


Figure 19: Performances as a function of CPU frequency: (a) Variation of run time of two interaction modes of the SIP user agent with varying the frequency CPU (b) Variation of run time of read or write operations of the SIP adapters with varying the frequency CPU (c) Variation of bandwidth for our SBC board with varying the frequency CPU

a value in entities is almost constant in Figure 19-(b), since this operation depends on the nature of the target proprietary bus. Thus, with threads, the maximum bandwidth provided by our SIP gateway for command or event is practically linear in the processor frequency. It is shown in Figure 19-(c). It allows users to scale the hardware to meet the requirements of the target environment. The result of our experimental study with ARM resource-constrained platforms has proven that our approach has the potential to be integrated into a gateway of lower power to reduce costs.

4.3 EVALUATION OF THE DIASPEC WITH A SIP BACK-END

In this section, we illustrate the practical benefits of our approach, which leverages DiaSpec with a SIP Back-End, through the realization of a wide range of applications (*e.g.*, three scenarios in Section 2.1 and various others scenarios defined by Orange Labs in the home automation domain). For example, we have also developed a multimedia content alert system that informs users about their preferred TV programs. Another example is an application displaying various information of interest on a suitable screen, including appointments and weather conditions. This application involves various type screens and presence detectors through various means (e. g. RFID badges and fingerprint).

These experiments allowed us to validate our SIP-based platform and our approach in practice. We now discuss the practical benefits of our integrated programming framework by doing this work.

Flexibility management for dynamic environments

The service discovery mechanism in our integrated programming framework has proven its effectiveness in dealing with highly-dynamic environments where network appliances are frequently introduced/turned on and removed/turned off.

Thanks to class-based definitions, the discovery mechanism allows applications to dynamically and transparently integrate various kinds of devices defined in a DiaSpec area. This capability enables our platform to evolve and grow without requiring a new support to be generated.

High-level accessibility

As mentioned earlier, SIP adaptations are mapped into our SIP based support framework. This support framework provides three interaction modes that cover all the encountered situations. We have also developed the adapters to wrap non-SIP compliant services into SIP services. Relying on these adaptations and these adapters, it is simple and easy to make these services to communicate each other with the high-level abstraction of interactions modes based on SIP. To do so, the developers require neither SIP knowledge nor others heterogeneous network protocols knowledge.

High-level expressiveness

The DiaSpec language, which is a lightweight Architecture Description Language (ADL) dedicated to the pervasive computing domain, provides globally meaningful high-level expression idioms for three type interactions (e. g. command, event and session) and data flows (e. g. requires and provides). These idioms make the architect to choose simply the appropriate interaction mode in practice and to write the application logic using the architectural style, without requiring SIP knowledge.

The ways in which high-level expression idioms are generated guarantee that any associated values have the right types. Relying on this generated programming framework allows the developer to quickly develop, adapt, and check our applications. As well, it facilitates the creation of new applications.

4.4 RELATED WORK

Numerous network protocols are available in pervasive computing environments. The simple X10 protocol allows any user to control his home devices. More complicated and feature-rich protocols like Jini, UPnP, HAVi, or ZigBee contains much more capabilities for appliance communications. But these existing protocols are mainly designed to work in a single home environment (i.e. local area network) and do not scale for Internet-wide communications. SIP is a very scalable protocol, working in both LAN and WAN conditions.

Gaia [51] uses CORBA relying on raw TCP/IP connections in order to transmit data. However, for CORBA, a port that is different from the port 80 for HTTP must be open in a firewall. Due to such restriction, the use of Web Services instead of CORBA is increasing, since Web Services is on top of HTTP. However, to communicate from WAN into LAN, the message of Web Services must go through firewalls. In SIP infrastructures, SIP proxies are placed on the edge of the home environment, behind the firewall, enabling an centralized access to inner entities. SIP messages pass through the firewall using a unique port. CORBA or Web Services need to translate address, ports, or URL into local IP addresses and ports using mechanisms such as Network Address Translation (NAT).

Several research projects use SIP beyond the telephony domain. TinySIP is a modified version of SIP for use with wireless sensors in a hospital environment [35]. Berger et al. [14] leverage the SIP event model to determine user location in pervasive computing environments. However, their approach has a limited scope. Moyer et al. [44] argue for the use of SIP in wide-area secure networked devices; however, this proposal has not been studied further. In contrast with our approach, these works rely on existing, low-level programming frameworks (*e.g.*, Jain-SIP, SIP Servlet). As a result, the programmer needs to manage the intricacies of the underlying technologies, such as protocols, network layers, and signaling.

Rosenberg et al. [54] have emphasized the need for programming support that is dedicated to Internet telephony services. They propose CPL, an XML-based scripting language for describing and controlling call services. Wu and Schulzrinne [68] propose another XML-based language, LESS, to program end-system services. Services written using CPL and LESS scripting languages are mostly limited to coarse-grained processing and dedicated operations. VisuCom is an environment to graphically create telephony services [36]. It supports the development of routing services with respect to non-telephony resources like

agendas and phone directories. VisuCom includes various verifications. Examples of errors detected in services include call loss, incorrect state transitions, and unbounded resource usage.

These high-level languages target non-programmers and provides static verifications. However, they focus on end-user services and call processing. As a consequence, they lack expressivity to program pervasive applications as targeted DiaSpec.

Existing, general-purpose middlewares (*e.g.*, CORBA, DCOM) are highly flexible and support a large number of features to ease the development of distributed applications in a wide range of application areas. However, these middleware don't address requirements from specific areas such as telephony, thus adaptation code must be developed to match the application's needs [11, 30]. Moreover, associated programming frameworks force developers to use unsafe and generic structures of code. In contrast, our integrated programming framework, by generating programming support, provides developers with structures of code typed with respect to the target area, as captured by the DiaSpec specification.

SPL is a domain-specific language (DSL) whose goal is to ease the development of robust telephony services [15]. Although SPL provides high-level programming abstractions, it is still close to SIP and only addresses the routing logic.

CINEMA is a SIP-based infrastructure that enables multimedia collaboration via IP phones, instant messaging and e-mail, among other forms of communication [29]. In CINEMA, service development relies on low-level programming support (*e.g.*, SIP Servlet) or end-user oriented languages (*i.e.*, CPL and LESS). In contrast, our approach provides high-level programming support and targets a wide range of applications involving a variety of telephony and non-telephony resources. Continuing the work on CINEMA, Shacham *et al.* address the use of heterogeneous devices in ubiquitous environments [63]. They introduce location-based device discovery and customization for session interactions using SIP. Unlike our integrated programming framework, they focus on a single interaction mode (*i.e.*, session). Moreover, they do not show how to interface and integrate these devices in distributed applications.

4.5 SUMMARY

In this chapter, we developed different SIP adapters for heterogeneous ad hoc communication busses. These adapters enable SIP to be used as a universal communication bus. This bus has been validated by an experimental study involving the coordina-

tion of a wide range of entities, including serial-based sensors (RS232, 1-Wire), ZigBee devices, X10 devices, PDA, native SIP entities and software components. Furthermore, we evaluated DiaSpec with a SIP back-end for application development in a home automation context. This evaluation focused on three aspects: flexibility, accessibility and expressiveness. Finally, we compared our approach with the related works.

5

SUMMARY

The first part of this thesis described the challenge for heterogeneity of pervasive computing systems and the importance to manage it. We have presented an approach to enabling homogeneous communications between heterogeneous distributed entities. This approach relies on the use of SIP as a universal communication bus for pervasive computing environments. We integrated the SIP communication bus into DiaSpec. We described a methodology and programming support to adapt heterogeneous entities to the SIP communication bus. Our approach has been used to make a wide variety of entities SIP compliant. These entities have then been integrated into a number of applications for home automation. Finally, our experimental study has proven that our approach is realistic for all classes of entities, and that our SIP gateway can run efficiently on resource-constrained platforms.

Part II

OFFLOAD METHODOLOGY FOR ENERGY EFFICIENT APPLICATION IN SMARTPHONE CONTEXT

6

BACKGROUND

Remote execution using wireless networks to access compute servers thus fills a natural role in pervasive computing, allowing applications to leverage both the mobility of small devices and the greater resources of stationary devices.

— Flinn, J. and Narayanan, D. and Satyanarayanan, M. [21]

We will discuss the requirements about energy concerns of applications on smartphone (highlighted in Figure 20) in this Chapter as mentioned in Chapter 2.

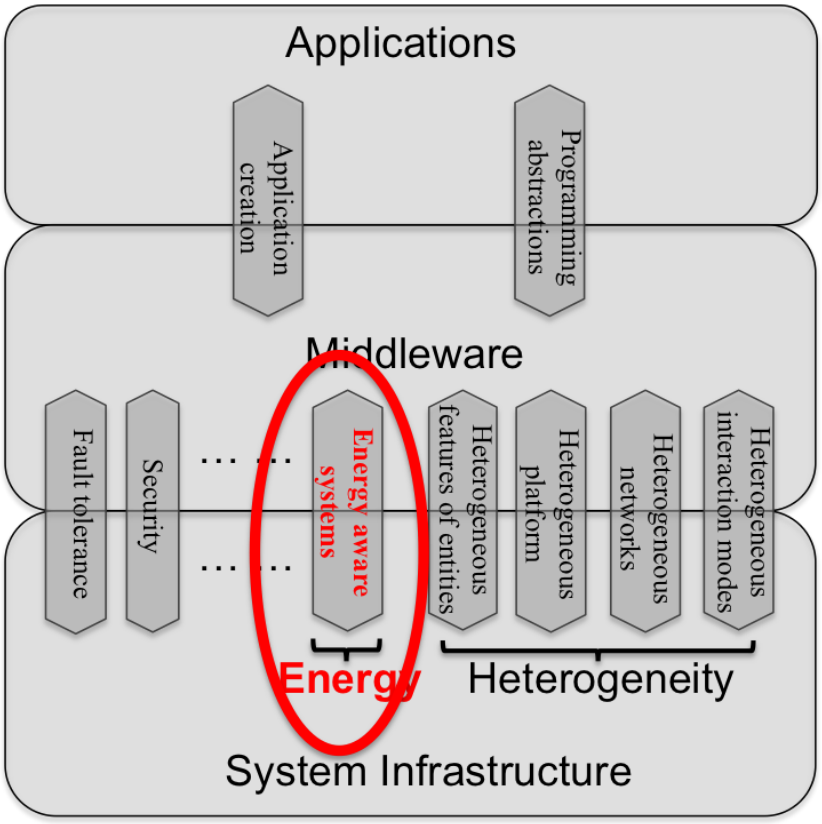


Figure 20: Special requirements of pervasive computing environments

The mobile phone is the world 's most popular computing platform. Over 2 billion people now own a mobile and in established

markets it is not uncommon for individuals to own more than one. Furthermore, modern smartphones are a rapidly growing segment of the market providing markedly more capability than conventional handsets. These devices become general-purpose devices like laptops, pocket PCs, and palm computers. They contain myriad communication interfaces, significant processing power and storage and numerous sensors ranging from simple light sensors to GPS tracking. With these devices, various services such as game, monitor, multimedia and mobile banking have become available from mobile device. Importantly today's smartphones are also programmable and come with a growing set of cheap powerful computation and communication capabilities. More and more services, which used to be available only by personal computers, are expected to be provided by mobile devices, especially with the increasingly pervasive smartphones. For executing various services, the performance of mobile device increases and the power consumption by mobile devices is also growing rapidly. However, battery capacity has shown little growth. Furthermore, battery consumption is clearly important for users when buying mobile phones and choosing applications and services to use [45]. Therefore, reducing power consumption of applications is effectively a crucial issue in smartphones. To do so, an approach to minimizing the energy cost of applications on smartphones is required.

Much progress has been achieved in the low-level software to manage hardware technologies. Several known power-conservation techniques include turning off the handheld computing device screen when it is not needed [19], optimizing I/O [23], slowing down the CPU [25], scaling down the wireless network [39] or optimizing the wireless protocols [46], among others. However, there is almost no application-level support to apply these technologies into the design phase of applications for smartphones. The programmer has to manage the intricacies of the low-level technologies.

The computation offloading has been studied as another approach to handling energy concern on smartphone. This approach doesn't require knowledge about hardware technologies. It offloads local processing on a smartphone to a server. Smartphones are connected to a selected server via a wireless connectivity. The idea of offloading computation on mobile computers has been studied in different aspects [21, 34, 66]. Flinn et al. [21] has explored middleware approach that lets applications simultaneously leverage smartphones with the mobility and servers with greater resources and no power limitations. Although this work takes only the context of execution into account, it doesn't consider energy consumption of applications as a criterion for deciding the offloading computation point. Several researches

concentrate on compiler technology [34, 66]. This approach estimates only the execution time by program analysis. However, this cannot give an accurate estimation for energy consumption, because energy consumption depends also on the power of hardware components such as Wi-Fi, CPU and screen. Recently, one sees some interests in exploring opportunities to offload the computation in cloud computing [43] by profiling the execution time. However, such work is conducted in a rather ad-hoc way by analyzing only some examples and represents only the potential energy benefit of offloading calculations. This approach cannot guide programmer in the development process to determine whether and how to offload the computation of a given program.

Computation offloading has the potential to save energy on mobile device but the savings from offloading the computation need to exceed the energy cost of the extra additional communication. The developer needs to identify, within an application, the parts performed on a server and the others parts on smartphone. The trade-off point is to find a compromise between the energy efficiency of communication and that of local computing. To achieve this goal, the developer needs to consider the use of computation offloading in the design and development of application. Thus there is a strong need for tools that can give energy consumption feedback for wireless communication and local processing during the application development cycle.

6.1 REQUIREMENTS FOR ENERGY CONCERNS IN MOBILE APPLICATIONS

This section presents two application examples of smartphones in pervasive computing environments. It also identifies the requirements and limitations of these applications for energy consumption issue. Additionally, based on these requirements, different execution and transfer modes of an application are defined to take energy efficiency into consideration.

6.1.1 *Motivating examples*

Let us examine two applications in one scenario from the area of surveillance and monitoring for healthcare: fall detection and activity monitoring. These applications involve the use of smartphones to process data from various sensors. These sensors may be integrated inside a phone or be connected to a phone by wireless communication. According to the need of applications, the level of complexity for the data processing may vary considerably. This leads to a change of the cost of using energy resources.

These examples illustrate the challenge of energy consumption for pervasive applications on smartphones.

6.1.1.1 *Health monitoring*

This health monitoring scenario is dedicated to healthcare for the aging population who are living independently. The system needs to provide to care giver valuable informations about daily activities of individual elderly people. These information permits care giver to monitor, record and analyze individual's daily activities i.e. running, walking, standing, sitting and dangerous activities such as falling. We introduce two typical applications in this scenario: fall detection and activity understandings.

FALL DETECTION. This pervasive application can monitor a falling accident that is a major health hazard for elderly person and an important obstacle to their independent living. It needs to detect a fall all the time everywhere. When a fall is detected, the application must alert to care giver and relevant people with a call, a SMS and an email.

ACTIVITY UNDERSTANDINGS. In this application, daily activities of individual elderly people are monitored by the system. Their activity and behavior monitoring can give significant information about their health. The application needs to record physical activities information, and then it analyses and learns to classify this information.

6.1.1.2 *Requirements*

This health monitoring scenario underlines the need for collecting the valuable information about daily activities anytime anywhere. A typical such system has two major functional components: the information collection component and the communication component. As their names imply, the information collection component records various informations to measure, recognize different activities and detect also user falls. In doing so, various technologies are available such as acceleration-based detection [49], image-processing techniques [22, 41]. However, many of them are only implemented within a small indoor environment since user can't bring the camera and some types of sensor everywhere. Smartphones including accelerometers as the platform meet this need. After data collection, how can we process them? The complexity of acceleration data processing may vary considerably in terms of used algorithm and amount of data. For fall detection application, the data section contains only several hundred

three-axis acceleration points and its size is relatively small (over ten kilobytes). The complexity of the fall detection algorithm will impact the execution time of the corresponding application. In this context, we need to take the energy consumption of the smartphone application into account. If we require calculations such as a threshold of absolute peak values of acceleration to determine falls [18], we can develop such a fall detection system on smartphones. Alternatively, we can use complex calculations to determine falls with more accuracy [**?**]. This fall detection algorithm uses 256 samples for FFT computation in order to extract the feature. It uses a Bagging meta-level classifier for machine learning. Data collected for a single subject on one day are used as training data and, data collected for the same subject on another day are used as testing data. Note that this algorithm performs intensive calculations on a large amount of data. If this algorithm is run on a smartphone, these calculations will consume a significant amount of energy. Thus we need to offload the computation to a server. For activity understandings application, we need acceleration data for one day for the activity recognition. The data size is over ten megabytes. We need to record all these data to analyze and recognize daily activities. Storing such data on smartphones everyday is not realizable. When we want to transfer a lot of data, using compression technology has a potential for saving energy on smartphones because it can reduce the amount of transmitted data. In this scenario, the communication component transfers collected data to a server and communicates with emergency contacts after a fall is detected.

After analyzing our working scenario, we draw an overview of health monitoring system in Figure 21 and provide a preliminary assessment of its capabilities with focus on computation offloading to answer energy consumption issue on smartphone.

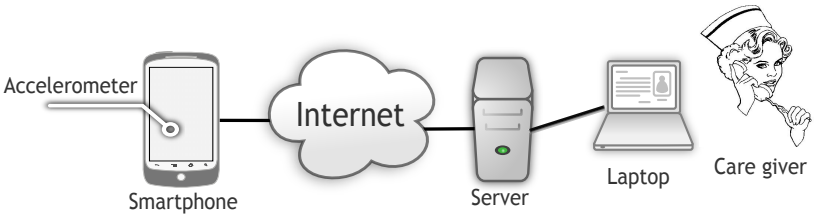


Figure 21: A system for health monitoring

Our working scenario illustrates the specific requirements for smartphones including accelerometers. Our working applications illustrate three potential modes for partitioning the computation for a given application to minimize energy consumption on smartphones in Figure 22: (1) computation may be performed on the smartphone (local mode) such as acceleration-based fall detection

approach, (2) on both (hybrid mode) or (3) on a server platform (remote mode) for a complex calculation.

Figure 22: Three execution modes for a given application on smartphone

As a smartphone is an energy-constrained device, we need to partition the application and integrate energy concern in the application design phase. Thus the usage of smartphones opens up a new important issue that is how to make a decision for one execution mode of a given program to save energy on smartphones in the application design stage.

6.2 SUMMARY

This chapter introduced the challenge of energy consumption in the context of smartphones. It presented the main hardware and software approaches aimed at minimizing the energy cost of applications on smartphones. Then, it described two application examples and outlined the requirements for energy consumption in the context of smartphones. The next chapter presents our approach, addressing our list of requirements.

7

COMPUTATION OFFLOADING AND DATA COMPRESSION FOR REDUCING ENERGY COST

This Chapter gives an overview of our approach in Section 7.1. Section 7.2 analyzes some important elements that can influence energy cost of computation and communication for a given program. Section 7.3 describes overview of three execution and transfer modes for mobile applications and their strategy scheme. Section 7.4 presents the overall design of our tool with the integration of our strategy scheme.

7.1 OUR APPROACH

Our research aims to address how to take energy consumption into consideration in the design and development of applications and services on pervasive smartphones for the developer. To handle this problem, the computation offloading, partially or completely, has been explored in our approach. To offload calculations, data transmission is necessary. Data compression can possibly reduce the energy cost by decreasing the amount of data to transfer. In this context, we integrate data compression into our approach. An application can be executed on a smartphone, on a server or on both. The developer decomposes an application into several parts according to the amount of data and the complexity of the algorithm for a program. Each part of an application with its input and output data is identified as a task. Taking computation offloading and data compression into account, we develop a task execution and transfer strategy scheme that determines whether and how to offload computation of a given task with or without compression data transfer. We apply this scheme to a tool named "OffDeci" that permits a developer to estimate accurately energy consumption of wireless communication and local processing for a given task. Through comparison between computation and communication cost, this tool decides the trade-off point.

Specifically, based on previous discussions, in our current strategy scheme, three program execution and data transfer modes (local, uncompressed remote and compressed remote) and five relevant stubs of these modes are defined. We develop our OffDeci tool with the use of this partition scheme. The tool takes each task as a parameter input. Then each task will be executed

in three modes mentioned earlier. Our approach focuses on the use of profiling to measure the execution time and the power consumption and not on program analysis. We collect first profiling information on time for local computation and wireless communication. We calculate then energy computation of local, uncompressed remote and compressed remote mode for a given task by multiplying power consumption and the time interval of local computation and wireless communication. Based on that, we construct an energy consumption graph for a task in three modes. According to the result of this graph, the tool proposes one specific mode of a task for the programmer to achieve the most efficient energy cost in the development process. We have validated our approach by studying two application examples on an HTC Hero smartphone running Android.

Moreover, the amount of input and output data may be diverse during the execution of a task. It is difficult to measure results for all possible input and output data volumes with our tool. In this context, we propose prediction estimation rules for the developer of applications on the basis of our energy analysis and experimental study. These prediction estimation rules are dependent on data volume and algorithm complexity. The programmer can choose one execution and transfer mode for a given task, using these prediction estimation rules. For the case that these rules cannot make an accurate decision, the programmer can use our tool for each specific task.

To handle energy concerns of mobile applications in pervasive computing systems, we sketch the integration of OffDeci into our high-level programming framework. We propose to raise the level of abstraction beyond the code level, by providing declarative support to minimize energy cost. Our programming framework is component-based. We extend existing DiaSpec specifications with energy-handling declaration to manage the communication back-end of a DiaSpec component that may be offloaded from the point of view of energy cost. This declaration is passed to OffDeci tool that generates five stubs for each relevant component. The implementation of a component is passed to OffDeci, in combination with these stubs. OffDeci allows deciding whether and how to offload a component.

7.2 ENERGY ANALYSIS FOR COMPUTATION AND COMMUNICATION

The applications that are discussed in previous Chapter demonstrate the need for computation offloading mechanisms to minimize energy cost of an application. In the context of computation offloading, the critical aspect for smartphones is the trade-off

between energy consumed by computation and the energy consumed by communication. In doing so, we need to calculate energy consumed for a given program by each execution mode. In the literature, two modeling techniques are proposed for calculating the energy consumed. One modeling method (proposed by Tiwari et al. [37]) is based on the instruction level by using the following formula.

$$E = P_{ave} * (n * \mu)$$

We use E to denote the energy consumed. P_{ave} represents the average power consumption, n is the number of cycles and μ is cycle period. Each instruction is executed in different number of cycles. To obtain the cycle cost for each instruction, we require extensive experimentation to build up an average cost for all instruction. Thus for a processor with "rich instructions sets", this technique is not realistic. The other issue is the use of virtual machine technology for mobile. This technique makes it almost impossible to obtain the cycle cost of an operation in a Java embedded platform such as Android. Because this technique works only in the assembly language by knowing process addressing modes and is not adapted for high level programming language. Taking these issues into account, we use another modeling method (proposed by Russell et al. [58]) to calculate the energy consumed by using the following formula.

$$E = P_{ave} * t$$

Where t is the execution time for a program (e.g., algorithm calculation, data transmission). Hence the total energy consumed by a program is the product of the time that it takes to execute and the average power consumption of a processor. Employing this method, we define the energy cost of performing the computation locally (E_{Loc}) versus the cost of transferring these data and offloading the computation (E_{NetCal}). Let us examine these energy costs on smartphones. First we have:

$$E_{NetCal} = E_{Net} + E_{CalDis}$$

Where E_{Net} is the energy cost of transferring these data and E_{CalDis} is the energy cost of computation on a server. From the point of view of energy consumption on smartphones, E_{CalDis} does not concern the mobile and is considered to be equal to zero. We then have:

$$E_{NetCal} = E_{Net}$$

The execution time and the power consumption of computation and communication hardware components influence significantly E_{Loc} and E_{Net}. To characterize energy costs associated with computation and communication, we make use of the following parameters.

P_{Net}: average power consumption of wireless communication component (e.g., Wi-Fi, 3G) in data transmission mode. According to the study in the literature, power for sending data and that for receiving data is very similar. We use P_{Net} to represent these two powers.

T_{Sock}: total time for a construction and destruction of a socket connection

T_{NetB}: average time to send a byte by the socket

v_B: amount of data to be transferred in bytes

t_{NetD}: total time for transferring the amount of data

P_{Loc}: average power consumption of CPU module during the CPU utilization 100 %

t_{Exe}: total execution time for a given program with input and output data

$$E_{Net} = P_{Net} * t_{NetD} = P_{Net} * (T_{Sock} + v_B * T_{NetB}) \quad (7.1)$$

$$E_{Loc} = P_{Loc} * t_{Exe} \quad (7.2)$$

Where P_{Loc}, P_{Net}, T_{Sock} and T_{NetB} are four constant parameters for a given platform and a given type of wireless network. These parameters can be measured. In this context, E_{Loc} and E_{Net} are highly dependent on v_B and t_{Exe}. In the formula 7.1, v_B is a variable parameter that relies on input and output data for a given program. The execution time value t_{Exe} depends on the complexity of algorithm and amount of data to process for a given program.

Based on the above discussions, we obtain the complexity of algorithm and the amount of data as two main parameters that determine computation offloading strategy for a given program.

7.3 A STRATEGY SCHEME FOR THREE PROGRAM EXECUTION AND DATA TRANSFER MODES

In the previous section, we have motivated the use of three execution modes to save energy consumption of an application on smartphones. We have also identified two important parameters for the determination of computation-offloading strategy. In this section, we first present the specification of our execution and transfer model and energy cost model. We then describe a program execution and transfer strategy scheme to model the program behavior and cost.

7.3.1 *Execution and transfer model*

To explore opportunities for application offloading, the developer needs to identify, within an application, server part and client part. The client part is run on a smartphone and the server part is offloaded on a server platform. As the complexity of algorithm and the amount of data have a great influence on the energy consumed by computation or communication for a program, the developer can divide an application into several parts according to these two parameters. In our program execution and data transfer model, a task corresponds to a part of an application that is chosen by the developer. For example, a task can be a component of a component-based application. Specifically, our execution and transfer model is based on energy consumption concern on smartphones. For a given task, the energy savings can be made with offloading local processing to a server. In the context of offloading a task, the amount of input and output data has large influence on energy consumption for data transmission. Apart from computation offloading, data compression has possibility to save mobile energy for a program, such as the case study of E-mail Xiao et al. [70]. Because the energy cost of compressed data transfer and the additional computation for data compression may not exceed that of data transfer without compression, at the same time, it may not also surpass that of data computation. In this context, we integrate computation offloading and data compression into our execution and transfer model. This is the specification of our execution and transfer model.

Based on the previous discussions, our classification of program execution and data transfer model uses two criteria. The first criterion is whether or not the input and output data of a chosen task is compressed before the transmission. The second criterion identifies whether or not a chosen task is offloaded. As shown in Figure 23, we classify three execution and transfer modes for a task in our model: (1) the computation may be performed on the smartphone (local mode), (2) on a server with uncompressed data transfer (uncompressed remote mode) or (3) on a server with compressed data transfer (compressed remote mode). We define the scope of the compressed remote mode. In our compressed remote mode for a task, the input data or the output data or both are compressed before the transmission between a server and a smartphone. As a result there is always a stage that is locally performed on a smartphone. This phase produces a compressed input data set or an uncompressed output data set or both. The single parameter is the amount of input and output data for a task.

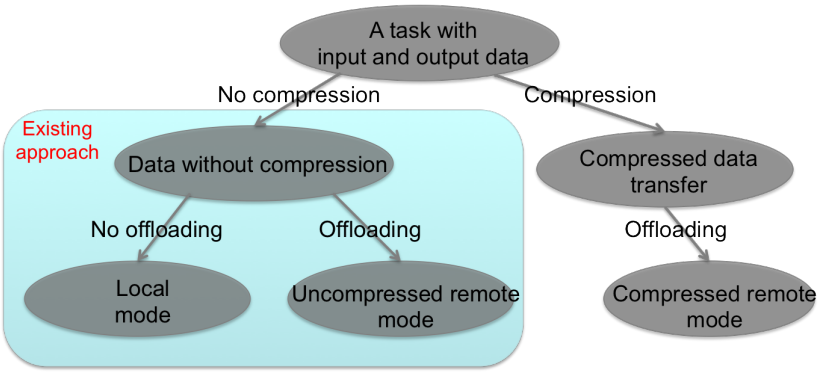


Figure 23: Classification of program execution and data transfer model

In fact, for an application, the set of task on the server side may be empty in our execution and transfer model. In such case, the whole application runs on a smartphone. In contrast, the set of tasks on the client side will never be empty, because realistic tasks always have some I/O operations for data acquisition that must be executed on a smartphone. However, these operations cannot change the fact that data may need to be transmitted between a server and a smartphone. Thus, if simple I/O operations are the only program that is performed on the client side, then the entire task run on the server side, in which case we consider a task executed on remote mode.

7.3.2 *Energy cost model*

According to our execution and transfer model, the energy cost of three execution and transfer modes for a task is modeled as follows: Besides the parameters defined in the formula (1) and (2), we refine vB with the utilization of data compression technology: the data can be transmitted in compressed or uncompressed form. The transfer time (t_{NetD}) varies with the utilization of different data forms. In this context, we utilize the following parameters.

v_{BIn}: amount of input data in bytes in uncompressed form.

v_{BOut}: amount of output data in bytes in uncompressed form.

$v_{BInComp}$: amount of input data in bytes in compressed form.

$v_{BOutComp}$: amount of output data in bytes in compressed form.

t_{InComp}: total time for compressing locally the input data of a task.

$t_{OutUncomp}$: total time for uncompressing locally the output data of a task.

$P_{NetIdle}$: average power consumption of wireless communication component (e.g., Wi-Fi, 3G) in connection mode without data transfer.

$t_{SockIdle}$: total waiting time for the reply of the task from the server.

$T_{SockOpen}$: average time for a construction of a socket connection.

$T_{SockClose}$: average time for a destruction of a socket connection.

t_{NetDS}: transfer time for input data that is uncompressed.

t_{NetDR}: transfer time for output data that is uncompressed.

$t_{NetDSComp}$: transfer time for input data that is compressed.

$t_{NetDRComp}$: transfer time for output data that is compressed.

We then have: The energy cost of sending uncompressed input data of a task E_{NetS}:

$$E_{NetS} = P_{Net} * t_{NetDS}$$

$$= P_{Net} * (T_{SockOpen} + v_{BIn} * T_{NetB}) \tag{7.3}$$

The energy cost of idling wireless communication to wait for the output sent by a server $E_{NetIdle}$:

$$E_{NetIdle} = P_{NetIdle} * t_{SockIdle} \tag{7.4}$$

The energy cost of receiving uncompressed output data of a task E_{NetR}:

$$E_{NetR} = P_{Net} * t_{NetDR}$$

$$= P_{Net} * (T_{SockClose} + v_{BOut} * T_{NetB}) \tag{7.5}$$

The energy cost of compressed input data transfer $E_{NetSComp}$ and the additional computation for compressing input data $E_{LocInComp}$:

$$E_{LocInComp} = P_{Loc} * t_{InComp} \tag{7.6}$$

$$E_{NetSComp} = P_{Net} * t_{NetDSComp}$$

$$= P_{Net} * (T_{SockOpen} + v_{BInComp} * T_{NetB}) \tag{7.7}$$

The energy cost of compressed output data transfer $E_{NetRComp}$ and the additional computation for uncompressing output data $E_{LocOutComp}$:

$$E_{LocOutComp} = P_{Loc} * t_{OutUncomp} \tag{7.8}$$

$$E_{NetRComp} = P_{Net} * t_{NetDRComp}$$

$$= P_{Net} * (T_{SockClose} + \nu_{BOutComp} * T_{NetB}) \tag{7.9}$$

As shown in equations 7.3 - 7.9, we can see that the energy cost for data transfer varies with the different possible combinations of input and output form. We have four combinations of input and output form for a given task: 1) uncompressed input and output data, 2) compressed input and uncompressed output data, 3) uncompressed input and compressed output data and 4) compressed input and output data.

7.3.3 *Program execution and data transfer strategy scheme*

We take advantage of two technologies in our approach: computation offloading and data compression. In this context, a given task can be performed on a smartphone or a server with the compressed or uncompressed data transfer. Specifically, we may compress the input data or the output data or both for the compressed data transfer. According to this analysis, we use five stubs to represent five combinations of three execution and transfer modes with compressed or uncompressed data as illustrated in Figure 24: (1) A given task may be performed on the smartphone. (2) A given task may be performed on the server with both the input and the output data that are uncompressed or (3) with input data only that is compressed or (4) with output data only that is compressed or (5) with both the input and the output data that are compressed. As a task is a part or a component of an application on a smartphone, the input data will never be empty. However, the output data may be empty for a task as shown in the stubs (2) and (3) in the Figure 24.

Based on five stubs in combination with the implementation of a task, the energy savings can be made with offloading the exceeded computation and minimizing the transmission energy cost with data compression. In doing so, we need to identify two trade-off points for a task: one consists in deciding how to save the energy during communication by transmitting uncompressed data or compressed data. The other involves the determination of

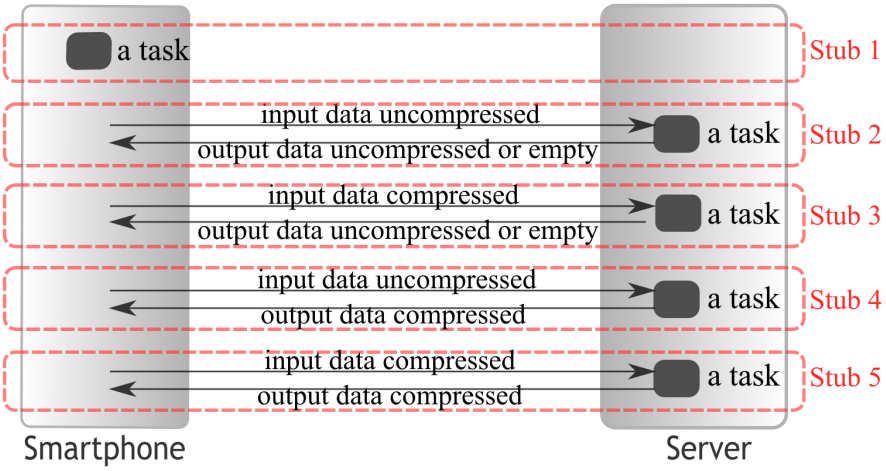


Figure 24: Five stubs for three execution and transfer modes

the minimum value between energy consumed by computation and the energy consumed by communication. We integrate these two trade-off points into our program execution and data transfer strategy scheme as shown in Figure 25.

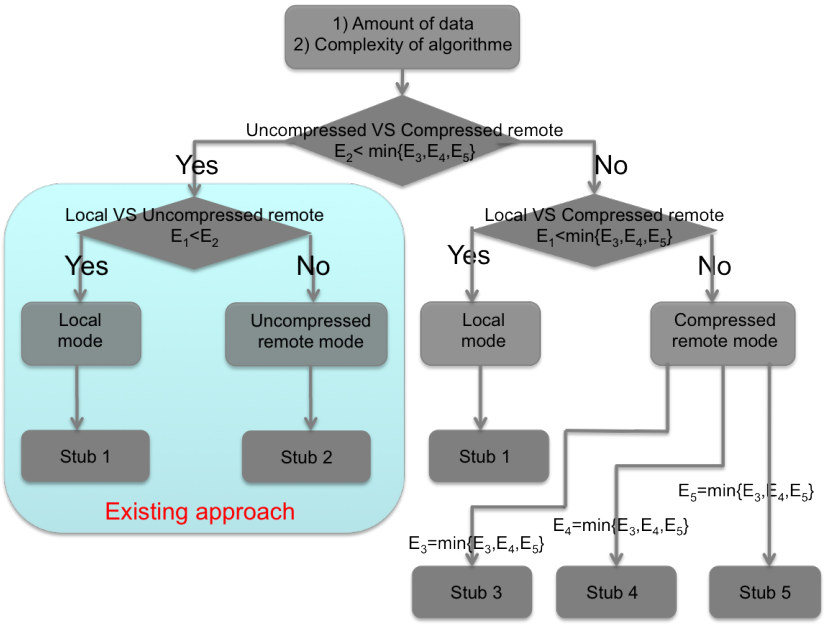


Figure 25: Program execution and data transfer strategy scheme

In our execution and transfer strategy scheme, the first phase consists in examining which mode consumes the least amount of energy for a task during the communication between uncompressed and compressed remote mode. We find the minimum energy consumed by communication with or without data compression. Then the minimum energy cost for communication is compared with the energy cost for local computation. The minimum energy cost for executing a task is obtained. The relevant

stub is selected as the most energy efficiency execution for a task. Compared with the existing approach [43, 66], our execution and transfer strategy scheme is more accurate and complete. Because the existing approach represents only the energy benefit of offloading calculations and doesn't take the possible energy savings from data compression for the communication into account.

Specifically, we make use of equations 7.1 and 7.3 - 7.9 to represent respectively the energy consumed by computation, communication and data compression in our five stubs. The energy cost set $E = E_1, E_2, E_3, E_4, E_5$ of five stubs in combination with the implementation of a given task is respectively defined as follows.

Energy consumed by executing a task with the stub 1:

$$E_1 = E_{Loc} = P_{Loc} * t_{Exe} \tag{7.10}$$

Energy consumed by executing a task with the stub 2:

$$E_2 = E_{NetS} + E_{NetIdle} + E_{NetR} = P_{Net} * t_{NetDS}$$

$$+P_{NetIdle} * t_{SockIdle} + P_{Net} * t_{NetDR} \tag{7.11}$$

Energy consumed by executing a task with the stub 3:

$$E_3 = E_{LocInComp} + E_{NetSComp} + E_{NetIdle} + E_{NetR}$$

$$= P_{Loc} * t_{InComp} + P_{Net} * t_{NetDSComp} + P_{NetIdle}$$

$$*t_{SockIdle} + P_{Net} * t_{NetDR} \tag{7.12}$$

Energy consumed by executing a task with the stub 4:

$$E_4 = E_{NetS} + E_{NetIdle} + E_{NetRComp} + E_{LocOutComp}$$

$$= P_{Net} * t_{NetDS} + P_{NetIdle} * t_{SockIdle} + P_{Net}$$

$$*t_{NetDRComp} + P_{Loc} * t_{OutUncomp} \tag{7.13}$$

Energy consumed by executing a task with the stub 5:

$$E_5 = E_{LocInComp} + E_{NetSComp} + E_{NetIdle} + E_{NetRComp}$$

$$+E_{LocOutComp} = P_{Loc} * t_{InComp} + P_{Net} * t_{NetDSComp}$$

$$+P_{NetIdle} * t_{SockIdle} + P_{Net} * t_{NetDRComp} + P_{Loc} * t_{OutUncomp} \quad (7.14)$$

Where P_{Loc}, P_{Net} and $P_{NetIdle}$ are three constant parameters for a given platform and a given type of wireless network, these parameters can be measured. In this context, the energy cost set E is highly dependent on the data transfer time (t_{NetDS}, t_{NetDR}, $t_{NetDSComp}$, $t_{NetDRComp}$ and $t_{SockIdle}$) and the local processing time (t_{InComp}, $t_{OutUncomp}$ and t_{Exe}). Based on the above, the overall energy optimization problem is to find the minimum energy cost E_i in the set E.

7.4 DESIGN OF EXECUTION AND TRANSFER MODE DECISION TOOL "OFFDECI" FOR ENERGY CONCERNS

Based on previous discussions, our execution and transfer strategy scheme points out the need for the measurement tool to bring rich informations about energy cost of the implementation of a task encapsulateid in each stub. In doing so, we use an execution profiling based approach to design a tool named "OffDeci". OffDeci provides a support with energy cost feedback in the application design phase for decision helping for the developer. The developer chooses a task of an application. A given task as input parameter invokes OffDeci to determine how and where the task will execute with uncompressed or compressed data transfer. OffDeci estimates first power consumption of CPU and Wi-Fi of a given mobile platform. OffDeci then executes respectively the task into five different stubs as shown in Figure 24. It measures the execution time for a task in each stub and then calculates its energy cost. Using our execution and transfer strategy scheme (illustrated in Figure 25), OffDeci chooses the best stub of a task in terms of minimizing energy consumption. According to the result, a developer can implement a task in the most energy-efficient way. The Figure 26 presents the architecture of OffDeci that consists of three principal components as shown in: 1) power estimation 2) time profiling 3) comparator of energy consumption.

7.4.1 *Power estimation*

The OffDeci power estimation component interacts with Sesame [20] that can acquire the power estimation for P_{Loc}, P_{Net}

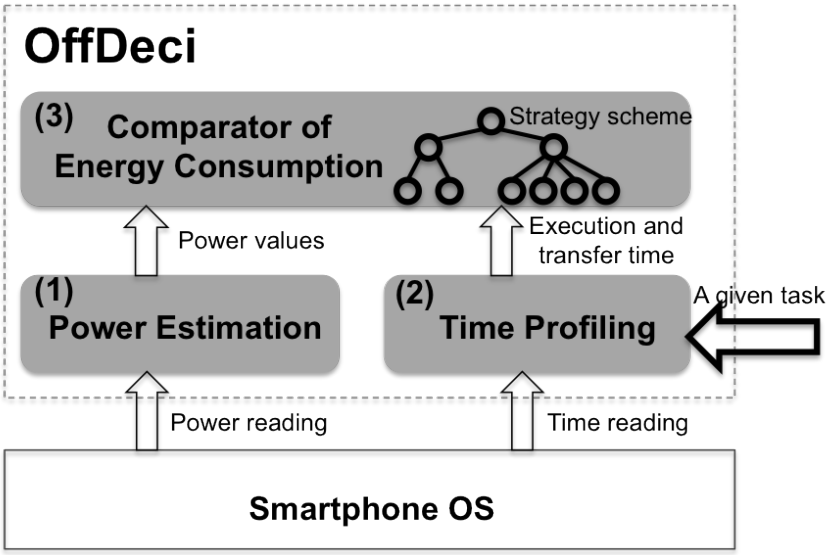


Figure 26: Architecture of OffDeci with three main components: power estimation, time profiling and comparator of energy consumption

and $P_{NetIdle}$. We use a program to reach almost a CPU utilization of 100 % for 200 seconds. The program execution for 200 seconds permits to obtain the most accurate CPU power consumption estimation (P_{Loc}), because Sesame has the lowest error at the reading rate 0.01 Hz. Idem for P_{Net} and $P_{NetIdle}$, we perform each of two socket programs for 200 seconds. The first one transmits continuously data to acquire P_{Net} estimation. The other one transmits nothing to acquire $P_{NetIdle}$ estimation. As the Internet socket is a basic communication mechanism and used by all others upper protocols across IP, from the programmer's point of view, we choose socket programs to characterize wireless network communication.

7.4.2 *Time profiling*

We have designed a stopwatch-benchmarking tool that can measure the amount of time for performing one or more operations. Our stopwatch measures the time spent in program execution, so it can trigger CPU utilization. We need to minimize the overhead of our stopwatch. In doing so, our stopwatch provides three simple functions to calculate time spent in a specific work. Three functions are defined as follows:

`starttime()`: This function call identifies the beginning of a work and stores the current time of the OS in milliseconds.

`stoptime()`: This function call identifies the end of a work and stores the current time of the OS in milliseconds.

elapsedtime(): This function call computes the amount of time spent in a work.

In the time profiling phase, the OffDeci tool generates five different stubs in combination with the implementation of a given task as illustrated in the Figure 25. These stubs use the client/server communication via sockets and data compression and decompression technology. We use three functions of our stopwatch to measure and log the program execution time (t_{InComp}, $t_{OutUncomp}$ and t_{Exe}) and data transfer time (t_{NetDS}, t_{NetDR}, $t_{NetDSComp}$, $t_{NetDRComp}$ and $t_{SockIdle}$) in the five stubs for a task as shown in equations 7.10 - 7.14. We repeat the time profiling phase for several times. The average value is obtained for each time variable that needs to be measured. By employing the average value (low-pass filter), artefacts in measurement of execution time are reduced. We acquire more accurate value for the measured time variables.

7.4.3 *Comparator of energy consumption*

The OffDeci comparator component takes the result of power estimation and time profiling components as input parameters. According to equations 7.10 - 7.14, these parameters allow calculating the energy cost set E by multiplying power consumption and the time interval of local computation and wireless communication. The OffDeci comparator component employs our program execution and data transfer strategy scheme as shown in the Figure 25 and compares the energy consumptions of five stubs in combination with the implementation of a given task in three execution and transfer modes. This component can determine how and whether to offload the computation of a given task and which data form is used for the input and the output data during the transmission. For data compression to be beneficial during communication, we require that the minimum energy cost among three stubs of a given task in compressed remote mode (min E_3, E_4, E_5) is smaller than the stub in uncompressed remote mode (E_2). For offloading to be beneficial we require that the energy cost of local mode (E_1) is smaller than E_2 or min E_3, E_4, E_5. Finally we can obtain the minimum energy cost E_i in the set E. The relevant stub of E_i is the most energy-efficient implementation for the given task.

7.5 SUMMARY

This chapter presented our approach for managing energy in the context of smartphones. For a given program and set of

inputs, we have identified two important parameters to the energy cost: computation and communication. We presented a model for the execution of an application and the transfer of its data, parameterized with respect to computation offloading and data compression. Finally, we presented the design of our tool, named OffDeci, that gives an approximation of the energy cost of an application with respect to combinations of offloading and data compression strategies. The next chapter evaluates our approach with an experimental study.

8

EVALUATION AND EXPERIMENTAL STUDY

In this Chapter, we validate the use of computation offloading and data compression. This validation is done in the context of our experimental platform. We first describe the specific smartphone platform and the measurement hardware setup we use in our experimental platform. We also explain the five stubs, each of our two motivating examples in three execution and transfer modes. We then present the energy consumption measurements for different stubs and compare these measurements with the result of our tool OffDeci. In doing so, we need to get the value of different parameters as shown in the Table 4. We outline prediction rules to help a developer to choose one program execution and data transfer mode for each task of mobile applications. Finally, we then discuss on the related works.

8.1 EXPERIMENTAL PLATFORM

This experimental platform was based on a HTC Hero smartphone running Android 2.1, a wireless router NETGEAR WGR614 and a MacBook Pro laptop with 4 GB RAM and the 2.26 GHz Intel a Core 2 Duo processor as a server. The HTC Hero features an ARM-based 528 MHz, a 288MB RAM and a 512MB ROM. It uses a 1350 mAh rechargeable lithium-ion battery. It allows super user access. We used the Android 2.0 software development kit, which supports both Java and C program development.

The smartphone current consumption is measured by using a high-precision current probe Tektronix TCP312 around one wire between the pin Vcc of the battery terminal and its connector on the smartphone. In doing so, we connect the battery of the smartphone to its connector with four wires as illustrated in Figure 27. We use a Tektronix DPO3014 oscilloscope to measure the current across the smartphone battery and also the voltage supplied by the battery and the software LabVIEW SignalExpress to register current traces into PC as shown in Figure 28.

The current probe Tektronix TCP312 allows us getting the real time current traces with a high precision, using the Hall effect theory. We then use an oscilloscope Tektronix DPO3014 to inspect a number of traces with the different sampling rate up to 2.5 GHz. We choose a sampling rate (100 KHz) that is enough to obtain all features of the trace.

Parameters	Estimation	Experimentation
P_{Loc}	Sesame in power estimation component of the OffDeci	current consumption measured by using a high-precision current probe with a oscilloscope
P_{Net}	Sesame in power estimation component of the OffDeci	current consumption measured by using a high-precision current probe with a oscilloscope
$P_{NetIdle}$	Sesame in power estimation component of the OffDeci	current consumption measured by using a high-precision current probe with a oscilloscope
t_{NetDS}	Stopwatch in time profiling component of the OffDeci	Oscilloscope
t_{NetDR}	Stopwatch in time profiling component of the OffDeci	Oscilloscope
$t_{NetDSComp}$	Stopwatch in time profiling component of the OffDeci	Oscilloscope
$t_{NetDRComp}$	Stopwatch in time profiling component of the OffDeci	Oscilloscope
$t_{SockIdle}$	Stopwatch in time profiling component of the OffDeci	Oscilloscope
t_{InComp}	Stopwatch in time profiling component of the OffDeci	Oscilloscope
$t_{OutUncomp}$	Stopwatch in time profiling component of the OffDeci	Oscilloscope
t_{Exe}	Stopwatch in time profiling component of the OffDeci	Oscilloscope

Table 4: Parameters for the energy consumption measurements

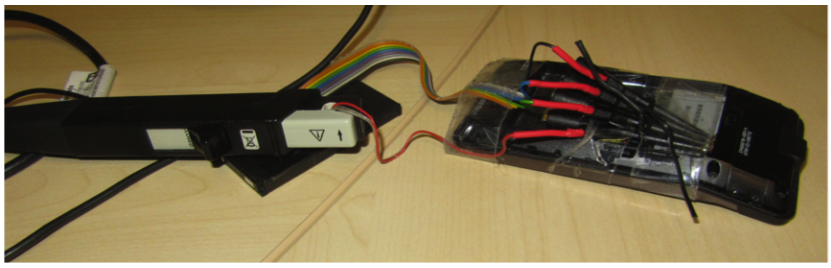


Figure 27: Battery replacement with four wires for current measurement

It is necessary for energy analysis to determine power consumption for each relevant hardware component. In doing so, we first measure power consumption of the HTC Hero in the idle mode. We then measure respectively power consumption of CPU and Wi-Fi. The power is calculated by multiplying the current and the relevant voltage.

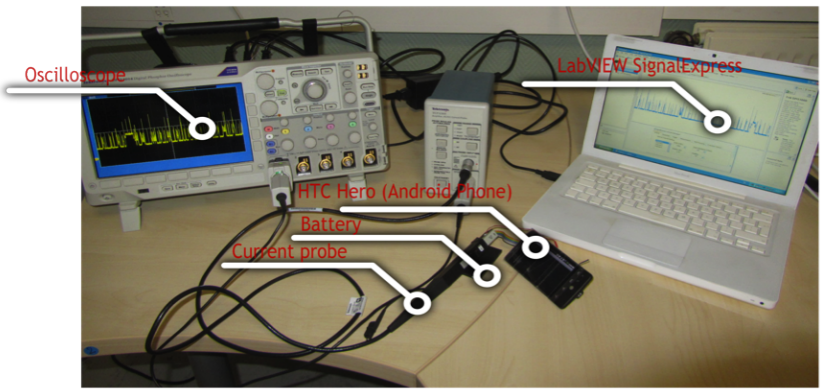


Figure 28: Our measurement platform for power monitoring

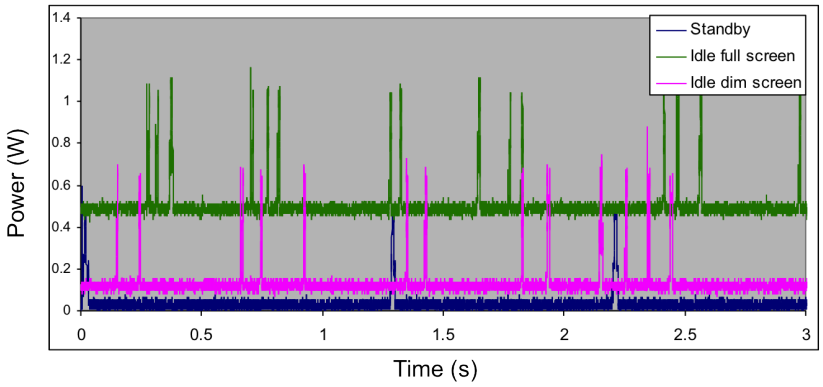


Figure 29: Power consumption of the screen of HTC Hero in three different modes

HTC Hero in idle mode	CPU	Wi-Fi	Screen
Average power (mW)	4.57	13.6	0

Table 5: Average power consumption of different hardware components in idle mode

As shown in Figure 29, power consumption of the HTC Hero in idle mode varies by switching the backlight of the screen in three modes: on, idle and off. Varying the backlight from maximum-to-minimum brightness decreases the power consumption of the smartphone by half a watt.

We illustrate the power consumption of CPU (a) and Wi-Fi (b) in idle mode in Figure 30. We can observe that the events are periodic in these modes. The average power consumption of different hardware components in idle mode is calculated in the Table 5. The Wi-Fi component in idle mode consumes more energy. Its average power (PNetIdle) is about 13.6 mW.

To obtain the power consumption of CPU, we exploit a program to vary the CPU utilization. The power consumption of CPU utilization of 100 % (a) and 40 % (b) is shown in Figure 31. We

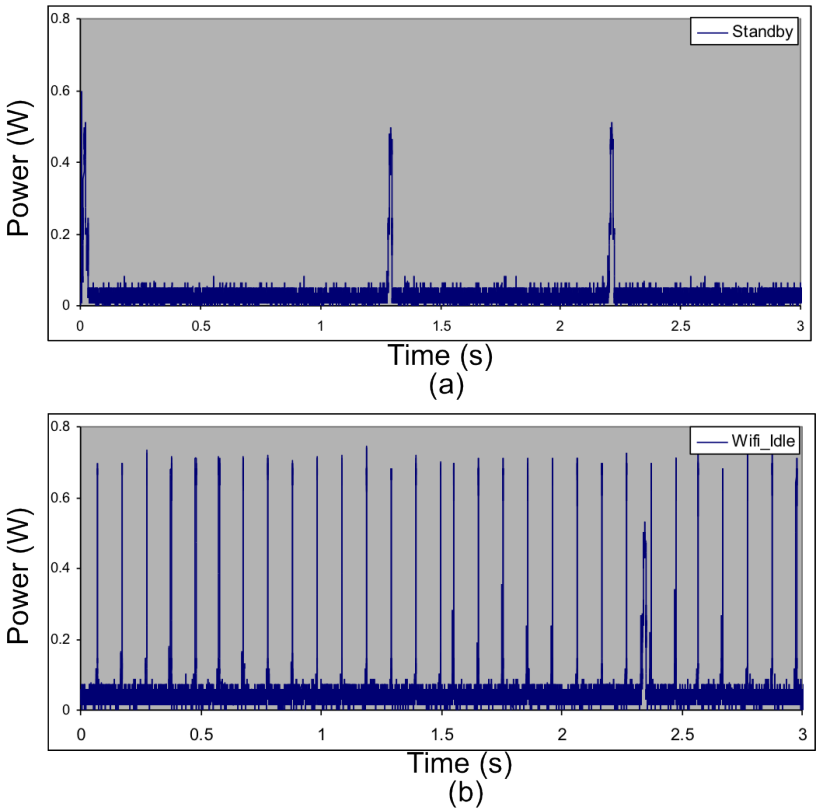


Figure 30: Power consumption of the HTC Hero: (a) CPU in idle mode, (b) Wi-Fi in idle mode

note that the power consumption of CPU utilization of 100 % is about 255.3 mW (P_{Loc}).

To acquire the power consumption of Wi-Fi interface, we use a program to switch data transmission in continuous (a) and discontinuous (b) mode as illustrated in Figure 32. In discontinuous mode, the process alternates transferring data and falling asleep. We observe that the power consumption of Wi-Fi data transmission is about 1196.2 mW (P_{Net}).

Based on these initial experiments for each component, we acquire some useful knowledge to understand energy costs of computation and communication. This knowledge allows us measuring energy cost of the implementation of our application examples on HTC Hero. It has a great importance for application developers to minimize the energy cost of applications on this kind of smartphone.

8.2 IMPLEMENTATION OF MOTIVATING EXAMPLES

Let us explain the development work for the five stubs of each our two motivating examples in three execution and transfer

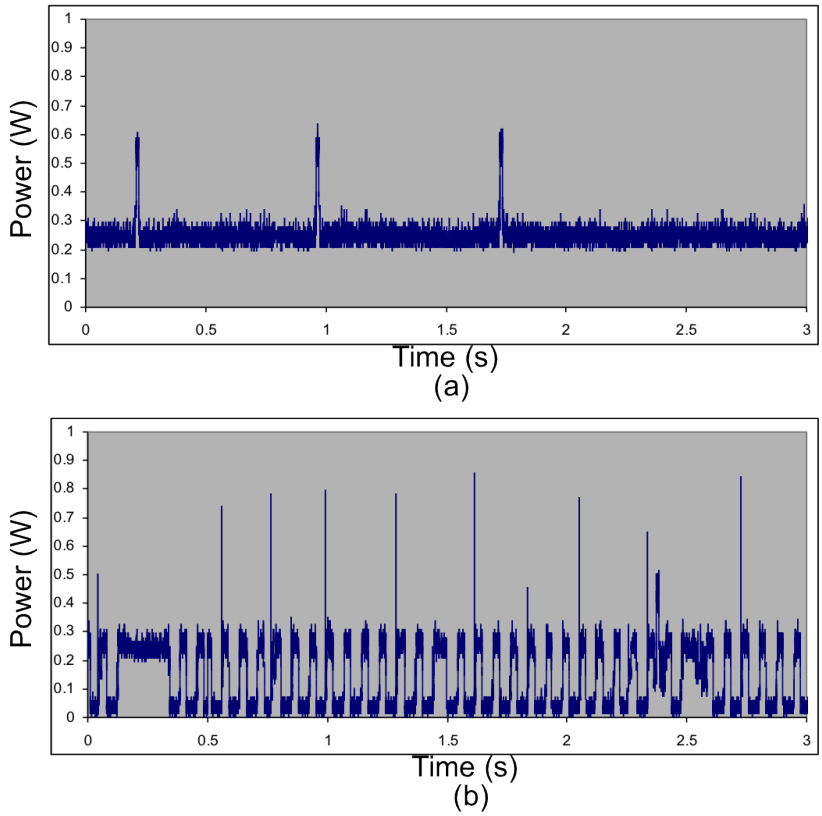


Figure 31: Power consumption of the HTC Hero: (a) CPU utilization of 100 %, (b) CPU utilization of 40 %

modes. We implement our applications in Java with Eclipse and Android 2.0 SDK.

FALL DETECTION. We implement PerFallD, a pervasive fall detection system tailored for smartphone, described by Dai et al. [18]. To detect a fall, features are computed on 4s time window size for 200 acceleration data sample (a sampling frequency of 50 Hz). In this algorithm, the amplitude of the acceleration in the absolute vertical direction compares to the thresholds. We pass this algorithm with input and output data as a task to the tool OffDeci. OffDeci generates the five different stubs for this task. For offloading the computation we use the client/server communication via sockets and for data compression and decompression we write and read data in zip format, using Java APIs with Eclipse and Android 2.0 SDK. These stubs in combination with the implementation of this task are respectively executed. OffDeci calculates their energy consumptions and proposes a stub that consumes less energy. We compare this result with that of our measurement platform for power monitoring.

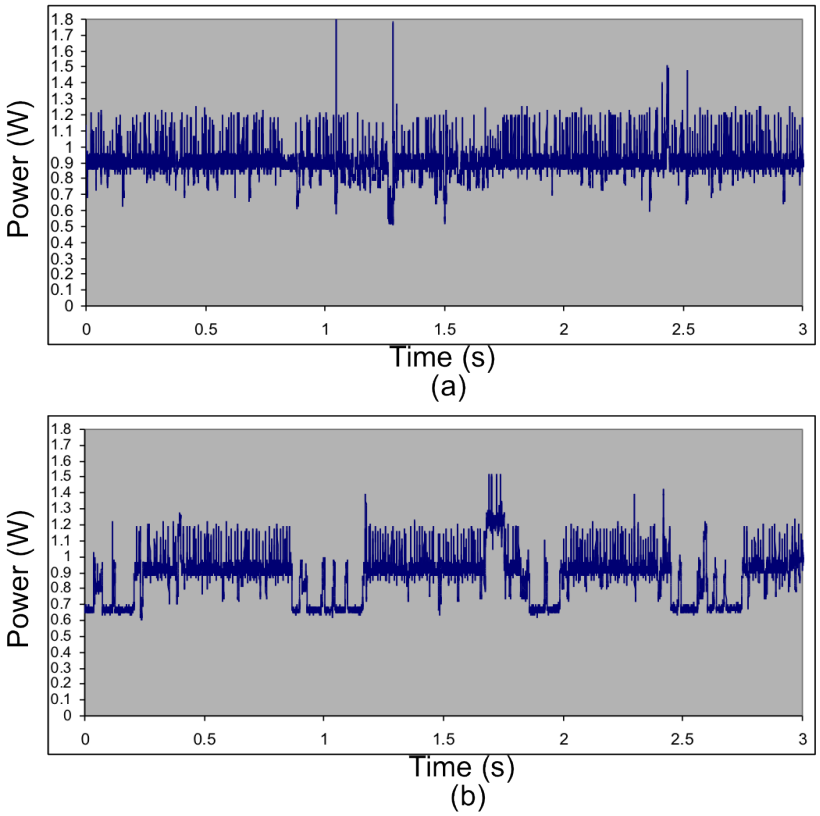


Figure 32: Power consumption of the HTC Hero: (a) Wi-Fi in continuous data transmission mode, (b) Wi-Fi in discontinuous data transmission mode

ACTIVITY UNDERSTANDINGS. The application contains three components: data collection, feature extraction and data interpretation [49]. Data is collected from the triaxial accelerometer with a sampling frequency of 50 Hz in the HTC Hero. To recognize the activities in daily life, we need to deal with a triaxial acceleration data during whole one day. We suppose that the study of activity understandings lasts ten hours for one day. In this hypothesis, 1800000 triaxial acceleration data need to be analyzed for one day. The size of the data is about 20.6M. To recognize the activities, we implement FFT algorithm to extract the features from the raw accelerometer data and decision trees classifier to interpret the accelerometer signal pattern as every corresponding activity. In feature extraction phase, FFT algorithm uses a window size of 256 acceleration data with 50 % overlapping between consecutive windows. This choice has illustrated success in the study [12]. We pass feature extraction and data interpretation with input and output data as a task to the tool OffDeci. Idem, OffDeci generates the five different stubs for this task.

type of stubs	E_1	E_2	E_3	E_4	E_5
OffDeci estimation (mJ)	1.18	22.63	26.37	26.22	28.96
Measurement (mJ)	1.28	25.87	29.15	28.97	32.25

Table 6: Energy cost of the task of the fall detection application in five different stubs

type of stubs	E_1	E_2	E_3	E_4	E_5
OffDeci estimation (mJ)	621025.93	33720.96	6080.82	33685.36	6086.21
Measurement (mJ)	667769.06	39139.91	7106.24	39142.78	7109.1

Table 7: Energy cost of the task of the activity understandings application in five different stubs

8.3 RESULTS

This section assesses the validity of our approach. This validation is done in the context of our experimental platform. The implementation of the two motivating examples have been discussed in previous section. We first use our OffDeci tool to estimate the energy consumption of five different stubs in combination with the implementation of a given task. We then measure the real energy consumption of these five stubs with the implementation in our measurement platform as illustrated in Figure 28. The estimated and measured energy cost are both calculated by the equations 7.10 - 7.14.

For the fall detection application, Table 6 shows respectively the estimation and the real-time measurement of the energy consumed by the task of this application in five different stubs. We observe that the result of the estimation is consistent with measurement. The most energy efficiency stub is to compute locally fall detection algorithm and can save more than 20 time energy cost compared with computation offloading.

For the activity understandings application, we note that the estimated and measured power consumption of five different stubs for the task of this application is also coherent in the Table 7. The result demonstrates that the stub 3 for the task of this application as shown in the Figure 24 is the most energy efficiency stub. This illustrates that data compression has the potential for saving energy.

The activity understandings application can also detect a fall. We implement FFT algorithm for fall detection. The energy cost of the stub 1 for fall detection is up to 37.1 mW. The stub 2 (25.87 mJ) of this application has the minimum value. In this context, the stub 2 (25.87 mJ) of this application becomes the minimum

value. This proves that the computation offloading depends on algorithm complexity.

Our measurement platform monitors power consumption of our smartphone HTC Hero in real time. The platform makes it clear when particular events are occurring. The platform has allowed us to run and analyze a large number of detailed test programs by monitoring the traces of power consumptions in real time. According to these results, the real-time measurement of energy cost validates our energy estimation profiling approach that uses our execution and transfer mode decision OffDeci tool in our experimental platform.

8.4 PREDICTION RULES

As discussed in the previous section, we can use our OffDeci tool to decide which is the best stub in terms of energy consumption for a task with the constant amount of input and output data. However, the amount of input and output data may be varied during the execution of a task. It is difficult to measure results for all possible input and output data volumes with our tool. In this context, we analyze the characteristics of energy cost of computation, data transmission and data compression. We then propose prediction estimation rules for decision helping for the developer of applications.

According to the earlier energy analysis and the result illustrated in the previous section, a task may be performed on three execution and transfer modes as shown in Figure 23. The prediction rules for these three modes is illustrated in Figure 33: a task contains only a small amount of computation with a relatively small amount of input and output data (local mode); a task contains only a large amount of computation with a relatively small amount of input and output data (uncompressed remote mode); a task contains only a large amount of computation with a relatively large amount of input and output data (compressed remote mode).

According to these prediction estimation rules, the programmer can choose one execution and transfer mode for a given task. The choice will be highly dependent on the amount of data to transfer and the computation complexity. For the case that these prediction rules cannot make an accurate decision, the programmer needs:w to use our tool for a task with each specific amount of data.

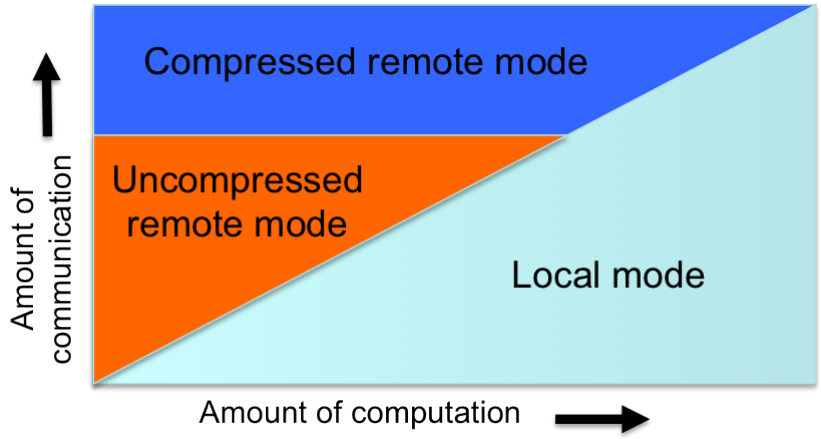


Figure 33: Prediction rules of three execution and transfer modes for a task

8.5 RELATED WORK

Energy efficiency has been always critical for mobile applications on smartphones and the importance seems to be increasing. Mobile applications are developing towards more and more intensive computation and increasing resource-demand. Evolution of battery technology has not been able to match the energy requirements of these mobile applications. The low-level middleware to manage hardware technologies have been explored in literature. They are focusing on optimizing energy efficiency of displays [28], processors [48], communication hardware [26], software transformation and scaling [39] and energy efficient data aggregation and communication [64]. These optimizations reduce energy consumption of their targeted subsystems by factors of 2 to 10 demonstrating the importance of energy scale-down in future designs. However, when we take into account architectures and power saving modes of hardware on smart mobile, the complexity of the software/hardware interactions is making the performance and energy consumption behavior increasingly non intuitive. There is no enough application-level support for these hardware techniques. The programmer has to manage the intricacies of the low-level technologies. Moreover, these approaches minimize independently energy usage of every hardware component. This is a complement of the remote execution approach that offloads the computation to a server for smart phones, using their wireless communication.

Computation offloading has been a challenging research problem of a number of studies. However, only a subset of those studies focus on the effect of the computation offloading on the energy cost of mobile applications on smartphones. In most cases

the focus is on response time and other resource consumption. Large part of the research uses modeling and simulation, like [47], which is an early study of computation offloading from mobile to a fixed server. This work has demonstrated that under certain conditions 20 % energy savings would be possible. The basic feasibility of moving tasks to cloud has been studied [43]. The thresholds for moving to the cloud vary significantly based on the used communication technology like 3G and Wi-Fi. However such work is conducted in a rather ad-hoc way and cannot guide programmer in the development process.

Compiler technology has been used in, e.g., [34, 66], where a program is partitioned into client and server parts. The client parts are run on a mobile device and the server part is offloaded. Even though the measurements show that this approach is able to save significant energy, compiler technology estimates only the execution time for program inputs and cannot give an accurate estimation for energy consumption.

Middleware based approach has been investigated in, e.g., [21]. The described framework uses only the snapshot of resource availability as criteria for deciding between local, remote and hybrid execution. The energy consumption of computation and communication for mobile applications is not considered as a criterion for deciding the offloading computation point.

We have reviewed existing approaches for handling energy efficiency on mobile device in hardware and software systems. To the best of our knowledge, there does not exist tool-based approaches to determining the trade-off point of computation offloading for the programmer. Our work complements existing research by providing a tool to measuring energy consumed by a given program in local, uncompressed remote and compressed remote mode and selecting one execution and transfer mode to guide the programmer in the development application process.

8.6 SUMMARY

In this chapter, we validated our approach based on computation offloading and data compression. We described the specific smartphone platform (HTC Hero) and measurement hardware setup that we have used in our experimental platform. We then explained the five stubs for our two motivating examples into the three execution and transfer modes. We presented energy consumption measurements for these stubs and compared these measurements with results from our tool. We outlined also prediction rules to help developers choose a program execution and data transfer mode for each task of a mobile application. Finally,

we discussed on the related works. The next chapter presents how to integrate OffDeci into DiaSpec in order to manage energy consumption of mobile application via a high-level programming framework.

9

DIASPEC EXTENSION FOR ENERGY HANDLING

The previous chapter introduced the study of two motivating examples that shows the potential to save energy cost of mobile application with the use of two technologies: computation offloading and data compression.

In this chapter, we introduce an approach to managing the energy consumption of mobile applications via a high-level programming framework. This approach allows taking energy concerns into account in the phase of application design by an energy-handling declaration. This approach facilitates the work of architects and developers by separating energy management tasks for mobile applications (illustrated in Figure 34).

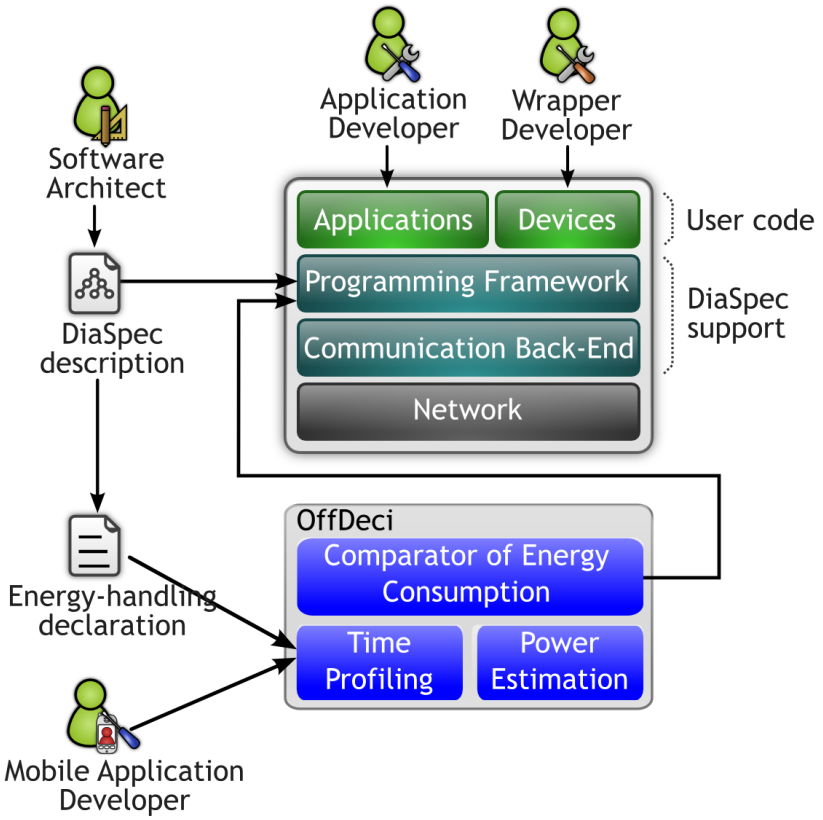


Figure 34: Integration of OffDeci into DiaSpec

Our approach leverages DiaSuite and extends our design language DiaSpec with energy-handling declaration that allows mobile application developer to characterize components of an

application from a energy-management viewpoint. This information pinpoints a component that may be offloaded for saving energy in the architecture description of an application.

To resolve energy concerns, we propose to raise the level of abstraction beyond the code level, by providing declarative support to manage energy cost. The mobile application developer uses energy-handling declaration to specify components that contain a large amount of calculation or processing data. Offloading these components has the potential to save their energy cost. Using this declaration in the architecture description, OffDeci generates five stubs for each declared component. The mobile application developer then passes the implementation of each relevant component. OffDeci combines the implementation with the five different stubs (shown in Figure 24) and executes them. It measures execution time of each stub for an implementation and then calculates its energy cost. Using our execution and transfer strategy scheme (illustrated in Figure 25), OffDeci chooses the best stub for a component in terms of minimizing energy consumption and generates the relevant programming framework, in combination with the implementation of component.

Let us now define our notion of energy concerns by characterizing DiaSpec components and examine the issues to be resolved within the DiaSpec development approach. The following Sections present our approach to energy management of mobile applications with the fall detection application.

9.1 CHARACTERIZING DIASPEC COMPONENT

This Section characterizes DiaSpec components of mobile applications and analyzes need of component offloading for energy saving.

Figure 35 shows a graphical architecture view of the fall detection application, including a smartphone with an accelerometer, call and SMS services. This application works as follows, the accelerometer (AccelSensor component) of the user's smartphone pushes periodically an acceleration value to the AccumulatorAccelContext component. AccumulatorAccelContext accumulates the acceleration values. It then pushes a list of values with a particular size (as event) to FallDetectorContext component. FallDetectorContext can also get a volume of acceleration values with a specific window size (as command) from AccumulatorAccelContext. It then processes a volume of acceleration data whose size depends on the used algorithm (*i.e.,* 200 acceleration data for thresholds method and 256 acceleration data for FFT) and determines whether there is a fall of the user. Eventually, if there

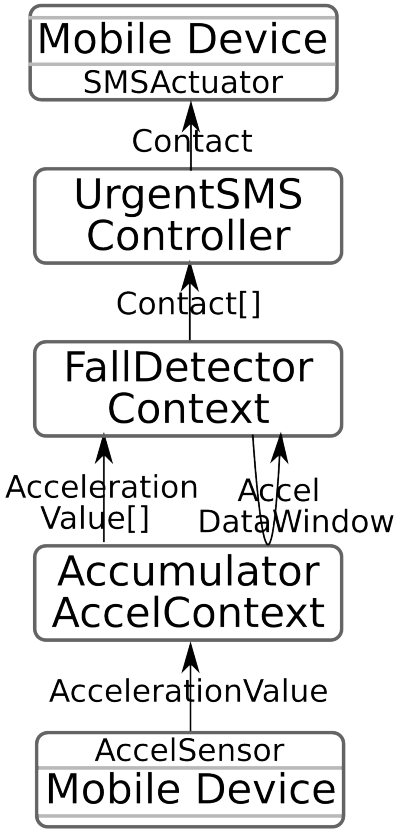


Figure 35: Architecture of the fall detection application

is a fall, FallDetectorContext sends a list of specific contacts to UrgentSMSController component that can invoke SMSActuator component. SMSActuator then sends SMS to a contact. The complete specification of the fall detection application can be found in Listing 4.

```
1  component MobileDevice (String id) { }
2
3  component AccelSensor extends MobileDevice {
4     provides event AccelerationValue to AccumulatorAccelContext;
5  }
6
7  component SMSActuator extends MobileDevice {
8     provides command SendSMS to UrgentSMSController;
9  }
10
11 component AccumulatorAccelContext {
12    requires event AccelerationValue from AccelSensor;
13    provides event AccelerationValue[] to FallDetectorContext;
14    provides command AccelDataWindow to FallDetectorContext;
15 }
16
17 component FallDetectorContext {
18    requires event AccelerationValue[] from AccumulatorAccelContext;
19    requires command AccelDataWindow from AccumulatorAccelContext;
```

```
20    provides event Contact[] to UrgentCallController;
21 }
22
23 component UrgentSMSController {
24    requires event Contact[] from FallDetectorContext;
25    requires command SendSMS from SMSActuator;
26 }
27
28 icommand AccelDataWindow {
29    AccelerationValue[] getAccelDataWindow();
30 }
31
32 icommand SendSMS {
33    void sendSMSContact (Contact contact);
34 }
```

Listing 4: DiaSpec specification of the fall detection application

According to the analysis of the fall detection application in previous Chapter, the energy cost of components of a mobile application is a critical issue because mobile platform is energy-constrained. Let us now categorize DiaSpec components from a energy consumption viewpoint. Our programming framework is component-based. Each DiaSpec component is often decomposed into three stages: acquiring input data (push or pull), processing input data and providing output data. These three stages influence directly on the energy cost of a component. Our programming framework provides also a underlying communication back-end (*i.e.,* local, SIP or RMI) to each DiaSpec component. In this context, a DiaSpec component can be executed either locally or remotely. Let us examine the characteristics of local and remote DiaSpec component.

9.1.1 *Local DiaSpec Component*

Using DiaSpec language, we decompose an application into several components in its architecture description. Some components contain only I/O operations on hardware platform and interact with the native OS. For example, AccelSensor reads an acceleration value on the smartphone and pushes it and SMSActuator sends SMS to a contact. These components are local components that need to be executed natively on the smartphone. Some components process data with few computation and data. For example, AccumulatorAccelContext only accumulates acceleration values. If this component is performed remotely, the energy cost of communication will be very expensive. For the same raison, UrgentSMSController that calls SMSActuator for one by one contact will be also executed locally. Thus, the local DiaSpec component is a component that contains only I/O operations

performed natively on the specific platform or that contains the small amount of calculation or processing data.

9.1.2 *Remote DiaSpec Component*

Contrary to the local DiaSpec component, the remote DiaSpec component is a component that contains the large amount of calculation or processing data. As discussed in the Chapter 7, computation offloading can save energy cost for the large amount of calculation and data compression can save energy cost for large amount of processing data (illustrated in Figure 33). For example, for obtaining higher precision, FallDetectorContext may process a larger volume of acceleration data with a complex algorithm ((*i.e.*, FFT and machine learning) that leads to the large amount of calculation like using the algorithm of activity understandings application to detect a fall.

9.2 DECLARING ENERGY MANAGEMENT

To resolve energy concerns, the mobile application developer analyzes components of an application in its architecture description. Based on the analysis in previous section, the developer classifies components. For local DiaSpec components, their declaration are preserved. For DiaSpec components that may be executed remotely, we extend DiaSpec and provide an energy-handling declaration with the energy keyword. The mobile application developer declares these components with this keyword. For example, the energy keyword applies only to FallDetectorContext in the fall detection application (shown in Listing 5).

```
1 energy {
2    component FallDetectorContext;
3 }
```

Listing 5: Energy-handling declaration of extended DiaSpec

9.3 ANALYZING ENERGY COST

After classification and declaration of components of a mobile application, the mobile application developer passes the implementation of the application with the energy-handling declaration to OffDeci tool. For each component declared by the energy keyword, the time profiling of OffDeci generates five stubs in combination with the implementation (discussed in the previous Chapter). OffDeci estimates respectively the energy cost of

the implementation with five different stubs and chooses the most energy efficiency stub for DiaSpec runtime. For example, FallDetectorContext component is the only component that may be executed remotely in the fall detection application. The others components will be performed locally and located on the smartphone. In this context, we focus on FallDetectorContext component that is declared by the energy-handling declaration. OffDeci takes this declaration and the implementation and generates five stubs as illustrated in Figure 36. If FallDetectorContext component is executed remotely, it must communicate with AccumulatorAccelContext and UrgentSMSController components using SIP or RMI communication bus. In Figure 36 red words denote the underlying communication back-end to use and green words point out whether the transmission data is compressed or not. After the phase of time profiling, OffDeci calculates the energy cost of stubs and chooses the best stub that consumes less energy for this application.

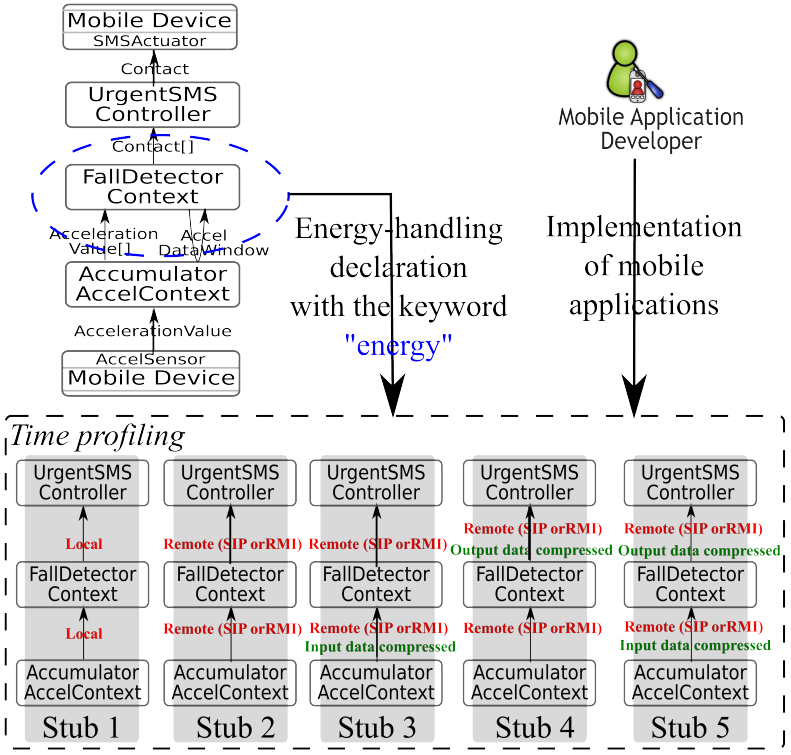


Figure 36: Implementation of extended energy-handling DiaSpec

9.4 SUMMARY

In this chapter, we presented our approach that allows mobile application developers characterize components of an application from an energy-management viewpoint. This approach

leveraged DiaSuite and extended our design language DiaSpec with energy-handling declarations. As a result, energy concerns can be expressed declaratively during the design phase of an application.

10

SUMMARY

We have described the energy issues of mobile applications in pervasive computing environments in the second part of this thesis. Since smartphones are energy-constrained platforms, we have presented the importance to manage the energy cost of applications on these platforms. We have presented an approach to help the mobile application developers to decide when and how to offload each task of a mobile application. This approach has relied on computation offloading and data compression technologies. According to these two technologies, we presented an execution and transfer model for a given task and its five different stubs for three program execution and data transfer modes. We have integrated these two technologies and the execution and transfer model into a strategy scheme for handling energy issues. This strategy scheme described a methodology to determine the most efficient stub in terms of energy cost. We have constructed the OffDeci tool, using this strategy scheme, to provide the energy feedback for the developer. Our experimental study has proven the feasibility of the strategy scheme of our approach. Finally, we extended DiaSpec with declaration dedicated to energy handling. We have sketched the integration of the energy-handling declaration and OffDeci into our high-level programming framework to manage energy concerns during the application design phase.

Part III

CONCLUSION

11

CONCLUSION

We now summarize the thesis results and draw overall conclusions. In this thesis we were mainly interested in creating pervasive applications and the various related problems. Pervasive computing environments are enriched by computing and communication capabilities. That promises new ways to integrate environments with the human user. Potential applications in the field of pervasive environments are numerous, ranging from building management to healthcare. Developing these applications is challenging because pervasive computing combines elements of several domains, thus inheriting and creating many challenges for the application developer (illustrated in Chapter 2). We focused on two main aspects: heterogeneity and energy concerns. Indeed, these two aspects complicated greatly the application development. The heterogeneity is one of these important challenges. This is due to the fact that the essence of these environments is entity-rich networked environments. Various entities are heterogeneous in many aspects (*eg*, network layers and communication modes). Another important challenge is energy consumption of mobile applications, because more and more applications and services on smartphones increase their energy demands. However, the smartphone is an energy-constrained device, using a battery. In this context, we have presented an approach to integrating heterogeneity and energy concerns into the development process of pervasive computing systems via a high-level programming framework.

In the first part of this thesis, we have proposed an approach to managing heterogeneity of pervasive computing environments. The increasing heterogeneity and dynamicity of pervasive computing environments calls for novel approaches to model and implement both middleware and system infrastructure parts of an application. While existing software engineering approaches support the developers for the most error-prone tasks and ad hoc techniques [44], they still lack a platform that relies on industrial standards to tackle the challenges of pervasive computing environments.

Our approach proposes a solution to this problem by leveraging an ADL (DiaSpec) dedicated to the pervasive computing domain and by using SIP as a universal communication bus for pervasive computing environments. Our approach describes a methodology and programming support to adapt heterogeneous

entities to the SIP communication bus and provides developers with an integrated programming framework for the application creation, using a high-level abstraction layer for all type entities.

We have detailed how to enable homogeneous communications between heterogeneous distributed entities. Our approach has been used to make a wide variety of entities SIP compliant. These entities have then been integrated into a number of applications for home automation and healthcare. Then, we have shown our implementation of a variety of entities running on different hardware platforms. Finally, we have analyzed and discussed our experimental measurements on memory footprint and communication performance and have demonstrated that our approach is realistic for all classes of entities, and that our SIP gateway can run efficiently on resource-constrained platforms.

In the second part of this thesis, we have proposed an approach to managing the energy concerns of pervasive computing environments. As smartphones get smaller, they become constrained on their computing power and battery energy. Yet, a lot of modern applications are resource-intensive, especially with great energy demands. To cope with these shortcomings, it is important to handle energy issues for mobile applications in the design phase. However, existing software approaches can't give energy consumption feedback in the design and development of application.

Our approach explores the potential of computation offloading and data compression technologies to save energy cost for mobile applications. According to these two technologies, we have constructed an execution and transfer model for a given task and introduced five different stubs for three program execution and data transfer modes. Based on these two technologies and the execution and transfer model, we have built up a strategy scheme for handling energy issues. This strategy scheme includes a methodology to determine the most efficient stub in terms of energy cost. We have constructed the OffDeci tool, using this strategy scheme, to give an energy consumption feedback of a given task of a mobile application without any external equipment assistance. Our experimental results show that, even under an ordinary wireless LAN environment, the scheme can result in significant energy-saving for our motivating examples. The measurement of energy cost in our experimental platform has validated our energy estimation profiling approach (our OffDeci tool).

To manage energy concerns during the phase of pervasive application design, we integrated our OffDeci tool into our high-level programming framework. In doing so, we enriched DiaSpec with the declaration dedicated to energy handling. OffDeci

uses the added information to generate five stubs for each declared DiaSpec component and combines with his implementation. OffDeci then executes them and helps the developer to select the best stub which consumes less energy.

12

FUTURE WORK

After drawing an overview conclusion in the previous Chapter, we now point out remaining problems and outline a number of possible future extensions to this work.

Concerning the heterogeneity-handling with the use of SIP protocol, as part of future work we will work on the design and implementation of an application scenario: an smart house. The application will use SIP as a universal communication bus. With the development of SIP network in the house, our application includes a number of the electronic appliances in a house which are controlled by a SIP gateway according to the user needs. This implementation allows us testing our approach with larger application scope.

Currently, our approach manages energy concerns via a high-level programming framework. We extend DiaSpec with the energy-handling declaration and integrate our OffDeci tool into our high-level programming framework. However, we haven't yet validated this integration. We plan to conduct more experiments on various mobile platforms to test coherence between the estimated and measured energy cost of each stub of a DiaSpec component. Furthermore, our generated and dedicated programming framework abstracts over some distributed communication layers (e. g. RMI or SIP). Our experimental study has only validated our OffDeci tool with use of a socket communication. RMI and SIP are upper protocols across IP and add more payload information for communication charge compared with Internet socket. Thus we also plan to conduct more experiments on these distributed communication layers and to analyze these influences.

For now, OffDeci generates five stubs for each DiaSpec component declared by the energy-handling declaration. However, it is not necessary to generate always five stubs to decide the best choice in terms of energy consumption. Data compression has the potential to save energy cost of mobile applications but the savings from reducing the volume of transmission data need to exceed the energy cost of the additional computation for compressing or uncompressing data on smartphones. For example, if the input or output data of a DiaSpec component is a single scalar value, the data compression can't save energy for these data. We don't need to generate the relevant stubs that compress or uncompress these data. Therefore, we plan to refine this phase of

generation with the existing architecture description. In DiaSpec, we can obtain the input and output data type of a component from its interaction mode declaration. An interaction mode is either command, event, or session that is combined with the name of a Java type to characterize the interaction. For a command, the Java type is an interface listing the relevant methods. We can obtain the data type from the return values of these methods. For an event or session, the Java type indicates the type of the data that are exchanged. Using this information, OffDeci can refine this phase of the generation and generate either two, three or five efficient stubs to decide the most efficient stub concerning energy cost.

Part IV

APPENDICES

BIBLIOGRAPHY

[1] Asn.1 encoding rules: Specification of basic encoding rules (BER), canonical encoding rules (CER) and distinguished encoding rules (DER). ITU-T X.690.

[2] Eukréa SBC Board, http://www.eukrea.com.

[3] The Heyu project, http://heyu.tanj.com.

[4] iButton SDK, http://www.maxim-ic.com/products/ibutton.

[5] kSOAP 2, http://ksoap2.sourceforge.net.

[6] The Linux kernel, http://kernel.org.

[7] The GNU oSIP library, http://www.gnu.org/software/osip.

[8] SIP for Instant Messaging and Presence Leveraging Extensions. IETF Working Group, http://www.ietf.org/html.charters/simple-charter.html.

[9] X10 communication protocol, http://www.x10.org.

[10] The ZigBee Alliance, http://www.zigbee.org.

[11] S. Apel et K. Bohm. Towards the development of ubiquitous middleware product lines. In *ASE'04 SEM Workshop*, pages 137–153, Linz, Austria, 2005.

[12] Ling Bao et Stephen S. Intille. Activity recognition from user-annotated acceleration data. pages 1–17. 2004.

[13] Len Bass, Paul Clements, et Rick Kazman. *Software Architecture in Practice*. Addison-Wesley Professional, 1997. isbn: 0201199300.

[14] Stefan Berger, Henning Schulzrinne, Stylianos Sidiroglou, et Xiaotao Wu. Ubiquitous computing using SIP. In *NOSSDAV '03: Proceedings of the 13th international workshop on Network and operating systems support for digital audio and video*, pages 82–89, New York, NY, USA, 2003.

[15] L. Burgy, C. Consel, F. Latry, J. Lawall, N. Palix, et L. Réveillère. Language technology for Internet-telephony service creation. In *IEEE International Conference on Communications*, 2006.

[16] B. Campbell, J. Rosenberg, H. Schulzrinne, C. Huitema, et D. Gurle. Session initiation protocol (SIP) extension for instant messaging. RFC 3428, IETF, 2002.

[17] D. Cassou, B. Bertran, N. Loriant, et C. Consel. A generative programming approach to developing pervasive computing systems. In *Proceedings of the 8th International Conference on Generative Programming and Component Engineering (GPCE'09)*, 2009.

[18] Jiangpeng Dai, Xiaole Bai, Zhimin Yang, Zhaohui Shen, et Dong Xuan. Mobile phone-based pervasive fall detection. *Personal Ubiquitous Comput.*, 14:633–643, 2010.

[19] J.W. Davis. Power benchmark strategy for systems employing power management. In *Electronics and the Environment, 1993., Proceedings of the 1993 IEEE International Symposium on*, pages 117 –119, 1993.

[20] Mian Dong et Lin Zhong. Self-constructive high-rate system energy modeling for battery-powered mobile systems. In *Proceedings of the 9th international conference on Mobile systems, applications, and services*, pages 335–348, New York, NY, USA, 2011.

[21] Jason Flinn, Dushyanth Narayanan, et M. Satyanarayanan. Self-tuned remote execution for pervasive computing. In *In Hot Topics in Operating Systems(HotOS-VIII*, pages 61–66, 2001.

[22] Zhengming Fu, Eugenio Culurciello, Patrick Lichtsteiner, et Tobi Delbrück. Fall detection using an address-event temporal contrast vision sensor. In *ISCAS*, pages 424–427. 2008.

[23] Kinshuk Govil, Edwin Chan, et Hal Wasserman. Comparing algorithm for dynamic speed-setting of a low-power cpu. In *Proceedings of the 1st annual international conference on Mobile computing and networking*, pages 13–25, New York, NY, USA, 1995.

[24] M. Handley et V. Jacobson. SDP: Session Description Protocol. RFC 2327, IETF, 1998.

[25] David P. Helmbold, Darrell E. Long, et Bruce Sherrod. A dynamic disk spin-down technique for mobile computing. pages 130–142, 1996.

[26] M. Honkanen, A. Lappetelainen, et K. Kivekas. Low end extension for bluetooth. In *Radio and Wireless Conference, 2004 IEEE*, pages 199 – 202, 2004.

[27] ITU-T. Recommendation Q.23: Technical features of push-button telephone sets, http://www.itu.int/rec/T-REC-Q.23/.

[28] Subu Iyer, Lu Luo, Robert Mayo, et Parthasarathy Ranganathan. Energy-adaptive display system designs for future mobile environments. In *Proceedings of the 1st international conference on Mobile systems, applications and services*, pages 245–258, New York, NY, USA, 2003.

[29] Wenyu Jiang, Jonathan Lennox, Sankaran Narayanan, Henning Schulzrinne, Kundan Singh, et Xiaotao Wu. Integrating Internet telephony services. *IEEE Internet Computing*, 6(3): 64–72, 2002.

[30] W. Jouve, N. Ibrahim, L. Réveillère, F. Le Mouël, et C. Consel. Building home monitoring applications: From design to implementation into the Amigo middleware. In *ICPCA'07: IEEE International Conference on Pervasive Computing and Applications*, pages 231–236, 2007.

[31] W. Jouve, J. Lancia, N. Palix, C. Consel, et J. Lawall. High-level programming support for robust pervasive computing applications. In *Proceedings of the 6th IEEE Conference on Pervasive Computing and Communications (PERCOM'08)*, pages 252–255, Hong Kong, China, 2008.

[32] W. Jouve, N. Palix, C. Consel, et P. Kadionik. A SIP-based programming framework for advanced telephony applications. In *Proceedings of The 2nd LNCS Conference on Principles, Systems and Applications of IP Telecommunications (IPTComm'08)*, 2008.

[33] G. Klyne et D. Atkins. Common Presence and Instant Messaging (CPIM): Message format. RFC 3862, IETF, 2004.

[34] Ulrich Kremer, Jamey Hicks, et James M. Rehg. A compilation framework for power and energy management on mobile computers. In *In International Workshop on Languages and Compilers for Parallel Computing (LCPC?01*, 2001.

[35] Sudha Krishnamurthy et Lajos Lange. Distributed interactions with wireless sensors using TinySIP for hospital automation. In *PerSeNS'08: The 4th International Workshop on Sensor Networks and Systems for Pervasive Computing*, Hong-Kong, China, 2008.

[36] F. Latry, J. Mercadal, et C. Consel. Staging Telephony Service Creation: A Language Approach. In *Principles, Systems and Applications of IP Telecommunications, IPTComm*, New-York, NY, USA, 2007.

[37] Mike Tien-Chien Lee, V. Tiwari, S. Malik, et M. Fujita. Power analysis and minimization techniques for embedded dsp software. *Very Large Scale Integration (VLSI) Systems, IEEE Transactions on*, 5(1):123 –135, 1997.

[38] Maxim. The 1-Wire Bus, http://www.maxim-ic.com/products/1-wire.

[39] Robert N. Mayo, Robert N. Mayo, Parthasarathy Ranganathan, et Parthasarathy Ranganathan. Energy consumption in mobile devices: Why future systems need requirements-aware energy scale-down, 2003.

[40] Nenad Medvidovic et Richard N. Taylor. A classification and comparison framework for software architecture description languages. *IEEE Transactions on Software Engineering*, 26(1): 70–93, 2000.

[41] S.-G. Miaou, null Pei-Hsu Sung, et null Chia-Yuan Huang. A customized human fall detection system using omni-camera images and personal information. *Distributed Diagnosis and Home Healthcare*, 0:39–42, 2006.

[42] SUN microsystems. Media Framework API (JMF), http://java.sun.com/products/java-media/ jmf.

[43] Antti.P Miettinen et Jukka.K Nurminen. Energy efficiency of mobile clients in cloud computing. In *2nd USENIX Workshop on Hot Topics in Cloud Computing*, 2010.

[44] S. Moyer, D. Marples, et S. Tsang. A protocol for wide area secure networked appliance communication. *Communications Magazine, IEEE*, 39(10):52–59, Oct 2001.

[45] Jukka.K Nurminen et Mikko Heikkinen. Consumer attitudes towards energy consumption of mobile phones and services. In *VTC-Fall 2010*, 2010.

[46] M N Nyan, Francis E H Tay, M Manimaran, et K H W Seah. Garment-based detection of falls and activities of daily living using 3-axis mems accelerometer. *Journal of Physics: Conference Series*, 34(1):1059, 2006.

[47] Mazliza Othman et Stephen Hailes. Power conservation strategy for mobile computers using load sharing. *ACM Mobile Computing and Communication Review*, 2:44–50, 1998.

[48] Padmanabhan Pillai et Kang G. Shin. Real-time dynamic voltage scaling for low-power embedded operating systems. *SIGOPS Oper. Syst. Rev.*, 35:89–102, 2001.

[49] Nishkam Ravi, Nikhil D, Preetham Mysore, et Michael L. Littman. Activity recognition from accelerometer data. In *In Proceedings of the Seventeenth Conference on Innovative Applications of Artificial Intelligence(IAAI*, pages 1541–1546. 2005.

[50] A. B. Roach. Session Initiation Protocol (SIP)-Specific Event Notification. RFC 3265, IETF, 2002.

[51] Manuel Román, Christopher Hess, Renato Cerqueira, Anand Ranganathan, Roy H. Campbell, et Klara Nahrstedt. A middleware infrastructure for active spaces. *IEEE Pervasive Computing*, 1(4):74–83, 2002.

[52] J. Rosenberg. A presence event package for the session initiation protocol SIP: Session Initiation Protocol. RFC 3856, IETF, 2004.

[53] J. Rosenberg et H. Schulzrinne. An offer/answer model with the session description protocol (sdp). RFC 3264, IETF, 2002.

[54] J. Rosenberg, J. Lennox, et H. Schulzrinne. Programming Internet telephony services. *IEEE Internet Computing*, 3(3): 63–72, 1999.

[55] J. Rosenberg, H. Schulzrinne, G. Camarillo, A. Johnston, J. Peterson, R. Sparks, M. Handley, et E. Schooler. SIP: Session Initiation Protocol. RFC 3261 (Proposed Standard), 2002. url: `http://www.ietf.org/rfc/rfc3261.txt`. Updated by RFCs 3265, 3853, 4320, 4916, 5393, 5621.

[56] J. Rosenberg, H. Schulzrinne, et O. Levin. A session initiation protocol (SIP) event package for conference state. RFC 4575, IETF, 2006.

[57] Rosenberg, J. et al. SIP : Session Initiation Protocol. RFC 3261, IETF, 2002.

[58] J.T. Russell et M.F. Jacome. Software power estimation and optimization for high performance, 32-bit embedded processors. In *Computer Design: VLSI in Computers and Processors, 1998. ICCD '98. Proceedings. International Conference on*, pages 328 –333, 1998.

[59] Debashis Saha et Amitava Mukherjee. Pervasive computing: A paradigm for the 21st century. *IEEE Computer*, 36(3):25–31, 2003.

[60] Debashis Saha, Amitava Mukherjee, et Somprakash Bandyopadhyay. *Networking Infrastructure for Pervasive Computing: Enabling Technologies and Systems*. Kluwer Academic Publishers, 2002. isbn: 140207249X.

[61] M. Satyanarayanan. Pervasive computing: vision and challenges. *Personal Communications, IEEE*, 8(4):10 –17, 2001.

[62] H. Schulzrinne, S. Casner, R. Frederick, et V. Jacobson. Rtp: A transport protocol for real-time applications. RFC 3550, IETF, 2003.

[63] Ron Shacham, Henning Schulzrinne, Srisakul Thakolsri, et Wolfgang Kellerer. Ubiquitous device personalization and use: The next generation of IP multimedia communications. *ACM Trans. Multimedia Comput. Commun. Appl.*, 3(2): 12, 2007.

[64] Mohamed A. Sharaf, Jonathan Beaver, Alexandros Labrinidis, Ros Labrinidis, et Panos K. Chrysanthis. Tina: A scheme for temporal coherency-aware in-network aggregation. In *In MobiDE*, 2003.

[65] H. Sugano, S. Fujimoto, G. Klyne, A. Bateman, W. Carr, et J. Peterson. Presence Information Data Format (PIDF). RFC 3863, IETF, 2004.

[66] Cheng Wang et Zhiyuan Li. A computation offloading scheme on handheld devices. *J. Parallel Distrib. Comput.*, 64:740–746, 2004.

[67] Mark Weiser. The computer for the 21st century. *SIGMOBILE Mob. Comput. Commun. Rev.*, 3:3–11, 1999.

[68] X. Wu et H. Schulzrinne. Programmable end system services using SIP. In *Proceedings of The IEEE International Conference on Communications 2002*. 2003.

[69] X. Wu, P. Koskelainen, et H. Schulzrinne. Use of session initiation protocol (SIP) and simple object access protocol (SOAP) for conference floor control. Internet draft, IETF, September 2003.

[70] Yu Xiao, Matti Siekkinen, et Antti Ylä-Jääski. Framework for energy-aware lossless compression in mobile services: The case of e-mail. In *ICC*, pages 1–6. 2010.

ABSTRACT

The topics of heterogeneity and energy are two fundamental considerations for pervasive computing environments. In this thesis, we describe our approach to manage heterogeneity and to handle energy concerns via a high-level programming framework. To manage heterogeneity, we describe a methodology and a programming support that use the SIP protocol as a universal communication bus in pervasive computing environments. Our work enables homogeneous communications between heterogeneous distributed entities. In doing so, we integrate the SIP communication bus into our programming framework. We rely on a declarative language named DiaSpec to describe the architecture of pervasive applications. This description is passed to a generator for producing a Java programming framework dedicated to the application area. We leverage the generated framework with SIP adaptations to raise the abstraction level of SIP operations. Regarding the energy concerns, we describe a methodology that uses two strategies, namely computation offloading and data compression, to minimize energy cost of mobile applications. In doing so, we present an execution and transfer model for a task of a mobile application and define its five different stubs for three program execution and data transfer modes. Based on this model and our two strategies, we construct a strategy scheme to determine the most efficient stub in terms of energy consumption. We then design the OffDeci tool, using this strategy scheme, to provide energy feedback for the developer and to analyze the balance between local and remote computing with consideration of data compression. We sketched the integration of this energy-handling declaration and OffDeci into our high-level programming framework.

KEYWORDS: Architecture Description Language, Domain-Specific Language, SIP, Heterogeneity, Energy consumption, Mobile application, Computation offloading, Data compression

RÉSUMÉ

L'hétérogénéité des objets communicants et la consommation d'énergie sont deux considérations fondamentales pour les environnements informatiques ubiquitaires. Dans cette thèse, nous présentons notre approche pour gérer l'hétérogénéité et pour économiser l'énergie via des canevas de programmation dédiés. Pour gérer l'hétérogénéité, nous proposons une méthodologie et un support de programmation qui visent à faire communiquer les différents objets communicants de l'environnement ubiquitaire, et ce, en utilisant le protocole SIP considéré alors comme un bus de communication universel. Nous avons intégré ce bus SIP dans les canevas de programmation générés à partir des descriptions d'applications d'informatique ubiquitaire, écrites dans le langage DiaSpec. Concernant la consommation d'énergie, nous proposons une méthodologie qui utilise les techniques d'offloading et de compression de données pour minimiser la consommation d'énergie des applications mobiles. Nous avons ainsi construit une stratégie d'aide à la conception au travers d'un outil qui permet de déterminer le meilleur mode d'exécution pour une tâche donnée. Nous proposons l'intégration de cette stratégie dans le langage de description DiaSpec.

MOTS CLÉS: Architecture Logicielle, Langage Dédié, SIP, Hétérogénéité, Consommation d'énergie, Applications mobiles, Calcul déporté, Compression de données