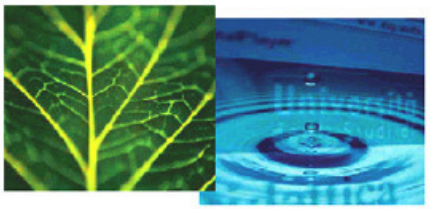
**PhD Dissertation**



**International Doctorate School in Information and Communication Technologies**

DIT - University of Trento

# Modeling and Compostion of Environment-as-a-Service

Tao Peng

Advisor:

Prof. Marco Ronchetti

Università degli Studi di Trento

February 2015

# Acknowledgement

I owe my greatest debt of gratitude to my advisor, Prof. Marco Ronchetti, who has constantly given me guidance and help. He has provided me invaluable advices on defining the research topic, developing the solution, and formalizing manuscripts. His dedication and enthusiasm to scientific research has encouraged me. He also cares about my student life besides doing research, which makes me feel backed up.

I would like to thank Centro Ricerche GPI for financially supporting my Ph.D. study and giving me the opportunity to experience the real health-care IT system development. Our "capo" (head) Giampaolo Armellin has created an environment for us to focus on research related activities. Anna-maria Chiasera gave me great advices on research and manuscript writing. I often miss the days working on interesting projects with great colleagues: Cristina, Dario, Jovan, Tefo, Manuella, Cesare...

I am grateful to Prof. Fabio Casati for advising me since my master study, and for bringing in the connections to academy and industry outside. Florian Daniel gave me great advices on doing research and formalizing manuscripts, with the expertise on web engineering and business process management and experiences of serving as scientific editorial committees. The senior Ph.D. students also helped me a lot through discussions.

Outside Trento, I have also received lots of help in different periods. Prof. Chi-Hung Chi hosted me at CSIRO Australia and provided insightful discussions and guidance on writing. Prof. Katia Jaffres-Runser and Prof. Gentian Jakllari at INPT-ENSEEIHT France gave me the opportunity to get hands on a data prefetching project and constructive comments. My thanks go to my tutor Keiichi Shima for the guidance during my stay at IIJ

Innovation Institute in Tokyo, Randy Bush for the advice on trust under decentralized architecture, and other colleagues.

I thank my fellow Chinese students and scholars in Trento for all the good memories. Finally, I would like to thank my family, especially my wife Xiaojuan Huang, for supporting and encouraging me through the Ph.D. study.

# Abstract

*Wireless-enabled electronic devices are becoming cheaper, more powerful and thus more popular. They include sensors, actuators, smartphones, tablets, wearable devices, and other complex devices. They can carry out complex tasks, cooperating with their "neighbors". However, it is difficult to develop mobile applications to exploit the full power of available resources because the computational capabilities on devices are not homogeneous, and their connectivity changes with physical movement. We propose a mobile environment model to describe the connected devices and study the structural and behavioral characteristics of the environments. Based on the model, we design the routing protocols and a language to support the composition of environments. We propose a framework to provide a unified, flexible and scalable service for task/process deployment and execution on top of the heterogeneous and dynamic mobile environments. We compare different architectures, and discuss the optimization of resources discovery and routing algorithm. A proof-of-concept framework is implemented and shows the feasibility of our Environment-as-a-Service approach. Finally, we explore the theoretical principles and practical techniques for performance optimization, including a data prefetching technique and a dynamic process/task allocation algorithm.*

**Keywords**
mobile device, cloud, service, business process.

# Contents

# List of Tables

# List of Figures

# Chapter 1

# Introduction

With the development of electronic and telecommunication technologies, devices are becoming smart and inter-connected. These connected devices (including sensors, actuators, smartphones, tablets, domestic electronics, smart vehicles, and other complex devices) are playing more and more important roles in our daily lives and industrial environment. According to CISCO [19], 8.7 billion objects were connected by 2012, and the number is expected to reach 50 billion by 2020. These devices provide very diverse capabilities and connect with each other via wired or wireless connections (Wi-Fi, Bluetooth, etc.). They assist us to monitor and interact with the environment, and carry out complex tasks. For example, devices such as smartphones are emerging as working equipments for workers in different industries [43]. Although each device provides a limited set of capabilities, integrated together, they have high potential in different domains, including healthcare, industry automation, emergency response, and many others.

However, it becomes more difficult to develop and deploy applications that can utilize the capabilities of the devices. We explain the challenges with examples under different scenarios.

## 1.1 Scenarios

Healthcare applications, application integration on smartphones and location-aware services are some example scenarios facing this challenge.

Mobile devices are more and more used to provide sophisticated personal healthcare assistance, such as in [61]. A simpler example scenario is: a patient can have monitoring devices such as blood pressure meter, and a clinic has more devices and access to remote repository of healthcare records of patients. To develop an application that assists the blood pressure monitoring, the developers need to analyze the available devices and accessible Application Programming Interfaces (APIs) in different environments, connect these resources, and integrate them to provide assistance to the monitoring. When the availability of devices in environment changes or we need to provide new healthcare services, developers need to go over the process again, because the application logic is tightly-coupled with the available hardware devices and other resources such as API access. The emergence of wearable devices those monitor the health status, including watches, chest belts, brings new possibilities for healthcare applications. The need of an efficient way to manage the diverse resources especially on mobile devices is also emerging.

Mobile application integration is another scenario that requires a framework to manage the bundled hardware/software resources. Many applications that a user installs have similar components, for example, barcode scanning, or Optical Character Recognition (OCR), etc. The development and integration of such components cost inefficient duplicated efforts, and occupies unnecessary space on phone. A framework to model the availability of these components and integrate them into new applications can make application more flexible, reduce the applications sizes, and allows the developers to focus on either the component development or the appli-

cation integration. Android provides a mechanism for the intra-application invocation, using Intent [1]. However, it is merely an interface to invoke applications on the same device, and is not capable for the across-device invocation and sophisticated resource management.

The prevailing of smartphones has created lots of opportunities for the location-aware services. One usage scenario for location-aware services is to apply the detected context information to help users to find the information of interest under the current situation. According to McKinsey, the number of horizontal Web searches from personal computers in France is outstripped by vertical and mobile searches [46]. People often search on mobile device for facilities (restaurants, hotels, parking places, shops, etc.) nearby current location, or near destination or en route [89]. Different with search on desktop, searching on a smartphone does not only implies a location to filter the search results, but also implies the urgent need of services at that time and that location, which has a high conversion rate into concrete business. However, current services on smartphone merely use the detected location information to filter the search results. The matching of users need and the services provided nearby still need to be done manually by the user: he/she has to check the rating of the hotels/restaurants on a website, and open another website to start booking. There are services of integration by mashing up information from correlated websites [98]. For example, the website first shows hotel ratings to user, then allows the user to book the room, and finally suggests car renting service in that city. Mobile context information (including the time, location, user preference, previous services, etc.) can be used as input to provide more accurate information for the successive service. In some situation where the connection is unreliable, it can increase the service availability to predict the functions that might be invoked and prefetch the correlated data onto mobile device

[1] http://developer.android.com/reference/android/content/Intent.html

[63]. It is more attractive if we have a framework that understands both the requirements of the user and the resources availability provided by the facilities, and the framework matches the requirements with resources and starting a user-specific process to take care of the potential needs of the user.

However, it is not easy to create such a framework. We need to cope with the challenges caused by the capability difference and connectivity vulnerability intrinsic to mobile devices and more specifically:

- The capabilities of devices are diverse. Devices are designed to perform different tasks by interacting with the information systems and the physical environment. These devices with different capabilities are scattered in the environment, constituting powerful but diverse environments. It is difficult to develop applications and connect these devices to exploit their capabilities.

- The connection across devices are unreliable and costly. Wireless connection is unreliable due to the possible obstacle, interference and device movement. And wireless connection is expensive in terms of battery and sometimes also in terms of money.

- The connection topology of devices is complex and changeable. The physical location of mobile devices are changing, thus the connectivity on mobile devices are also changing. The changing topology makes it difficult for application to route a task to the proper destination.

To solve these problems, we first need to answer these questions:

- How can we model the capacity to carry out computation and the capability to provide services?

- How can we enable the scalable management of the environments, allowing the environment to grow and shrink with the joining and

leaving of devices?

- How can we provide an abstract description of environmental resources and the tasks' requirements, to ease the deployment of tasks in the environment?

All these challenges are difficult with mobile devices in the environments.

## 1.2 State of the Art

With the advance of network technologies and the emergence of network connected devices, developing software across heterogeneous devices has been a challenge for researchers. In this section, we introduce the state of the art of researches from relevant topics, including Service-Oriented Architecture, Cloud Computing, Business Process Management and Wireless Sensor Networks. Detailed comparison between existing work and ours will be given later in Chapter 6.

Under the topic of Service-Oriented Architecture (SOA) [28, 76], a set of concepts and tools are developed. Standards and protocols are proposed to bridge the gap between the device interfaces and the business applications [16]. Legacy softwares together with the underlying hardware resources are bundled as autonomous services [76]. Services are invocable using an interface description language (e.g., Web Service Definition Language (WSDL) [17]). Implementations are hidden behind the interfaces. And services can compose with each other to provide more complex functions. Web service (WS) defines a Web-based communication interface to implement SOA [4]. Researchers extend Service-Oriented Architecture to involve mobile devices, either to consume or to provision the services [87, 37, 75].

Cloud Computing utilizes a group of networked servers to provide logically centralized services [12, 11, 34]. According to the type of services

provided, Infrastructure-as-a-Service (IaaS), Platform-as-a-Service (PaaS), and Software-as-a-Service (SaaS) are some of the popular concepts in cloud computing industry [8, 71, 92]. There are also similar concepts such as "grid computing" [33], "utility computing" [72]. The idea behind these concepts is to provide computing resources as a on-demand, elastic and scalable utility. Michael Armbrust stated in [8]: "This elasticity of resources, without paying a premium for large scale, is unprecedented in the history of IT." According to [25], "The term mobile cloud computing was introduced not long after the concept of cloud computing." Mobile cloud computing refers to the architecture that mobile devices offload resource-intensive computing tasks to the cloud infrastructures. There are researches that integrate the resources available on mobile devices to provide various cloud services, including image processing, natural language processing, crowd computing, GPS/internal data sharing, sensor data application, multimedia search, social networking [30, 18, 81, 100, 52, 13].

However, existing researches fail to address some major concerns on mobile devices. They seldom base on the heterogeneous resources on mobile devices, but assume that mobile devices provide homogeneous functions, such as computation, data store, or certain type of sensing. The diverse capabilities of mobile devices are difficult to exploit.

And the security and privacy concerns on mobile devices still remain: on one hand, devices owners may not be aware of to whom and how the devices will be shared; on the other hand, resources users may not know where their requested functions are actually provisioned, and to what extend the usage footprint may be disclosed and/or logged.

## 1.3 Our Approach: Environment-as-a-Service

We propose a model of environment which allows the composition of environments and the task execution in composed environments. We name the model "Environment-as-a-Service", which is inspired by the concepts (Infrastructure-as-a-Service, Platform-as-a-Service, Software-as-a-Service) from cloud computing.

An Environment contains a collection of connected devices. It is modeled as a service, which implies:

- An Environment is autonomous. An Environment has the necessary resources to carry out the functions that it provides. Resources management and message exchange are performed within the Environment, but do not depend on external resources.

- An Environment provides a service: it can execute a set of functions in the form of task execution via predefined interfaces. Powered by the composing devices, an Environment can receive computational tasks either from outside of the Environment or from an internal node, and execute the tasks. The service offered by an Environment is ready-to-use functions, so users do not need to worry about device utilization, resource management, internal messaging, and other contained services. Internal nodes or external users can utilize the published functions on-demand, by assigning tasks to the Environment.

- Environments are composable. Because Environments are defined and implemented under the same model and using the same architecture, they are able to connect with each other to form larger Environments. In a composed Environment, the children Environments remain autonomous and their internal structures are not altered. So it is possible to decompose into individual smaller Environments. Being compos-

able and decomposable, Environments are able to scale up or shrink down. The service provided is thus elastic and scalable.

An Environment is modeled as a graph: a vertex represents a device, and an edge represents the capability of connecting two devices. An Environment exposes an interface to outside, allowing the deployment and execution of tasks/processes. It manages its internal resources, including hardware devices and software APIs. Environments can compose into larger Environment, while each composite Environment manages the resources and routing on its level. By introducing such composite Environment model, we are able to model the complex environment with heterogeneous devices, and provide a solution for the resources management and task routing across devices.

To implement the environment model, we explore different possible architectures and compare the their strength and weakness dealing with different devices and tasks. These architectures include centralized architecture, Peer-to-Peer architecture and hybrid architecture. Their differences are discussed in detail in Chapter 3.

We adopt the hybrid architecture in our proof-of-concept framework, because hybrid architecture fits better with this type of environment. Devices deployed in vicinity are more likely to interact with each other on tasks, and they occasionally need to communicate with devices far way.

Based on the hybrid architecture, we design a protocol for service discovery and task routing across environments, and a language to describe the environments and the requirements of tasks. The hierarchical composite environments allow devices to join or leave the environments without manual efforts to reconfigure or redeploy the applications, and provides a unified, elastic and scalable service on top of the heterogeneous and unreliable devices.

The basic service discovery protocol simply returns the first deployment

solution. We study further optimization considering the resources allocation efficiency, routing performance and robustness to mobile network or device failure.

## 1.4 Contributions

Our contributions include:

- The model of Environment-as-a-Service, which provides a theoretical foundation to manage the resources (especially on mobile devices) and route the tasks/processes across connected devices.
- The study of architectures to support the Environment-as-a-Service model. We explored the three possible architectures and adopted the hybrid architecture for proof-of-concept framework.
- An implementation of proof-of-concept framework on Android platform. The framework includes a language to describe the available resources on devices and the requirement of task/process, an algorithm to manage the resources and routing, and the implementation on Android platform.
- The approach to optimize the resources discovery and task routing.

## 1.5 Structure of the Thesis

This thesis is organized as follows: In Chapter 2, we describe our model of Mobile Environment, the composition of Environments, and a framework to route tasks across Environments to the destination; In Chapter 3, we discuss different architectures to integrate the devices, to support the composition of environments; Chapter 4 presents the mechanism to enact the process/task in destination Environment, and a mobile process engine

prototype; Chapter 5 discusses optimization of task allocation and process decomposition; Chapter 6 presents the related work; Chapter 7 concludes the thesis, and discusses its limits and future work.

# Chapter 2

# Modeling Environment-as-a-Service

As we mentioned, mobile devices provide various capabilities to support applications in different industries. To better utilize the capabilities on mobile devices and reuse the domain knowledges encompassed in software modules, one way is to decouple the implementation of software modules and the high level business logic design on top of the available software modules. Existing software architectures such as Service-Oriented Architecture (SOA) [28], focus more on stationary devices, assuming the connectivity and availability of resources are stable. We need a more flexible model to describe the resources on mobile devices and unreliable wireless connectivities.

We propose the model of Environment-as-a-Service with the following principles:

1. Hierarchical. In the complex industrial environments, organizations of different level manage different scope of resources. The model should allow different granularity of resources management.

2. Autonomous. In the model, a Environment manages the internal resources and provides essential services without depending on external resources.

3. Composable. To support the complex process deployment, Environ-

ments should be able to compose with each other into large Environment and together provide different services.

## 2.1 Modeling an Environment

**Definition 1 (Device)** *A device (denoted by $D$) is an electronic equipment that can perform certain task(s) and is able to communicate with other equipment. $\mathcal{D}$ denotes the set of all known devices.*

Devices include a spectrum of wired and wireless-connected electronic equipment, from sensors, actuators, to smartphones, personal computers, and other complex electronic equipments.

**Definition 2 (Capability)** *A capability of a device is a function that it provides. We assume that all devices share the same taxonomy of capability description. Each device has a set of capabilities, and we model the capabilities of a device as an attribute, denoted by $Cap(D)$. The set of all capabilities is denoted as $\mathcal{CAP}$.*

For example, for a printer $D_1$: $Cap(D_i) = 'print'$, and for a barcode reader $D_2$ with a LCD: $Cap(D_2) = 'readBarcode', 'display'$.

Each device has at least one connection method, which allows it to connect to other devices those have the same connection method.

**Definition 3 (Connectivity)** *We denote the set of all connection methods between devices as $\mathcal{CON}$. Each device is able to communicate with other devices over a set of connection methods: $Con(D_i) \subseteq \mathcal{CON}$. We assume that the connection between devices is bi-directional: if a device $D_i$ can directly start the communication and send information to device $D_j$, $D_j$ can also directly start the communication and send information to $D_i$. We indicate this relation as: $D_i \leftrightarrow D_j$. Obviously, they need to have one common connection method: $D_i \leftrightarrow D_j \Rightarrow Con(D_i) \bigcap Con(D_j) \neq \emptyset$.*

A group of devices can form an Environment, if and only if any two devices can reach each other without passing any device that does not belong to the same Environment:

A group of devices can form an Environment $\Leftrightarrow$

$$\forall D_i, D_j \in Env, \exists D_{x_1}, D_{x_2}, ...D_{x_m} \in Env, s.t. D_i \leftrightarrow D_{x_1} \wedge D_{x_1} \leftrightarrow D_{x_2} \wedge \ldots \wedge D_{x_{(m-1)}} \leftrightarrow D_{x_m} \wedge D_{x_m} \leftrightarrow D_j \quad (2.1)$$

**Definition 4 (Environment)** *An Environment comprises a group of elements, which can be devices or other Environments, and the connections among the elements. A tuple represents an Environment $Env = (V, E, C)$: a vertex $V_i$ ($\in V$) represents a device or a nested Environment which is an element of $Env$, and an edge $(V_i, V_j) \in E$ represents the two devices or environments that are connected ($V_i \leftrightarrow V_j$); for a $V_c \in C \subseteq V$, $V_c$ is the vertex (device or Environment) that serves as a controller in $Env$. We denote the set of all Environments as $\mathcal{E}$.*

**Definition 5 (Device joins an Environment)** *$D \in Env$ represents that a device $D$ joins an Environment $Env(V, E, C)$. As defined in Equation (2.2), when a devices $D$ joins an Environment $Env$, it means that either (directly) it is vertex of the Environment $Env$, or (recursively) it joins an Environment $Env'$ which is a vertex of the Environment $Env$:*

$$D \in Env(V, E, C) \Leftrightarrow D \in V \text{ or } \exists Env'(V', E', C'), s.t., D \in V' \text{ and } Env' \in V. \quad (2.2)$$

**Definition 6 (Atomic/Composite Environment)** *If the vertices of an Environment are all devices, the Environment is an "Atomic Environment". More formally:*

$$Env(V, E, C) \text{ is atomic} \Leftrightarrow \forall V_i \in V, V_i \in \mathcal{D}$$

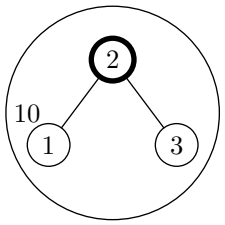



Figure 2.1: An Environment composed by Three Atomic Environments

*If one or more vertices of an Environment are Environments, the Environment is said to be a "Composite Environment". More formally:*

$$Env(V, E, C) \text{ is composite} \Leftrightarrow \exists V_i \in V, V_i \in \mathcal{E}$$

Since Atomic Environments are the finest elements that we consider, we restrict Atomic Environments to have simple topology: there is exact one controller, and it connects other devices in the Environment. A Composite Environment can have one or more controllers. In Fig. 2.1, $E1$, $E2$ and $E3$ are Atomic Environments, while $E10$ is Composite Environment.

### 2.1.1 Capability

The capability of an Environment is the union of capabilities of all its devices.

$$Cap(Env(V, E)) = \bigcup_{V_i \in V} Cap(V_i)$$

The controller serves as a registry of all the capabilities of the devices in the Environment. Given a request of a certain capability, the controller is able to decide whether there are some devices in the Environment have this capability and how to connect those devices.

### 2.1.2 Computational Capacity

Different from the diverse capabilities that devices provide (e.g., "print", "readBarcode"), we also model the generic computational power of devices

as computational capacities.

**Definition 7 (Computational Capacity)** *A Capacity of an Environment is a parameter that measures an aspect of generic computational resources that it can provide. A set of capacities are specified at the same time, with the only condition that their values can be aggregated:*

$$\{CC\} \text{ can be a set of capacities} \Leftrightarrow \forall CC_i \in \{CC\}\ \exists F$$
$$\forall Env(V, E, C)\ CC_i(Env) = F(\{CC_j(V_k)\}_{CC_j \in \{CC\}, V_k \in V}) \quad (2.3)$$

As example, we define three categories of Computational Capacities of a controller: CPU, memory, availability. They can be substituted by other parameters that are of interest, such as communication delay within the Environment, remaining battery duration, cost of resource usage, etc. The only restriction is that the set of capacity parameters should be able to be aggregated (Equation (2.3)).

**Definition 8 (Parameters)** *The device capabilities and computational capacities are called "parameters" of Environments.*

A controller of an Environment has both internal and external responsibilities: internally it manages the resources (children Environments) and calculates the aggregated parameters of the Environment; externally it can receive a task, replies whether the task is executable in the Environment, and returns the execution results.

### 2.1.3 Connection Topology

We assume that in an Atomic Environment the contact point to outside is the controller, because it controls all other devices in the Environment. A controller has several connection methods for the outside. For example, a smartphone can have Wi-Fi, 3G and Bluetooth connection, while a laptop may only have Wi-Fi and Bluetooth.

### 2.1.4 Atomic Environment as a Service

An Atomic Environment provides a service to the higher level application. More specifically the Atomic Environment provides on top of the single-hop wireless service discovery such as Bluetooth, OSGi [26], and Apple Bonjour [6]. An Atomic Environment in our model provides the abstraction of the capacities and capabilities of the hardware devices. In the latter section, we are going to present an XML-based language to describe the Environments as unified interfaces for task execution.

## 2.2 Composition of Environments

Multiple Environments can compose a higher level Environment. Depending on the trigger and configuration, the composition can be:

- Passive. The user or an application can trigger a composition and specify the children Environments to be composed. The only condition is that any two selected children Environments can reach each other without passing other Environment, as defined in Equation 2.1.

- Active. The framework can automatically decide when to compose the Environments, and it also works out the configuration of how to compose.

**Definition 9 (Environment Composition)** *Environment Composition forms a new Composite Environment from a set of Environments ($\{V_i\}$) and the connections cross them. More formally, given:*

$$\{V_i\} \subseteq \mathcal{E}, \text{ and } E_{cross} = \{(D_j, D_k)\}, D_j \in V_a, D_k \in V_b, V_a \neq V_b$$

*the Environment Composition results in:*

$$Composition(\{V_i\}, E_{cross}) = Env(V', E', C') = \\ \forall D_i, D_j \in \{V_i\}, \exists D'_i, D'_j \in V', \textit{the connectivity of } D_i, D_j \\ \textit{is the same as } D'_i, D'_j \quad (2.4)$$

Fig. 2.1 shows an example of an Environment Composition: given three Atomic Environments ("1", "2", and "3"), and the new connections $(1, 2)$ and $(2, 3)$, the Composition result is an Environment that comprises all the Atomic Environments and the connections among them. Fig. 2.3 illustrates the composition result on top level.

**Definition 10 (child/parent/sibling Environment)** *In a Composite Environment $Env(V, E)$, if $V_i \in V$ and $V_i \in \mathcal{E}$, then $V_i$ is a child Environment of $Env$ and $Env$ is the parent Environment of $V_i$. If $V_i, V_j \in V$ and $V_i, V_j \in \mathcal{E}$, $V_i, V_j$ are each other's siblings.*

**Definition 11 (descendent/ancestor Environment)** *Descendent Environments of an Environment include: its children Environments; and the children Environments of any of its descendent Environments. Ancestor Environments of an Environment include: its parent Environment; and the parent Environment of any of its ancestor Environments.*

In other words, Environments that are not Atomic Environments are Composite Environment. The same condition as Equation (2.1) applies for Composite Environments.

### 2.2.1 Aggregation of Computational Capacities and Capabilities

As shown in Equation (2.3), the set of capacities can be aggregated. We defined the following set of capacities as an example: availability of the Environment, highest/lowest CPU, highest/lowest memory size.

These parameters can be aggregated effectively from the children Environments, without involving the parameters from lower level Environments. Each controller manages the aggregated parameters of its Composite Environment and the parameters of its children. When changes happen within an Environment, the changes propagate up to the top level Composite Environment. Threshold can be applied to reduce the change propagation of aggregated parameters.

### 2.2.2 Three Types of Environment Composition

As in the Definition 9, any output Environment comprises the same set of Atomic Environments and connections as input is a valid Composition. For the same input, there can be multiple valid Composition results. We categorize all the possible compositions into three types:

- Hierarchical Composition. The input Environments keep their structures, and form a higher level Environment. Each input Environment manages its children, and exposes its controller to others. Hierarchical Composition generates a loosely coupled Environment, having these benefits: a) it is easier to decompose into original input Environments; b) the communication within input Environments is efficient, because input Environment is tightly connected and remains unchanged in size; c) each input Environment retains control of its descendants. However, the disadvantage is that the hierarchy is one more level deeper and the communication across input Environments is less efficient; and the composition algorithm is complex.

- Merge - under one controller. The input Environments break their borders and form an Environment. The controller of one input Environments becomes the new controller. The benefits of this type are: a) the structure of output Environment is simple; b) the composition

algorithm is simpler - we just need to rerun the composition algorithm. The disadvantage is that the output Environment is larger, has higher communication delay and is more vulnerable to controller failure.

- Merge - retaining multiple controllers. The input Environments break their borders and form an Environment. All controllers of input Environments retain as controllers of the new Environment and share their control. This type of Composition has the following benefits: a) more robust to failure; b) controllers partially retain control; c) the communication is efficient in vicinity of a controller. However, it is more complex to set up and maintain the routing information of the network.

### 2.2.3 Routing in Composite Environment

In this subsection, we first show how a task is routed across Environments, then describe how the routing tables are created and managed during Environment composition and when the device connections are changed. Since our Environment model is hierarchical, we describe the composition on one level, and based on the assumption that: within any Environments ($E_i$), the information is able to be delivered between any two children Environments of $E_i$.

For simplicity, we introduce the routing in single controller Environment. Routing in multi-controller Environment is similar except that message to an unknown destination is broadcast to all controllers of the Environment.

Fig. 2.2 shows our example of Environment composition.

Our routing is similar with traditional Internet routing protocols [70]. The differences include:

- Our model of Environments is hierarchical. An Environment only sees

Table 2.1: Examples of Routing Tables

| Table of | Dest. | Int. | //Comment |
|---|---|---|---|
| RT7 | 6 | a | T1, neighbor |
| | 20 | 6 | T2, to controller |
| RT6 | 4 | a | T1, neighbor |
| | 7 | b | T1, neighbor |
| | 10 | c | T1, neighbor |
| | 5 | 4 | T3, direction to child |
| | 10 | 4 | T1, neighbor of 20 |
| | 10 | 6 | T1, neighbor of 20 |
| | 30 | 4 | T1, neighbor of 20 |
| | 100 | 10 | T2, to controller |
| | -100 | 30 | T2, from controller |
| RT3 | 2 | a | T1, neighbor |
| | 20 | b | T1, neighbor |
| | 10 | 2 | T2, to controller |
| RT2 | 1 | a | T1, neighbor |
| | 3 | b | T1, neighbor |
| | 20 | 1 | T1, neighbor of 10 |
| | 20 | 3 | T1, neighbor of 10 |
| | 30 | 20 | T3, direction to child |
| RT4 | 5 | a | T1, neighbor |
| | 6 | b | T1, neighbor |
| | 30 | c | T1, neighbor |
| | 30 | d | T1, neighbor |
| | 20 | 6 | T2, to controller |
| | -20 | 5 | T2, from controller |
| RT9 | 8 | a | T1, neighbor |
| | 20 | b | T1, neighbor |
| | 20 | 9 | T1, neighbor of 30 |
| | 100 | 20 | T2, to controller |

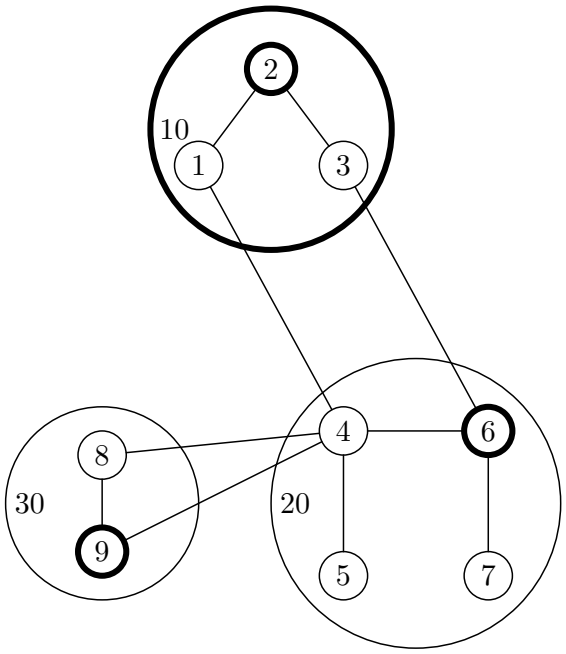



Figure 2.2: A more complex composition of Environments

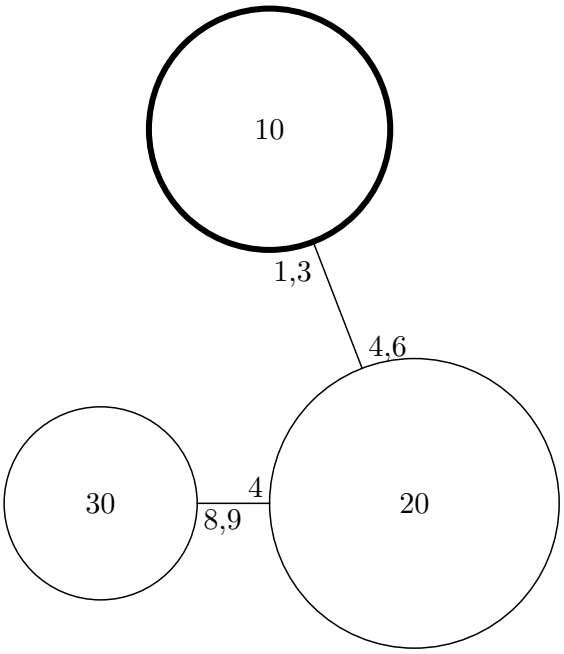



Figure 2.3: The top level Composite Environment

its parent, children and siblings. So the routing is also not from any point to any point.

- Routing in Environments accompanies the Environment discovery. Besides pure message routing, our routing is concerned about the routing of tasks. Depending on task and Environment descriptions, decisions need to be made before routing to parent/child Environment.

- The Environments are not always well-connected. Because of the dynamic nature of mobile devices, Environments and the connections are unreliable. The routing information needs to reflect the changes of connections in an efficient way.

#### Discovering a satisfying Environment

Our discovery algorithm is similar to the Domain Name System (DNS) [57]. As shown in Fig. 2.5, each Environment decides if it satisfies the require-

ment of a task. If yes, it passes the task to a satisfying child Environment; If not, it passes to the parent Environment.

Our example is based on the completed Composite Environment as shown in Fig. 2.2. To better illustrate the process of discovering the Environment, we simplify the graph into a tree (Fig. 2.4). Each node in the tree is an Environment and its children nodes are the children Environments.

In the example, the Atomic Environment 7 generates a task, whose requirements are only satisfied on Environment 9. Here are the steps to discover the Environment (for convenience, when we say that an Environment performs a certain action, actually its controller performs the action):

a. Atomic Environment 7 checks its capacity and capability and decides that it does not satisfy the requirements, then it passes the task to its parent Environment 20;

b. Environment 20 checks its aggregated parameters and decides that it does not satisfy the requirements, then it passes the task to its parent Environment 100;

c. Environment 100 satisfies the requirement, then it finds the satisfying child Environment 30 and passes the task;

d. Environment 30 finds the satisfying child Environment 9 and passes the task;

e. Environment 9 is the Atomic Environment that satisfies the requirement, so it executes the task.

#### Routing across Environments

Now the problem is how the controllers route messages across Environments. We introduce Routing Tables (RT) in controllers. The controller of

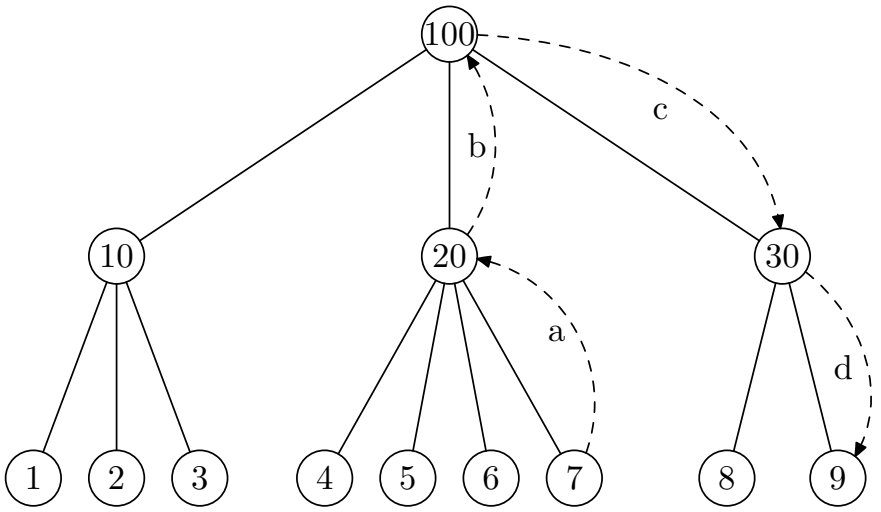



Figure 2.4: Tree of Environments

each Atomic Environment has a Routing Table. The Routing Table also includes the information about the Composite Environment controlled by this controller, if any.

**Definition 12** *A link is the connection between two Environments. There are two types of links: Atomic Link is the connection between two Atomic Environments; Composite Link is the connection between two Environments and at least one of them is Composite Environment.*

A record in the Routing Table has two columns:

- Destination. The destination of a link. For an Environment, the visible destinations include: neighbors within the same Composite Environment (e.g., 4 knows 5 and 6); neighbors across Environment border (e.g., 4 knows 10 and 30); its controller (e.g., 4 knows 20 is via 6); paths to its children Environments (e.g., 20 (6) knows paths to 4, 5, 7).

- Interface. The next intermediate target that leads to the destination. Depending on the type of destination, the interfaces have different meanings.

```
function DISCOVER(Env, T)
    if Env.satisfies(T) then
        return FindSatisfyingAtomicEnv(Env, T)
    else if Env.hasParent() then
        return Discover(Env.parent, T)
    else
        return null
    end if
end function
function FINDSATISFYINGATOMICENV(Env, T)
    if Env.isAtomic() then
        if Env.satisfies(T) then
            return Env
        else
            return null
        end if
    end if
    for all childEnv ∈ Env do
        if childEnv.satisfies(T) then
            re = FindSatisfyingAtomicEnv(childEnv, T)
            if re ≠ null then
                return re
            end if
        end if
    end for
    return null
end function
```

Figure 2.5: Algorithm: Discover a satisfying Environment

Table 2.2: Type of Routing Records

| Type | Description | Type of Link | Meaning | Example |
|---|---|---|---|---|
| T1 | to neighbor | Atomic | Connection interface to the neighbor | 4 to 5 via int. a |
| | | Composite | The border Environment that connects to the neighbor | 20 to 30 via 4 |
| T2 | to/from controller | | The next hop leading to the controller or the reverse direction | 4 towards 20 via 6, and outwards via 5 |
| T3 | direction to child | | The next hop from a controller to its child | 20 (6) to 5 via 4, 100 (2) to 30 via 20 |

As shown in Table 2.2, there are three types of records in the Routing Table of a controller:

- T1, neighbor Environment. The connections between neighbor Environments define the whole network connectivity. For an Atomic Environment E, a link to a neighbor in the same Environment is an Atomic Link, and the corresponding T1 record points to the network interface that connects to that neighbor; and a link to a neighbor outside the Environment is also an Atomic Link, but E only knows the Composite Environment that it connects (e.g., 4 only knows 10 and 30, but it does not know 8, 9, or 1). For a Composite Environment, the interface to a neighbor is its own child Environment on the border.

- T2, next hop towards controller, or the reverse. Each Environment knows its next hop that leads to the controller, then a message can be routed from any child Environment towards its controller. Each Environment also knows the next hop that goes away from the controller. When an Environment receives a message originated from the controller and the destination is not itself, it passes the message to

```
function GetNextHop(CurrentEnv, dest, source)
    if dest == CurrentEnv then
        next = SELF
    else
        interface = RT.lookup(dest)
        if interface ≠ null then
            next = interface
        else if source == CurrentEnv.controller then
            next = RT.lookup( - CurrentEnv.controller)
        else
            next = ERROR
        end if
    end if
end function
```

Figure 2.6: Algorithm: Find the next hop to route a message

the interface that goes away from the controller.

- T3, direction to each non-neighbor child, if the controller also controls a Composite Environment. When a controller sends a message to a non-neighbor child, it passes the message to a neighbor toward that direction. When an Environment on the path receives such a message addressing for another Environment, it passes the message to the outward direction. Then a controller is able to route a message to any child.

Fig. 2.6 shows the algorithm of looking up the Routing Table in an Environment. Because the connection information is encoded in the Routing Table, the look up is straight forward: if the destination is current Environment, it is done; otherwise if the destination is in the Routing Table, then forward it; otherwise if it is from the controller, forward it to the "away from controller" direction; otherwise the destination is unknown.

**Setting up the Routing Tables**

When several Environments are composing a new Composite Environment, connections are established in different levels of Environments. To represent the new connections, records are inserted into the Routing Tables of the newly connected Environments.

The procedure can be divided into three phases:

- Connecting neighbor Atomic Environments. First, both Atomic Environments (e.g., 4 and 1) along the new connection insert a new T1 record containing the destination and interface information into their Routing Tables. However, the connection crosses the borders of existing Composite Environments, and they (4 and 1) cannot see each other but only the top level Composite Environment that the other belongs. For example 4 sees 10 and 1 sees 20. So 4 inserts a record "10,c", while 1 inserts "20,b".

- Propagating neighbor connections to top level Environments. After new connections are established in an Atomic Environment, this connection propagates to the parent Environment. The new record uses the child Environment who has propagated the connection as the interface, because the controller already knows the path to this child Environment. If the parent Environment has a parent Environment, the connection propagates further up to the top level Environment before composition. For example, the connection from 4 to 10 is propagated to its controller 6. A new record "10,4" is inserted into RT6. Since 6 is already one of the top level Environments before composition, the propagation stops here.

- Connecting controller in new top level Environment. After all connections between neighbors are established, it is time to establish the

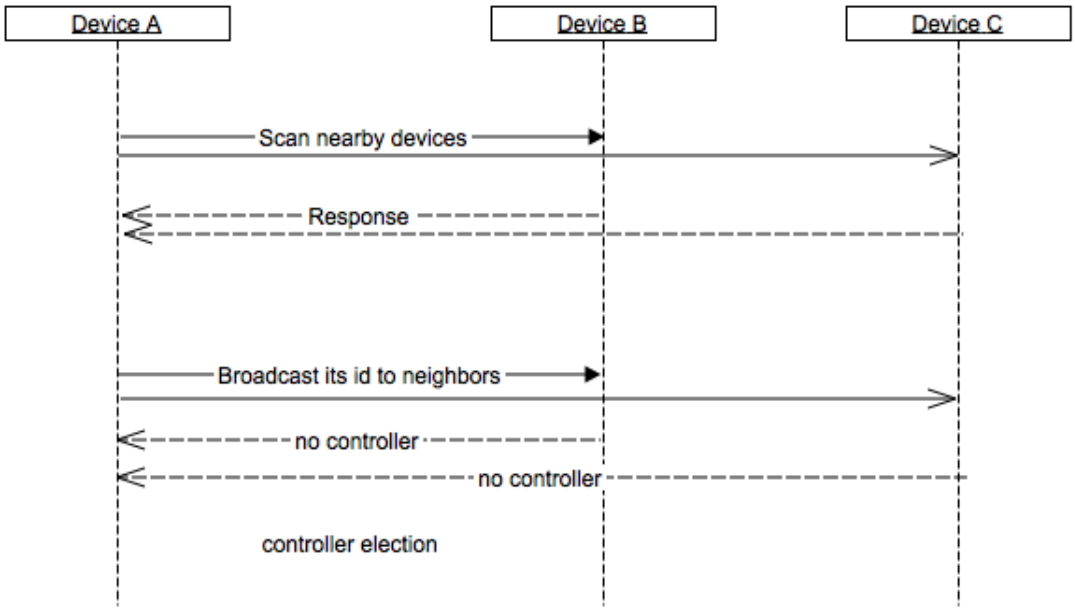



Figure 2.7: Initialization an Environment with no Controller

connections between the new top level controller and its children Environments. The protocol works in three steps: 1. (Flooding) first the new controller sends its address along all its neighbors. When one children Environment receives this message for the first time, it spreads this message further to its neighbors increasing the distance by 1, and inserts a T2 record to its Routing Table. If the children Environment receives another controller message again, if the distance is shorter than current routing record, it updates the record and spreads the message; otherwise, it discards the message; 2. (Setting up paths) At the end of the protocol, each children knows the next hop on the shortest path to the controller. They send a message to the controller to report. 3. The Environments those receive such message add a record "away from" controller. When the messages arrive in the controller, the controller adds routing records to its children, specifying the next hop.

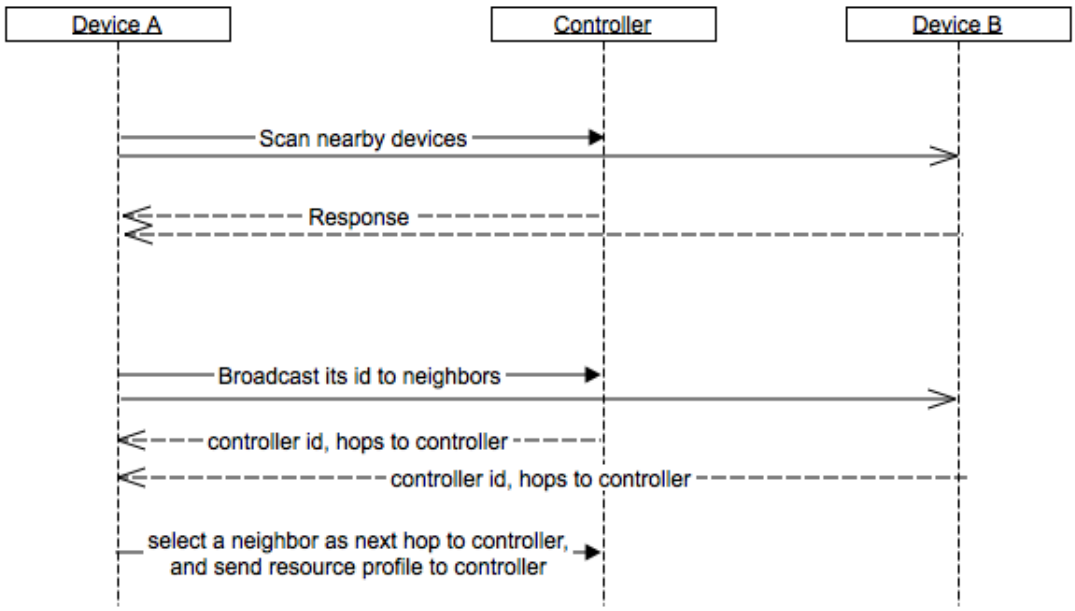



Figure 2.8: Join an Environment with Controller

Figure 2.8 shows the communication when a new device $A$ finds an Environment with controller. Multiple neighbors can return a metric about the distance to controller. Device $A$ selects one neighbor to connect. Figure 2.7 shows the communication sequence when a new device $A$ discovers other devices in neighborhood but no controller is available. It is the procedure of setting up an Environment. Controller election algorithm is executed to elect a controller and set up the Environment. More discussion about election algorithm will be given later in this chapter.

#### Mobility Support

As we mentioned before, the challenges brought by mobile devices include:

- The wireless connections are unreliable. Many factors, such as interference, low battery etc. can compromise the wireless connection. When a connection is lost, the routing information depending on this

connection becomes invalid.

- The devices themselves are unreliable. When a device dies, the connections linked to it becomes invalid. And the routing information that it holds is lost.

- The devices may physically move to different locations, changing the connection topology. Even when the connectivity between devices remain unchanged, the movement of devices can cause the previous Environment Composition depreciated.

Our composition considers the above challenges by corresponding Routing Table adjustment.

When the connection between two Atomic Environments is lost or a device is dead: The routing records are removed in the remaining Atomic Environments; If the connection crosses the Environment boundary and it is the only connection from the Atomic Environment to the top level neighbor, the controller of its parent Environment removes the record to that neighbor; If it is the only connection from the controller, the disconnected Atomic Environment asks all its neighbors for the distance to controller and pick the shortest as the new record, and propagates the same update along the previous path from controller; otherwise, the Routing Table of controller remains unchanged. If a controller becomes unreachable, a new controller is elected and the Routing Table is recreated by repeating the Routing Table setup process.

When an Environment moves across composition boundary without interrupting any connection between underlying devices, there may be the following cases: If the Environment is not the controller, we only need to notify the old neighbors of the moved Environment and its old/new controllers; If the Environment is the controller, we also need to elect a new controller and setup the new Routing Tables in the old Composite Envi-

ronment. When the moved Environment was at the end of the path from controller, the procedure becomes simpler, because it is not affecting the T2 and T3 routing records of other Environments in the old Composite Environment.

#### Controller Election in the Composite Environment

The controller is a role in the Composite Environment. A child Environment serving as the controller of the Composite Environment has the following responsibilities:

- It calculates and maintains the composite capacity and capability for the Composite Environment as a whole. By aggregating the capacity and capability parameters from children Environments, the controller computes the composite parameters for the composite Environment. When a child Environment or the parent Environment passes a task with requirements on the capacity and capability of the Environment, the controller needs to check if the Composite Environment that it manages satisfies the requirements.

- It manages the index of capacities and capabilities of children Environments of the Composite Environment. If the Composite Environment satisfies the requirement, the controller needs to find out which child Environment satisfies the requirement. Then it passes the task to the satisfying child Environment.

- It maintains the routing information to other children Environments in the Composite Environment. One benefit of having a controller in the network is that it simplifies the routing in the Composite Environment. Since the task routing is always from child to parent Environment or from parent to child, the children Environments only need to know how to send a message to the controller, and how to pass a

message from controller. The controller only needs to know the first hop of the path that leads to each child Environment.

- It maintains the routing information to the neighbor Composite Environments. When the Composite Environment becomes a child of a higher level composite Environment, the controller needs to maintain the information of routing information to the neighbor Composite Environment.

During the composition, a child Environment is elected as the controller of the Composite Environment. There are algorithms to elect the controller (coordinator) of a peer-to-peer network [35, 42, 97, 5]. Depending on the characteristic of the network, several factors can be considered:

- The availability of the controller. The availability of the controller is decided by its underlying Atomic Environment. If the availability of devices in the network is the major concern, we can consider the availability as the most important factor.

- (Degree Centrality) The connection degree of the controller. The connection degree of the controller decides probability that the controller is connected to the rest of the Environment. In the network with heterogeneous connections, we can attach weights to the connection to calculate the connection degrees. If the reliability of connections in the network is low, the connection degree can be the major concern in controller election.

- (Closeness Centrality) The average distance from the controller to other children Environments. The average distance to children Environment affects the communication delay and cost. Instead of using hops to measure the distance, more sophisticated metrics, such as the delay or cost of the connection can be used to measure the distance.

When the communication delay or cost is the major concern, the average distance to children Environments can be the factor that we need to consider.

These factors can be combined into a single metric to address multiple concerns in the controller election.

If we choose a metric that can be computed locally, e.g., the availability or connection degree, we can use a flooding-based protocol: Each Environment computes its metric value and stores it in a buffer; Each Environment sends its metric to all the neighbors, containing its metric value and identity (ID); When an Environment receives a metric, if it is larger than current buffered metric, the Environment refresh the buffer with the greater value and sends the greater metric to all neighbors; otherwise, if the received value is smaller, the Environment discards the received metric; if the received value equals the buffered, the one with higher ID wins. At the end, each Environment buffers the greatest value of metric and the ID of that Environment. The election result is sent in broadcast in the Composition Environment, so each child knows the controller.

## 2.3 Environment Description Language

We design an XML-based Environment Description Language (EDL), which can describe the resources that an Environment can provide and the resources that a task requires. EDL supports the composition of Environments: the description for the Composite Environment can be generated from the descriptions of children Environments. By comparing the Environment with requirement, the framework can decide whether the Environment is able to execute the task.

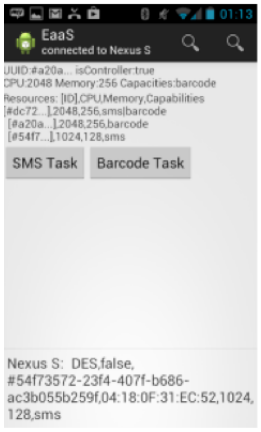



Figure 2.9: Console of a controller

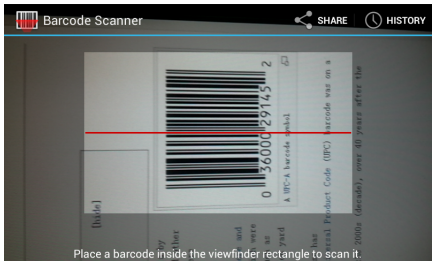



Figure 2.10: Barcode task received


Figure 2.11: Console of a child

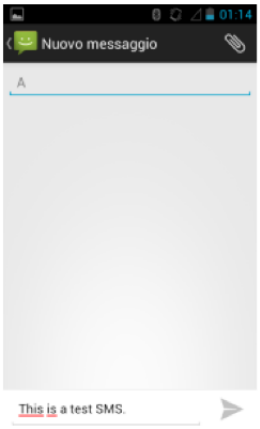



Figure 2.12: SMS task received

## 2.4 Implementation

We implemented a framework to verify the model and the algorithm design. The framework design is divided into two related parts: the overlay network that manages the resources in Environments and routes process/-tasks, which was discussed in this chapter; the process engine on atomic Environment that executes the received process/tasks, which will be described in Chapter 4,

The implementation is based on smartphones with Android operation system. The communication between smartphones is via Bluetooth connection. Figure 2.9 shows the console screen of a controller Environment

who is capable of reading barcodes, and Figure 2.11 shows a child Environment who is capable of sending message. In every Environment, a user can start a task of sending Short Message Service, or reading barcode, and the task will be routed to the Environment with required resources and executed (as shown in Figure 2.10 and 2.12).

## 2.5 Conclusion

In this chapter, we presented the model of Environment-as-a-Service. An Environment is modeled as a nested graph with the children Environments as vertices and connections between children Environments as edges. An Environment manages the resources and routing of tasks within the Environment. Multiple Environments can compose into larger Environment. The resources management and routing algorithms are also described. Part of this chapter was published on IEEE Mobile Cloud conference 2014 [64].

# Chapter 3

# Environment-as-a-Service Architectures

In last chapter, we defined the model of Environment: a group of connected devices is modeled into Environment. Environment is the building block of our framework, and it is represented as a node in the graph. In this chapter, we are going to discuss the architecture to organize the Environments in the overlay network.

In this thesis, "architecture" refers to the overlay network structure on top of the physical network of devices. Devices can connect with each other using different types of links, such as Wi-Fi, Bluetooth, or wired connections. We base the framework on transportation layer, taking advance of the existing connections for message exchange.

With the devices joining or leaving, the size of network can grow or shrink, and the Environments dynamically compose or decompose. Given the same collection of devices and connections among them, using the same Environment-as-a-Service model, there are different architectures to organize the devices. One essential difference is the scope of resources allocation and routing information: which nodes establish and track the resources allocation and routing information, and to which scope is the information shared.

In the Environment-as-a-Service model, resources include the hardware capabilities and API access privileges on nodes. In the model, resources are described as tags. Routing information is the information about the connection topology. Depending on the routing mechanism, different information is collected and used for routing. For example, in centralized and hybrid architecture, the controller can maintain a map of destination-path; while in P2P architecture, each node can maintain a table of destination-nexthop.

We introduce different architectures of Environments, explaining the resource allocation and message routing approach.

## 3.1 Three Architectures

According to the connection topology among devices, there can be three different architectures: centralized (Fig. 3.1), peer-to-peer (Fig. 3.2) and hybrid (Fig. 3.3). By analyzing their network characteristics, we make the following analysis:

1. Centralized. A single central controller manages all resources and routing across devices within one environment. Many current tasks/process management systems adopt centralized architecture: a process engine controls the workflow and the access to resources. In many industrial scenarios, we do have a center which manages all the resource within the environment, and authorizes the access to resources. For example, a hospital has the control of all its resources and authorizes access privileges to possible users (doctors, patients, government agents, etc.) From the implementation point of view, the centralized architecture is also easier to design. The essential logic of resources management is done on the controller and the other nodes only need to perform a small set of actions; The trust and authorize problem is

trivial here: each node just need to request the control to approve. However, this architecture is not scalable from the aspect of system performance. When the number of devices and users grows, it becomes inefficient to manage the resources, and the network becomes too large to perform routing efficiently; From the development effort aspect, it is easier to implement and the overhead of network setup is small.

2. Peer-to-Peer (P2P). There are no controllers in Environments. All children Environments share the information about resources and routing in the Composite Environment. P2P architecture can either be flat, or hierarchical. In flat P2P architecture, there is only one Composite Environment containing all the children Environments. And each child shares part of the knowledge of resources and routing information about other Environments. Hierarchical P2P architecture allows the composition of Environments, children Environments share the resource and routing information of the parent, but it only knows the aggregated information of its sibling Environments, but it does not know the descendant Environments inside the sibling Environments. P2P architecture is more robust to device or network failure due to the redundancy of connectivity and network management. However, the resource allocation can be slower and less effective, because no Environment has complete vision of resources and routing information. And the trust and authorize problem is more difficult to solve, because there is no central authority in the network in the initial state.

3. Hybrid. Hybrid architecture combines the centralized control with distributed Environment composition. Within each Composite Environment, a controller manages the resource and routing of its children Environments. Multiple Environments compose a higher level Com-

posite Environment and the new controller is elected to manage the resource and routing at the new composed Environment. Devices deployed in vicinity are more likely to interact with each other in tasks, and they occasionally need to communicate with several devices far way. By dividing the network into sub-networks (Environments), hybrid architecture can perform better on heterogeneous networks. Our first prototype adopts this architecture.

Because we are focusing on the deployment of task/process, the most important usage of the resources allocation and routing protocol is to route the task to the node with required capabilities. Considering the sequence of resource allocation and task routing, There are two possible designs:

1. Lookup a destination node first, and then route the task to that destination. The destination lookup and the task routing are separated. The framework first initiates the resource discovery algorithm, finds a destination node which satisfies the requirement. The result can be in the form of node ID, or a piece of routing information (e.g., a path) which leads to the destination. And then the task is routed to the destination node.

2. Lookup a destination and route the task at the same time. The task is also forwarded during the resource discovery process. When the framework finds the destination node with satisfying capabilities, the task is already at that node.

The first design avoids forwarding the task to unnecessary path, thus reducing the traffic of task routing. The tasks go through less nodes, if it is a concern of privacy or security. In the second design, resource discovery and task routing are done at the same time. The total latency is shorter, and it is easier to forward the task to multiple nodes if the framework allows the execution on multiple nodes.

Figure 3.1: Centralized Architecture. All devices form an Environment. A controller (thick circle) manages resources and routings of all devices.

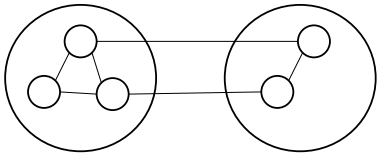


Figure 3.2: Hierarchical P2P Architecture. No controller. Children of an Environment manage resources and routing together. Children can connect to outside and expose aggregated information.

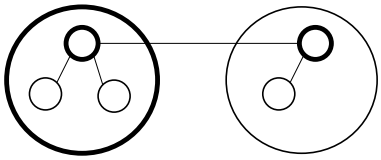


Figure 3.3: Hybrid Architecture. An elected controller manages resources and routing. Only controller can connect with outside. Our first prototype adopts this architecture.

## 3.2 Centralized Architecture

As shown in Fig 3.1, in centralized architecture, there is only one controller. The controller serves as the repository for resources on all devices, and manages the routing information for the whole network.

The setup of the centralized architecture is straight forward. A node is preconfigured as controller. Its ID can either be preconfigured in other nodes. It can also be broadcast in the network, given that the network is trustworthy.

### 3.2.1 Resources Management

During the setup phase of a node, it reports its available resources to the controller. The controller inserts it in the resource table.

In the task allocation phase, the controller looks up the resource table and allocates a node with available resources to execute the task.

### 3.2.2 Routing

Different routing mechanisms can be used in centralized architecture. The same routing mechanism described in last chapter also works under centralized architecture: the controller knows the next hop for a destination, and a node knows the next hop to/from controller.

Another solution is to maintain the mapping of destination-path in the controller, and each node maintains the next hop to the controller. When a non-controller node receives a message, if the destination of a message is the controller, it passes the message to the next hop to the controller; if the destination is not to controller, the message contains a path from controller to destination, and the node just passes the message as instructed by the path. When a controller receives a message which is not addressed to itself, it attaches the path to the message and passes it to the next hop.

During the network setup phase, each node detects its neighbors and this information is flooded in the network. The controller collects the neighbor information of all the nodes and generate the graph of the whole network.

Other routing protocols also work without treating the controller as a center. However, routing mechanisms which take advantage of the central controller are easier to implement and work better with the centralized resources allocation algorithm.

## 3.3 Peer-to-Peer Architecture

In Peer-to-Peer (P2P) architecture, there is no central controller. Resources and routing information are shared among all nodes in an Environment. P2P architecture can be either flat (only one level) or hierarchical (multiple levels). On a single level, the resources management and routing work in the same way for both flat and hierarchical P2P architectures. The difference is that for hierarchical P2P, a task is first allocated to the top

level node that satisfies the requirements, and then the same allocation algorithm repeats in that node, until the task is allocated to the satisfying atomic Environment.

### 3.3.1 Resources Management

There exist P2P resources management protocols [24, 5, 78, 74, 88, 101, 22]. One simple solution is to share all the resources information on each node. It is expensive to set up and maintain the complete resources index on each node. It generates more traffic to broadcast the resources information during the setup phase, and it occupies larger storage space to store the resource index. However, the resource discovery is fast, because it can be done on a single node.

More sophisticated protocols (for example Distributed Hash Table [9, 48]) can distribute resources allocation information multiple nodes. But the resources discovery need to communicate with other nodes and thus takes longer time.

### 3.3.2 Routing

Many routing protocols proposed for P2P networks can also be used [55]. Most P2P routing protocols keep small portion of the routing information per node, assuming the network is aggressively dynamic. However, the lookup latency is large because several nodes need to be contacted. Other P2P routing protocols keep more information per node to reduce the lookup contacts.

## 3.4 Hybrid Architecture

Hybrid architecture implementation was described in detail in previous chapter. In hybrid architecture, each node implements the same set of functions and potentially can work as a controller. In each Environment, one node is assigned or elected as controller and manages the resources and routing. Our first version implementation adopts hybrid architecture. Hybrid architecture is easier to implement comparing to P2P architecture, because it can adopt simpler resources management and routing algorithm; and it is more scalable comparing to the centralized architecture, because the controller is a role that each node can take over.

## 3.5 Comparison

In this section, we compare the above three different architectures focusing on the aspect of resource management and message routing. We first conclude the differences of architectures in Table 3.1.

Because centralized architecture and unstructured P2P architecture are more common and easier to understand, we focus on the comparison of hierarchical P2P and hybrid architecture.

Hybrid architecture has the same connectivity topology with hierarchical P2P architecture: so they have the same representation as nested graph. A node in the graph is an Environment and it can be composed by lower level Environments. The difference is whether there is a controller for each Environment. The hierarchical P2P architecture does not have a controller in an Environment, and the resource allocation and routing information is shared by all children nodes of that Environment. The hybrid architecture has a controller for each Environment, which manages the resources and routing information for that Environment. With the controllers, the hybrid

| | Centralized | Unstructured P2P | Hierarchical P2P | Hybrid |
|---|---|---|---|---|
| Number of Overlay Network Level | one | one | multiple | multiple |
| Controller | one controller in the whole network | no controller | no controller | one controller in every Environment |
| Resource Allocation Information | in controller | shared by nodes | shared by children nodes of an Environment | in controller of an Environment |
| Routing | by controller | P2P protocols | P2P protocols | by controller |

Table 3.1: Comparison of Architectures

architecture is easier to implement, because we do not need to manage the complex P2P protocols.

Since hybrid architecture and hierarchical P2P architecture have the same connectivity topology, we can use the same example network in Figure 2.2 to explain the different ways they handle the resource management and routing. Assuming that both architectures form the same overlay networks as shown in Figure 2.3, both architectures have the same resource tree as shown in Figure 2.4.

For example, hybrid architecture uses Environment 10 (the controller of whole composite Environment) to manage the resources (including resources on Environment 10, 20, 30). When the composite Environment (100) receives a task, the controller (10) is responsible to decide where (Environment 10, 20 or 30) to assign the task. Then the task is routed to the destination with the help of the routing information stored in the controller.

In hierarchical P2P architecture, the resources information of Environ-

ment 10, 20 and 30 are shared among themselves. When a task is received, the P2P network formed by children Environments (10, 20, 30) of the composite Environment (100) is responsible to locate the destination to pass the task, using P2P protocols (such as flooding, or Distributed Hash Table [9]). The task routing is done either at the same time with destination lookup, or as the second step after the destination is known.

One common characteristic for hybrid architecture and hierarchical P2P architecture is that the nodes are organized in multiple levels. Resource management and routing are done first on separate levels, and then one level in if the destination is reached or one level out if the destination cannot be found. So the resource lookup and routing follows the same procedure in the resource tree in Figure 2.4: The framework will first decide which child Environment to look into, and then the task is passed one level down. If no child Environment satisfies the task, the task will be passed to the parent until to the top level Environment.

## 3.6 Justification of Adopting Hybrid Architecture

We adopt hybrid architecture in our framework to fit our usage scenario. The framework is designed for use mainly in smaller areas with a few organizations involved, for example a healthcare system with government agents, several hospitals and patients' houses. Centralized architecture does not satisfy the requirement that more than one organization are managing their devices and services. Unstructured P2P architecture also has difficulty to control the access to resources that belong to different organizations. Only the hierarchical P2P architecture and hybrid architecture divide the network into subsets which fits the management convenience or different organizations. Because we are able to divide the network into subsets, the size of each sub-network is not large, and centers naturally

exist in each organization, the hybrid architecture fits most with its controller design, while hierarchical P2P architecture has high complexity in the P2P protocols implementation. So we adopt hybrid architecture in the framework.

# Chapter 4

# Deploying Processes and Tasks onto Mobile Devices

In previous chapters, we described the model of Environment-as-a-Service and the architectures. With the resource management function, the framework can allocate the required resources in the composite Environment. With the routing function, any two nodes in the framework can communicate with each other. Process and task deployment are based on these two functions: we define a task's requirements on resources and the resources availability in the Environment using the resource management function; and the communication for controlling the Environments and deploying the tasks are based on the routing mechanism.

In this chapter, we are going present how we deploy the processes and tasks, after they are assigned and routed to a destination Environment that satisfies the requirements.

Traditional business process have two forms of composition: orchestration and choreography [62]. In orchestration, a process engine controls the workflow of process execution across domains, whereas in choreography, each participant obeys the predefined rules and fulfills their roles in the process. Our framework works differently with both orchestration and choreography: a process or task is forwarded to the atomic Environment

without a central process engine, and executed in the destination Environment by a lightweight process engine. On the framework level, a process is first deployed in a way similar to choreography: the deployment logic is predefined but a central process engine is not necessary; After a process is deployed onto the atomic Environments, the process engine executes the process or task on the devices which compose the atomic Environment.

We designed a lightweight process engine in atomic Environment, which executes the received process or task. The process engine supports the automatic assignment and distributed execution of tasks on wireless connected devices within an atomic Environment. The contributions of this chapter include:

- A mobile process management approach for design and execution of business processes on mobile devices. The approach is based on four phases: service preparation, process design, activity assignment and activity execution;

- An extension of the Business Process Model and Notation (BPMN) 2.0 specification [60] to allow context-aware activity assignment, particularly: to model context constraints on activity assignment and execution, and model invocations of services offered by mobile devices inside business process models;

- A mobile process engine that executes processes on Android smartphones, a UI Framework for rendering user interfaces on mobile devices and a server that parses the extended BPMN processes and manages activity assignment.

## 4.1 Scenario

Our motivating scenario for executing processes/tasks on mobile devices is from a real-world project called MOPAL [21] in which nurses deliver healthcare services at patient's house with the assistance of mobile devices. The services are configured and monitored by a coordinator located in the hospital that schedules and assigns the healthcare services (i.e. a list of tasks for each patient) that nurses need to deliver. Task assignment considers criteria such as nurses' qualifications, their location and service history in order to obtain the most efficient task execution and meet the requirements of the healthcare service.

The nurses receive the tasks and instructions elicited by a coordinator, such as the list of patients and the activities to perform on mobile devices through specific developed applications. One of such mobile-assisted healthcare service is given by the blood pressure examination. In such scenario the nurses use the mobile device to perform a set of tasks and collect patients' blood pressure data through the following sequence of steps:

1. The application allows the nurse to search for a patient by the Social Security Number (SSN). It then loads patients data and shows them to the nurse.

2. The nurse, once measured the blood pressure enters the data filling specific forms.

3. If the pressure is too high, the application shows a warning message and suggests to the nurse to give to the patient an appropriate medicine. Otherwise this step is skipped.

4. The application composes a report recording the measured blood pressure and whether the medicine has been administered.

5. After the nurse confirms, the report summarizing the set of activities is sent to the hospital.

6. Finally the coordinator can examine the activities and can update the task-lists for next visits.

Developing applications that support such care delivery scenarios is not cost effective and is time consuming because of the need to support many different healthcare processes and provide a high degree of customization. In our real-world project the design and development suffered many difficulties due to the need for flexible task definition, assignment and execution and frequent updates of the mobile applications on all mobile devices to ensure that all of them were running the last versions.

The task-intensive nature of analyzed healthcare services suggested the need for a more flexible assistance process definition approach using technologies such as BPMN. Namely, process-modeling technologies such as BPMN have been demonstrated to provide an appropriate solution for fast changing contexts where the continuous evolution, monitoring and improvement of performed activities represent a crucial requirement.

A process model of the described blood pressure measurement service is shown in Figure 4.1. The coordinator's lane defines the process of managing the health examination service, and it runs on the central process engine used by coordinators; nurses' lane defines the process of carrying out the health examination, and it runs on mobile devices used by nurses.

Although it is very easy to model such healthcare delivery process, there are many challenges related to its potential execution on mobile devices that is still unsupported by current BPMN frameworks. To achieve the goal of business process assignment and distributed execution on mobile devices, we are facing several challenges:

- Current BPMN 2.0 specification is inadequate to support process ex-

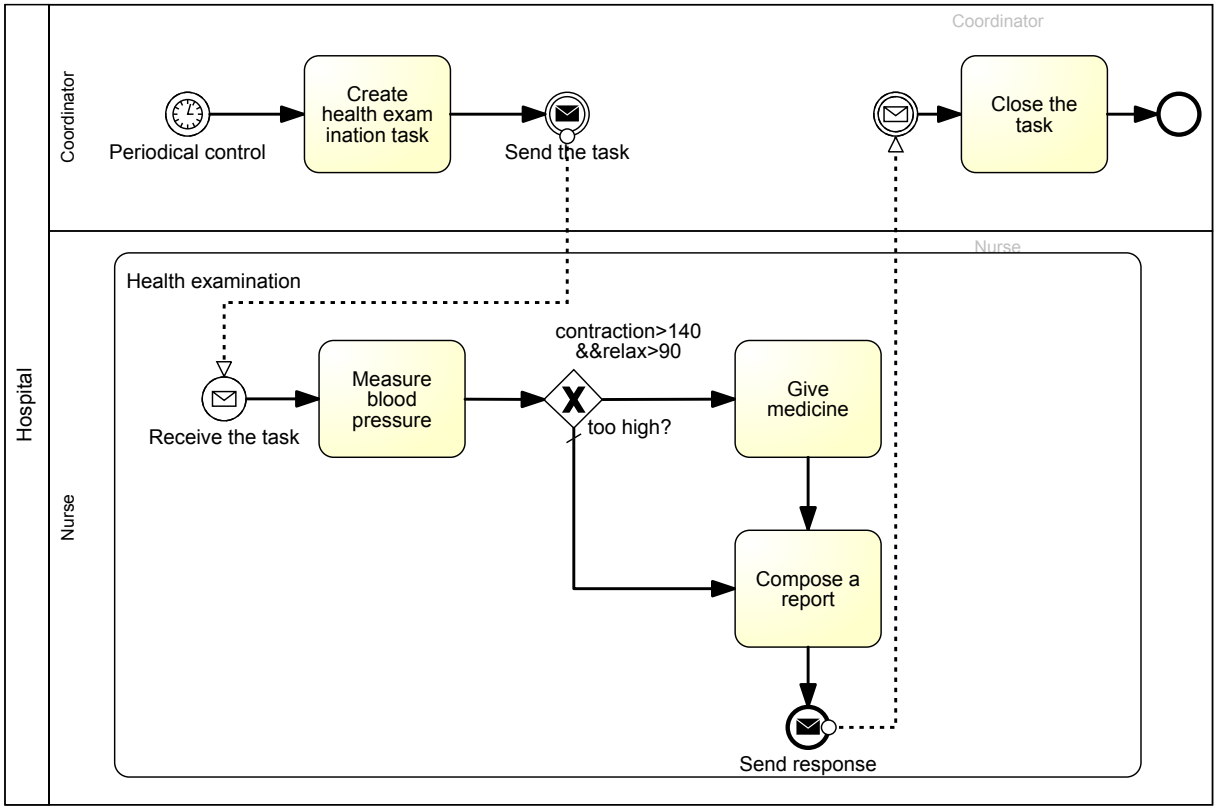



Figure 4.1: Business process model of blood pressure examination

ecution on mobile devices, as it does not exploit natively the functionalities and services offered by mobile devices. Such service can be any sensor-based collected information such as locations, network status or phone signal or it can be specific service such as Short Message Service (SMS) or calls that are present on mobile devices. The set of basic services that can be accessed by the process execution on a mobile device needs to be provided as a standard and lightweight library.

- Network connection on mobile devices is not always reliable (e.g. if the patients house is in a rural area). It is not acceptable to have loss of functionalities or latencies due to a disconnection of network. To tolerate the unreliable connection, the mobile process engine should be able to execute process and related tasks in an offline modality. Furthermore, it is needed a mechanism to prepare the required data before disconnection and to synchronize the data with the server once finished.

- Process and single activity assignments should be able to consider context related information specific to mobile devices (e.g., current location of the nurse or required qualification). The modeling framework should support modeling of such context-aware constraints for activity assignment and execution on mobile device. It is important to exploit the available context information in order to assign the tasks in a smart and automatic way.

- The process models executed on mobile should be compatible with current BPMN 2.0 specification while supporting extra defined semantics of constraints and services present on mobile devices.

To the best of our knowledge, none of existing state of the art tools and

engines is able to face the identified challenges and to provide a comprehensive solution to scenarios such as the one we faced in MOPAL project [21]. It is inefficient to run traditional process engines on mobile devices where computational resources are restricted. Mobile devices have slower CPUs, smaller Random-Access Memories (RAMs), and restrictions on energy consumption. Therefore a custom mobile engine and modeling framework is needed.

## 4.2 Mobile Process Management Approach

We tackle the previously described challenges by identifying a methodology that describes a sequence of steps performed by different participants each of them having different competencies and using different tools.

The four steps of our methodology identify the phases starting from requirements analysis to process execution. In particular it starts with the service preparation phase in which the developers prepare the services following the requirement analysis; then domain experts, with the help of developers, can compose and annotate processes on top of these services according to the business requirements; finally, the semi-automated assignment of tasks and process execution on mobile devices are performed by the framework.

### 4.2.1 Services Preparation

To understand how the business processes execution can be achieved on mobile devices, we need to analyze what are the implications of the shift of the execution environment; from desktop/server to mobile. We analyze these aspects according to the availability and location of the services invoked by the processes, classifying the services involved in the mobile business processes execution into two categories:

1. Services provided by local device: some mobile platforms (e.g. Android) allow the cross-application invocations and thus allowing the mobile engine to easily invoke local services and available applications on the device. In our project we consider the Android platform which allows applications to broadcast "intent" to start another application. Such "intents" can be triggered by the processes running on mobile devices to interact with applications. To facilitate such interaction we provide custom BPMN extensions that are executed on mobile process engine to support the execution of mobile specific tasks/events.

   On platforms that restrict the cross application communication, different solutions need to be designed such as using URL style invocation, or by implementing the required services within the process engine instead of using the ones available on the device.

2. External services: web services, or any other external resources, can be invoked by processes deployed on the mobile engine. For example, Web Services represent a popular implementation standard that are exposed through standard services interfaces defined by WSDL [17]. We provide the possibility to invoke such external resources from the mobile engine through the definition of specific modeling elements that are described later.

In service preparation phase, developers do not need to implement by themselves the required code to invoke the mobile specific services from the business processes. Our framework facilitates their invocation by providing a specific library of the mobile process engine that is used to access to them.

Despite the diversity of existing mobile platforms and devices, most popular mobile platforms provide a standard set of essential functions to access to email, calendar, browser, location sensor, motion sensor, etc. We provide developers with the library to access to some commonly used

services and in particular. Currently, we provide the following components inside our process engine:

- FormService turns the process description into a form that shows predefined instructions to perform a task, accepts user input, and guides the task performer through the given process.
- EmailService composes an email draft and initializes the mandatory fields (e.g., receiver, subject, body) to incoming parameters.
- ShortMessageService enables editing and sending a Short Message to other phone numbers.
- BarcodeService scans and recognizes a barcode or QR Code (Quick Response Code).

We plan to publish our platform under an open source license and thus allowing developers to implement additional accesses to services available on the mobile platforms and to share their services with others in need.

To invoke external services, we provide process components to invoke Web Services that are exposed through WSDL interfaces and RESTful services:

- SOAPService sends a request to a Web Service with SOAP protocol [95] and receives the response.
- RESTfulService similarly to SOAPService, it sends a request to a RESTful service [31] and parses the response.

As for the local services, also in this case we want to enable developers to develop custom code to invoke other types of existing remote services (e.g., legacy systems, or APIs in the cloud) from the mobile process engine.

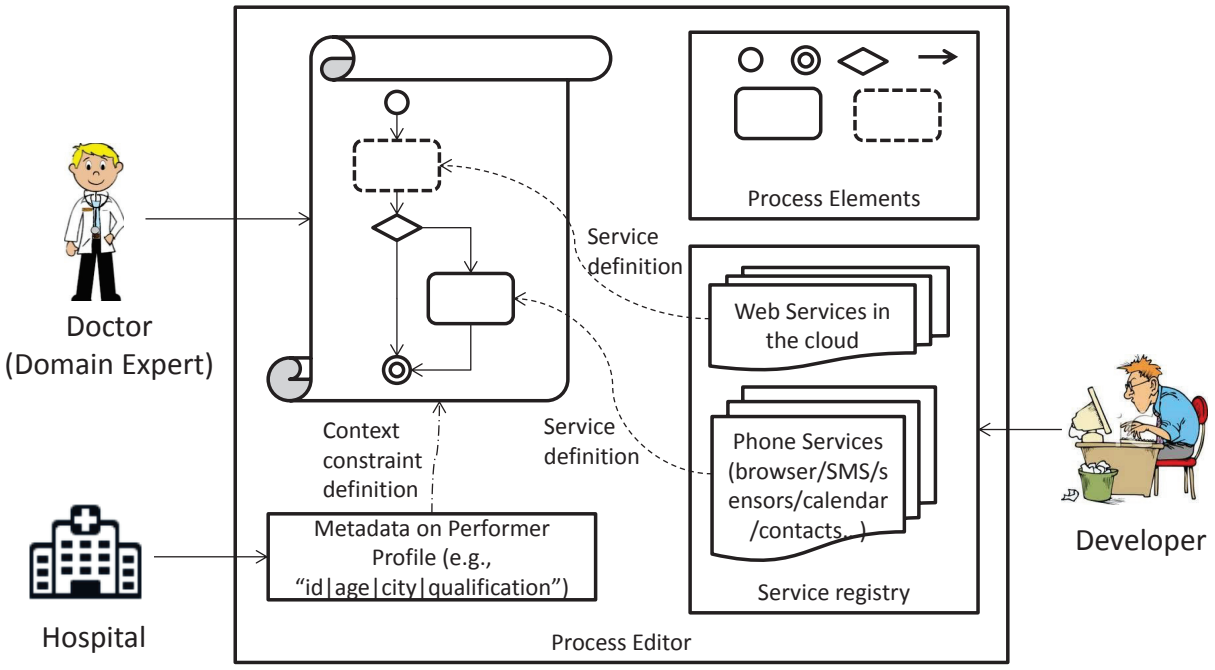



Figure 4.2: Process Design Phase

### 4.2.2 Process Design

Once the services have been prepared in the previous steps of our approach, the business process modeling can take place. We do not provide at this stage a specific process modeling framework allowing users to use any process editor (e.g. Signavio [85]) that is compliant with BPMN 2.0 specification.

Figure 4.2 shows the sequence of steps that need to be performed to design the process model and add additional custom extensions to execute it on the mobile engine. The design starts with the Domain Experts (e.g. Doctor) that define the process model without any concrete execution semantic specified. After the model is defined, Developers can export the process models and customize them to be executed on our mobile engine. When the context information and underlying services interfaces are defined, developers can deploy it to the central process engine and be ready to be deployed and executed on mobile devices.

The first step is to annotate the process models using our BPMN extension to support the automatic activity (simple tasks or sub-processes) assignment and distributed execution on mobile engine. Our extensions of the BPMN 2.0 specification consist of two aspects:

1. **Constraints -** specification and annotation of process models to be executed on the mobile engine. In this phase, the available context information on mobile devices (e.g. geolocation) and the information on task performers (e.g. nurse qualifications) need to be specified with the help of domain experts. With the help of developers, domain experts are able to specify the constraints that need to be satisfied before assigning the tasks and before executing the process on mobile devices. The context information on mobile devices are captured and provided by the mobile process engine.

2. **Services -** invocation definition. As already mentioned, the list of available service is exposed through a developed library inside the mobile process engine. Domain experts only need to consider the business logic and the interaction between the process execution and task performers. Developers will take care of configuration details of the services, such as invocation and data exchange interfaces across services. The framework supports parameter passing from task to task.

Once the process is designed and the process model is customized with annotations designed specially to exploit the characteristics of mobile devices, the process tasks can be assigned to performers.

### 4.2.3 Activity Assignment

The activity assignment on mobile devices is done in four steps and can be represented as a state diagram shown in Figure 4.3. The steps are:

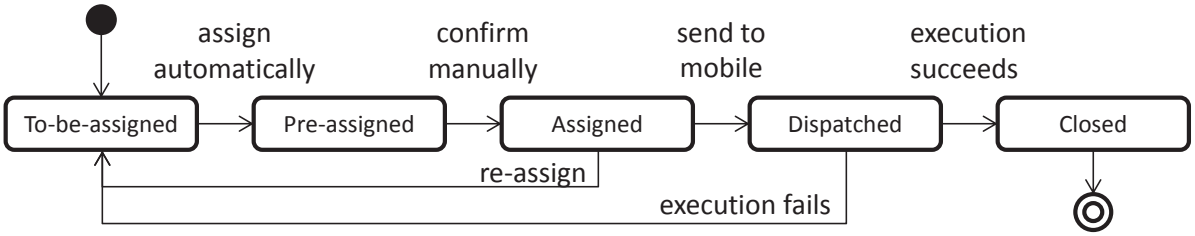



Figure 4.3: Activity Assignment

automatic pre-assignment by the assignment application, confirmation of assignment manually by coordinator, dispatch to mobile device, and the update of execution result.

1. **Pre-assignment.** When the coordinator starts to schedule the process execution, the framework checks if a task is annotated with assignment constraint. If yes, then the task enters the state of *to-be-assigned*. For each task in the *to-be-assigned* state, the framework filters the list of available performers and recommends the best matching ones. The state of the task is then changed to *pre-assigned*.

2. **Assignment confirmation.** By default, the *pre-assigned* tasks need to be confirmed by the coordinator to be *assigned*. Optionally, the task assignment tool can configured to by-pass the manual confirmation.

3. **Dispatching to mobile device:** Now the framework is ready to send the process model to the mobile device. When the process is successfully sent to the performer, it enters the *dispatched* state.

4. **Result Update.** Depending on the execution result, a *dispatched* process can either be *closed* upon successful execution, or return to the *to-be-assigned* state when the execution fails and automatic re-assignment is enabled.

Once assigned, the whole activity can be executed on mobile process engine and results committed and synchronized with the central engine.

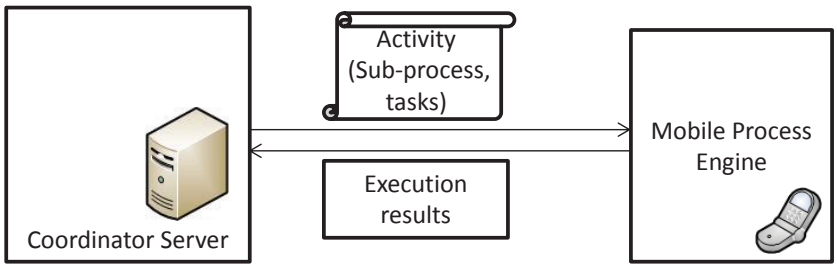



Figure 4.4: Activity Execution on Mobile Device

### 4.2.4 Activity Execution on Mobile Devices

The activity (single task or sub-process) execution on mobile devices is performed by the process engine and the task performers.

Figure 4.4 shows the interaction between the central server and mobile devices process engine. When the performer receives the process on her device, she can read the descriptions of the tasks assigned to her explaining when to start the tasks and how to execute them. When the performer starts the task execution, the mobile process engine checks if the execution constraint matches the task constrains and if the task can be executed under current context situation. If the execution constrains are satisfied, the performer can follow the instructions attached to the assigned tasks (e.g. FormService or EmailService) on the mobile device while the process engine will execute all the other service tasks such (e.g. SOAPService) or triggering of events.

Current engine implementation does not enforce the automatic returning of activity execution results and process state. It is up to the domain experts to decide when and how a process should return collected data to the central server. As it is shown in Figure 4.1, it can be easily defined in the process model how to send back the execution result inside the Send Event elements that interacts with the central engine.

## 4.3 An Extension of BPMN

We extend the BPMN 2.0 specification to support the definition of constraint for assigning and executing sub-process on mobile device, and to enable the mapping of tasks and events to mobile specific services.

The extension is defined in the *extensionElements*, thus does not alter the predefined elements in process definitions. The traditional process editors and engines can still work with the process models that contains *extensionElements* defined by this extension, only that the extended semantics are ignored.

### 4.3.1 Context constraints for activity assignment and execution

We defined two categories of context constraints according to when the checking is performed: activity assignment and activity execution.

**Assignment constraints.** Activity assignment constraints are checked by the mobile server when the coordinator is seeking proper worker to bind to the activity execution instance. This assures the activity is sent to the mobile device of a worker who satisfies the constraints for later execution. Such constraints can impose requirements on: mobile worker (e.g., roles, qualifications, affiliation), or mobile devices configuration (e.g., CPU capacity, free storage available, availability of specific APIs), or on any other context information (e.g., current geography location) available at the time of activity assignment. If there is no constraint to assign the activity, an assignment constraint with $expression = true$ is defined to annotate that it is an activity to assign to mobile devices.

**Execution constraints.** Activity execution constraints are checked when the process engine on mobile device is going to start the execution of an activity. These execution constraints can impose requirements on context information available on mobile devices when starting the activity

execution (e.g., current geography location, time).

Comparing these two categories of constraints, the assignment constrains are about relatively stable parameters characterizing the device and the user along process execution; while the execution constraints can be more transient parameters, since the mobile process engine is going to verify these conditions at the last second before process execution.

The difference can be illustrated with an example: the same parameter of geographic location can appear both in assignment constraint or execution constraint. When it is an assignment constraint, it is more reasonable to be the "low definition" location (e.g. the city where the mobile work is in). When it is an execution constraint, it can be "high definition" location (e.g. the position of mobile worker at patient's house).

It should be noted that the context constraints that we defined can also be expressed with conditional flows. In particular condition expression can be associated with exclusive gateway's outgoing flows to decide which path to take. However, we decided that it is better to detach those constraints that do not alter the structure of business process but only enable/disable the execution of the process. There are two reasons: the process structure is simpler and easier to evolve; it can exploit the rich context information available on mobile devices, and still remain interchangeable with traditional processes.

### 4.3.2 Services provided on mobile devices

We define a *service* element to map tasks and events to mobile specific services. The *service* element can be inserted in *extensionElements* of tasks and events. An attribute *class* is defined in *service* element, which specifies the supporting component on mobile process engine. When the mobile process engine is finishing the execution of a task or event, the value of sub-elements in *service* will be passed to the next task or event.

Our BPMN extension syntax allows third-parties to define their own services. They can extend our mobile process engine or even implement their own engine to support the defined service. The XML schema of sub-elements for services is not restricted. It is up to the corresponding component on mobile process engine to consume the sub-elements of *service*.

So far, our mobile process engine has provided *FormService*, *EmailService*, *SmsService*, *BarcodeService*, and *SOAPService*. More services are under development.

### 4.3.3 Example usage of extension

Below is a fragment of the subProcess definition, for the blood pressure examination scenario:

*Fragment of Blood Pressure Examination Process Model*

```
<subProcess name="Health examination">
 <extensionElements>
   <mpe:constraint type="assignment" expression=
     "performer.hasNurseQualification=true"/>
   <mpe:constraint type="execution" expression=
     "time.hour&gt;9&amp;&amp;time.hour&lt;10"/>
 </extensionElements>

  <task name="Measure blood pressure">
    <extensionElements>
      <mpe:service class="it.unitn.disi.peng.process.
        engine.service.FormService">
        <mpe:text id="hint" value=
          "Enter the measured value of blood pressure (mm Hg)" />
```

```
        <mpe:text id="label_patient_id" value="Patient ID:"/>
        <mpe:input id="patient_id" type="text" />
        <mpe:text id="label1" value="Contraction"/>
        <mpe:input id="contraction" type="text" />
        <mpe:text id="label2" value="Relax"/>
        <mpe:input id="relax" type="text" />
        <mpe:input id="submit" type="submit" value="Submit"/>
      </mpe:service>
    </extensionElements>
 </task>
 <!-- More tasks, events, flows etc. -->
</subProcess>
```

The extension elements in this subProcess define: the condition for assigning this subProcess is that the performer should have a nurse qualification; the condition for executing this subProcess is that the time should be between 9:00 and 10:00 AM; and the component on mobile process engine that supports the execution of task "Measure blood pressure" is the *FormService*; other tasks and irrelevant attributes are omitted here due to limited space.

## 4.4 Mobile Process Engine and UI Framework

The framework that we have developed to support the mobile process definition, BPMN extensions injection to exploit mobile devices characteristics and the process execution includes the following components:

- A lightweight process engine for smartphones with Android operating system.
- A UI Framework that renders the user interfaces on mobile device

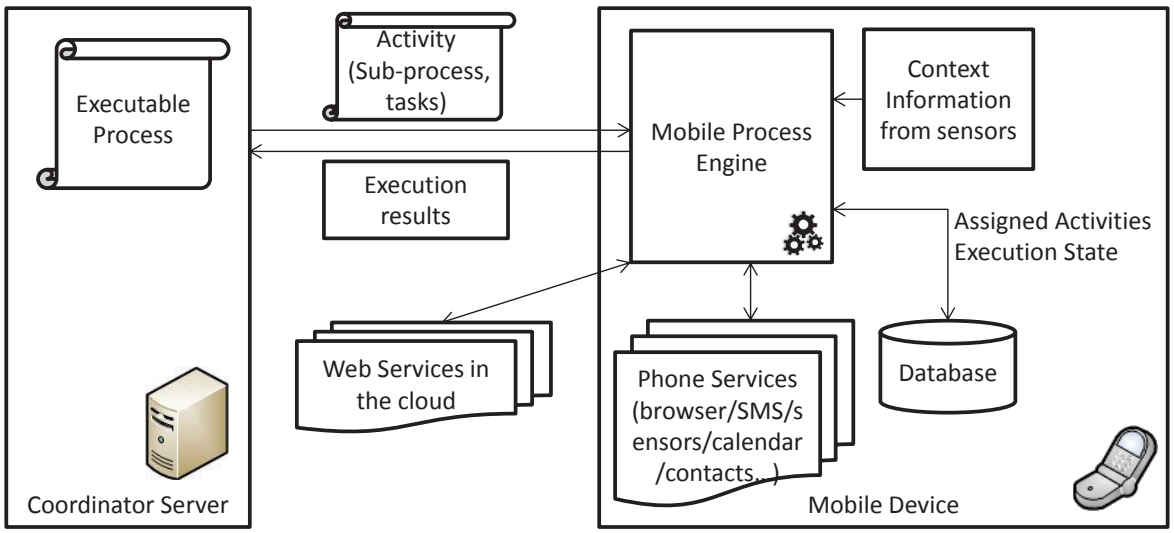



Figure 4.5: Mobile Process Engine

according to the FormService definition. It manages also invocations of existing Android services such as mail service.

- A remote central server that hosts the process, checks the annotations, and assigns activities to the matched performers.

### 4.4.1 Mobile Process Engine

To test the process execution on mobile devices and to solve the identified challenges of process mobility under partially connected environment, we developed the mobile process engine for Android operating system. We implemented the engine as a standalone Java library that parse the BPMN 2.0 XML file and executes a subset of BPMN modeling elements.

Figure 4.5 shows the deployment phase of the processes from central server to the mobile process engine. It shows also a high-level architecture of the engine. When the process is deployed on the device, the engine uses XPath [94] library to parse the process definitions. It checks annotated execution constraints and verifies if their execution is supported on the current device. The contextual information is gathered from device sensors

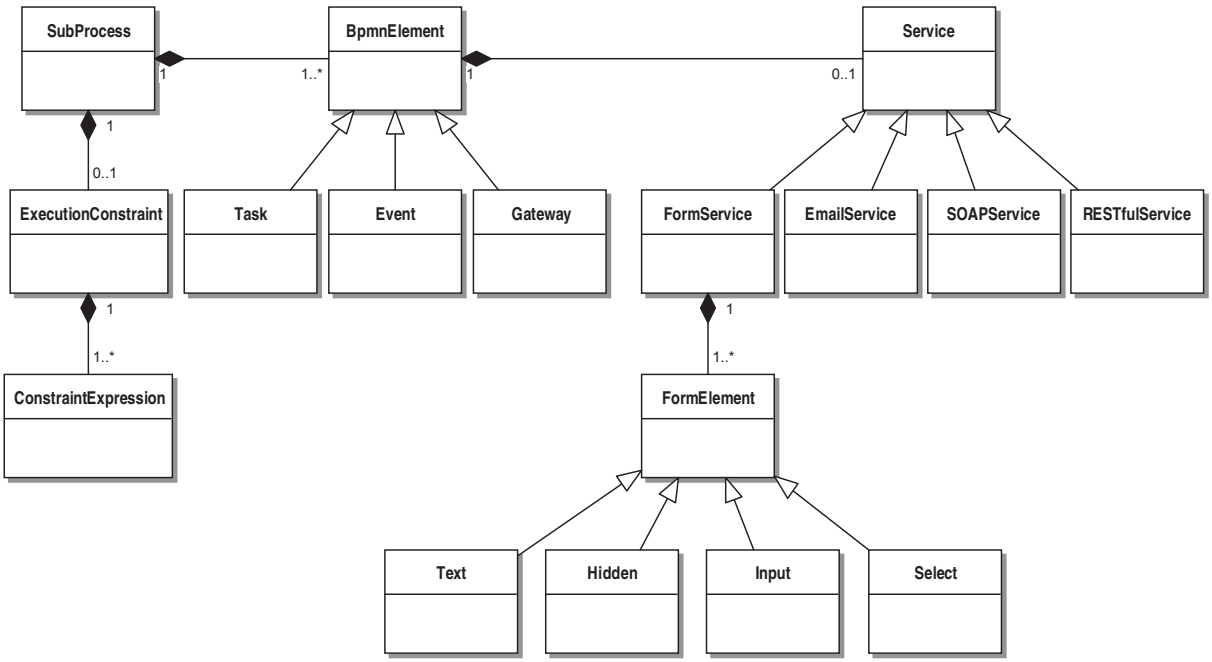



Figure 4.6: Subset of classes in Mobile Process Engine

and saved in the process session or internal database. Data required for the process execution are loaded at deployment time from central server and saved on the device local database that uses a SQLite instance [86] that is natively available on Android operating system.

Figure 4.6 shows the internal structure of the mobile process engine. Starting from the left we can see how a sub-process, that is deployed and executed on a device, is composed by one or many BPMN elements. Elements can be Tasks, Events or standard BPMN gateways that are used to control the flow of the process models. BPMN elements can implement a service. The current version supports the FormService, EmailService, SOAPService or RESTfulService.

When the performer starts the task execution, the mobile process engine checks if the execution constrains attached to the SubProcess, matches the current context information that is collected from the device sensors. If the execution constrains are satisfied, the performer can execute the process.

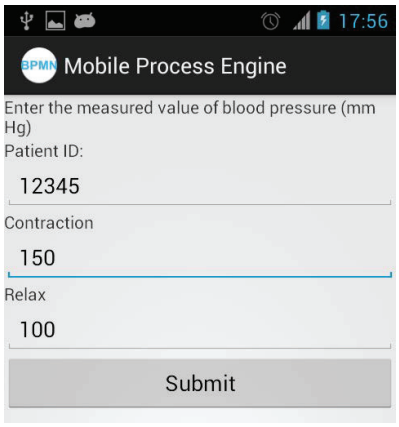



Figure 4.7: Measure blood pressure

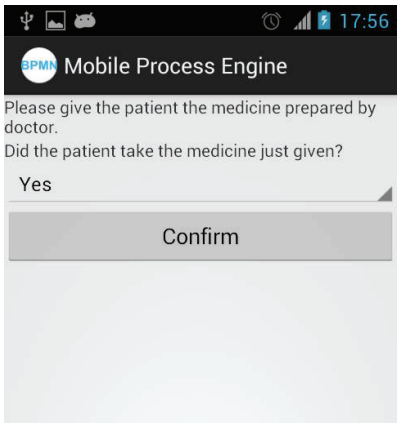



Figure 4.8: Give medicine

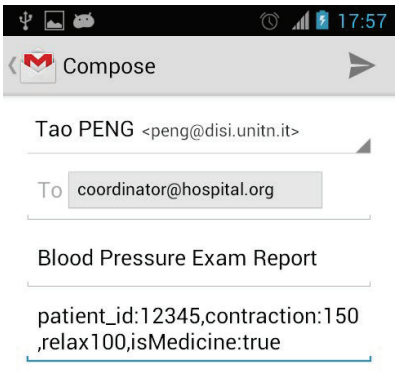



Figure 4.9: Send response

### 4.4.2 UI Framework

The services that interact with users are supported by the mobile process engine. They include the interactive form service that implements the graphical interface to interact with activity performer, the email service that is used to compose and send the email, the short message service that sends short messages, and a barcode service that recognizes barcodes and QR codes.

Figure 4.7 and 4.8 show the usage of the FormService tasks to input patients' data, administer the medicine, and confirm the measurement data before sending them with an email in Figure 4.9 to the coordinator.

### 4.4.3 Controller of Atomic Environment

The coordinator application runs on the controller of the atomic Environment on which an Activiti [1] process engine is deployed. The central process engine supports the process definition and it has been extended to support the parsing of context constraints by checking and automating the activity assignment. As previously mentioned, the supported context constraints include: constraints on environment such as time and geoloca-

tion; constraints on performer profile where any profile attributes defined by domain experts and coordinators can be used to match the activity assignment.

## 4.5 Conclusion

In this chapter, we presented the Business Process deployment and execution framework on mobile devices. We designed an extension of BPMN, which allows the annotation of constraints of processes/tasks and availability of resources on devices. With the constraints and resources availability defined, it is possible to deploy the subprocesses/tasks on devices regarding the constraints. The BPMN extension includes the way to map between tasks definition and the implementation on Android system. We implemented a lightweight Business Process Engine for mobile devices, supporting a few common tasks and a UI framework. With the BPMN extension and mobile process engine, the controller in an atomic Environment can deploy the process and execute it on the devices that it manages. This chapter was published in [67].

# Chapter 5

# Task Allocation Strategies and Optimization

In previous chapters, we described the model of Environment-as-a-Service, the architecture design and the implementation of process/task execution. The implemented framework proves the correctness and feasibility of the model and design. When the framework finds a satisfying Environment, However, we have not discussed the performance optimization of the framework so far. In this chapter, we focus on the framework performance: Based on the implemented framework, we explore various task allocation strategies to improve performance. We also bring up several techniques as possible enhancements of the framework.

## 5.1 Constraint Satisfaction

In the Environment-as-a-Service model, requirements of tasks and the resources constraints are defined as key-value pairs. We compare the requirements and constraints, and get a result in boolean value: either the resources satisfy the requirements, or not. Enforcing the strict requirements ensure that the process/tasks are enacted under the condition decided by the domain experts. However, there are two limitations of applying strict

requirements:

1. The framework cannot decide the best node to allocate the task, when more than one nodes satisfy the requirements.

2. The framework cannot describe and enact tasks which can degrade when the required resources are not fully available.

To describe how much an Environment satisfies a requirement in a finer granularity, we extend the satisfaction between a task requirement and an Environment. In previous model, we define the requirements of tasks, and the constraints of resources. An Environment satisfies a task if and only if it match all the requirements of a task.

We have only two values for the satisfaction between a task and an Environment: true or false. To provide an indicator to differentiate the Environments with the same matching result (satisfied, or not), we need an indicator with finer grain value.

### 5.1.1 Optional Requirement

We first introduce the optional requirement of a task, which is not compulsory but a "good-to-have" condition.

**Definition 13 (Optional Requirement)** *An optional requirement of a task is a condition of task such that: if it is not satisfied, it does not affect the executability of the task; while if it is satisfied, the task has better performance given that other conditions remain the same.*

An optional requirement of a task can be a constraint on a parameter which is not in other requirements. In this case, that parameter is not essential for the execution of the task. An optional requirement can also be about the same parameter defined in a requirement. In this case, that

parameter is essential for the task execution and the essential value is defined in the requirement; additionally the optional requirement defines a higher standard to execute the task.

### 5.1.2 Satisfaction Factor

As we mentioned, we need a value to compare the satisfaction between task and Environment. We call such value Satisfaction Factor.

**Definition 14 (Satisfaction Factor)** *Satisfaction Factor (denoted by $sf$) is a value $sf \in [0, 1]$ that indicates how much all the requirements of a task are satisfied in an Environment.*

Satisfaction Factor is a function of task and Environment ($sf(t, e)$). Domain experts or developers can define the evaluation of Satisfaction Factor arbitrarily, as long as the conditions below establish. For any task $t$, $t'$, and any Environment $e$, $e'$:

- Condition 1: $e$ satisfies $t$, and $e'$ does not satisfy $t' \Rightarrow sf(e, t) > sf(e', t)$
- Condition 2: Both $e$ and $e'$ satisfy $t$, $e$ satisfies all the Optional Requirements those are satisfied by $e'$, and $e'$ does not satisfy at least one Optional Requirement which is satisfied by $e' \Rightarrow sf(e, t) > sf(e', t)$
- Condition 3: Neither $e$ nor $e'$ satisfies $t$, $e$ satisfies all the requirements those are satisfied by $e'$, and $e'$ does not satisfy at least one requirement which is satisfied by $e \Rightarrow sf(e, t) > sf(e', t)$

There are two possible ways to define how to evaluate the Satisfaction Factor given a task and an Environment:

1. The domain experts or developers define the evaluation equation. The design of evaluation equation need to meet the above conditions.

```
function SatisfactionFactor(Env, T)
    if Env.satisfies(T) then
        sf = 1
        if T.hasOptionalRequirement then
            sf− = unsatisfied Optional Requirement number/total Optional Requirement number
        end if
    else
        sf = 0 − unsatisfied requirement number/total requirement number
    end if
end function
```

Figure 5.1: Algorithm: calculate the Satisfaction Factor

2. The framework defines an equation to calculate the Satisfaction Factor. If the domain experts and developers do not specify the evaluation equation, the framework adopts a simple but extendible algorithm to calculate the Satisfaction Factor. We describe the simple algorithm in Figure 5.1. In this algorithm, the Satisfaction Factor is normalized, thus it is possible to compare the Satisfaction Factor of any two pairs of task-Environment.

From:

$0 \leq$ unsatisfied Optional Requirement $\# \leq$ total Optional Requirement $\#$

we can infer:

$$e \text{ satisfies } t \Leftrightarrow sf(e,t) \in [0,1] \tag{5.1}$$

By definition, we have:

$e$ does not satisfy $t \Leftrightarrow 0 <$ unsatisfied requirement $\# \leq$ total requirement $\#$

we can infer:

$$e \text{ does not satisfy } t \Leftrightarrow sf(e,t) \in [-1,0) \tag{5.2}$$

Now, we can prove that this algorithm complies with the conditions for Satisfaction Factor calculation:

- Condition 1: Given $e$ satisfies $t$, and $e'$ does not satisfy $t'$. From Equation 5.1, we have $sf(e,t) \geq 0$. From Equation 5.2, we have $sf(e',t') < 0$. Then we have $sf(e,t) > sf(e',t')$.
- Condition 2: Both $e$ and $e'$ satisfy $t$, $e$ satisfies all the Optional Requirements those are satisfied by $e'$, and $e'$ does not satisfy at least one Optional Requirement which is satisfied by $e' \Rightarrow (e,t)$ has less unsatisfied Optional Requirements than $(e',t) \Rightarrow sf(e,t) > sf(e',t)$.
- Condition 3: Neither $e$ nor $e'$ satisfies $t$, $e$ satisfies all the requirements those are satisfied by $e'$, and $e'$ does not satisfy at least one requirement which is satisfied by $e \Rightarrow (e,t)$ has less unsatisfied Requirements than $(e',t) \Rightarrow sf(e,t) > sf(e',t)$.

The Satisfaction Factor can be extended. Domain experts can assign different weights to the requirements and Optional Requirements. For example, if among the Optional Requirements, the network bandwidth is most important to execute a data uploading task, the developer may want to assign higher weight to the Optional Requirement on bandwidth.

## 5.2 Approximate Allocation vs. Optimal Allocation

With the definition of Satisfaction Factor, we can compare how much an Environment satisfies the requirements and Optional Requirements of a task.

**Definition 15** *Optimal Allocation An Optimal Allocation is an allocation of process on a composite Environment, which has higher satisfaction than any other allocation.*

In this thesis, we calculate the satisfaction of a process by adding up the Satisfaction Factor of all tasks. The resource allocation algorithm we

introduced in previous chapters gives an approximate allocation instead of optimal allocation. More exactly, it gives the first valid allocation that the algorithm finds. Approximate Allocation can emphasize on different performance parameters, including Satisfaction Factor, algorithm efficiency.

Furthermore, a process is not simply a set of unrelated tasks. Tasks in a process are organized in structure, and there are control flow and data flow dependence among tasks. Another consideration in task allocation is the dependence among tasks.

## 5.3 Algorithm Complexity

Finding the optimal allocation for a task has no efficient solution, because we need to examine all allocations. Fortunately, finding an allocation for a task can be solved in logarithm time, using the resource allocation algorithm. Finding an approximate allocation has intermediate complexity, depending on what and to which extent the algorithm optimizes.

## 5.4 Optimization Technique: Data Prefetching

In last two sections, we discussed the optimization strategies of task allocation. Our current framework deploys and executes tasks in Environments with required resources. Mobile Environments are important targeted devices for the task deployment.

Previous sections in this chapter is about task allocation optimization, studying the strategy to better allocate subprocesses and tasks across Environments. This section is about a data prefetching technique which is restricted in atomic Environment. The assumption is that a subprocess or a set of tasks is already deployed in an atomic Environment. The data prefetching algorithm predicts and fetches data into the atomic Environ-

ment for later usage.

Due to the limitation of hardware resources on mobile devices, data access to remote server greatly extends the functionalities of these mobile applications. Unfortunately, the wireless connectivity is still unreliable for current mobile devices, especially in developing countries, remote areas and certain working environments (e.g., some areas in hospitals with wireless signal restriction). On the other hand, more complex mobile applications supported by remote server are designed. The demand of remote data access on mobile devices is increasing rapidly. When the mobile worker is working in an area with unreliable wireless connection, the operations that depend on remote data access become unavailable.

Data prefetching has been proposed as a method to reduce the application response latency caused by slow network communication. The data prefetching systems predict the data objects that are going to be needed by the application in the future, and retrieve them from the remote server into the local cache before they are needed. The application can use the data object in local cache if the needed data is prefetched there. By doing so, the application can reduce the on-demand data access over the unreliable connection and shorten the response latency.

However, data prefetching is a double-edge sword. The wrong prediction causes the system to prefetch data objects that are never used by the application. It brings unnecessary energy consumption, wastes network data usage and congests the precious connection bandwidth. Careful study must be done to balance the cost and benefit of data prefetching. Current prefetching systems focus more on using the limited cache capacity or network connection resources to prefetch as much data as possible. And the design goal is more concerned about the hit ratio of the prefetching, or the trade-off between access latency and prefetching cost.

However, with the recent development on hardware, storage capacity on

on mobile devices is no longer a major concern. Under the context of mobile workforces, power supply (in the working field or in car) is frequently available to recharge the working device, and most working devices would have sufficient or even unlimited data plan to use. The major concern for mobile workers is the availability of essential services under various contexts (connectivity, remaining battery amount, current working status, etc.) The availability of services depends on data availability in run-time. And the less important services can be delayed until the connectivity is available again later.

In this section, we propose a model to modularize the mobile application into operations, and to describe dependency on data objects. The model allows the domain experts to specify the priorities of different operations. On top of the model, two different algorithms are proposed to schedule the prefetching of operations.

The Markov Chain based algorithm fits procedure-oriented applications better. It modularizes the operations in mobile application as Markov Chain. Based on the Markov Chain model, two strategies are studied: Incremental Prefetching is more pessimistic, and focuses on value of next one prefetched operation; while Complete Prefetching is more optimistic, and considers not only the value of next prefetched operation itself but also the additional value that it makes successive operations reachable. Dependency Graph based algorithm is more suitable for content-oriented applications. It can generate operation probabilities based on the Dependency Graph or from execution history. Max priorities are passed through operation dependencies, thus guarantees the dependencies are satisfied in prefetched operations. Both algorithms differ from existing solutions in exploiting the priority of operations specified by domain expert. And both allow dynamic adjustment to adapt to changing contexts, such as connectivity, power status.

The data prefetching technique introduced in this section can apply to applications constructed by different components, including the processes on our framework. With the data prefetching technique, the framework can accelerate the task execution in atomic Environments and increase the service availability.

### 5.4.1 Scenario and Challenges

In this subsection, we are going to depict a motivating scenario of mobile-assisted healthcare service and the challenges from it.

A hospital provides healthcare services to elderlies who are in high risk of various health problems. Nurses are equipped with smartphones, and mobile applications are developed to assist the periodical healthcare tasks. A nurse follows similar routine every day: In the morning she arrives at the hospital, fetches her working device - a smartphone installed with application designed to assist home-visiting tasks. She launches the mobile application, and receives a list of patients that she needs to visit, which is planned by the hospital considering the locations of patients' houses.

The nurse arrives at the first patient's house and carries out the healthcare service with the assistance of the mobile application. The mobile application executes the healthcare service tasks composed by operations and guides the nurse through the process task by task. During the process, the mobile application retrieves information from the server in hospital and sends back the information entered by nurse or measured by connected devices. After the task at a patient is finished, the nurse goes to the next patient on the list. After all the patients on today's list are visited, she returns to the hospital.

By communicating with the remote server, the mobile application is able to show more related information of the patients and collect the data in real-time fashion. However, due to the limitation of current mobile devices

and wireless network, there are several challenges from this scenario:

1. The patients' residences can be sparsely distributed in the area, and some of them can be in the location where wireless data connection on phone is weak or even unavailable. And many patients are elderlies and have not installed Wi-Fi access point in house. In these locations where the connection is weak or lost, the nurse is not able to carry on the tasks whenever the mobile application needs to access the data from the server or send data back.

2. The data access latency makes the mobile application less responsive, thus hinders the usability and user experience of the application.

3. The energy consumption due to the data communication drains the limited battery of the mobile devices.

It is difficult to predict the data that are going to be required later in runtime. The misconducted data prefetching creates unnecessary network traffic and occupies precious mobile network bandwidth. Ad-hoc design and tuning of data prefetching and synchronization are techniques to overcome these challenges under different contexts [40, 54, 15]. These approaches contributed in solving the challenges in their own settings. However, mobile workforces first would like to ensure the availability of essential service before try to improve responsiveness. To better utilize the increased resources to tackle the challenges faced by mobile workforces, we need a data prefetching solution focuses on increasing the availability of essential operations. More specifically, we need:

- a model to modularize the mobile application. Data dependency of the application modules and dependencies across modules need to be specified. And it allows domain experts to specify the priorities of operations;

- a prefetching algorithm that considers the priority of application modules to be first-class citizen, and exploit both current and future available network bandwidth and cache capacity to maximize the availability of essential operations;

- a framework to handle the details of prefetching algorithm implementation, and still retains the control of operation priority for domain experts. Domain experts only need to define the priorities of operations. Developers can help to specify the data dependencies of operations. Then the framework takes care of other parameters (e.g., the probabilities of operations) for the prefetching algorithm, generates the prefetching schedule and makes necessary adjustment in run-time.

### 5.4.2 Model of Data and Operations

To support the data prefetching in mobile application, we first model the data and specify the concerned parameters of data, then the application is modularized into operations, and the parameters concerning the dependencies are also defined. To be general, we separate the abstract models with the framework design that adopts the models to carry out the data prefetching. Thus, the data and operation models defined here are reusable for different frameworks.

#### Data Objects

Here, we are only interested in the data objects that are going to be fetched from the remote server onto the mobile devices. These data objects are serializable, and it is up to the application to parse or parcel the data objects. An data object is denoted by $d_i$, set of all communicated data $D$, and data size $Size(d_i)$.

The framework is going to provide a data access Application Programming Interface (API) for the application. So it can extract the data dependency of operations from the application without extra efforts from the developers. If the application does not know the data size before starting prefetching algorithm, the algorithm can give a approximate solution, which is also sufficient for our purpose.

#### Operation

We modularize a mobile application into operations. An operation contains tightly-coupled computation, user interaction and data access. For example, it can correspond to an Activity [1] in Android programming.

Here we focus on the remote data access, i.e., the operation fetches the data from the remote server in runtime. When an operation needs to fetch the data from the remote server using the API, we say that it depends on that data object.

An operation is denoted with $o$, and the set of all operations in an application is $O$. All data dependencies of an operation can be denoted with a set of relation $DataDependencies = \{(o_i, D_i)|D_i \subseteq D\}$.

Operation priority $o_i.priority$ is an essential parameter in our model. In a large mobile application, the size of data required by all operations is large. The scarce resources on mobile devices restricts the prefetching of all the data required. However, in many cases, we are interested in ensuring the availability of an essential operations. Assigning priorities to operations is then necessary. Priority of an operation is the importance for it to be available when it is needed, and it represents the business value of that operation. So the domain experts need to specify the priorities of different operations and the framework should not alter such specification in runtime, unlike the dynamic adjustment of execution probability.

[1] http://developer.android.com/reference/android/app/Activity.html

The priority is specified by domain expert as an integer ranging in $[1, 10]$ with 10 as the highest priority. For example, the doctor specifies that in a blood pressure examination, the operation with the highest priority is to record the blood pressure value, while the advices on food and exercise are with lower priority.

The hospital can assign priorities to different tasks to be performed by a nurse in a day. For example, the hospital assigned the priority of a task $t_k$ in the nurse's daily schedule to $t_k.priority$ and the doctor specified the priority of an operation $o_i$ within the process $t_k$ to $o_i.priorityInProcess$, then the priority of the operation within the day is:

$$o_i.priority = t_k.priority * o_i.priorityInTask$$

The most valuable operation that worth the cost of data prefetching to ensure its availability is then the operation that with the highest priority and execution probability. More generally, the value of an operation can be modeled as $o_i.value = o_i.priority \times o_i.probability$.

### 5.4.3 Markov Chain based Prefetching Algorithm

This subsection describes the prefetching algorithm based on Markov Chain. It addresses the procedure-oriented applications which are more structured, and the transition operations are well defined.

Using Markov Chain model, an execution instance of an application is a random process of operations executions, and the probability of transitions between operations depends only on current state (operation). In procedural applications, it is often the case that the sequence of operations are decided during the application development, and the probability to transit to the next operation depends on current operation. This probability can be extracted from the execution history. The set of transitions between

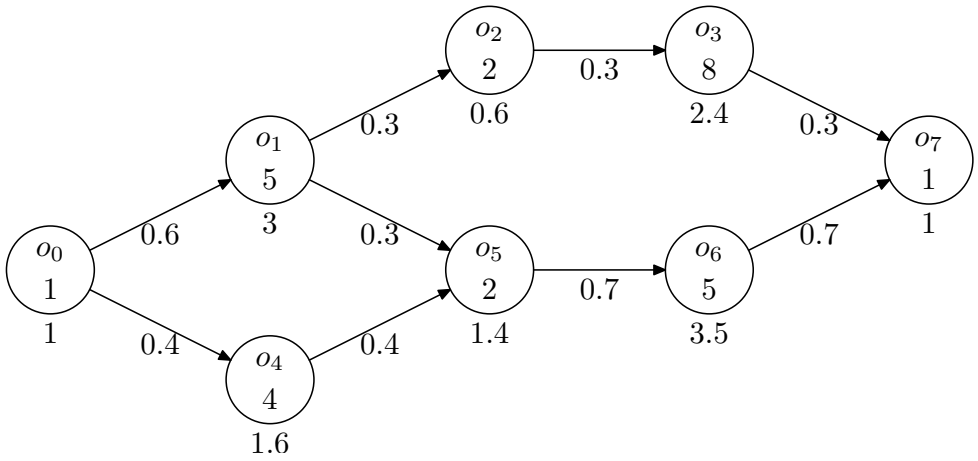



Figure 5.2: Operation Dependency Graph. A vertex represents an operation $o_i$, with specified priority $o_i.priority$ labeled in the circle. An arrow represents a transition, with probability $o_i.probability$ calculated by the framework labeled on the arrow. The number below the vertex is the value of operation $o_i.value$.

operations is denoted as:

$T = \{(o_i, o_j)|\text{if current operation is } o_i\text{, the next operation is possible to be } o_j\}$

#### Calculating Probabilities of Operations

Fig. 5.2. shows how an application is structured as operations and transition between operations. The probability of an operation being executed is initially calculated from the operation execution sequence graph, based on the provided probability of taking different branches. If such branching probability is unknown, they are set to uniform distribution. And after later iterations, the probabilities are adjusted according to the execution history. The calculated conditional probabilities of transitions are denoted as:$P(o_j|o_i)$, and the probability of a transition is denoted as $P(o_i, o_j) = P(o_i) \times P(o_j|o_i)$

**Scheduling Algorithm based on Markov Chain Model**

Depending on the prefetching limitation, there are two different scheduling algorithms for Markov Chain Model.

**Algorithm 1** Incremental Prefetching Algorithm based on Markov Chain

```
function SCHEDULE(O, T, o_0)
    PrefetchQueue ← {o_0}
    PrefetchRecall ← (o_0, 1) ▷ map, key: operation o; value: the probability that it is
executed and the execution sequence from o_0 to o falls in PrefetchQueue
    OpenOperations ← NextOperations(o_0)
    while OpenOperations ≠ ∅ do
        find out the operation o ∈ OpenOperations with highest o.recall × o.priority
        PrefetchQueue.push(o)
        OpenOperations.remove(o)
        for all o' ∈ NextOperations(o) do
            if o' ∉ PrefetchQueue then
                OpenOperations.add(o')
                PrefetchRecall.put(o', o.recall)
            else
                PrefetchRecall.put(o', o.recall)
                for all o'' ∈ {o''|o'' ∈ OpenOperations ∧ o'' is reachable from o'} dox
                    PrefetchRecall.put(o'', o.recall)
                end for
            end if
        end for
    end while
    return PrefetchQueue
end function
```

Incremental Prefetching: Whenever possible (e.g., available cache size increases due to resources release or network is idle), Incremental Algorithm incrementally prefetches the next operation with the highest value whenever it is possible. The incremental algorithm (Algorithm 1) only considers the value of the next prefetched operation, but neglects the value of

**Algorithm 2** Complete Prefetching Algorithm based on Markov Chain

**function** SCHEDULE($O, T, o_0$)
  **for all** $o \in O$ **do**
    $o.value = o.priority \times o.probability$
    $o.av = o.value$
  **end for**
  $Decided \leftarrow \emptyset$
  **while** $Decided \neq O$ **do**
    take an undecided operation $o$ that all its next operations are decided ($o \in (O - Decided) \wedge \forall o' | (o, o') \in T \Rightarrow o' \in Decided$)
    **for all** $\{o' | (o, o') \in T\}$ **do**
      $o.av+ = o'.av \times P(o|o')$
    **end for**
    $Decided.add(o)$
  **end while**
  $Queue \leftarrow \{o_0\}$
  **while** $Queue \neq O$ **do**
    Among operations that are reachable in one hop from Queue, find the one with max accumulative value $av$, and add to the Queue
  **end while**
  **return** $Queue$
**end function**

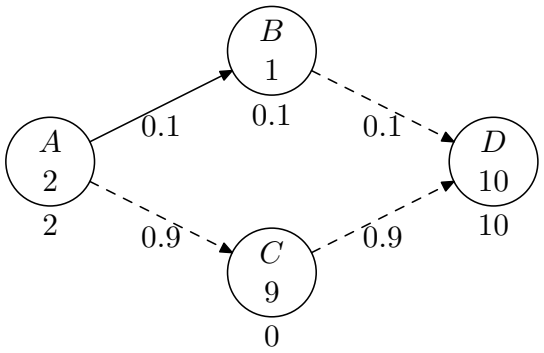



Figure 5.3: Incremental Prefetching (step 1) (The number below the vertex is the value of operation): $B$ and $C$ are one hop reachable and $B$ has the highest value, so the prefetch $B$; then $C$ and $D$ are one hop reachable and $D$ has the highest value, so prefetch $D$; prefetch the last one - $C$.

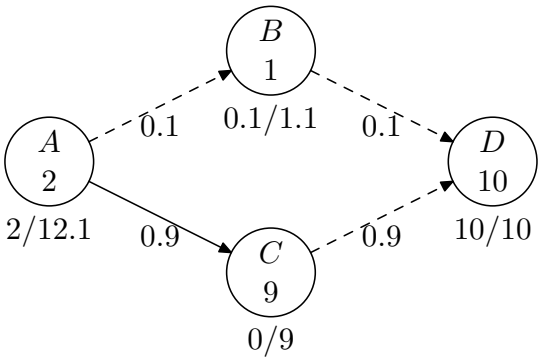



Figure 5.4: Complete Prefetching (step 1) (The pair of number below the vertex is the value/(accumulated value) of operation): $B$ and $C$ are one hop reachable and $C$ has the highest accumulated value, so the prefetch $C$; then $B$ and $D$ are one hop reachable and $D$ has the highest accumulated value, so prefetch $D$; prefetch the last one - $B$.

its successive operations. It is more suitable when only one operation can be prefetched, or the responsiveness is more concerned than availability.

Complete Prefetching: Complete Algorithm (Algorithm 2) always considers the value of successive operations when calculating the value, even only one more operation can be prefetched. It is more suitable when more operations can be prefetched, or the availability is more concerned than responsiveness.

#### Dynamic Adjustment

The application execution transits to a new operation, dynamic adjustment of prefetching can be applied. Both Incremental Algorithm and Complete Algorithm can be adjusted instead of calculating all over again.

When a new schedule is calculated, the next step is to find out in the prefetched operations:

- The operations that are also in new schedule. The prefetching of these operations are skipped.
- The operations that are not in new schedule but are still reachable. The prefetched data of these operations remain in the cache until the cache is full, then among them those with lowest $MaxDependingValue$ are replaced by the new prefetched data.
- The operations that are no longer reachable. The prefetched data are removed.

### 5.4.4 Dependency Graph based Prefetching Algorithm

Different from procedure-oriented applications, content-centric applications are composed by loosely coupled operations: they can be executed in any order as long as the dependency between operations are satisfied. Transition graph of the operations becomes less useful for these applications since the number of transitions is large and the probabilities of each transition is low. To better deal with prefetching of these applications, we propose Value Passing Algorithm which is based on the Dependency Graph (DG) among operations.

If an operation $o_i$ is available only if another operation $o_j$ is available, we say that $o_i$ depends on $o_j$, denoted by $o_i \rightarrow o_j$.

$$o_i \rightarrow o_j \iff (o_i \text{ is available } \Rightarrow o_j \text{ is available })$$

Dependency among operations can be extracted from the application structure (e.g. $o_i$ needs to invoke $o_j$ to fulfil its task), or inferred from the execution log (e.g., whenever $o_i$ is executed, $o_j$ is also executed), or specified by developers. Loop is not allowed in our operation dependency definition. If such loop exists, we can merge operations on the loop into one, sacrificing some precision.

The operation dependency relation is *transitive*: $o_i \rightarrow o_j \wedge o_j \rightarrow o_k \Rightarrow o_i \rightarrow o_k$ The closure set of *OperationDependencies* ($ODC$) is denoted by:

$$ODC = \{(o_i, o_j) | o_i \text{ is available } \Rightarrow o_j \text{ is available}\}$$

or equivalently:

$$(o_i, o_j) \in ODC \iff o_i \rightarrow o_j$$

*DirectOperationDependencies* ($DOD$) is the set of dependencies among two operations that no third operation lies in between:

$$DOD = \{(o_i, o_j) | o_i \rightarrow o_j \wedge ((\forall o_k \mid o_i \rightarrow o_k \wedge o_k \rightarrow o_j) \iff (o_k = o_i \vee o_k = o_j))\}$$

**Calculating Probabilities of Operations**

**Algorithm 3** Probability Generation Algorithm based on Dependency Graph

```
function GeneratePROBABILITIES(O, DOD)
    for all o ∈ O do
        o.probability = 1/|O|
    end for
    Decided ← {o|no other operation depends on o}
    while Decided ≠ O do
        take one undecided operation that no other undecided operation depends on it:
o ∉ Decided ∧ ∀o' ∉ Decided, (o', o) ∉ DOD
        o.probability = P(o or o_1 or o_2 or ... or o_n) o_1, o_2, ..., o_n directly depend on o
        Decided.add(o)
    end while
    Normalize probabilities of all operations
end function
```

Probabilities of operations are calculated in three steps: assign initial probability; accumulate probabilities according to the operation dependency; finally normalize the probabilities (see Algorithm 3). There are three ways to initialize the probabilities of operations:

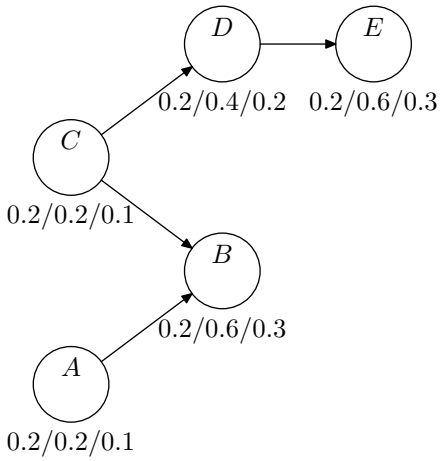



Figure 5.5: Simple Probability Generation in DG. Vertices - operations; arrows - dependencies; probabilities under vertices - initial / accumulated / normalized.

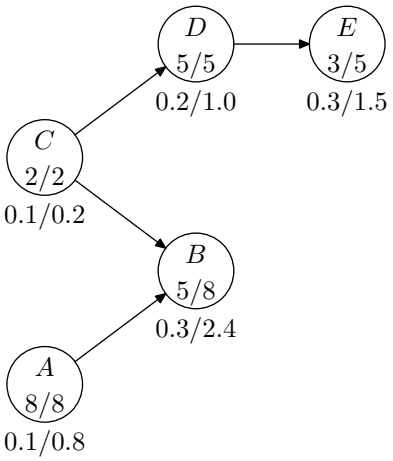



Figure 5.6: Priority Passing in DG. Number in vertices - priority / max dependent priority; number under vertices - probability / value. Generated schedule: BEDAC

1. **Simple Probability Generation.** Without knowledge about the probabilities of operations, we assume that the probabilities of the intend to execute operations are equal. Operations gain extra probability because other operations depend on them. From operation dependency, probabilities of operations are accumulated and normalized.

2. **Pre-assigned Probability Generation.** This algorithm is similar with Simple Probability Generation except that the initial probabilities are specified by domain experts.

3. **Probability extraction from execution history.** When execution history data is sufficient, we can also extract the probabilities of operations directly from it.

**Prefetching Scheduling Algorithm based on Dependency Graph**

Once the probabilities of operations are decided, we are able to schedule the prefetching of operations. The algorithm (Algorithm 4) is divided into two steps:

1. Passing the priority of operations along the dependency relations. A *max dependent priority* (*mdp*) is defined of an operation, representing the max priority of those operations that depend on it.

2. The value of each operation is calculated by $maxdependentpriority \times probability$. And the prefetching algorithm schedules from operations with the highest value.

**Algorithm 4** Prefetching Scheduling Algorithm based on Dependency Graph

**function** SCHEDULE($O, DOD$)
  **for all** $o \in O$ **do**
    $o.mdp = o.priority$
    $o.value = o.mdp \times o.probability$
  **end for**
  $Decided \leftarrow \{o|$no other operation depends on $o\}$
  **while** $Decided \neq O$ **do**
    take one undecided operation which is not depended by other undecided operation: $o \notin Decided \wedge \forall o' \notin Decided, (o', o) \notin DOD$
    $o.mdp = max(o'.mdp)\ (o', o) \in DOD$
    $o.value = o.mdp \times o.probability$
    $Decided.add(o)$
  **end while**
  Sort $O$ according to *value* in descending order
**end function**

**Dynamical Adjustment**

After the prefetching is scheduled, the framework can greedily prefetch as much as possible until the *DataSizeLimit* is met. However, the stor-

age capacity of current smartphones is no longer a very scarce resource, comparing to the network data usage and the battery consumption. It is not always profitable to prefetch until the cache is full, under the circumstances where the energy and data usage is restricted. For example, in the healthcare scenario, when the nurse has been working without charging for a long time, the remaining battery may become a potential threat to the availability of operations (and the whole device).

Looking back to the dependency graph based algorithm, operations on the prefetching schedule has lower and lower value, and it becomes not worth to prefetch any more, even the $DataSizeLimit$ is not exhausted. We introduce another constraint $ValueThreshold$ to decide whether to prefetch more operations. Since we know the max depending value of an operation is larger or equal to all operations that depends on it: $\forall o_j|(o_j, o_i) \in DOD, o_i.value \geq o_j.value$. We can stop to prefetch when the algorithm meets an operation with lower max depending value than the threshold.

When the context switches (an operation finishes, the mobile device connects to a different network, or the power charger is plugged, etc.), a new schedule is computed using these updated parameters:

- Initial operation is set to current operation. Operations that become unreachable from current operation is pruned from the graph.

- Probability of operations. The probabilities of reachable operations are updated.

- $ValueThreshold$ considering the new context (connectivity, power status). The $ValueThreshold$ corresponding to different contexts can be tuned in experiments.

### 5.4.5 Simulation

We simulate both procedure-oriented and content-oriented mobile applications under Android Emulator and compare the performance of different prefetching algorithms.

#### Markov Chain based Algorithms

The application simulates the same process in Figure 5.2. Data related with each operation are simulated by a binary file with 10 K Byte of randomly content. All the data are stored on a remote server, which is common for mobile applications. Using the binary file with random content eliminate the possible distortion of simulation result caused by compression techniques implemented in network. We simulate the unstable network by adding random network delay in the emulator. When the delay is longer than a threshold, we assume that this network connection fails.

Then we run the application for 50 times, which is sufficient for the example process with 3 possible traces. The application makes random decision to move to the next operation, according to the predefined probability at each branch. So the application takes a possible trace of operations in each round. When the application is at an operation, it first checks in the cache whether the data was prefetched. If the data was in the cache, the operation succeeds, and the application runs the prefetching algorithm to prefetch data for the next steps. If the data was not in the cache, we assume that the operation fails and the application stops this round and starts a new trace from the initial operation.

We set a cache size for each round, starting from 0 to 80 K Bytes (large enough to prefetch the data for all operations). For each round, the simulation outputs the sum of priorities of all the successfully executed operations. Figure 5.7 shows the simulation result of Markov Chain based

algorithms. X-axis represents the size limit of each prefetching (i.e., cache size if we do not consider other limits such as network traffic limits or energy consumption limit), and Y-axis represents the (average of 50 rounds) sum of priorities of successful operations.

We compare the Complete Prefetching Algorithm, Incremental Prefetching Algorithm and Simple Prefetching Algorithm. Simple Prefetching Algorithm prefetches the next-hop operations with highest probabilities at each operation. We first observe from the diagram: when the cache size is 0, no data can be prefetched so the application always fails, and the sum of priorities is 0 for all algorithms; when the cache size is as large as the total size of related data, the application is able to prefetch all the data, so all the algorithms have the maximum sum of successful priorities, and the values are equal. The difference is in the range between 0 and maximum size: Both Markov Chain based algorithms have higher priority sum than Simple Prefetching Algorithm; Complete Prefetching has similar performance with Incremental Prefetching, and performs better when the prefetch limit is 20 K Bytes or 50 K Bytes.

#### Dependency Graph based Algorithms

We implemented another Android application to simulate the content-centric application corresponding to the Dependency Graph in Figure 5.5. The simulation adopts the same procedure when executing an operation as in the Markov Chain based Algorithm simulation: At each operation, the application first checks whether the related data is cached. If not the application fails; if yes the application runs the prefetching algorithm to decide and prefetch data for the later operations.

One difference is that, for content-centric application simulation, the application does not follow a predefined process but initiates operations randomly. If the chosen operation depends on other operations, the appli-

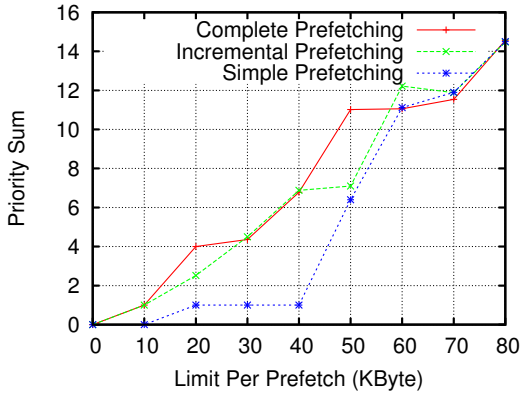



Figure 5.7: Simulation result - Markov Chain based algorithms

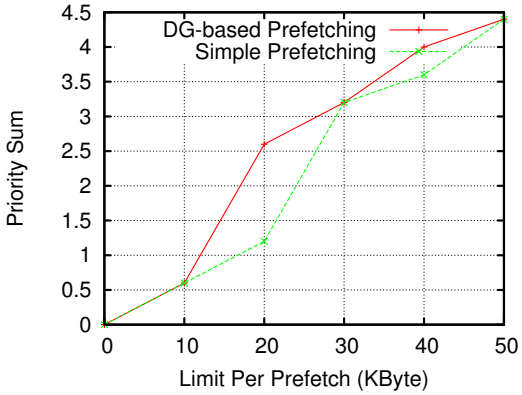



Figure 5.8: Simulation result - Dependency Graph based algorithms

cation will execute the depended operations first. As the baseline, we also implement a simple prefetching algorithm: it prefetches starting from the operations that are depended by most operations.

Figure 5.8 shows the simulation result. When the prefetch size limit (cache size) is 0, no operation will be successful. When the limit is large enough to prefetch data for all operations, the priority sum is largest, and different algorithms have the same result. When the limit is in between, our Dependency Graph based Algorithm performs as well as Simple Algorithm at some points, and outperforms it at other points.

### 5.4.6 Work Related with Data Prefetching

Data prefetching (or caching, which is related) has been a fundamental approach to improve performance of different types of systems. It involves predicting the possible demanded data for the applications, and the mechanism to retrieve the data from the remote server onto the local devices. Accompanied with data prefetching, another related technique is data caching, which focuses on predicting the possible future reuse of the already acquired data. While both prefetching and caching rely on the pre-

diction of future data access, researches on prefetching in different types of systems focus on balancing the cost and benefit of prefetching, due to the possible increase in traffic and waste of energy.

#### Mobile Data Management

Trifonova and Ronchetti proposed an approach to hoard the learning material on mobile devices [91]. For each learning session, it first predicts the starting point to be the index page for the first time access, and then ending point of last session for the later access. Then it predicts the required material along the learning process, according to the links across pages. It uses user profile to capture the preference and style of different groups of learners, and materials are prioritized to increase the chance of more important materials to be hoarded. This approach is similar with our Dependency based Algorithm. The difference is that Trifonova's approach exploits the rich domain knowledge on e-learning (learner profiling, learning material prioritizing); while our approach is more general and depends only on the usage probability and user predefined priority.

For structured data, data ming techniques and materialized view are popular techniques in predicting the data for prefetch or caching. For example, the algorithms proposed by Agrawal and Srikant can extract association rules from database usage to decide which "basket" of data objects should be prefetched as a batch [2]; Neto and Salgado proposed to mine the SQL history of the mobile user and assign priority to a subset of data for cache [59]. Jane et al. proposed to use association rule mining to determine the data to prefetch and use Dual Valid Scopes to invalidate the data in cache [47]. These researches focus more on the lower level data record prediction, which is difficult in the changing context. Higgins et al. proposed "Informed Mobile Prefetching" to let the application to inform the intend of prefetch of data [44], then the prefetching system decides the

intension with the highest value and prefetch the data opportunistically according to the network status and available energy and data quota. The "Informed Mobile Prefetching" inspired us with the abstract data access API design and the idea of dynamic prefetching. While "Informed Mobile Prefetching" requires extra development efforts to send prefetching intend, and it also aims to trade off between latency and energy / traffic. Our approach focuses on operation level prediction and our goal is to increase the availability of operations.

#### Distributed File Systems

Accessing files under disconnections has been a research topic since the early development of distributed file systems. Kistler and Satyanarayanan proposed the three-state transition among "hoarding", "emulation" and "reintegration" of prefetching systems [49]. They studied the issues of designing and implementing disconnected operation. SEER is another hoarding framework for file systems [51]. It measures lifetime semantic distance among files, clusters them into overlapping clusters and maintains measurements and clusters as the file system evolves. Thanks to their pioneer work, we can reuse the methodology of data reintegration, and focus on the new challenges brought by the modern mobile devices. Different from file prefetching, our approach address the availability of higher level operations, the actual data usage is performed by application through predefined API. Today's powerful mobile devices makes it possible and promising for our prefetching approach.

#### World Wide Web

The fast response is one factor for the success of World Wide Web. The data prefetching research in web application can be categorized into content-based prefetching and history-based prefetching. Content-based prefetch-

ing predict the future access by analyzing objects and links on the web pages. [27] predict future requests based on keywords in anchor text of URL. History-based prefetching uses dependency graph, Markov chain, cost function, or data ming approaches to predict the future user request [102]. Lymberopoulos et al. propose to use a machine learning approach based on stochastic gradient boosting techniques to to predict the web access on a per user basis [56]. Reda et al. proposed to a solution for user to notify the kiosk by SMS to prefetch private data [77]. These studies provide us inspiring ideas. Our major difference is that mobile workforce applications has more clear scope of data access and structure of operations. Traditional web prefetching/caching techniques care more about the access latency, and the intensiveness of network communication forces them to conserve resources such as cache size and network bandwidth. While we are more concerned on the availability of operations, and the operation-oriented mobile working encourages the utilization of available storage and traffic to improve the service availability.

### 5.4.7 Conclusion on Data Prefetching

The computational capability and wireless connectivity of mobile devices are improving. Mobile devices constitute important part of the atomic Environments in our model and framework. They increase the diversity of executable tasks/processes and the service availability. However, the environmental context of such mobile devices is becoming more complex, and connectivity remains a threat to the availability of the mobile applications used in field.

We proposed to differentiate the priority of different tasks/operations in an application, and utilize the available resources (storage capacity, network data usage etc.) to maximize the availability of essential operations. Based on an abstract model, we design a framework and two different

data prefetching scheduling algorithms. Markov Chain based algorithm has more strength for applications that are procedure-oriented and structured as sequential operations, while the Dependency Graph based algorithm is more suitable for applications that are content-oriented and described by operation dependency rules. Both algorithms take the priority of tasks/-operations as input from the domain experts, infers the tasks/operation execution probability, maximizes the availability of essential tasks/operations, and allows dynamic adjustment to changing contexts. Our data prefetching algorithms were published in [66].

## 5.5 Task Allocation in Resource-limited Environments

In previous sections, we discussed the strategies and techniques to optimize the execution performance of processes/tasks. To start from a simpler setting, we had two assumptions in processes/tasks allocation:

1. Environments have sufficient resources in runtime for the allocated processes/tasks, so multiples processes/tasks allocated in the atomic Environment in the same time period can be executed.

2. All processes/tasks to be allocated are known at the beginning. Allocation algorithm has the complete information about tasks and Environments at initial state.

These assumptions establish in scenarios with smaller number of tasks. They simplify the initial design and implementation of the framework. However, when the number of processes/tasks to deploy and execute becomes too large, the Environments cannot deploy and execute them efficiently. The first assumption does not establish.

In a continuously running framework, processes and tasks are generated or assigned dynamically. One way to server the dynamically incoming

processes is to hold them and execute them in a cycle. In a cycle, The framework can collect a number of incoming processes and tasks, and then deploy and execute them, after that start the next cycle. This method solve the allocation problem but is inefficient because it holds the processes and defers the execution of the earlier received process.

To improve the process/task allocation performance for resources-limited Environments, we enhance the processes/tasks allocation to address the limited resources, and further to cope with the dynamically incoming processes/tasks.

### 5.5.1 Task Allocation Optimization for Resource-limited Environments

The problem of allocating tasks in given Environments is similar with "Bin Packing Problem". [20] Bin Packing Problem is the problem of finding a solution that uses least containers to pack a set of given objects. Our problem is to find a solution that packs most objects (tasks) into a set of given containers (Environments).

We use a vector to represent the requirements of a task: $req =< req_1, req_2, \ldots, req_n >$, and a vector to represent the resources available on an Environment: $res =< res_1, res_2, \ldots, res_n >$. For example, if three types of requirements are modeled: bandwidth, available memory size, and availability of Bluetooth connection, we use $< 100, 128, 1 >$ to represent the requirements of a task which requires 100 (kbps) bandwidth, 128 (MB) available memory size, and Bluetooth connectivity. We use $< 800, 256, 1 >$ to represent the resource of an Environment at a certain state, which has 800 (kbps) bandwidth, 256 (MB) available memory and Bluetooth connectivity. The vectors are comparable but addition and subtraction does not work, because allocating two or more tasks together may not occupy the resources as the sum of them. To decide whether a task fits in an Envi-

ronment, we need to detect the available resources on that Environment in run-time, and compare it with the requirements of the task. The more types of requirements are modeled, the higher dimension the vector has.

We have one assumption here: allocating a task in an Environment will not make the previously allocated tasks unsatisfied. When we detect the satisfaction in run-time, this assumption establishes, because the function should return "unsatisfied" if the new task will hamper an allocated task.

Here is the formal statement of the problem: given a set of independent tasks ($\mathbb{T} = \{T_1, T_2, \ldots, T_m\}$) and their priorities ($\{P_1, P_2, \ldots, P_m\}$), vectors ($(Req)$) those represent the requirements of the tasks, a set of atomic Environments ($\mathbb{E}$), and a function to detect the run-time available resources on the Environments ($res(e)$), find an allocation of tasks to Environments with largest sum of priorites ($\mathbb{A} = \{< t, e > | t \in T \wedge e \in E \wedge e \text{ satisfies } t\}$)

A trivial solution is to compare all the possible allocations to find out the one with highest sum of task priorities allocated. For $m$ tasks and $n$ Environments, each task can be allocated to one of the $n$ Environments, or not allocated. There are $n^{m+1}$ possible allocations, so the time complexity of this trivial solution is $O(n^{m+1})$.

Base on the hybrid architecture that we adopted for the first version of framework, we propose an approximate algorithm: we first sort the tasks by priority, and then start to allocate from the task with highest priority exploiting the function of controllers in hybrid architecture. When a controller receives a task, it checks the resource tree to see if a child Environment satisfies the requirements. If yes, the task is sent to that child Environment; otherwise, the controller responds that this Environment does not satisfy the task.

Sorting the tasks has the time complexity of $m * log(m)$, and allocating the tasks in the hierarchical Environments has the time complexity of $m * log(n)$, so the overall time complexity of the algorithm is $m * log(m) + m *$

$log(n)$, i.e. $m * (log(m) + log(n))$.

The algorithm does not always produce the optimal solution. The trick is on the algorithm for a controller to decide whether a child Environment satisfies a task or not. Depending on the relationship between the controller decision and the reality, we have four situations:

1. Controller decision matches the reality. It means that the controller always make the correct decision about whether a child Environment satisfies a task or not.

2. When a controller decides that a child Environment satisfies a task, it is always correct. When a controller decides that a child Environment does not satisfy a task, it is not always correct.

3. When a controller decides that a child Environment does not satisfies a task, it is always correct. When a controller decides that a child Environment satisfies a task, it is not always correct.

4. Whatever decision (a child Environment satisfy a task or not) a controller makes, it is not always correct. In this case, the controller make the "best effort" decision based on the incomplete information on children Environment resources or incomplete calculation.

In situation 1, to get the exact result, the controller has to check all the children Environments as well as all the descendants. The time complexity of such operation is $n$, making the overall algorithm $m * n$. In situation 2, 3 and 4, approximate algorithms can make fast decisions based on the aggregated information on children Environments. Algorithm for situation 2 is too pessimistic, whereas algorithm for situation 3 is too optimistic. An example algorithm is to use the upper/lower bound of the resources in the children Environments in calculation. The error comes from the variance on the resources of children Environments. When the children

Environments have smaller variance on resources, the algorithm decision error is smaller.

When multiple children Environments satisfy the same task, we apply the Satisfaction Factor to select a child Environment with the highest Satisfaction Factor.

To address the problem of limited resources, the algorithm needs to track the available resource in run-time. One solution is to measure and report the resources. When a task is allocated to an Environment or a task execution is finished, the controller updates the amount of available resources.

### 5.5.2 Task Allocation Optimization for Dynamically Incoming Tasks

This problem is similar with the "Dynamic Bin Packing Problem". "Bin Packing Problem" is proven to be an NP-hard problem [20]. The difficulty comes from the unpredictable arrival of tasks, because the previous sorting solution fails without knowing all the tasks.

We propose the optimization algorithm based on the previous subsection. For those tasks which are known, we apply the same algorithm to allocate them to the Environments. Later, when a new task arrives, we find an Environment that satisfies the task. We run the algorithm twice, first by checking the static resources, and second by checking the run-time available resources. The two results are two candidate locations for the new task: the first is the location where the task can be deployed when the Environments have no other tasks running; and the second is the location when the Environments are deployed with previous tasks. We can deploy the new task onto either one of these locations if they are different. Deploying onto the first location requires migrating at least one previous deployed task out of the Environment, which causes an overhead. The

decision depends on the comparison of cost and benefit of task migration, and is not further discussed in this thesis.

### 5.5.3 Conclusion on Task Allocation Optimization

Optimizing tasks allocation is no easy task, because there are many factors to consider: the priority of tasks, resources available on Environments, satisfaction between task's requirements and the Environment's resources, run-time status of task execution and Environments' workload.

Finding the optimal allocation requires high time complexity. Approximate solutions can exploit the hierarchical resources information of Environments to reduce the time complexity of the allocation algorithm. The price is that some tasks may miss to be allocated when the Environments do have sufficient resources.

Allowing tasks to arrive and leave further complicates the problem. Our algorithm handles this dynamic allocation well, although the performance is to be tuned. More sophisticated optimization techniques were discussed in [65].

# Chapter 6

# Related work

The same as the structure of this thesis, our work can be divided into several parts: model, architecture, task deployment and execution, optimization. Works related with optimization techniques and algorithms such as prefetching algorithms and task allocation, were discussed in Chapter 5, because they are specific topics and relatively independent of the rest of the thesis. In this chapter, we discuss the related work focusing on abstract model, overlay network architecture, and business process/tasks deployment and execution on mobile.

## 6.1 Model

Modeling software systems has been a research topic for software engineering and related communities. In the past years, research on service science and engineering and cloud computing has developed a set of concepts, methodologies, tools and prototypes, which inspire us on the model of Environment-as-a-Service. We present the related work from different aspects:

### 6.1.1 Service on Devices

Researchers have invested efforts to abstract the physical interfaces of pervasive devices into software services. Abstract interfaces separate the software development into different layers, and increase the reusability and interoperability of software components.

Frameworks that involve devices were proposed for proprietary programming lanuaguages. The industrial OSGi framework proposes a service gateway to manage the devices as Java modules. It decouples the work of system integrators and the devices developers, allowing discovery and dynamic integration of devices in enterprise systems [26]. Based on OSGi framework, Helal et al. developed the Gator Tech Smart House (GTSH), which is an experimental smart home instrumented with a range of sensing and smart technologies [41]. OSGi is only for Java platform, and there is a center to manage the service lifecycle (install, uninstall, start, stop) and a registry that manages the process of "publish, find, bind". So it can be used in cross-platform environment and without a unique center to mange the resources. In contrast, our service model is not limited to a specific platform (such as Java for OSGi), and Environments are designed to be autonomous and composable with the resources management and routing mechanism.

A recent software architecture is Service-Oriented Architecture [29]. It models software applications into pieces (services) which provide self-contained functions to other applications. Services can be invoked across the network via a vendor-independent protocol. Web Service is a common implementation of the communication protocol for SOA. Tergujeff et al. demonstrated that it is possible to consume Web Services on light-weight J2ME-enabled mobile devices [90]. To use the resources on mobile devices, we need to provide services on mobile devices, instead of only consuming

services on network. Srirama et al. experimented to implement a Web Service Host on smartphones [87]. Due to the resource limits, hosting the traditional Web Services on mobile devices is expensive. As experimental work, it did not mention specific functions on mobile devices. The privacy and security concern is also not addressed.

"SODA" (Service Oriented Device Architecture) proposes an abstract model to bridge the device interface and the SOA bus [23]. SODA uses device adapters to talk with devices of proprietary or industry standard, and provides interfaces complying Service-Oriented Architecture. SODA focuses on converting physical devices into standard invocable services, while the services discovery and composition are left to the traditional Service-Oriented Architecture standards and tools. In our argument, traditional SOA tools are usually designed for stationery computers thus are too "heavy" for mobile devices. And traditional way of service registry management does not fit well with mobile devices, because they are unstable difficult to address. The resource scarcity and primitiveness to private environments make people reluctant to convert their devices into services and publish to a public (or even limited access within an organization) registry. Our framework supports mobile devices better, and provides finer granularity management and access control for groups of devices, which fits the organization hierarchy.

Huerta-Canepa et al. proposed a solution to share the resources on mobile devices to provide virtual cloud services [45]. Depending on the context, resource-intensive computation is offloaded to nearby mobile devices via an ad-hoc network. The devices in vicinity are discovered in P2P scheme. This approach does not solve the resource scarcity and security/privacy issues. With the cloud infrastructure and ever improving wireless connection coverage, it becomes easier to offload the heavy computation tasks to more powerful cloud infrastructure. Our approach is

focused on exploiting the specific functions of devices in field.

Jae Yoo Lee et al. designed a framework to capture mobile context for applications [53]. On client side, the framework has a three layer design (physical, service, application), and it uses sensors on mobile devices to collect and infer context situation for different application use. On server side, it uses a Model-View-Controller (MVC) architecture to store the collected context information as well as the context knowledge database. This framework supports an interesting way of application development: define rules to react on different contexts. However, the framework only provides services on context sensing, and the effectiveness of rule-based application development is not discussed.

### 6.1.2 Service Discovery

In their survey, Ververidis and Polyzos analyze existing research in service discovery for Mobile Ad Hoc Networks (MANETs), and pointed out one open issue is that these protocols and standards lack of interoperability [93]. Chakraborty et al. proposed a de-centralized architecture to support the service composition in ad-hoc environment. This approach modeled devices as the basic components in the system and installed middleware on these devices, thus it is not able to address the emerging devices with limited capacities. And the one-layer architecture is not suitable for the networks those are not managed by a unique organization [14].

Rasch, Li et al. proposed to personalize the service discovery based on the context [73]. They proposed a model Hyperspace Analogue to Context (HAC) to describe context, service, and user preference. The proposed approach proactively captures the user's context, and presents the most relevant services in response to the change of context, services, or user preferences. The approach provides useful hints to our future improvement on context-aware service discovery. Despite the fact that it captures the

disruption of services, it does not address the challenges of services running on mobile devices, such as mobility and more strict access control.

### 6.1.3 Service Composition in Pervasive Environment

Kyusakov et al. deployed SOAP protocol directly on sensor nodes, translating into the lower level TCP/IP based API invocation [37]. It takes lots of efforts to implement the invocation conversion from SOAP to TCP/IP. And the problem of service discovery and management is still to be solved.

Ravindranath, Lenin, et al. [75] proposed a task execution framework for non-expert users to compose tasks for single or multiple devices. However, the dependence across multiple devices needs to be hardcoded, which is difficult at design time and prune to failure.

The aforementioned researches of service on devices focus on abstracting individual devices and enabling service discovery in a single environment, our model focus on composition of mobile environments as well as their services, thus is more scalable to network size.

### 6.1.4 Distributed Application Processing on Mobile Devices

Cyber foraging is a technique to offload resource-intensive tasks from mobile devices to more powerful surrogate devices nearby [80, 83, 84]. Flinn reviewed the development of the cyber foraging research, and discussed how cyber foraging systems partition and offload data and computation [32]. Kristensen et al. designed a framework "Locusts" [50]. They modeled computational tasks as directed graphs composed by services. The resource-intensive services can be offloaded at runtime to surrogates. A Lucusts daemon uses UDP broadcast over Wi-Fi to detect nearby surrogates. Ha et al. proposed "Just-in-time" provisioning for cyber foraging [38]. It improves the Virtual Machine migration efficiency by provisioning

a set of VM images of common systems, and a compressed binary difference that encompasses the customization such as installed libraries. This VM-based solution is too "heavy" for mobile devices.

Balan et al. proposed a solution for developers to modify mobile applications to partition and offload computation to servers [10]. However, the framework helps only in generating the API stubs. It requires developers to manually locate the part of application that worth offloading, and reimplement the offloaded component on server.

## 6.2 Architecture

In Chapter 3, we presented three possible architectures for our framework: centralized architecture, P2P architecture, and hybrid architecture. In this section, we discuss the literatures on architectures, network topology and communication protocols of different network systems, and compare with our framework.

DNS (Domain Name System) decomposes domain name look up service into different levels [57]. Each domain server resolves a part and forward the look up request to the next subdomain server. DNS has a different environment setting: servers are well connected, and provide similar services.

Sensor Network Systems form ad-hoc networks from distributed resource-restricted devices [3]. The sensor nodes usually have similar capacities and share the same connection protocol. Traditional network routing protocols enable the end-to-end message transmission [70]. Peer-to-peer (P2P) systems enables resources sharing and allocating among peers [5].

The distributed network systems provide a set of useful protocols and network structures. We adopt similar hierarchical controller design as DNS and reuse underlying mechanisms (e.g., flooding, election) in WSNs and P2P systems. However, our approach works on top of heterogeneous de-

vices and targets for execution of tasks that have dependency among them. Our proposal of Environment Composition and routing also addresses the specific concern on unreliability of devices and connections.

## 6.3 Process Deployment and Execution on Mobile Devices

Light-weight business process engines have been developed to support mobile process execution. Sliver [39] is BPEL process engine for mobile device, but as it stated, the task allocation and data distribution challenges are not solved. The ROME4EU project [79] enables the single task assignment from a team leader's smartphone to other members' phones. It does not support assignment of process other than single tasks, and relies on the network during process deployment. Presto [36] is a pluggable platform that allows mobile users to perform different tasks depending on roles, physical environment, and process state. Since its focus is on process development on Internet of Things, physical deployment of process on mobile device during run-time is not mentioned in the paper and the linked project website.

Efforts have been investigated to tolerate the unreliability of mobile devices in business process execution. Philips et al. designed a new workflow language "NOW" to support dynamic service discovery and communication to tolerate the communication or service failure in nomadic network [68]. Similarly, Mostarda et al. described an approach that can automatically generate a distributed choreographic implementation of a logically centralized orchestration process [58]. Different from these works, our focus is to enable dynamic activity assignment. Zaplata and Lamersdorf proposed a process management resource sharing and billing mechanism [99]. But it still depends on connection, if not worse due to its peer-to-peer process

engine sharing. Our process engine differs in the distributed way of process execution. Processes are decomposed into subprocesses or tasks, and then assigned to the Environments with required resources for execution.

Another topic related with process enactment on mobile is context constraints of business process. It has been studied for business process task access control [82, 96]. Our context constraints serve the similar purpose of activity assignment and execution. The proposed model of context constraints differs from others in the separation of assignment constraints and execution constraints. Under partially connected environment, this two-step control on constraints diminishes invalid activity assignment at early stage, and still enforces an accurate control of constraints on execution context.

# Chapter 7

# Conclusion, Limitations and Future Work

## 7.1 Conclusion

More and more devices other than traditional computers, are becoming "smart". They include mobile phones, tablets, wearable devices, domestic electronics, vehicles, etc. They have more powerful computational capacity, and are connected with each other through different types of connection (WI-FI, Bluetooth, etc.). When these diverse devices are connected together, there is high potential to create powerful, diverse, and valuable applications for different industries. However, traditional software development as well as recent research on is facing the difficulty in deploying onto such complex environment: devices have different hardware configuration; their connectivities are not reliable; their connection topologies changes with the movement of physical devices; different users have different access privilege to different groups of devices.

We studied the characteristics of these connected devices and proposed a theoretical model of the network of devices. We modeled the connected devices into hierarchical "Environments", which provide services on top of the underlying heterogeneous devices. To support infrastructural functions

of Environment-as-a-Service model, we proposed the resource management and message routing mechanism. Different network architectures are discussed, focusing on the resource management and message routing design and performance regarding to scalability and protocol efficiency.

We developed a proof-of-concept framework, based on Android smartphones. It proves the feasibility of model, resource management and routing protocol. Tasks can be deployed onto the devices with required resources. For atomic Environments, we designed and implemented a light-weight process engine to orchestrate the enactment of assigned processes/-tasks. Several examples of tasks are supported by the light-weight process engine, including email, Short Message Service, barcode reading, form generation (generate a form according to the specified parameters for user interaction).

At the end, we presented the possible optimization for the framework, including a data prefetching mechanism for facilitate the task execution on atomic Environments, and approximate algorithms for dynamic task allocation.

The innovation of this work includes:

- The theoretical Environment-as-a-Service model that abstracts the heterogeneous devices to a hierarchical and composable structure.
- The process and task routing protocol design based on the resource management.
- Implementation of a proof-of-concept framework based on the model. Processes and tasks generated in any Environment can be allocated to the destination Environment with required resources.
- A mobile process engine for process orchestration for Android smartphones.

- A data prefetching mechanism for task execution on mobile devices and an approximate algorithm to allocate tasks on resouce limited devices.

## 7.2 Limitations

Our Environment-as-a-Service model adopts hybrid architecture with controllers in Environments. The number of overlay network level and the size of Environments cannot be too large. Otherwise it compromises the performance of resource management and routing, because: the workload of controller increases linearly with the Environment size; and the effectiveness of resource allocation algorithm decreases when the network level increases. Although the test of preliminary framework gained satisfying result, we have not done test on large scale network to experiment the maximum supported number of levels and size of Environments.

Current framework includes a set of predefined services, which are ready to use for invocation and service composition. Tasks or processes can only invoke predefined services so far. Although the framework is open, and it allows new services definition and publishing, we have not implemented the mechanism for service discovery for runtime.

We support limited set of business process structures. Some structures require complex concurrency control and are not supported in current implementation. Our learned the lesson that it is difficult to support traditional business process enactment on mobile devices due to the resource limitation. We restricted the effort to supporting the essential set of BPMN structures, and focused on the mobile specific services which are more attractive.

## 7.3 Future Work

In the future, the framework can be better improved by providing more services, such as process/task deployment and execution accounting. The Environment-as-a-Service model can be a real business model only with the corresponding accounting.

We need to make the new services publication and discovery easier. One idea is to build a market (like Apple AppStore [7] or Google Play [69]), which allows the publication and acquisition of services/processes developed by third party developers. It can motivate the process design and encourage the reuse of good process design.

The performance study under large scale networks is yet to be evaluated. More services can be implemented on mobile process engine to reduce the repetitive implementation of common functions.

# Bibliography


[1] Activiti BPM Platform. http://activiti.org/.

[2] Rakesh Agrawal, Ramakrishnan Srikant, et al. Fast algorithms for mining association rules. In *Proc. 20th Int. Conf. Very Large Data Bases, VLDB*, volume 1215, pages 487–499, 1994.

[3] Kemal Akkaya and Mohamed Younis. A survey on routing protocols for wireless sensor networks. *Ad hoc networks*, 3(3):325–349, 2005.

[4] Gustavo Alonso, Fabio Casati, Harumi Kuno, and Vijay Machiraju. *Web services*. Springer, 2004.

[5] Stephanos Androutsellis-Theotokis and Diomidis Spinellis. A survey of peer-to-peer content distribution technologies. *ACM Computing Surveys (CSUR)*, 36(4):335–371, 2004.

[6] Apple AppStore. https://itunes.apple.com.

[7] Michael Armbrust, Armando Fox, Rean Griffith, Anthony D Joseph, Randy Katz, Andy Konwinski, Gunho Lee, David Patterson, Ariel Rabkin, Ion Stoica, et al. A view of cloud computing. *Communications of the ACM*, 53(4):50–58, 2010.

[8] Hari Balakrishnan, M Frans Kaashoek, David Karger, Robert Morris, and Ion Stoica. Looking up data in p2p systems. *Communications of the ACM*, 46(2):43–48, 2003.

[9] Rajesh Krishna Balan, Darren Gergle, Mahadev Satyanarayanan, and James Herbsleb. Simplifying cyber foraging for mobile devices. In *Proceedings of the 5th international conference on Mobile systems, applications and services*, pages 272–285. ACM, 2007.

[10] Roger Barga, Dennis Gannon, and Daniel Reed. The client and the cloud: Democratizing research computing. *IEEE Internet Computing*, (1):72–75, 2011.

[11] Apple Bonjour. http://www.apple.com/support/bonjour/.

[12] Rajkumar Buyya, Chee Shin Yeo, Srikumar Venugopal, James Broberg, and Ivona Brandic. Cloud computing and emerging it platforms: Vision, hype, and reality for delivering computing as the 5th utility. *Future Generation computer systems*, 25(6):599–616, 2009.

[13] Rodrigo N Calheiros, Christian Vecchiola, Dileban Karunamoorthy, and Rajkumar Buyya. The aneka platform and qos-driven resource provisioning for elastic applications on hybrid clouds. *Future Generation Computer Systems*, 28(6):861–870, 2012.

[14] Dipanjan Chakraborty, Anupam Joshi, Yelena Yesha, and Tim Finin. Toward distributed service discovery in pervasive computing environments. *Mobile Computing, IEEE Transactions on*, 5(2):97–112, 2006.

[15] Naveen Chauhan, LK Awasthi, and Narottam Chand. Data caching with intelligent prefetching in mobile ad hoc networks. In *Communication Systems and Network Technologies (CSNT), 2011 International Conference on.*

[16] Liang Ben Chen and Maria Prokopi. A resource-aware pairing device framework for ubiquitous cloud applications. In *Innovative Mobile*

*and Internet Services in Ubiquitous Computing (IMIS), 2012 Sixth International Conference on*, pages 252–258. IEEE, 2012.

[17] Erik Christensen, Francisco Curbera, Greg Meredith, Sanjiva Weerawarana, et al. Web services description language (wsdl) 1.1, 2001.

[18] Byung-Gon Chun and Petros Maniatis. Augmented smartphone applications through clone cloud execution. In *HotOS*, volume 9, pages 8–11, 2009.

[19] CISCO. Connections counter: The internet of everything in motion, July 2013.

[20] Edward G Coffman Jr, Michael R Garey, and David S Johnson. Approximation algorithms for bin packing: A survey. In *Approximation algorithms for NP-hard problems*, pages 46–93. PWS Publishing Co., 1996.

[21] Manuela Corradi. Design Collaborativo e Soluzioni Tecnologiche per lHealthcare: il caso MOPAL (Mobile Palm for Assisted Living) [Collaborative Design and Technological Solutions for Healthcare: the case MOPAL]. In *Master Degree Thesis*. University of Trento, 2010.

[22] Frank Dabek, Ben Zhao, Peter Druschel, John Kubiatowicz, and Ion Stoica. Towards a common api for structured peer-to-peer overlays. In *Peer-to-Peer Systems II*, pages 33–44. Springer, 2003.

[23] Scott de Deugd, Randy Carroll, Kevin E Kelly, Bill Millett, and Jeffrey Ricker. Soda: Service oriented device architecture. *Pervasive Computing, IEEE*, 5(3):94–96, 2006.

[24] Giuseppe Di Modica, Orazio Tomarchio, and Lorenzo Vita. Resource and service discovery in soas: A p 2 p oriented semantic approach.

*International Journal of Applied Mathematics and Computer Science*, 21(2):285–294, 2011.

[25] Hoang T Dinh, Chonho Lee, Dusit Niyato, and Ping Wang. A survey of mobile cloud computing: architecture, applications, and approaches. *Wireless communications and mobile computing*, 13(18):1587–1611, 2013.

[26] Pavlin Dobrev, David Famolari, Christian Kurzke, and Brent A Miller. Device and service discovery in home networks with osgi. *Communications Magazine, IEEE*, 40(8):86–92, 2002.

[27] Josep Domenech, Jose A Gil, Julio Sahuquillo, and Ana Pont. Using current web page structure to improve prefetching performance. *Computer Networks*, 2010.

[28] Thomas Erl. Service-oriented architecture. *Concepts, Technology, and Design*, 2004.

[29] Thomas Erl. *Service-oriented architecture*, volume 8. Prentice Hall New York, 2005.

[30] Niroshinie Fernando, Seng W Loke, and Wenny Rahayu. Mobile cloud computing: A survey. *Future Generation Computer Systems*, 29(1):84–106, 2013.

[31] Roy Thomas Fielding. *Architectural styles and the design of network-based software architectures*. PhD thesis, University of California, Irvine, 2000.

[32] Jason Flinn. Cyber foraging: Bridging mobile and cloud computing. *Synthesis Lectures on Mobile and Pervasive Computing*, 7(2):1–103, 2012.

[33] Ian Foster, Yong Zhao, Ioan Raicu, and Shiyong Lu. Cloud computing and grid computing 360-degree compared. In *Grid Computing Environments Workshop, 2008. GCE'08*, pages 1–10. Ieee, 2008.

[34] Armando Fox, Rean Griffith, A Joseph, R Katz, Andrew Konwinski, Gunho Lee, D Patterson, Ariel Rabkin, and Ion Stoica. Above the clouds: A berkeley view of cloud computing. *Dept. Electrical Eng. and Comput. Sciences, University of California, Berkeley, Rep. UCB/EECS*, 28:13, 2009.

[35] Hector Garcia-Molina. Elections in a distributed computing system. *Computers, IEEE Transactions on*, 100(1):48–59, 1982.

[36] P. Giner, C. Cetina, J. Fons, and V. Pelechano. Presto: A pluggable platform for supporting user participation in smart workflows. In *MobiQuitous, 2009*.

[37] Jonas Gustafsson. Integration of wireless sensor and actuator nodes with it infrastructure using service-oriented architecture. *IEEE TRANSACTIONS ON INDUSTRIAL INFORMATICS*, 6(1):1, 2011.

[38] Kiryong Ha, Padmanabhan Pillai, Wolfgang Richter, Yoshihisa Abe, and Mahadev Satyanarayanan. Just-in-time provisioning for cyber foraging. In *Proceeding of the 11th annual international conference on Mobile systems, applications, and services*, pages 153–166. ACM, 2013.

[39] Gregory Hackmann, Mart Haitjema, Christopher Gill, and Gruia catalin Roman. Sliver: A bpel workflow process execution engine for mobile devices. In *ICSOC*, 2006.

[40] JunZe Han and Xiang-Yang Li. Network agile preference-based prefetching for mobile devices. *arXiv preprint arXiv:1208.0054*, 2012.

[41] Sumi Helal, Chao Chen, Eunju Kim, Raja Bose, and Choonhwa Lee. Toward an ecosystem for developing and programming assistive environments. *Proceedings of the IEEE*, 100(8):2489–2504, 2012.

[42] Pedro Henriques and Adolfo Garo. A lightweight distributed super peer election: Algorithm for unstructured dynamic p2p systems on the internet of things. 2012.

[43] Steffen Hess, Felix Kiefer, Ralf Carbon, and Andreas Maier. mConcAppt–A Method for the Conception of Mobile Business Applications. In *Mobile Computing, Applications, and Services*, pages 1–20. Springer, 2013.

[44] Brett D. Higgins, Jason Flinn, T. J. Giuli, Brian Noble, Christopher Peplin, and David Watson. Informed mobile prefetching. In *MobiSys '12*, 2012.

[45] Gonzalo Huerta-Canepa and Dongman Lee. A virtual cloud computing provider for mobile devices. In *Proceedings of the 1st ACM Workshop on Mobile Cloud Computing & Services: Social Networks and Beyond*, page 6. ACM, 2010.

[46] Olivia Nottebohm Michael Chui Borja de Muller Barbat Remi Said Jacques Bughin Laura Corb, James Manyika. The impact of internet technologies: Search. *International Journal of Mobile Communications*, 2011.

[47] F Jane, N Ilayaraja, M Ashwin Raghav, R Nadarajan, and Safar Maytham. Cache prefetch and replacement with dual valid scopes for location dependent data in mobile environments. In *iiWAS '09*.

[48] Brad Karp, Sylvia Ratnasamy, Sean Rhea, and Scott Shenker. Spurring adoption of dhts with openhash, a public dht service. In *Peer-to-Peer Systems III*, pages 195–205. Springer, 2005.

[49] James J Kistler and Mahadev Satyanarayanan. Disconnected operation in the coda file system. *ACM Transactions on Computer Systems (TOCS)*, 10(1):3–25, 1992.

[50] Mads Darø Kristensen and Niels Olof Bouvin. Developing cyber foraging applications for portable devices. In *Portable Information Devices, 2008 and the 2008 7th IEEE Conference on Polymers and Adhesives in Microelectronics and Photonics. PORTABLE-POLYTRONIC 2008. 2nd IEEE International Interdisciplinary Conference on*, pages 1–6. IEEE, 2008.

[51] Geoffrey H Kuenning. The design of the seer predictive caching system. In *First Workshop on Mobile Computing Systems and Applications, 1994.*

[52] Karthik Kumar and Yung-Hsiang Lu. Cloud computing for mobile users: Can offloading computation save energy? *Computer*, (4):51–56, 2010.

[53] Jae Yoo Lee, Soo Dong Kim, et al. A comprehensive framework for mobile cyber-physical applications. In *Service-Oriented Computing and Applications (SOCA), 2011 IEEE International Conference on*, pages 1–6. IEEE, 2011.

[54] Junghoon Lee, Gyung-Leen Park, Sang-Wook Kim, Hye-Jin Kim, and Sung Y Shin. A hybrid prefetch policy for the retrieval of link-associated information on vehicular networks. In *Proceedings of the 2010 ACM Symposium on Applied Computing.*

[55] Eng Keong Lua, Jon Crowcroft, Marcelo Pias, Ravi Sharma, Steven Lim, et al. A survey and comparison of peer-to-peer overlay network schemes. *IEEE Communications Surveys and Tutorials*, 7(1-4):72–93, 2005.

[56] Dimitrios Lymberopoulos, Oriana Riva, Karin Strauss, Akshay Mittal, and Alexandros Ntoulas. Pocketweb: instant web browsing for mobile devices. *SIGARCH Comput. Archit. News*.

[57] Paul V Mockapetris. Domain names-concepts and facilities. 1987.

[58] Leonardo Mostarda, Srdjan Marinovic, and Naranker Dulay. Distributed orchestration of pervasive services. In *Advanced Information Networking and Applications (AINA), 2010 24th IEEE International Conference on*, pages 166–173, 2010.

[59] Mariano Cravo Teixeira Neto and Ana Carolina Salgado. Hoarding and prefetching for mobile databases. In *ICIS-COMSAR*, pages 219–224. IEEE, 2006.

[60] Object Management Group. Business Process Model and Notation (BPMN) Version 2.0. PDF, Januar 2011. http://www.omg.org/spec/BPMN/2.0/PDF.

[61] Cristhian Parra, Tao Peng, Cristina Matteotti, Vincenzo Dndrea, Aleksey Kashytsa, and Davide Martintoni. Allenavita: Motivating lifestyle improvement in a telemedicine scenario. In *Persuasive Technologies*, page 53, 2014.

[62] Chris Peltz. Web services orchestration and choreography. *Computer*, 36(10):46–52, 2003.

[63] Tao Peng, Giampaolo Armellin, Dario Betti, Annamaria Chiasera, Tefo James Toai, and Marco Ronchetti. Mdo: framework for context-

aware process mobility in building-maintenance domain. In *Software Maintenance and Reengineering (CSMR), 2013 17th European Conference on*, pages 449–452. IEEE, 2013.

[64] Tao Peng, Chi-Hung Chi, Annamaria Chiasera, Giampaolo Armellin, Marco Ronchetti, and Cristina Matteotti. Modeling and composition of environment-as-a-service. In *Mobile Cloud Computing, Services, and Engineering (MobileCloud), 2014 2nd IEEE International Conference on*, pages 250–251. IEEE, 2014.

[65] Tao Peng, Chi-Hung Chi, Annamaria Chiasera, Giampaolo Armellin, Marco Ronchetti, Cristina Matteotti, Cristhian Parra, Aleksey Oleksiy Kashytsa, and Alessio Varalta. Business process assignment and execution in mobile environments. In *Collaboration Technologies and Systems (CTS), 2014 International Conference on*, pages 267–274. IEEE, 2014.

[66] Tao Peng, Marco Ronchetti, Annamaria Chiasera, and Giampaolo Armellin. Dynamic data prefetching for mobile workforces. In *Mobile Web Information Systems Workshops*. Springer, 2013.

[67] Tao Peng, Marco Ronchetti, Jovan Stevovic, Annamaria Chiasera, and Giampaolo Armellin. Business process assignment and execution from cloud to mobile. In *Business Process Management Workshops*, pages 264–276. Springer, 2014.

[68] E. Philips, R. Van Der Straeten, and V. Jonckers. NOW: Orchestrating services in a nomadic network using a dedicated workflow language. *Science of Computer Programming*, pages 1–27, November 2011.

[69] Google Play. https://play.google.com/store.

[70] Jon Postel. Internet protocol. 1981.

[71] Radu Prodan and Simon Ostermann. A survey and taxonomy of infrastructure as a service and web hosting cloud providers. In *Grid Computing, 2009 10th IEEE/ACM International Conference on*, pages 17–25. IEEE, 2009.

[72] Michael A Rappa. The utility business model and the future of computing services. *IBM Systems Journal*, 43(1):32–42, 2004.

[73] Katharina Rasch, Fei Li, Sanjin Sehic, Rassul Ayani, and Schahram Dustdar. Context-driven personalized service discovery in pervasive environments. *World Wide Web*, 14(4):295–319, 2011.

[74] Sylvia Ratnasamy, Paul Francis, Mark Handley, Richard Karp, and Scott Shenker. *A scalable content-addressable network*, volume 31. ACM, 2001.

[75] Lenin Ravindranath, Arvind Thiagarajan, Hari Balakrishnan, and Samuel Madden. Code in the air: simplifying sensing and coordination tasks on smartphones. In *Proceedings of the Twelfth Workshop on Mobile Computing Systems & Applications*, page 4. ACM, 2012.

[76] Maryam Razavian and Patricia Lago. A survey of soa migration in industry. In *Service-Oriented Computing*, pages 618–626. Springer, 2011.

[77] Azarias Reda, Brian Noble, and Yidnekachew Haile. Distributing private data in challenged network environments. WWW '10.

[78] Antony Rowstron and Peter Druschel. Pastry: Scalable, decentralized object location, and routing for large-scale peer-to-peer systems. In *Middleware 2001*, pages 329–350. Springer, 2001.

[79] Alessandro Russo, Massimo Mecella, and Massimiliano Leoni. Rome4eu–a service-oriented process-aware information system for mobile devices. *Software: Practice and Experience*, 2012.

[80] Mahadev Satyanarayanan. Pervasive computing: Vision and challenges. *Personal Communications, IEEE*, 8(4):10–17, 2001.

[81] Mahadev Satyanarayanan, Paramvir Bahl, Ramón Caceres, and Nigel Davies. The case for vm-based cloudlets in mobile computing. *Pervasive Computing, IEEE*, 8(4):14–23, 2009.

[82] Sigrid Schefer-Wenzl and Mark Strembeck. Modeling Context-Aware RBAC Models for Business Processes in Ubiquitous Computing Environments. In *Mobile, Ubiquitous, and Intelligent Computing (MUSIC), 2012 Third FTRA International Conference on*, pages 126–131. IEEE, 2012.

[83] Mohsen Sharifi, Somayeh Kafaie, and Omid Kashefi. A survey and taxonomy of cyber foraging of mobile devices. *Communications Surveys & Tutorials, IEEE*, 14(4):1232–1243, 2012.

[84] Muhammad Shiraz, Abdullah Gani, Rashid Hafeez Khokhar, and Rajkumar Buyya. A review on distributed application processing frameworks in smart mobile devices for mobile cloud computing. *Communications Surveys & Tutorials, IEEE*, 15(3):1294–1313, 2013.

[85] Signavio Process Editor. http://www.signavio.com/.

[86] SQLite database. http://www.sqlite.org/.

[87] Satish Narayana Srirama, Matthias Jarke, and Wolfgang Prinz. Mobile web service provisioning. In *Telecommunications, 2006. AICT-ICIW'06. International Conference on Internet and Web Applica-*

*tions and Services/Advanced International Conference on*, pages 120–120. IEEE, 2006.

[88] Ion Stoica, Robert Morris, David Liben-Nowell, David R Karger, M Frans Kaashoek, Frank Dabek, and Hari Balakrishnan. Chord: a scalable peer-to-peer lookup protocol for internet applications. *Networking, IEEE/ACM Transactions on*, 11(1):17–32, 2003.

[89] Jaime Teevan, Amy Karlson, Shahriyar Amini, AJ Brush, and John Krumm. Understanding the importance of location, time, and people in mobile local search behavior. In *Proceedings of the 13th International Conference on Human Computer Interaction with Mobile Devices and Services*, pages 77–80. ACM, 2011.

[90] Renne Tergujeff, Jyrki Haajanen, Juha Leppanen, and Santtu Toivonen. Mobile soa: Service orientation on lightweight mobile devices. In *Web Services, 2007. ICWS 2007. IEEE International Conference on*, pages 1224–1225. IEEE, 2007.

[91] Anna Trifonova and Marco Ronchetti. Hoarding content for mobile learning. *International Journal of Mobile Communications*, 4(4):459–476, 2006.

[92] Luis M Vaquero, Luis Rodero-Merino, Juan Caceres, and Maik Lindner. A break in the clouds: towards a cloud definition. *ACM SIGCOMM Computer Communication Review*, 39(1):50–55, 2008.

[93] Christopher N Ververidis and George C Polyzos. Service discovery for mobile ad hoc networks: a survey of issues and techniques. *Communications Surveys & Tutorials, IEEE*, 10(3):30–45, 2008.

[94] W3C. XML Path Language, 2010. http://www.w3.org/TR/xpath20/.

[95] Don Box; David Ehnebuske; Gopal Kakivaya; Andrew Layman; Noah Mendelsohn; Henrik Frystyk Nielsen; Satish Thatte; Dave Winer. Simple object access protocol (SOAP) 1.1, W3C note 08 may 2000. Available from `http://www.w3.org/TR/SOAP`.

[96] Christian Wolter and Andreas Schaad. Modeling of Task-Based Authorization Constraints in BPMN. In *Business Process Management*. 2007.

[97] Hirokazu Yoshinaga, Takeshi Tsuchiya, and Keiichi Koyanagi. Coordinator election using the object model in p2p networks. In *Agents and Peer-to-Peer Computing*, pages 161–172. Springer, 2005.

[98] Jin Yu, Boualem Benatallah, Fabio Casati, and Florian Daniel. Understanding mashup development. *Internet Computing, IEEE*, 12(5):44–52, 2008.

[99] Sonja Zaplata and Winfried Lamersdorf. Towards mobile process as a service. In *Proceedings of the 2010 ACM Symposium on Applied Computing*, pages 372–379, 2010.

[100] Xinwen Zhang, Anugeetha Kunjithapatham, Sangoh Jeong, and Simon Gibbs. Towards an elastic application model for augmenting the computing capabilities of mobile devices with cloud computing. *Mobile Networks and Applications*, 16(3):270–284, 2011.

[101] Ben Y Zhao, Ling Huang, Jeremy Stribling, Sean C Rhea, Anthony D Joseph, and John D Kubiatowicz. Tapestry: A resilient global-scale overlay for service deployment. *Selected Areas in Communications, IEEE Journal on*, 22(1):41–53, 2004.

[102] Ban Zhijie, Gu Zhimin, and J Yu. A survey of web prefetching. *Journal of computer research and development*, 2009.

# Appendix A

# List of Publication

1. Tao Peng, Giampaolo Armellin, Dario Betti, Annamaria Chiasera, Tefo James Toai, and Marco Ronchetti. Mdo: framework for context-aware process mobility in building-maintenance domain. In Software Maintenance and Reengineering (CSMR), 2013 17th European Conference on, pages 449–452. IEEE, 2013.

2. Tao Peng, Chi-Hung Chi, Annamaria Chiasera, Giampaolo Armellin, Marco Ronchetti, and Cristina Matteotti. Modeling and composition of environment-as-a-service. In Mobile Cloud Computing, Services, and Engineering (MobileCloud), 2014 2nd IEEE International Conference on, pages 250-251. IEEE, 2014.

3. Tao Peng, Marco Ronchetti, Jovan Stevovic, Annamaria Chiasera, and Giampaolo Armellin. Business process assignment and execution from cloud to mobile. In Business Process Management Workshops, pages 264-276. Springer, 2014.

4. Tao Peng, Chi-Hung Chi, Annamaria Chiasera, Giampaolo Armellin, Marco Ronchetti, Cristina Matteotti, Cristhian Parra, Aleksey Oleksiy Kashytsa, and Alessio Varalta. Business process assignment and execution in mobile environments. In Collaboration Technologies and

Systems (CTS), 2014 International Conference on, pages 267-274. IEEE, 2014.

5. Tao Peng, Marco Ronchetti, Annamaria Chiasera, and Giampaolo Armellin. Dynamic data prefetching for mobile workforces. In Mobile Web Information Systems Workshops. Springer, 2013.

6. Cristhian Parra, Tao Peng, Cristina Matteotti, Vincenzo Dndrea, Aleksey Kashytsa, and Davide Martintoni. Allenavita: Motivating lifestyle improvement in a telemedicine scenario. In Persuasive Technologies, page 53, 2014.