UNIVERSITY OF TARTU
FACULTY OF MATHEMATICS AND COMPUTER SCIENCE
Institute of Computer Science
Software Engineering Curriculum

Wei Ding

A Framework for Collaborative Content Mashup with Pervasive Services

Master's Thesis (30 ECTS)

Supervisor(s): Chii Chang
Satish Narayana Srirama

Tartu 2015

A Framework for Collaborative Content Mashup with Pervasive Services

Abstract:

By composing pervasive services, mobile phones can support various industrial and commercial needs. However, the pervasive services composition involves discovering and processing a large amount of data in order to identify and interpret the content. Due to the limitation of the single device capability, it is advisable to collaborate with other devices via a wireless network to accomplish common goals. In this thesis, we propose and develop a generic framework that supports service-oriented content mashup and integrating pervasive services composition in the Business Process Execution Language (BPEL)-based collaboration. A resource-aware offloading scheme to collaborative devices has been proposed and implemented as a proof of concept. The evaluation results have shown that the framework supports collaborative task-offloading scheme that reduces the resource usage of mobile devices.

Keywords:

Task Offload, Collaboration, Workflow, Pervasive Services

Kombineeritud teenustega koostööl põhineva sisu mashupi raamistik

Lhikokkuvte:

Kombineerides erinevaid teenuseid saavad mobiiltelefonid rahuldada paljusid tööstus ja ärivajadusi.Samas tuleb teenuste kombineerimise raames sisu õigesti tuvastamiseks ja tõlgendamiseks avastada ja töödelda suurt hulka andmeid. Kuna ainult ühe seadme kasutamine mingi ülesande lahendamiseks ei ole väga efektiivne on ühiste eesmärkide saavutamiseks soovitatav tööd mitme seadme vahel jagada. Pakume välja ja arendame üldraamistikku, mis toetab teenustele orienteeritud sisu segunemist ning laialt levinud teenuste loomise integreerimist, mis toimuks Business Process Execution Language (BPEL)-tuginevale kollaboratsioonile. Esitasime kollaboratsioonis seadmetele ressursisäästliku teisaldamise plaani ja implementeerimise selle proof of concept'ina (kontseptsiooni tõestus). Hinnangu tulemused näitavad, et raamistik toetab kollaboratiivset ülesannete teisaldamise kava, mis vähendab mobiilsete seadete ressursside kasutamist.

Märksnad:

Task teisaldada, koostöö, töökorraldus Läbivad teenused

Table of Contents

List of Figures

1 Introduction

1.1 Preamble

According to Moore's law, the transistors in electric circuits doubled every two years, and manufacturer cost is also lower (Lanter, 2013). The electronic devices are mass-produced at a low cost rate. As a consequence, people are exposed in an environment where many electronic devices are surrounding them, e.g. mobile phone, wireless sensors, actuators. With application logic embedded into those electronic devices, they are capable of fetching, analysing, recording various environmental information or spatial information or point of interest. By hosted Web servers (Srirama, Jarke, & Prinz, 2006), they can provide various services that interact with humans.

In the meantime, people use the mobile phone more to perform their daily activities. The evolved hardware and software change grants mobile phones not only allows for making a phone call, sending text messages but also capable of performing complex tasks as the computation power grows. The phones can play the role to communicate with the sensors in the pervasive services environment. There are potentially various sensors providing different kind of services. By composing pervasive services, mobile phones can support various industrial and commercial needs.

Mashup represents aggregating different existing services into one composite service towards providing customized service to fulfil the need of users. Content Mashup, which derived from a Web 2.0, is one of the standard approaches to realise service composition. It represents a content-driven approach that utilising the technology to fetch desired content from multiple content service providers and aggregate them together (O'reilly, 2007).

In the past, researchers used different approaches to performing the mashup in pervasive services environments. In (Chang, Srirama, & Ling, 2014), the authors propose a workflow-based SPiCa framework to enable content mashup in Mobile Social Network in Proximity (MSNP). In (Spiess et al., 2009) using Device Profile for Web Services (DPWS), which is a subset of standard Web service interface that was designed as a set of

guideline based on WS-* standard to enable interoperability among heterogeneous devices. In (Guinard, 2010), authors utilised a lightweight RESTful Web service to achieve service composition.

However, mashup in pervasive services environment face more challenge because of the environment consists of a large number of heterogeneous devices. Sensors' hardware are usually made by different manufacturers with different standard and characters. As a result, it is hard to discover and integrate with the service because different protocol used by the devices cannot communicate directly. The level of heterogeneous become even complicated when assemble the chips into a diverse device like actuator nodes. The content mashup will face overhead, high latency issues in the service discovery phase (Gama, Touseau, & Donsez, 2012). The content mashup in pervasive services composition involves discovering and processing a large amount of data in order to identify and interpret the content. It will face the challenge as following describes.

1.2 Motivation

In order to clarify the main objectives of this research, we use the following scenarios to describe the motivation of the topic.

1.2.1 Scenario

Zhang is driving to Beijing with his friends Li and Wang during the Golden Week Holiday (Zhang, Song, & Qin, 2008). However, it is hard to find a parking lot during such hot holiday season. Zhang wants to find a parking lot quickly with his smartphone. In Beijing, there are numerous closed-circuit television camera that are available for public to access (Klein, 2008). However, they have different kinds of usage; some of them provide services of the real-time telescope to view tourist attraction spot like Forbidden City, Triple of Heaven. There are cameras that provide service for checking the parking lot status. However, those cameras belong to different organizations. Some of them belong to different department stores to monitoring their parking lot status. They are different in both syntax and semantics (the different organization has their rules). Others are observing the street conditions but provide service to monitor the street parking lot status as well. However, the question is:" how does Zhang can rapidly find the parking lot information with such a crowded environment?" When it has a large number of information providers, Zhang may discover too much unwanted information. It is very challenge for Zhang to

identify and filter the information by using his device due to the resource-constrained issues. Why not delegate the task to his friend Wang and Li if their devices are idle as described in Figure 1.1. This is assuming they are friends, and they have installed the compatible standard-based Web service-oriented applications to perform the tasks.

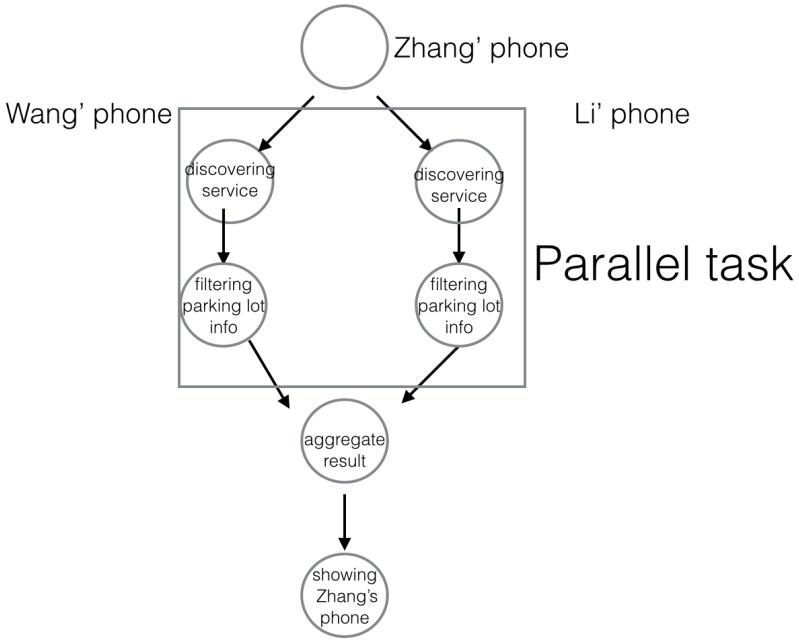

Figure 1.1 The workflow describes offloading the tasks to friends

By collaborating with other mobile phones, it extends Zhang's phone capability to discovering the service and filtering the information he wants. However, because different mobile phones have different capabilities, the tasks cannot be randomly delegated. Hence, it requires a feasible approach to identify how the task delegation process can be performed based on the phone's capabilities in order to take full advantage of their performance.

1.3 Research Challenge

One way to establish a composite service is that utilizes external computational resources to perform the tasks (Chen et al., 2003). There are typically two ways to achieve it.

1. Offloading tasks to external cloud service like Google App Engine, Amazon Cloud (Dornemann, Juhnke, & Freisleben, 2009).
2. Offloading task in a group of mobile devices that belong to friends or a same community or even belong to the same user (Chang et al., 2014).

In approach (1), it usually requires extra cost for using external cloud services, and it requires additional mobile network bandwidth as well. However in approach (2), it is less cost but it is assuming everyone has installed the compatible applications. A standalone approach such as (Bottaro, Gérodolle, & Lalanda, 2007) was built on top of OSGi. The framework maintains a list of available services and enables service composition at runtime. In such a design, developers have to implement and maintain the application for all platforms (e.g. Android, iOS, Windows Phone, Firefox OS etc.), which is less flexible and costly for development.

Although there were many mobile workflow engines had been proposed in last decade, they are complex, difficult to extent, no support for RESTful services and no support for CoAP service interaction (Dar, Taherkordi, Baraki, Eliassen, & Geihs, 2014; Kim, Lee, Kim, Park, & La, 2014). Additionally, they did not support context-awareness when performing the task allocation.

1.4 Research Objective and Contributions

- To investigate, develop and validate a solution for a content mashup in the pervasive environment by enabling service-oriented service composition among mobile resources.
- To investigate, develop and validate an approach to support task offloading in content mashup in a collaborative environment
- To investigate, develop and validate a lightweight service-oriented workflow engine for a mobile device that can achieve above two mechanisms.

To accomplish the objectives, we propose a framework for collaborative content mashup with pervasive services, which supports the following features:

- *Content-aware service discovery*

  In order to achieve the content mashup in the pervasive services environment, the framework support content-aware service discovery based on user defined ontology. In another word, the framework can proactive discover and filter the desired service providers based on user preference in a large number of information providers.

- *Decentralisation*

The framework supports decentralized service composition that consists of collaborative devices discovery and interaction without relying on stationary mediators.

- *Energy saving*

The framework supports collaborative task-offloading scheme that reduces the resource usage of mobile devices. Also, the framework supports CoAP protocol in the constrained pervasive services environment.

1.5 Research Scope

This thesis focuses on developing a generic framework that supports service-oriented content mashup and integrating pervasive services composition in the Business Process Execution Language (BPEL)-based collaboration. The proposed framework is implemented and evaluated on real mobile devices: Nexus 5, LG G3, and Nexus 7. The pervasive services environment is based on simulation, in which a number of different Web services are hosted in the mobile phone.

The service composition in pervasive services environment needs to consider the privacy and security problem. However, they are not in the scope of this thesis.

1.6 Thesis Outline

This thesis organized as follow:

Chapter 2 discusses some related background information like workflow system and service composition, context-ware workflow system and related technologies to understand pervasive services composition in BPEL-related technologies.

Chapter 3 introduces our proposed framework for enabling Collaborative Content Mashup with Pervasive Services and how to make a decision on collaborates task scheduling.

Chapter 4 describes the implement detail of the proposed the framework.

Chapter 5 evaluates the advantage of the framework in terms of performance and energy consumption.

Chapter 6 concludes the research project and the direction of future research.

2 Background

The content mashup in pervasive services environment needs a loosely couple standard to enhance the interoperability. Interoperability can be classified into two types: physical interoperability and logical interoperability (Chang, 2013). Physical interoperability means the nodes, which connect with each other via a wireless network, are able to exchange data under physical network layer. Logical interoperability means the nodes understand what kind of service other nodes provides. When a node needs to autonomously discover a particular content provided by the other nodes in an environment that consists of hundreds of nodes, the system needs to support a machine-readable way to enable the autonomous content discovery and filtering. In (Srirama et al., 2006), authors used a mobile peer-to-peer network to support physical interoperability and using a set of Web services standards to enable logical interoperability. The content mashup in pervasive services requires dealing with four challenges: service discovering and selection, scalability, fault tolerance and flexible (Bakhouya & Gaber, n.d.). This chapter provides the related background to understand content mashup with pervasive services.

2.1 Overview of Web Services Composition

Web service composition is the process by composing multiple services into one complex service to create the value-added application. In the subsection, we identify some basic concept and terminologies related to Web service composition.

2.1.1 Web Service

Web service was designed to enable interoperability among machine-to-machine in a connected network. In a classic design, Web service is described using Web Services Description Language (WSDL) for describing network services. There are mainly two ways to enable Web service: Simple Object Access Protocol (SOAP)-based and Representational State Transfer (REST)-based.

2.1.2 SOAP-based Web Service and Restful Web Service

SOAP stands for Simple Object Access Protocol. It uses Extensible Markup Language (XML) for messaging exchange and can be transmitted in HTTP, SMTP, or other protocols. It consists of an envelope which includes header and body and a set of encoding rules (Hirsch, Kemp, & Ilkka, 2007).

REST was created by Roy Fielding in his Ph.D. thesis (Fielding, 2000). In the RESTful Web service, the resource is manipulated based on Uniforms Resource Identifiers (URI) that uses stateless communication protocol, typically HTTP method: GET, POST, PUT, and DELETE. The data format used in REST can be XML or JSON for sending and receiving data.

2.1.3 Semantic Web Service

Before composing any Web service, firstly it requires discovering the service which providing the functionality that match the requirement. To be more specific, the service discovery needs to find all the service that matches the desired operation in the pervasive services environment. To fulfil this requirement, semantic Web for describing the functionality of Web service is widely used. Several XML-based standards have been introduced to enable semantic for service discovery: Resource Description Framework (RDF), Web ontology language (OWL), Semantic Annotation for WSDL and XML schema (SAWSDL) (Kopecky, Vitvar, Bournez, & Farrell, 2007).

World Wide Web Consortium (W3C) has introduced RDF, which is an XML-based metadata for describing Web resources. The RDF terminology *triples* utilises the form of subject-predicate-object expression to describe entity relationship including subject and object represent the resource, and the predicate denotes the relationship between the resources. List 2.1 illustrates an example of RDF document.

List 2.1 an example of RDF

```
<?xml version="1.0"?>
<rdf:RDF  xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-
ns#"xmlns:university="http://purl.org/dc/elements/1.1/univer
sity">
<rdf:Description
```

```
rdf:about="  http://purl.org/dc/elements/1.1/university/tartu
university">
    < university:rank>Top 400</university:rank>
    < university:location>Tartu</university:location>
</rdf:Description>
</rdf:RDF>
```

Because RDF is XML based, the first line is the XML declaration. Following by <rdf:RDF> tag that represents the XML is a RDF document, and it contains the RDF namespace. <rdf:Description> contains the particular resource as well as the attributes the resource contains. Usually, the attributes are defined in the global ontology. In this case, rank, location is defined in the http://purl.org/dc/elements/1.1/university namespace. In this example, it describes two triple relationships. “Tartu University ranks top 400” and “Tartu University location in Tartu.” in which the subject is “Tartu University”, the predication are “rank” and “location”, and the object are “Tartu” and “Top 400”.

OWL was introduced for machines to process and to interpret the Web content by defining additional vocabulary (nouns and verbs) instead of just presenting content to users. The nouns represent the classes of objects, and the verbs represent the relation between the objects. It is built on top of RDF to support larger vocabulary with greater machine interpretability than RDF. On the other words, OWL extends the RDF for representing the relationships to support for cardinality, disjointness and symmetry.

2.1.4 Web Services Composition

There are several approaches to realise Web service composition. In general, The Web service composition can be classified into two categories: orchestration and choreography (Peltz, 2003). Further, the approach can be autonomous or manual service composition.

Service orchestration utilises predetermined business logic and execution order to composing multiple services into one complex service. This is achieved with a central messaging engine (See Figure 2.1). The central engine coordinates the flow control, business logic. A common approach is to utilise a service-oriented workflow engine (e.g. Business Process Execution Language for Web services [BPEL4WS] engine) to enable the composition.

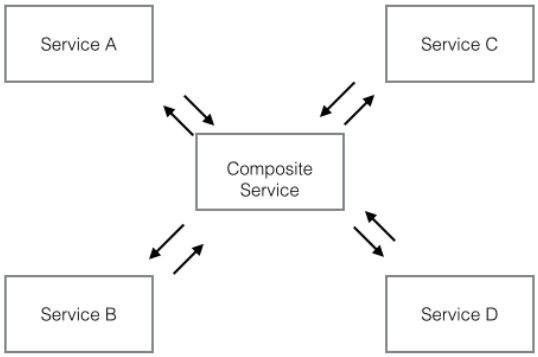

Figure 2.1 Service orchestration

Service choreography does not require a central controller. By allowing each participant message exchange to other participants, each participative service knows the business logic and rules of interaction according to the behaviour of other participants (See Figure 2.2).

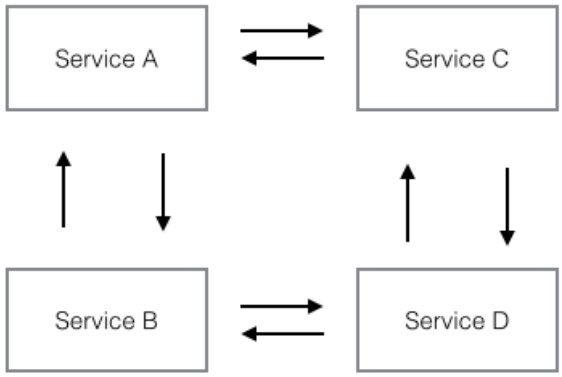

Figure 2.2 Service Choreography

Web service composition can be divided into two categories (static or dynamic) based on the time when the services are proposed. The static service composition means choosing the service and composing them at design time. This is assuming that the business logic and process usually fixed and can be predicted at design time. It does not fit when the requirement of service frequent change. Static service composition is not flexible enough to support service change at the run time when the binding service becomes unavailable.

Dynamic service composition is more flexible that it supports discovery, selection, binding the service at run time. However, it is very difficult to implement dynamic service composition that many factors need to take into consideration, fault tolerance, correctness (Sheng & others, 2006).

Web service composition can be achieved with three approaches, manual, automated, semi-automated. The manual composition is a tedious and error-prone procedure. It requires designer manually binds the service and composing the services. However, composing services usually involve complex business process, which may contain multiple tasks and interact between those tasks. The manual composition is time-consuming and not always meets the requirement. Automated composition take advantage of the semantic Web (Berners-Lee, Hendler, Lassila, & others, 2001). Based on the user specification, the automated composition will select services based on the semantic and compose the services automatically. However, because of Web service cannot full understanding Web semantics, which will affect the automatic selection of the services. Fully automated service is very hard to achieve (Berardi, n.d. , 2005).

2.1.5 Service Composition With WS-BPEL

WS-BPEL, also known as BPEL, is an XML-based language that enable Web services exchange message in a service-oriented architecture (SOA) which specifying business process for data flow. In each BPEL process file, all the composition services and business process are defined by *process* tag. Inside *process* tag, the participating services are defined in the *partnerlink* tag. A process contains a set of activities that define the business process for service composition. It identifies three primary activities that enable message interaction with Web service.

- **Receive**. The purpose of receive is receiving a message from Web services. Usually, it represents a variable that hold the reply data from the Web service
- **Invoke.** The purpose of invoke is used for call a Web service.
- **Reply.** The purpose of reply is used in conjunction with the previous receive activity that send the reply message to the previous Web service.

With the combination of above activities, it can support both synchronous and asynchronous message exchange of Web service.

For example, List 2.2 shows the asynchronous invoke of Web service

List 2.2 Asynchronous invoke

```
<invoke name="invokeSemanticMatching"
     partnerLink="matchingSemantic"
     inputVariable="ontologyData"
     outputVariable="mathcingResult"/>
```

It is blocks until it receive a response from Web service. This invokes contains four attributes. The name is important which used for identify this particular activity in the process. The *partnerLink* defines that Web service to interact and BPEL engine use this name to identify the Web service for actual invoke. The *inputVariable* define the data that to be sent to the Web service. The *outptutVariable* define the synchronous behaviour that the invoke blocks until it gets reply data from the Web service. If the *outputVariable* omitted, invoke activity is blocked because it does not expect the reply message from Web service.

BPEL also defines the structure execution order of the composing activities. It can be a sequence or parallel execution flow.

The *sequence* defines the sequential order of the activities. The activities are executed in a pre-defined order. List 2.3 shows an example of *sequence* process.

List 2.3 An example of sequence process

```
<sequence>
     <invoke name="CoapServiceDiscovery"
          partnerLink="getWellKnowInCoap"
          operation=".well-known/core"
          outputVariable="coapServiceResponse"/>
     <assign name="assign">
          <copy>
               <from variable="coapServiceResponse" />
               <to variable="postData" />
          </copy>
     </assign>
     <invoke name="invokeMatchingCoap"
          partnerLink="matchingCoap"
          operation="POST"
          inputVariable="postData"
          outputVariable="mathcingResult"/>
</sequence>
```

We can use a directed, acyclic graph to show the representation of the above sequence process (See Figure 2.3). The edges show the execution flow of the connected nodes. Each node represents one activity.

Figure 2.3 Acyclic graph represents sequence process

The *flow* activity defines parallel tasks. Inside the *flow* activity usually consist two or more sequence activities. Those sequence activities do not depend on each other and run asynchronously. This enables support for complex concurrency composition scenarios. List 2.4 shows an example of *flow* activity with two parallel sequence tasks inside.

List 2.4 Flow activity with two parallel sequence tasks

```
<flow>
     <sequence>
          <invoke name="getData1"
          partnerLink="getDataPL"
          outputVariable="output1"/>
          <assign name="assign1">
               <copy>
                    <from variable="output1" />
                    <to variable="copy1" />
               </copy>
          </assign>
          <invoke name="postData1"
          partnerLink="postDataPL"
          inputVariable="copy1"
          outputVariable="POSTOutput1"/>
     </sequence>
     <sequence>
          <invoke name="getData2"
          partnerLink="getDataPL"
          outputVariable="output2"/>
          <assign name="assign2">
               <copy>
                    <from variable="output2" />
                    <to variable="copy2" />
               </copy>
          </assign>
          <invoke name="postData2"
          partnerLink="postDataPL"
```

```
            inputVariable="copy2"
            outputVariable="POSTOutput2"/>
        </sequence>
</flow>
```

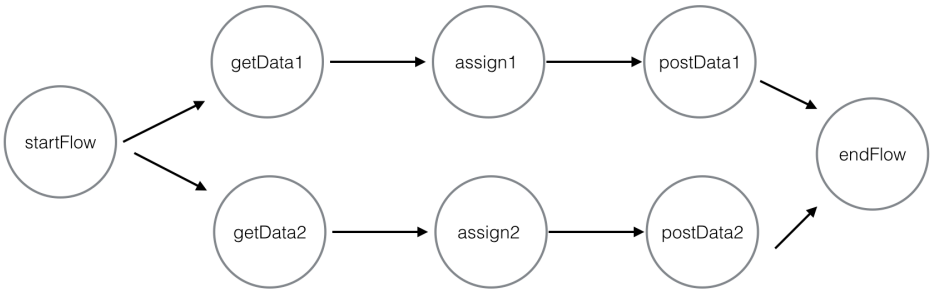

Figure 2.4 Acyclic graph represents flow activity

In the graph above, the *startFlow* activity follows two sequence activities. That two sequence activity does not depend on each other. The *endFlow* activity will be executed after two sequence activities finish.

2.1.6 Service Composition with Semantic Web (OWL-S)

The semantic Web provides a set of machine-readable ontologies that makes Web resources accessible by content or keyword. OWL-S consists of three upper ontologies (See Figure 2.5).

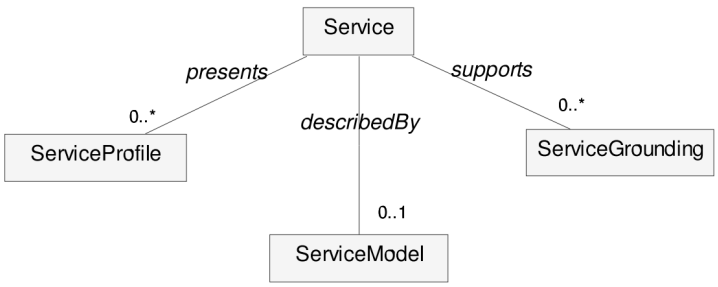

Figure 2.5 OWL-S consists of three upper ontologies

- **Service profile.** Providing the information about what kind of input expected from users and presenting all the information for service discovery. It contains functional properties (input, output, and preconditions) and non-functional properties (Quality of Service parameters).
- **Process model.** OWL-S defines three type of process: atomic, simple, composite. The Atomic process does not contain sub process and can be directly invoked. A simple process cannot be directly invoked and does not provide binding to any service thus it used as an abstraction for service or process. The composite process contains sub process and can define complex workflows.
- **Grounding.** Define how to access and use the service.

List 2.5 An example of OWL-S

```
<rdf:RDF
xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
xmlns:rdfs="http://www.w3.org/2000/01/rdf-schema#"
xmlns:owl="http://www.w3.org/2002/07/owl#"
xmlns:dc="http://purl.org/dc/elements/1.1/">
<owl:Ontology
rdf:about="http://www.ontology.com/temperature">
     <dc:title>The temperature Ontology</dc:title>
     <dc:description>An example for describe the temperature
     </dc:description>
</owl:Ontology>
<owl:Class
rdf:about="http://www.ontology.com/temperature#temperatureTy
pe">
     <rdfs:label>The temperature type</rdfs:label>
     <rdfs:comment>The class of temperature types.
     </rdfs:comment>
</owl:Class>
</rdf:RDF>
```

2.2 CoAP Protocol In Constrained Pervasive Services Environment

In the pervasive services environment, all kind of sensors or actuators are expected to access and interact in everywhere and anytime (Palattella et al., 2013). However, the manufacture is more concerned about the reduce the cost and energy consumption of the device instead of purely increase the computation power nowadays (CoAP, 2012). As a consequence, the traditional communication protocol (HTTP etc.) does not suitable in this constrained environment. CoAP, as a new protocol, was designed to enable communication between small low power sensors, actuator through standard Internet network. CoAP especially meet the requirement for low overhead and support RESTful operations (GET,

PUT, POST, DELETE). In addition, CoAP provides additional functionality for discovery new service (provide a `well-known` URI).

2.2.1 CoAP Message Format

Similar to client/server model of HTTP, CoAP using a binary-based header format to represent the request (client) and response (server) message. However, unlike HTTP that is based on Transmission Control Protocol (TCP), the CoAP is based on User Datagram Protocol (UDP). Therefore, it uses additional message layer to guarantee reliability. They are four type of message: confirmable, non-confirmable, acknowledgment, reset.

CoAP messages (See Figure 2.6) are encoded in a binary format. The message contains a 4-byte header, 0-8 byte long token value, a sequence of zero or more CoAP options, the rest followed by a payload (Shelby, Hartke, & Bormann, 2014).

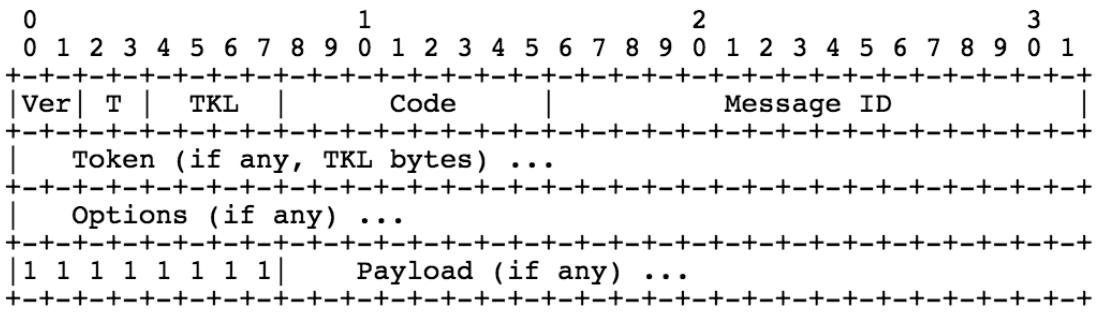

Figure 2.6 CoAP message format

- **Version** represents CoAP version number in a 2-bit length and by default it must be set to 1. Other value are reserved for future use.
- **Type** represents the four type of message in a 2-bit length also. 0 for confirmable. 1 for non-confirmable. 2 for acknowledgment. 3 for reset.
- **Token length** represents the length of the following token length in a 4-bit length
- **Code** is 8-bit length long and divides into two categories: 3-bit in most significant bits represent the class. The class can further denote request message (0), a success response (2), an error occurs in client response (4), an error occurs in server response (5). Other values are reserved for future use. The remaining 5-bit represents detail.
- **Message ID** is 16-bit length long that representing identify for the message to detect message duplication.

2.2.2 Method Definition

Similar to HTTP method, CoAP support GET, POST, PUT, DELETE operation. GET retrieve the resource identified by request URI. POST usually defines the operation to be processed for the entity enclosed in the client request. A typically use the POST request is for create a new resource on the server. PUT is used for update resource identified by request URI, and DELETE is for delete resource identified by request URI.

2.2.3 CoAP Discovery

The client has to have a URI for discovery CoAP server in order to discovering the services on the server. The default port number for CoAP server is 5683, and this port number must be supported by CoAP server for resource discovery. In order to maximize interoperability in pervasive services scenario, a CoAP service should include CoRE link format to describe hosted resources, their attributes for the machine to interpret the services. By default, a `well-known` URI "/.well-known/core" is the entry point for requesting the links hosted by the server and performing CoRE Resource Discovery (Shelby, 2012).

CoRE link format also provides a set of attributes that provides information of the target link. The "rt" (resource type) attributes can be used to describe the semantic of a target resource. In case of humidity resource, The "rt" attributes could provide semantic type link "indoor-humidity" or a URI referencing a global ontology that defines the concepts and relationships like "http://www.ontology.com/phys.owl/#Humidity"

For example, the CoRE link format can be used in building automation environment that enable the client to find and interact with humidity sensors without human intervention. The resource discovery can begin either unicast or multicast. If the client already knew a particular server IP through Domain Name System (DNS), the client wants to know whether this server provides the humidity information. The client can issue a request to the entry point "/.well-known/core" on the server. Then the client can match the appropriate resource type, description for locate temperature information through the server response. List 2.6 shows an example of CoAP request and response.

List 2.6 CoAP request and response

```
REQ: GET /.well-known/core
RES: 2.05 Content
</sensors/light>;if="sensor"
</sensors/humidity>;rt="http://www.ontology.com/phys.owl/#Hu
midity"
```

This example shows the server provide light and Humidity information. It also provides humidity with additional ontology defined in “rt” attribute that let the client know what specific humidity information it provides.

2.3 Fuzzy Logic

Fuzzy logic (Klir & Yuan, 1995) has been applied to many fields, from industrial process control to artificial intelligence in the past decade. Unlike traditional binary logic where variables only permit propositions having true or false values, the fuzzy logic variable allows partial truth, that ranges in degree between completely true and completely false (Perfilieva & Močko\vr, 1999). Fuzzy logic makes a decision closer to human thinking. It is suitable for decision-making application in which using classical control strategies are difficult to have a mathematical model.

If categorizing a person based on the height, the criteria define a tall person whose height is above 170 cm. In the traditional set definition, a person whose height is 171 cm regards as tall. However, another person whose height is 169 cm does not regard as tall. In reality, a person whose height is 169 cm is no much different with a person whose height is 171 cm. In the fuzzy set, it can easily solve the problem with a degree range.

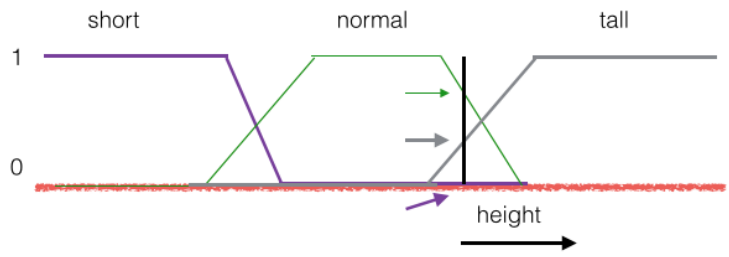

Figure 2.7 Mapping height in fuzzy logic

As shown in the picture above (See Figure 2.7), the meanings of the expressions short, normal, and tall are represented by functions mapping a height scale. A point has three "true values" based on three membership function. The vertical line denotes a particular person that three true values gauge. Because the purple arrow maps to 0, this person can regard as not short at all. The green arrow pointing at 0.8 could describe "fairly normal height" and the grey arrow pointing at 0.2 may describe slight tall.

Fuzzy Logic Controller (FLC), proposed by Zadeh (Zadeh, 1965), is based on the fuzzy logic. It consists of three parts: input, inference processing, output. The input are a set of truth values in the Knowledge Base ranges from 0 to 1 representing degrees of membership in the set. The Knowledge Base denotes a combination of expert knowledge: data base knowledge and rule base knowledge. Data base knowledge defines a linguistic term that is membership function for fuzzy sets. The rule base knowledge is consists of a set of fuzzy control rules. The inference processing using each rule in the form of IF-THEN statement and transform the crisp value of the input into fuzzy sets (also known as the Fuzzification), and then perform the reasoning process. The output combined the result generate each rule and map it into crisp values for the control variables (Cingolani & Alcalá-Fdez, 2013).

The Fuzzification works as follows:

$$A = F(i)$$

Where, i is a crisp value defined in the input variable set. A is a fuzzy set defined in the same universe as input variable set. F denotes the fuzzifier operator (Cingolani & Alcalá-Fdez, 2013).

Fundamentally, the inference processing is the process for fuzzy implication function. Generalized modus tollens is the fuzzy implication inference rules in approximate reasoning in which:

Premise 1: y is B'

Premise 2: if x is A then y is B

Consequence: x is A'

Where x and y are linguistic variables, A and B are fuzzy sets, B' is the inferred fuzzy set.

2.4 Comparison of Existing Works

2.4.1 A REST Architecture for Interconnecting Business Processes with IoT Resources

(Dar et al., 2014) authors propose an architectural model which using the resource-oriented approach for designing that integrating Internet of Things (IoT) service with enterprise level Business Process (BPs). HTTP REST and CoAP were used for communication protocol for IoT service to be integrated into BPs. They use Contiki-based Tmote Sky[1] for supporting REST in IoT device. WSDL and WADL were used for service description that describe what methods provided and in which way to invoke the service. BPMN was used to compose those smart objects. In order to find the available service, it needs to search available service using JUDDI (service discovery protocol) to fetch WADL to find available service. Activity BPMN execution engine was used to support disturbed business process execution in Android phone.

The advantage of the above approach is the application developer can easily integrate IoT services using drag and drop fashion with little understanding the underlying technologies.

2.4.2 A Notation for Representing the Behaviour of Things to Enable Complex Mashups

Devices Profile for Web Service (DPWS) takes full advantage of Web services that allows integration heterogeneous device provided certain service seamlessly. In (Cubo, Brogi, & Pimentel, 2012) authors made a proposal to extend the DPWS specification by introducing new tags in the WSDL file to add a set of behavior "constraints" to automate the behavior of things properly composed at run time without human involved.

[1] http://www.eecs.harvard.edu/~konrad/projects/shimmer/references/tmote-sky-datasheet.pdf

2.4.3 Web Mashups for Embedded Devices with RESTful Resources

In (Guinard & Trifa, 2009), the authors propose two ways to integrate IoT devices into RESTful resources that addressed over HTTP. If the device is resource constrained that not capable to access through IP. Then authors propose a gateway that hides the commutation detail with the device and providing RESTful information related to the device. By this way, the gateway can orchestrate the composition those services with more functionalities. For example, it can get the battery consumption by all the devices connected to the gateway. The second way directly turn the device into RESTful service. The prototype was build based on Sun SPOT RESTful API.

2.4.4 Smart Objects as Building Blocks for the Internet of Things

In (Kortuem, Kawsar, Fitton, & Sundramoorthy, 2010), authors identify the smart object as three categories used for a building block of Internet of things. They are activity-aware, policy-aware, process-aware smart objects. Awareness, Representation, Interaction is the three criteria that identify the smart object in the proposed categories. In summary, the activity-aware smart object only stores activity record and using recognition algorithms to detect activity. It is the simplest type that doesn't interact with users. The policy-aware smart object can consist predefined policy that interprets activity accordingly. It provides interactive action that alert users if they violate policies. It is like activity model with additional policy integrated. The process-aware smart object understands the real-world scenario. It consists of context-ware workflow model that integrates company's workflow process. More specifically, it knows what users suppose to do now and what activity ought to be done next.

2.4.5 Summarize and Compares the Technologies in Related Work

Title	Lower Layer	Centralized	App Layer	Technology	Messaging	Composition	Application
A resource oriented integration architecture for the Internet of Things: A business process perspective (Dar et al., 2014)	Bluetooth author mention "It should be noted that due to the lack of Zigbee-based communication in most of the smartphones."	Decentralized(choreography)	HTTP AND CoAP(REST)	WSs using Contiki-based and Tmote Sky nodes CoAp using Erbium	JSON	BPMN (mention it supports orchestration, as well as choreography) it supports a larger set of workflow patterns and events, has a standardized diagram interchange format, and provides user tasks for human interaction	Use case: Ambient Assisted Living Help elderly people to live independently at home with bio-medical devices attached to measure essential signs of the patient
Towards Physical Mashups in the Web of Things (Guinard, Trifa, Pham, & Liechti, 2009)	n/a	N/A	HTTP(REST)		JSON	N/A	demonstrate a sensor that provide temperature status and integrate it into ERP through SAP MII
Multidimensional reputation network for service composition in the internet of things (Bossi, Braghin, & Trombetta, 2014)	n/a	N/A	N/A	N/A	N/A	N/A	N/A
Towards Behavior-Aware Compositions of Things in the Future Internet (Cubo et al., 2012)	ipv4/ipv6/ip multicast	N/A	HTTP	DPWS	N/A	By introducing new tags (constraints) in WSDL file to ensure the execution sequence	airport surveillance system
Towards the Web of Things: Web Mashups for Embedded Devices (Guinard & Trifa, 2009)	Bluetooth	Decentralized	HTTP	Wireless sensor network	JSON	N/A	monitoring battery consumption
Smart Objects as Building Blocks for the Internet of Things (Kortuem et al., 2010)	N/A	Decentralized(p2p)	N/A	N/A	N/A	Ad Hoc	Road-patching at road construction site

3 System Design

3.1 A Framework for Collaborative Content Mashup with Pervasive Services

3.1.1 Framework Requirement

The proposed framework consists of following mechanisms in order to realise the run-time collaborative content mashup with pervasive services.

- **Workflow execution** – the framework is able to parse and to execute the predefined BPEL-based workflow.
- **Collecting resource and context information –** In order to decide whether or not the workflow tasks should be performed solely or be performed collaboratively, the framework needs to be capable of collecting relevant resource and context information such as CPU usage, battery status and network condition from the collaborative mobile devices.
- **Decision-making –** Once the relevant resource and context information are collected, the framework should be able to decide whether to partition and offload the tasks to other mobile devices or run it locally.
- **Enable collaboration –** The framework should be able to modify the workflow and generate new workflow in order to partition the tasks to the collaborative mobile devices.

3.1.2 Architecture Overview

Figure 3.1 shows the high-level architecture of the proposed framework.

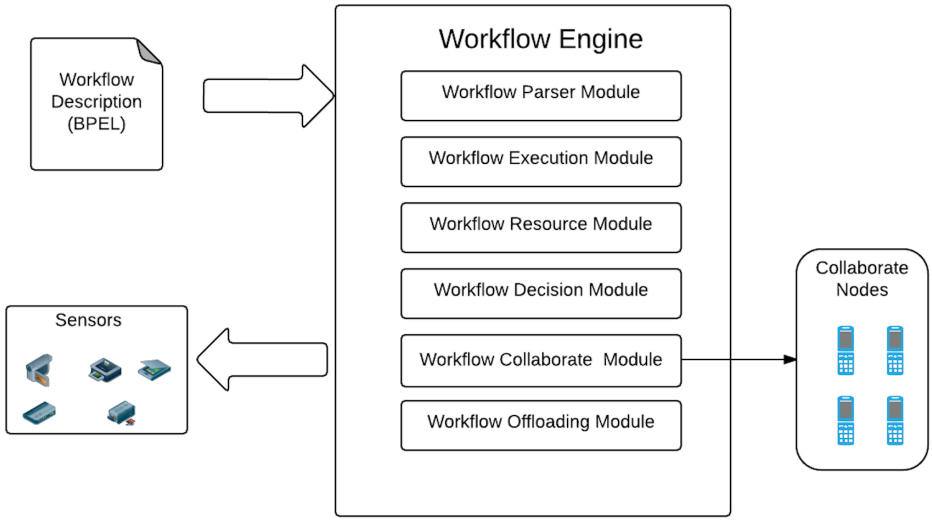

Figure 3.1 The high-level architecture of the proposed framework

This framework consists of the following components:

- **Workflow Description** defines the workflow process. Specifically, it describes the automatic selection, composition services, sequence or parallel execution and synchronization or asynchronization mechanisms.
- **Workflow Parser Module** reads the structure of the BPEL process as a directed graph and stores all the BPEL-related information (`partnerlink`, `variable`) into the memory.
- **Workflow Execution Module** plays an important role in this framework. It will interactive with external services based on the tasks defined in the workflow description.
- **Workflow Resource Module** provides hardware information of the mobile device. For instance, CPU usage, battery status and network condition.

- **Workflow Decision Module** makes the decision based on the information get from *Workflow Resource Module* to decide whether to offload the current task or partition the following parallel tasks to collaborative devices.

- **Workflow Collaborate Module** maintains a list of collaborative devices, as well as its status such as CPU, RAM, and battery condition.

- **Workflow Offloading Module** generates new BPEL file based on the decision made by *Workflow Decision Module* and modify the existing BPEL file to enable collaboration with the other devices.

3.2 Decision on Collaborates Task Scheduling

The key contribution of this thesis is designing task delegation model for collaborative content mashup in pervasive services scenario, which is achieved by the BPEL-based workflow controlled resource-aware task-scheduling scheme. Since this framework is a BPEL workflow, we can use Business Process Model and Notation (BPMN) (White, 2004) to describe the BPEL workflow models (Ouyang, Dumas, Van Der Aalst, Ter Hofstede, & Mendling, 2009). The task delegation model is based on two types of BPEL workflow, sequential task delegation, and parallel task delegation.

3.2.1 Fuzzy Logic for Decision Making

Applying fuzzy logic in a mobile environment for decision-making has several advantages. Firstly, it can generate the decision result without limiting the input variables. Any input variable that provide an indication of the system's action is sufficient, and the input variables does not necessary to be precise and noise-free. Moreover, it is flexible and extensible by letting users freely add/remove fuzzy logic rules or even input variables. The output control is stable even with a wide range of input variables.

In this thesis, we will use JfuzzyLogic (Cingolani & Alcalá-Fdez, 2013) for making a decision whether to offload certain activity to external collaborative device In order to design a fuzzy control system for our decision-making scheme, the first step is to define the input and the output variables.

We define the input variables as following:

- Battery
- CPU
- RAM
- Bandwidth

Input variables are defined under VAR_INPUT section as following shows.

List 3.1 Define input variables

```
// Define input variables
VAR_INPUT
     CPU : REAL;
     BATTERY : REAL;
     RAM:REAL;
     BANDWIDTH : REAL;
END_VAR
```

The output variable of the decision-making system is to decide whether to offload or not. We define it as “decision”. Output variables are defined under VAR_OUTPUT sections.

List 3.2 Define output variable

```
// Define output variable
VAR_OUTPUT
     decision : REAL;
END_VAR
```

Fuzzy sets are divided into two categories, FUZZIFY and DEFUZZIFY. FUZZIFY defines a linguistic term and its corresponding membership function for each input variable. We use different membership functions for each input variables. For example, we define CPU, RAM and Bandwidth in triangular function. However, we define battery function in piece-wise linear functions. Because the mobile phones commonly use lithium-ion batteries. The cell voltage of the lithium ion chemistries discharge curve looks like below.

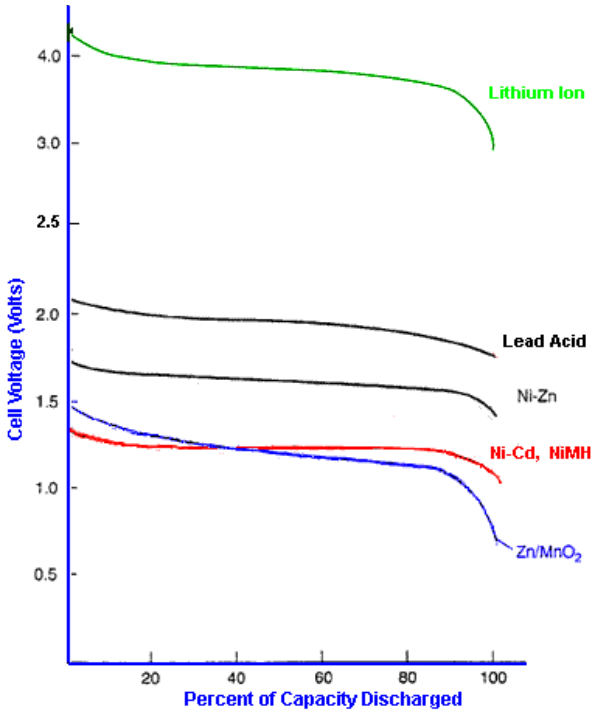

Figure 3.2 Cell voltage of discharge curve[2]

Figure 3.2 shows that when the battery level is less than 20%, the cell voltage drops dramatically. We regard the battery in absolute poor condition when they are less than 20%. Therefore, the piece-wise linear function defines the level of battery from 0%-20% in an absolute poor condition where the function maps to 1 in the y-axis (List 3.3).

[2] http://www.mpoweruk.com/performance.htm

List 3.3 Define FUZZIFY function

```
// Fuzzify input variable 'CPU': {'low', 'medium' ,  'high'}
FUZZIFY CPU
     TERM low := (0, 1) (40, 0) ;
     TERM medium := (30, 0) (50,1) (70,0);
     TERM high := (60, 0) (100, 1);
END_FUZZIFY
// Fuzzify input variable 'RAM': {'low', 'medium' ,  'high'}
FUZZIFY RAM
     TERM low := (0, 1) (40, 0) ;
     TERM medium := (30, 0) (50,1) (70,0);
     TERM high := (60, 0) (100, 1);
END_FUZZIFY
// Fuzzify input variable 'BATTERY': { 'poor', 'excellent' }
FUZZIFY BATTERY
     TERM poor := (0, 1) (20, 1) (70,0) ;
     TERM excellent := (30,0) (80,1) (100,1);
// Fuzzify input variable ' BANDWIDTH ': { 'poor', 'fast',
'excellent' }
FUZZIFY BANDWIDTH
     TERM poor := (0, 1) (8,0);
            TERM fast := (6,0) (10,1) (14,0)
     TERM excellent := (12,0) (20,1) ;
END_FUZZIFY
```

We use three terms “high”, “low” and “medium” to indicate RAM and CPU usage. If the RAM is less than 30%, it means *in low usage*. We use the term “poor” and “excellent” to describe the battery usage. If it is less than 20%, it means the battery is in absolute poor condition. If it is in 60%, it means the battery is in fairly excellent condition. The 60% mapping to piece-wise linear functions we defined, we can see 0.75 points to the excellent category and 0.25 point to the poor category from the y-axis.

DEFUZZIFY defines membership function of the output variables. The definition is mostly similar to the previous input variables described. The jFuzzyLogic provides several defuzzification methods for calculating the output variables. The study shows the “Centre Of Gravity” approach as the best and the most popular defuzzifer method (Runkler, 1997). Therefore, we define the DEFUZZIFY as follows.

List 3.4 Define DEFUZZIFY function

```
// Defuzzify output variable 'decision' : {'offloading',
'notoffloading' }
DEFUZZIFY decision
     TERM offloading := (0,0) (5,1) (10,0);
     TERM notoffloading := (10,0) (15,1) (20,0);
     METHOD : COG;
     DEFAULT := 0;
END_DEFUZZIF
```

The inference logic is defined by a list of RULEBOLCK, in which the fuzzy rules are declared. The order of the fuzzy rule does not matter by default; the jFuzzyLogic treats them equally. However, we can assign a weight of each rule by using "IF condition THEN conclusion with weight" clause. Also, it can use AND connector to bind several conditions. We define our rules as follows

List 3.5 Define rules

```
// Inference rules
RULEBLOCK No1
     AND : MIN;
     ACT : MIN;
     ACCU : MAX;
     RULE 1 : IF CPU IS high OR BATTERY IS poor THEN deci-
sion IS offloading;
     RULE 2 : IF CPU IS low AND BANDWIDTH IS fast THEN deci-
sion IS notoffloading;
     RULE 3 : IF CPU IS medium AND BATTERY IS poor THEN de-
cision IS offloading;
     RULE 4 : IF RAM IS high AND CPU IS medium THEN decision
IS offloading;
     RULE 5 : IF RAM IS medium AND CPU is medium AND BATTERY
is excellent THEN decision IS notoffloading;
END_RULEBLOCK
```

Setting the fuzzy rules is very import because the inference is based on the rules and they should be defined very carefully in order to obtain the accurate decisions. For instance, if the CPU usage is high that means the device currently is performing some heavy computation task, which also consume more battery at the same time. Hence, the decision can be: offloading the task.

Figure 3.3 A test case example

Figure 3.3 shows one test case. As the figure illustrates, currently, the phone status are: RAM is in 50% usage, CPU is in 80% usage, the battery status is 20% and the Bandwidth is 15Mbit/s. The decision is to offloading. This is the expected behaviour because the CPU current doing heavy task and battery is in poor condition, which also indicates the phone already in poor condition, and it is suitable to offloading the task.

The key contribution of this thesis is how to design a workflow-controlled framework with the resource-aware adaptive task-scheduling scheme that enables collaboration with other devices. We propose two task-scheduling scheme corresponding to the sequence workflow and parallel workflow.

3.2.2 Sequential Task Delegation

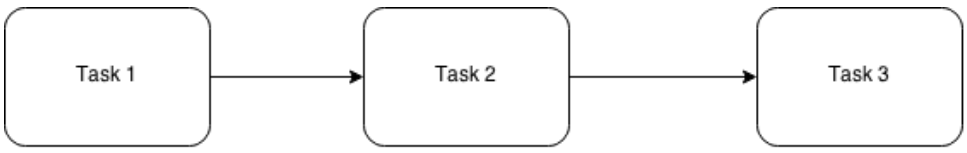

Figure 3.4 Sequence workflow

Figure 3.4 illustrates a sequence workflow using BPMN notation. Before each task execution, our framework will perform the offloading decision as the following sequence diagram shows (See Figure 3.5).

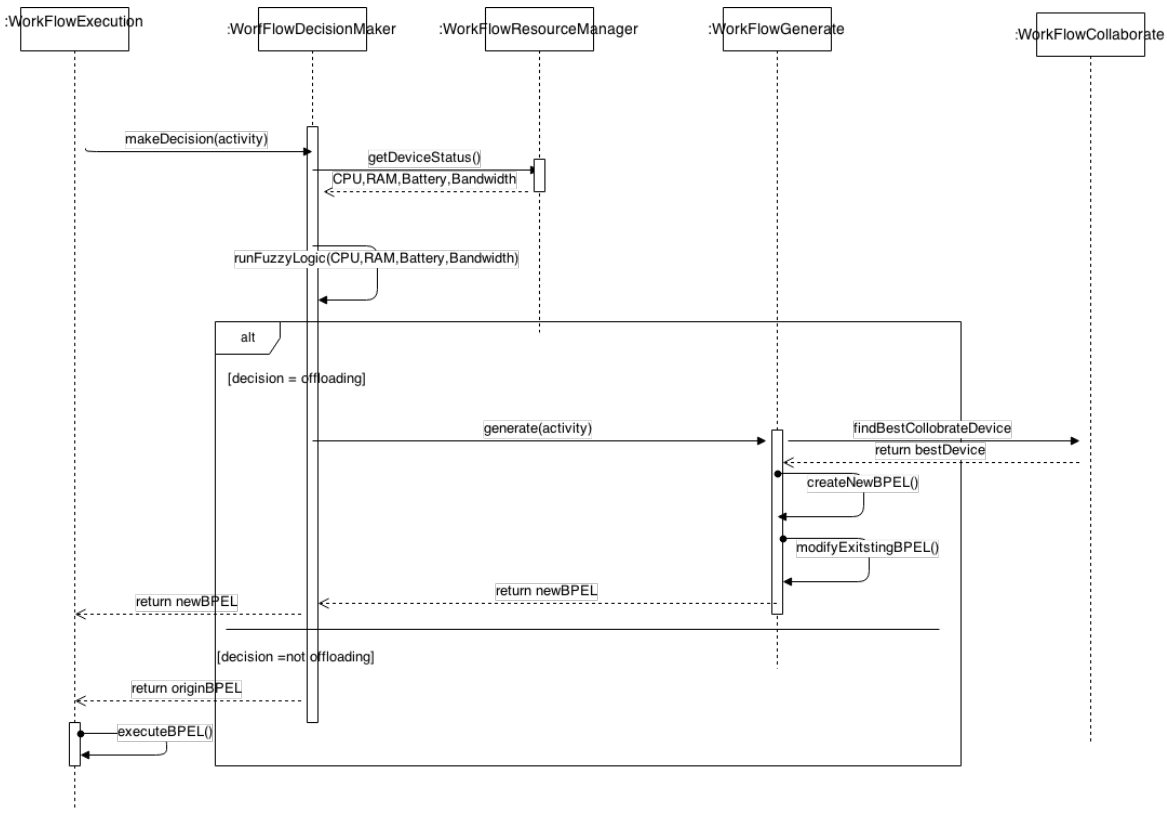

Figure 3.5 Sequence diagram for sequence delegation

Before executing each activity, the *Workflow Execution Module* will ask the *Workflow Decision Module* whether to offload the activity or not. In order to run the fuzzy logic to make the decision, the *Workflow Decision Module* needs to get the input variables (e.g. CPU, RAM, Battery and Bandwidth). It communicates with the workflow resource manager to get the current CPU, RAM, Battery and Bandwidth usage. The fuzzy logic will evaluate and make the decision based on those input variable. If the decision is to offloading the current task, the *Workflow Offloading Module* will generate a new BPEL file for this specify activity.

However, in order to decide which collaborative device will handle this activity. We have to find the device with the best capability to process it. We introduce a ranking scheme based on the collaborative device current capability, such as available CPU, RAM, Battery and Bandwidth. We expect each collaborative device provides a service about whether it is

capable to handle extra tasks currently. This can be achieved by using the same fuzzy logic to make the decision whether it is capable to handle extra tasks or not. If they are capable to be assigned extra tasks, the service will reply their current available CPU, RAM, Battery and Bandwidth value. The *Workflow Collaborate Module* will invoke this service on each collaborative device to get the available CPU, RAM, Battery and Bandwidth values. We use the following algorithm to calculate the ranking for each collaborative device.

Definition 1 (Collaborator Profile). *Collaborator profile is defined as a tuple* $(\mathcal{D}, \varepsilon, \lambda)$ *where:*

- $\mathcal{D}$ *is a set of mobile devices of collaborators.*
- $\varepsilon: \mathcal{D} \rightarrow \mathcal{R}$ *maps mobile devices to available resource values (e.g., CPU, RAM, etc).*
- $\lambda: \mathcal{R} \rightarrow \mathcal{W}$ *maps resources to the weight of resource.*

Let ϵv_r^D be $\sum_{d \in \mathcal{D}} v_r^d$ where v_r^d denotes the value of one of the available resource—$r \in \varepsilon(d)$ in device—$d \in \mathcal{D}$.

The ranking of a device—$d_x \in \mathcal{D}$ is computed by:

$$rank_x = \sum_{r \in \varepsilon(x)} \frac{v_r}{\epsilon v_r^D} \cdot \lambda(r)$$

Equation 1

where $rank_x$ denotes the ranking score of d_x. $\varepsilon(x)$ denotes the function that generates a set of available resource values of d_x. v_r represents the value of a $r \in \varepsilon(x)$ where the resource denoted by $r \in \varepsilon(x)$ is same as the resource denoted by ϵv_r^D. $\lambda(r)$ is the function that generates the weight value of the r.

Based on different activity, the $\lambda(r)$ change accordingly. For example, the invoke activity usually involved the networking connection. The available resource bandwidth in device $r \in \varepsilon(x)$ will be assigned a high weight value $\lambda(r)$, which means the device with a large network bandwidth will get a high score. Similarly, if the task requires a high computation power, the available resource CPU in device $r \in \varepsilon(x)$ will be assigned a high weight value $\lambda(r)$, which means the device with a better CPU will get a high score.

We will offload the task to the device with the best ranking score. The *workflow generate module* will get the best capability device from the *Workflow Collaborate Module*, and it will modify the original BEPL activity to the invoke activity that will offload the task to the selected device (See Figure 3.6).

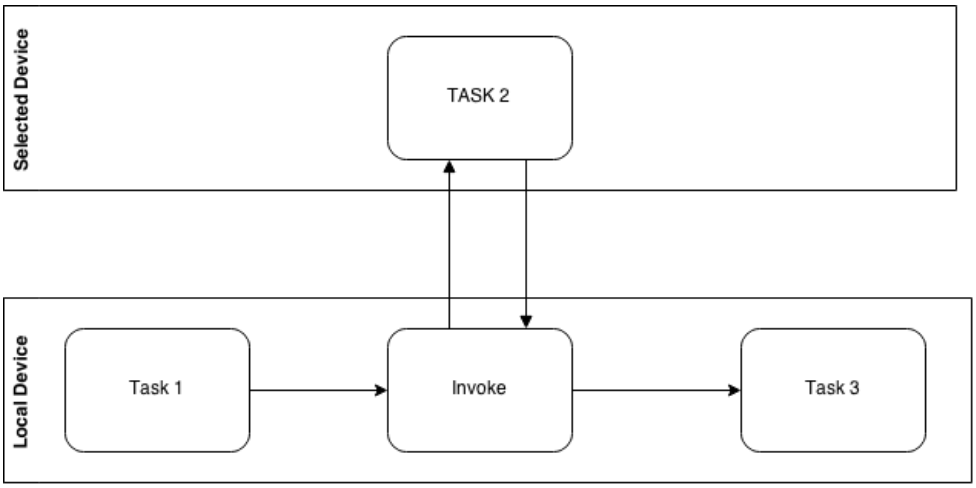

Figure 3.6 Sequence task delegation

3.2.3 Parallel Task Delegation

This section will discuss how parallel task delegation works. The parallel task is suitable for invoking a large of activities asynchronously. For example, the content mashup in service composition needs to discover and to process a large amount of data in order to identify and interpreter the consent. This usually involves interactive with a large number of service providers. The interaction with service providers does not depend on each other and can be done asynchronously.

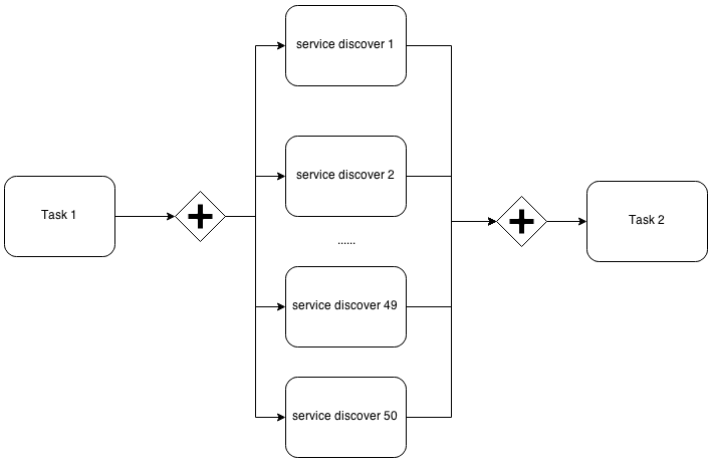

Figure 3.7 Parallel workflow

Figure 3.7 shows an example for content mashup in service discovery phase. Since this process may consume a lot of resources for single mobile phone, it is feasible to split the parallel tasks and delegate to the collaborative devices.

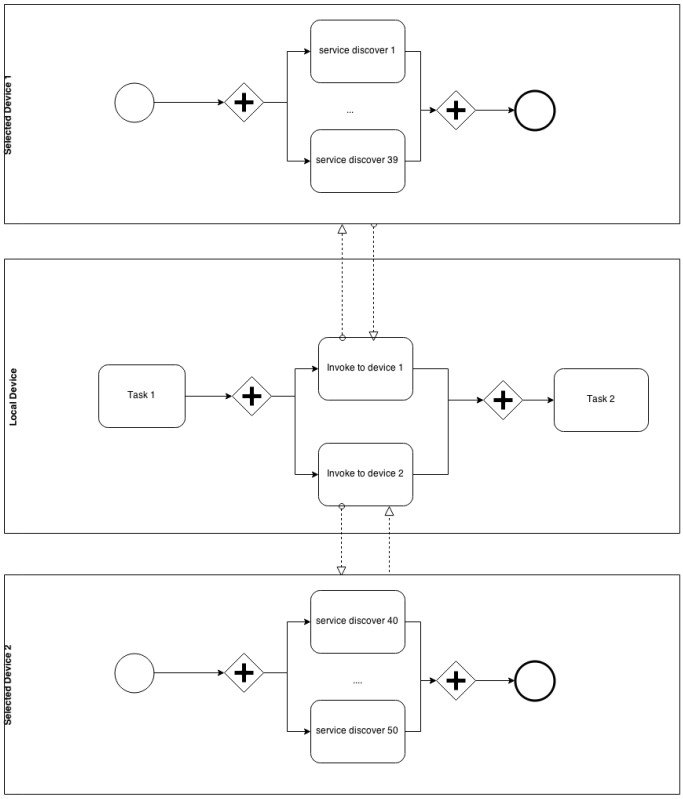

Figure 3.8 Parallel task delegation

Figure 3.8 shows an example of partition previous example of the parallel activities to two collaborative devices. The original BPEL file contains 50 parallel activities for discovering services. After the *Workflow Execution Module* decides to offload the parallel tasks, the *Workflow Offloading Module* creates two new BPEL file. One BPEL files contains 40 parallel activities; the other contains ten parallel activities. The parallel activities of the original BPEL file change to two invoke activity, which will offload the parallel activities to two external collaborative devices.

Unlike the sequencing task delegation, which we only select one collaborative device with the best capability currently, the parallel task delegation involves delegate the tasks to multiple collaborative devices, and each collaborate will assign different portion of the parallel tasks. We have the following question need to consider.

- How the workflow engine decides how many collaborative devices to support partition?
- How the workflow engine determines the portion of parallel tasks to each collaborative device for the partition?

To answer the question one, we use the same scheme as we describe in the sequence part. The *workflow collaborates module* firstly invoke the service on the collaborative device asking if they are capable to assign tasks currently. The *workflow collaborates module* maintain a list of collaborative devices that are capable to assign the tasks. We divide the parallel task based on the number of available collaborative devices.

To answer the second question, we using the same ranking algorithm as we describe in sequence part, and we add an extra algorithm to calculate the portion of parallel tasks to each collaborative device.

Definition 2 (Collaborator Portion). *Collaborator Portion is defined as a tuple (D, λ)*

Where:

- *D is a set of mobile devices of collaborators.*
- *λ is the ranking score of a device defining in* Equation 1

The portion of a device —$d_x \in \mathcal{D}$ is computed by:

$$portion_r = \frac{r_\lambda}{\sum_{r \in D}^{D} r_\lambda}$$

Equation 2

Where $portion_r$ denotes the portion of the collaborative device r. r_λ denotes the ranking score λ of the device r. $\sum_{r \in D}^{D} r_\lambda$ is the addition of a sequence ranking score λ in the set of collaborative devices D.

For example, the *workflow collaborates module* get three available devices with the status as List 3.6 shows.

List 3.6 An example of three available devices status

	available RAM(MB)	available CPU(MHZ)	available Battery(mAH)	available Bandwidth(Mbit/s)
Device 1	400	1610	1890	4
Device 2	2000	3000	3000	10
Device 3	1000	1000	1000	3

After normalized value (List 3.7)

List 3.7 Normalized value

	Normalized RAM	Normalized CPU	Normalized Battery	Normalized Band-width
Device 1	$\frac{400}{400+2000+1000} = 0.12$	$\frac{1600}{1600+3000+1000} = 0.29$	$\frac{1800}{1800+3000+1000} = 0.31$	$\frac{4}{4+10+3} = 0.23$
Device 2	$\frac{2000}{400+2000+1000} = 0.59$	$\frac{3000}{1600+3000+1000} = 0.54$	$\frac{3000}{1800+3000+1000} = 0.52$	$\frac{10}{4+10+3} = 0.59$
Device 3	$\frac{1000}{400+2000+1000} = 0.29$	$\frac{1000}{1600+3000+1000} = 0.17$	$\frac{1000}{1800+3000+1000} = 0.17$	$\frac{3}{4+10+3} = 0.18$

If the task involves networking connection, the ranking weight for the bandwidth will become 2. Therefore, The ranking score for each device is the summary of the normalized RAM, CPU, and Battery. Therefore

Device 1 ranking = 0.12 + 0.29 + 0.31 + 0.23 *2 = 1.18

Device 2 ranking = 0.59 + 0.54 + 0.52 + 0.59 * 2= 2.83

Device 3 ranking = 0.29 + 0.17 + 0.17 + 0.18 * 2 = 0.99

Therefore the portion of parallel task assign to device 1 is

$$\frac{1.18}{1.18 + 2.83 + 0.99} = 24\%$$

If the parallel task contains 100 asynchronized task, the device 1 will handle 100 * 24% = 24 tasks.

Similar to device 2

$$\frac{2.83}{1.18 + 2.83 + 0.99} = 57\%$$

The device 2 will handle 100 * 55% = 57 tasks.

4 Implementation Description

In the last chapter, we introduced our proposed a framework for enabling collaborative content mashup with pervasive services. In this section, we describe the implementation detail of each component of the framework.

4.1 Workflow Parser Module

This module is for reading and parsing BPEL-based workflow description into memory. The BPEL-based workflow description fundamentally is an XML file. We use the `XMLPullParser`, which is an Android native library for processing XML file. The traditional DOM-based parser, which reads the whole document as a tree structure in the memory for dynamically access and updates the content, occupies more memory. Hence, `XMLPullParser` was chosen.

The purpose of *Workflow Parser Module* is to provide all the information that described in the BPEL workflow description file for the *Workflow Execution Module* to execute the workflow. The BPEL workflow description file could have different tags to describe the structure of the workflow. Currently, the prototype support to process `<sequence>`, `<flow>`,`<forEach>`,`<invoke>`,`<assign>`,`<partnerLink>` and `<variable>` tags.

The following class diagram shows *Workflow Parser Module* components and their relationship.

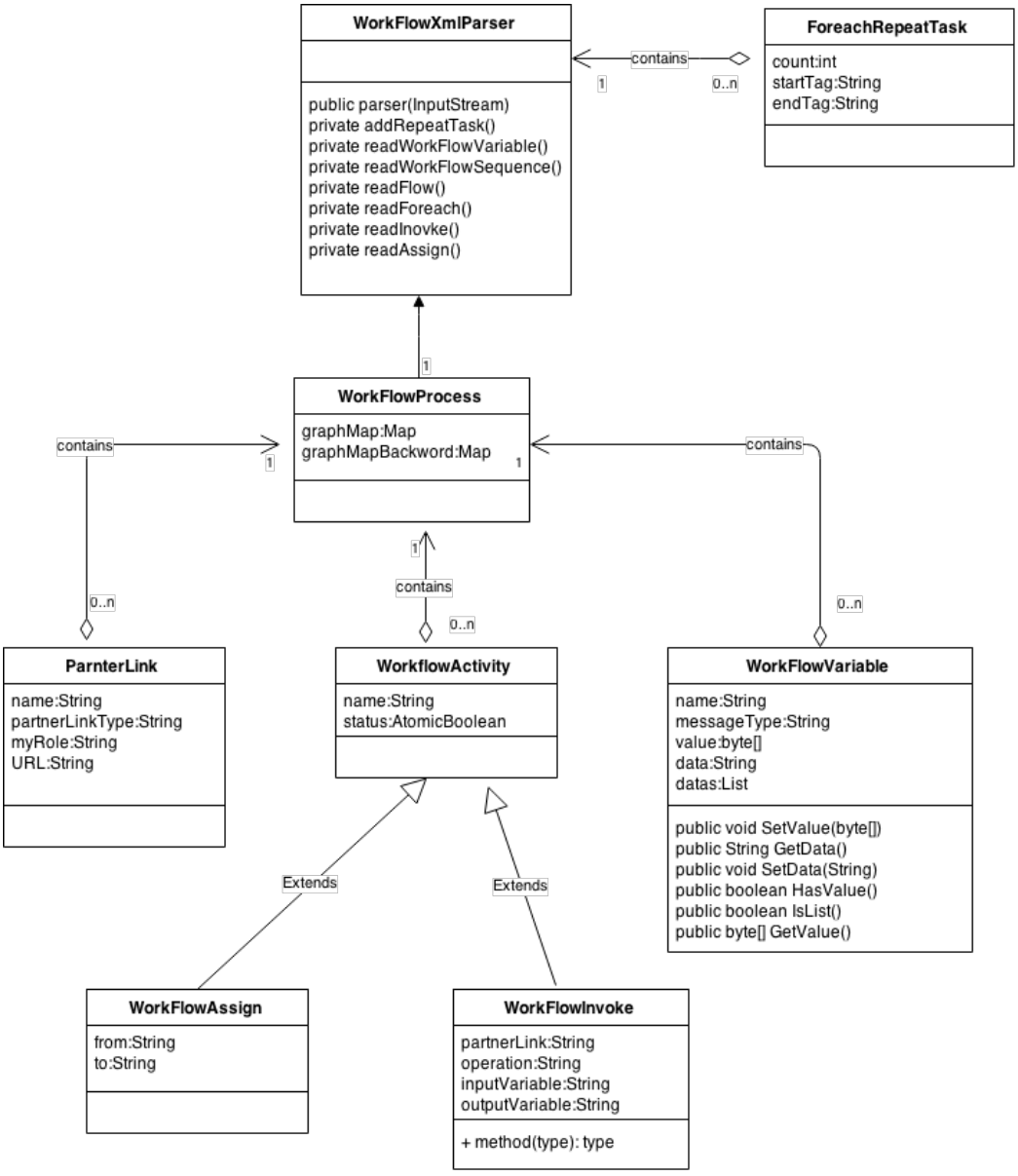

Figure 4.1 Class diagram of Workflow Parser Module

Figure 4.1 shows `WorkFlowXmlParser`, which is the entry point for processing the BPEL workflow description. Once it created, it directly executes `parser()` function that reading stream data of the BPEL workflow description file. Inside the `parser()` function, we instantiate an object of a `XmlPullParser` class for retrieving and processing BPEL tags. The `WorkFlowXmlParser` call corresponding method to handle different tags. For example, the `readFlow()` function is for retrieving the parallel structure of the workflow process. The `WorkflowProcess` class contains all the information related the BPEL workflow. The `graphMap` and `graphMapBackword` attribute inside `WorkflowProcess` class store the workflow structure in a directed graph using Map inter-

face. For one particular activity inside workflow, the `graphMap` provides information about next activity/activities after that activity in the workflow. If the next activity is more than one, it means the next execution is a parallel task. If the next activity is only one activity, it means a sequence task.

4.2 Workflow Execution Module

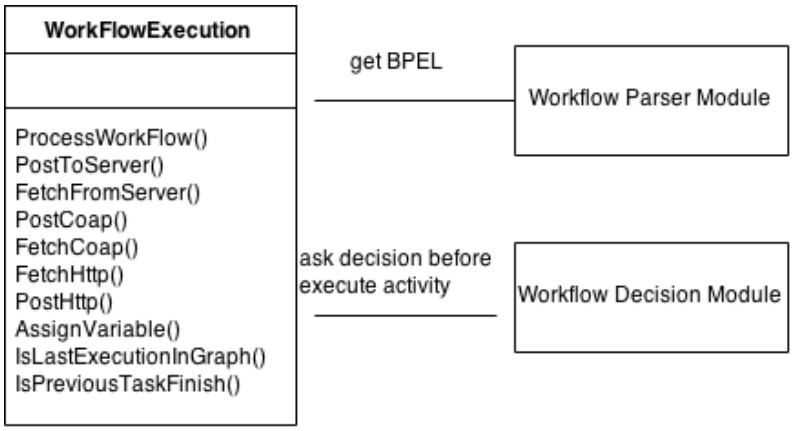

Figure 4.2 Class diagram of Workflow Execution Module

This module is for executing the workflow based on the information from the *Workflow Parser Module*. The *Workflow Execution Module* will get the reference of `workflowProcess` class that contains the workflow graph and list of `partnerlinks` and `variables`. The *Workflow Execution Module* will execute the activity defined in the workflow graph map. Before execute each activity, the *Workflow Execution Module* will communicate with the *Workflow Decision Module* for deciding whether to offload this activity or not based on the device current condition of the device. The implementation for *Workflow Decision Module* will be discussed later.

The `invoke` activity defines the behavior to invoke external services that was defined in the `partnerlink`. The *Workflow Execution Module* supports CoAP and HTTP protocol. Based on the URI signature defined in the `partnerLink` variable, the *Workflow Execution Module* invokes CoAP or HTTP service dynamically.

For the sequence workflow, the activities run sequentially, which means the current activity needs to be accomplished before the execution of the next activity. Conversely, the parallel task in the workflow is running asynchronously. The *Workflow Execution Module* identifies the following workflow is parallel or sequential task based on the size of next activity in the graph map. If next activity size is more than one, it means that next task in the workflow is a parallel task. The module will create multiple threads to support the asynchronous process of the parallel task execution in BPEL (See Figure 4.3).

```
private void ProcessWorkFlow(String graphKey) {
    if (!IsLastExecutionInGraph(graphKey) && IsPreviousTaskFinish(graphKey)) {
        decisionMaker.MakeDecision(graphKey);
        ArrayList<String> graphValues = graphMap.get(graphKey);
        for (int i = 0; i < graphValues.size(); i++) {
            ExecutionTask task = new ExecutionTask(graphValues.get(i));
            task.start();
        }
    }
}
```

Figure 4.3 The code of support parallel task execution

The `ExecutionTask` class implements the java runnable interface. If the `graphValues.size()` is more than one, each parallel task will execute on its own thread that means running asynchronously (See Figure 4.4).

```
class ExecutionTask implements Runnable {
    private String activityName;
    private Thread t;

    ExecutionTask(String _activityName) {
        activityName = _activityName;
    }

    @Override
    public void run() {
        WorkFlowActivity activity = activityMap.get(activityName);
        if (activity instanceof WorkFlowInvoke) {
            WorkFlowInvoke workFlowInvoke = (WorkFlowInvoke) activity;
            if (workFlowInvoke.operation != null && workFlowInvoke.operation.contains("POST")) {
                try {
                    PostToServer(workFlowInvoke);
                } catch (IOException e) {
                    e.printStackTrace();
                }
            } else {
                try {
                    FetchFromServer(workFlowInvoke);
                } catch (ClientProtocolException e) {
                    e.printStackTrace();
                } catch (IOException e) {
                    e.printStackTrace();
                }
            }
        } else if (activity instanceof WorkFlowAssign) {
            AssignVariable((WorkFlowAssign) activity);
        }
        activity.status.compareAndSet(false, true);
        ProcessWorkFlow(activityName);
    }

    public void start() {
        Log.d(TAG, "Starting " + activityName);
        if (t == null) {
            t = new Thread(this, activityName);
            t.start();
        }
    }
}
```

Figure 4.4 Code for running workflow asynchronously

4.3 Workflow Offloading Module

4.3.1 Overview

Once the *Workflow Decision Module* decides to offload the task to collaborative device, the *Workflow Execution Module* will pass the execution flow to *Workflow Offloading Module*. The *Workflow Offloading Module* will generate new BPEL file that describe the workflow about the offloading task. Afterward, the original task in the BPEL file will be changed to an invoke activity in which the `inputvariable` is the new BPEL file, and the `partnerlink` is the collaborative device IP address. The *Workflow Execution Module* will execute the modified invoke activity that offloads the task to the collaborative device.

4.3.1 Implementation Detail

When the *Workflow Offloading Module* generate new BPEL file, the following requirements need to be meet.

- When generating new BPEL file with the offloading tasks, the corresponding `variable` and `partnerlink` in the BPEL file need to be added as well.
- When offloading the parallel tasks, the *Workflow Offloading Module* is generating multiple new BPEL files when it needs to partition the parallel tasks to several collaborative devices with a different portion.
- The original BPEL file needs to be modified correspondingly in order to achieve collaboration with external devices.

To achieve the above requirements, the *Workflow Offloading Module* use the following processing steps:

Step 1 *Find the corresponding variables and partnerlinks with the offloading tasks*

When the initiator device decides to offload the tasks, the *Workflow Offloading Module* will identify the start task and end task on the graph map. The start task means the starting point of the workflow task that needs to be offloaded; the end task means the ending point of the workflow task that needs to be offloaded. For each task, the module calls the `FindCurrentTaskVariableAndPartnerLink()` function to find the corresponding variable and partner link in this task.

```
private void FindCurrentTaskVariableAndPartnerLink(String currentTag) {
    WorkFlowActivity activity = activityMap.get(currentTag);
    if (activity instanceof WorkFlowInvoke) {
        WorkFlowInvoke invoke = (WorkFlowInvoke) activity;
        for (WorkFlowVariable variable : variables) {
            if (variable.name.equals(invoke.inputVariable)) {
                if (!offloadingVariables.containsKey(variable.name)) {
                    offloadingVariables.put(variable.name, variable);
                }

            } else if (variable.name.equals(invoke.outputVariable)) {
                if (!offloadingVariables.containsKey(variable.name)) {
                    offloadingVariables.put(variable.name, variable);
                }
            }
        }

        for (PartnerLink partnerLink : partnerLinks) {
            if (partnerLink.name.equals(invoke.partnerLink)) {
                if (!offloadingPartnerLinks.containsKey(partnerLink.name)) {
                    offloadingPartnerLinks.put(partnerLink.name, partnerLink);
                }
            }
        }
    } else if (activity instanceof WorkFlowAssign) {
        WorkFlowAssign assign = (WorkFlowAssign) activity;
        for (WorkFlowVariable variable : variables) {
            if (variable.name.equals(assign.from)) {
                if (!offloadingVariables.containsKey(variable.name)) {
                    offloadingVariables.put(variable.name, variable);
                }

            } else if (variable.name.equals(assign.to)) {
                if (!offloadingVariables.containsKey(variable.name)) {
                    offloadingVariables.put(variable.name, variable);
                }
            }
        }
    }
}
```

Figure 4.5 Code for adding variables and partnerlinks

As Figure 4.5 shows, for each activity, the corresponding `variables` and `partnerlinks` will be added.

Step 2 *Generate a new BPEL activity with the current activity*

```
private void CreateCurrentXMLTag(String currentTag) throws IllegalArgumentException, IllegalStateException, IOException {
    WorkFlowActivity activity = activityMap.get(currentTag);
    if (activity instanceof WorkFlowInvoke) {
        WorkFlowInvoke invoke = (WorkFlowInvoke) activity;
        CreateInvoke(invoke);
    } else if (activity instanceof WorkFlowAssign) {
        WorkFlowAssign assign = (WorkFlowAssign) activity;
        CreateAssign(assign);
    }
}
```

Figure 4.6 Code for generate new activity

If the current activity is an `<invoke/>` activity, the module will call the `CreateInvoke()` method to create new BPEL invoke. If the current activity is `<assign/>`, the module will call the `CreateAssign()` method to create new BPEL assign.

Step 3: *Modify the original BPEL workflow*

If the offloading task is a sequence task, we only need to change the task to an `<invoke/>` activity in the original BPEL file. The graph map and graph backward map need to be modified as to describe the new BPEL workflow.

```
public void ModifyBpelMap(String startTask, String endTask,StringWriter writer, String IP){
    //Create offloading input variable
    WorkFlowVariable offloadingInput = new WorkFlowVariable("offloadingInput","tns:String");
    WorkFlowVariable offloadingOutput = new WorkFlowVariable("offloadingOutput","tns:String");
    offloadingInput.SetData(writer.toString());

    variables.add(offloadingInput);
    variables.add(offloadingOutput);
    //Create Offloading partnerlink

    PartnerLink offLoadingPartnerLink = new PartnerLink("offLoadingPartnerLink","tns:PostData","",IP);
    partnerLinks.add(offLoadingPartnerLink);

    WorkFlowInvoke invoke = new WorkFlowInvoke("InvokeOffloading",offLoadingPartnerLink.name,"POST",offloadingInput.name,offloadingOutput.name);
    activityMap.put(invoke.name,invoke);
    //update graomap the next task is new invoke
    ArrayList<String> newArrayList = new ArrayList<>();
    newArrayList.add(invoke.name);
    graphMap.put(startTask,newArrayList);

    //update next inovke activity to endtask
    ArrayList<String> newArrayList2 = new ArrayList<>();
    newArrayList2.add(endTask);
    graphMap.put(invoke.name,newArrayList2);

    graphMapBackword.put(endTask,newArrayList);
}
```

Figure 4.7 Code for modify original sequence workflow

As the Figure 4.7 shows, the `ModifyBpelMap()` method will change the original workflow with a new `<invoke/>` activity together with an input variable that contains the new BPEL workflow description. Afterward, the original BPEL workflow was modified with the new graph map and graph backward map structure.

Unlike to sequence task, the parallel task delegation involves modifying the original BPEL workflow to multiple `<invoke/>` activities that support task delegation to several collaborative devices. The `ModifyBpelParallel()` method will receive a list of IP address of the collaborative device. The new invoke activity is created based on the IP address size. Also, the original BPEL workflow was modified with a list of invoke activity added (See Figure 4.8).

```
public void ModifyBpelParallel(String startTask, String endTask, List<CollaborateDevice> IPs,ArrayList<String> inputVariable){
    ArrayList<String> invokes = new ArrayList<>();
    for(int i = 0 ; i < IPs.size() ; i++){
        WorkFlowVariable offloadingInput = new WorkFlowVariable("offloadingInput" + i,"tns:String");
        WorkFlowVariable offloadingOutput = new WorkFlowVariable("offloadingOutput" + i,"tns:String");
        offloadingInput.SetData(inputVariable.get(i));
        variables.add(offloadingInput);
        variables.add(offloadingOutput);
        PartnerLink offLoadingPartnerLink = new PartnerLink("offLoadingPartnerLink" + i ,"tns:PostData","",IPs.get(i).IP +":8080");
        partnerLinks.add(offLoadingPartnerLink);
        WorkFlowInvoke invoke = new WorkFlowInvoke("InvokeOffloading" + i ,offLoadingPartnerLink.name,"POST",offloadingInput.name,offloadingOutput.name);
        activityMap.put(invoke.name,invoke);
        invokes.add(invoke.name);
        // next activity joint activity
        ArrayList<String> jointActivity = new ArrayList<>();
        jointActivity.add(endTask);
        graphMap.put(invoke.name,jointActivity);
    }

    graphMap.put(startTask,invokes);
    graphMapBackword.put(endTask,invokes);
    for(String invokeName : invokes){
        ArrayList<String> previousActivity = new ArrayList<>();
        previousActivity.add(startTask);
        graphMapBackword.put(invokeName,previousActivity);
    }
}
```

Figure 4.8 Code for modify the original parallel workflow

4.4 Workflow Decision Module

As describes in Chapter 3.2.1, the *Workflow Decision Module* will use battery status, CPU, RAM, and bandwidth as input variables for the fuzzy logic to make the decision of offloading. Figure 4.9 shows the class diagram for the *Workflow Decision Module*.

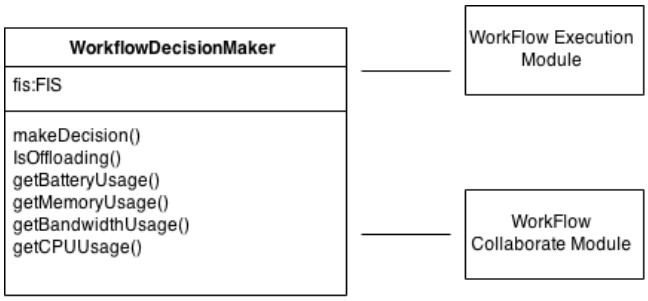

Figure 4.9 Class diagram for Workflow Decision Module

The FIS object is the core element in the jFuzzyLogic library. Firstly, it read the fuzzy logic rules that we describe in Chapter 3. Inside the `IsOffloading()` method, the FIS object will run the fuzzy logic based on the current CPU, RAM , Battery and Bandwidth through the `evaluate()` method. After the evaluation, we get the output value, which has been defined in Chapter 3. The output value is the decision point to tell whether to offload the current task (See Figure 4.10).

```
private boolean IsOffloading() {
    if (fis == null) {
        Log.e(TAG, "FAILED TO OPEN");
    }
    double defuzzifiedValue;

    fis.setVariable("CPU", getCPUUsage());
    fis.setVariable("BATTERY", getBatteryUsage());
    fis.setVariable("RAM", getMemoryUsage());
    fis.setVariable("BANDWIDTH", getBandwidthUsage());
    fis.evaluate();
    Log.e(TAG, "" + fis.getVariable("decision").getLatestDefuzzifiedValue());
    defuzzifiedValue = fis.getVariable("decision").getLatestDefuzzifiedValue();
    if (defuzzifiedValue > 10) {
        return false;
    } else {
        return true;
    }
}
```

Figure 4.10 Code for running fuzzy logic

4.5 Workflow Collaborate Module

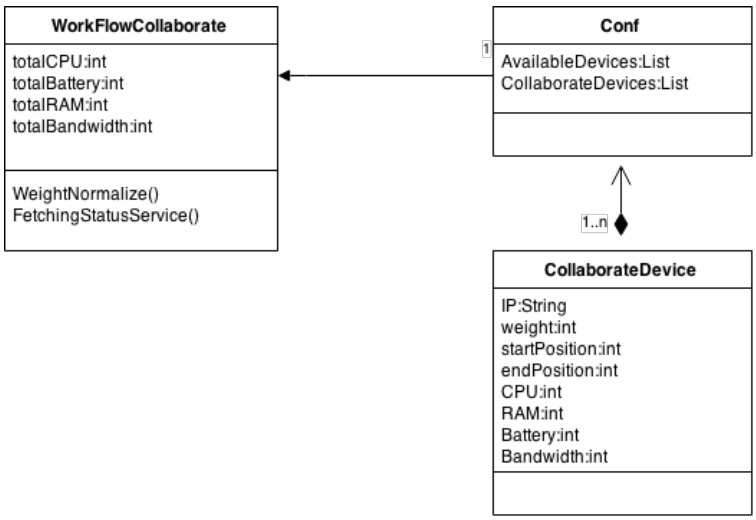

Figure 4.11 Class diagram for Workflow Collaborate Module

The *Workflow Collaborate Module* maintains a list of currently available collaborative devices. The term "available" means that the collaborative device can handle extra tasks currently. When the *Workflow Decision Module* makes a decision, the *Workflow Collaborate Module* will communicate each collaborative device and asking if then are capable to assign tasks currently. The FetchingStatusService() function will get a list of available collaborative devices and as well as their current CPU, RAM, Battery and Bandwidth.

```
private class fetchingStatusService extends AsyncTask<Void, Void, Void> {
    @Override
    protected Void doInBackground(Void... params) {
        // TODO Auto-generated method stub
        try {
            for(int i = 0 ; i < Conf.CollaborateDevices.size() ; i++){
                String response = "";
                try {
                    response = fetchHttp(Conf.CollaborateDevices.get(i)+":8081");
                } catch (IOException e) {
                    e.printStackTrace();
                }

                List<String> items = Arrays.asList(response.split("\\s*,\\s*"));
                if(items.size() > 1){
                    CollaborateDevice device = new CollaborateDevice(Conf.CollaborateDevices.get(i));
                    device.RAM = Integer.parseInt(items.get(0));
                    device.Battery = Integer.parseInt(items.get(1));
                    device.CPU = Integer.parseInt(items.get(2));
                    device.Bandwidth = Integer.parseInt(items.get(3));
                    totalRAM += device.RAM;
                    totalBattery += device.Battery;
                    totalCPU += device.CPU;
                    totalBandwidth += device.Bandwidth;
                    Conf.AvailableDevices.add(device);
                }
            }
        }catch (Exception e){

        }
        return null;
    }
```

Figure 4.12 The code for getting available device with their conditions

As the code above, we assume each collaborative device is running a service on port 8081 that providing the information whether they are capable to handle extra tasks or not. If they are available, they will reply their current available CPU, RAM, battery, and bandwidth. The *Workflow Collaborate Module* will store the currently available collaborative device as well as the CPU, RAM, battery, and bandwidth. The `WeightNormalize()` function will calculate the ranking for each available collaborative device as we describe in Chapter 3 (See Figure 4.13).

```
private void WeightNormalize(){
    float normalizeCPU = 0;
    for(CollaborateDevice device : Conf.AvailableDevices){
        if(totalCPU !=0) {
             normalizeCPU = (float) device.CPU / (float) totalCPU;
        }
        float normalizeBattery = (float) device.Battery / (float) totalBattery;
        float normalizeRAM = (float) device.RAM / (float) totalRAM;
        float normalizeBandwidth = (float) device.Bandwidth / (float) totalBandwidth;
        float weight = normalizeCPU + 3 * normalizeBattery + 5 * normalizeRAM + normalizeBandwidth;
        Log.e(TAG, "DEVICE WEIGHT IS " + weight);
        if(weight == 0){
            device.weight = randInt(3,6) * 10;
        }else {
            device.weight = (int) (weight * 100);
        }
    }
}
```

Figure 4.13 Code to perform normalize

5 Evaluation

5.1 Introduction

In the previous chapters, we have illustrated the proposed solution to resolve Collaborative Content Mashup with Pervasive Services. In Chapter 3, we have presented the architecture design of the proposed framework. A proposed fuzzy model-based resource-aware offloading among mobile devices. Also, we introduced two offloading scheme for how to partition and offload the activities in sequence and parallel tasks. The framework is light-weight and supports CoAP protocol.

In this chapter, we evaluated our framework based on two scenarios. The first scenario demonstrated the advantage of supporting CoAP protocol for content mashup in a constrained pervasive services environment in terms of energy saving and performance. The second scenario demonstrated the advantage of supporting task offloading using our proposed algorithm.

5.2 Scenario 1

The first scenario is a 'service discover scenario'. The workflow of this scenario defines parallel tasks to perform service discovery. A classic service provider uses a standard WSDL to describe its operation. When a client performs service discovery for the desired service, the client matches the keyword based on the vocabularies provided by the server WSDL. On the other hand, the CoAP service discovery is based on the `well-known` address. The `well-known` address includes CoRE link format to describe hosted resources, their attributes for the client to interpreter the services.

1. **Setting**

In order to compare the advantage of adopting CoAP protocol in our framework, we compare the WSDL service discovery and CoAP service discovery. In the workflow, we define 10-200 parallel tasks for the service discovery. We measure and compare the average response time, throughput, and battery consumption for using CoAP service discovery and WSDL service discovery.

List 5.1 Test case for WSDL service discovery

```
    <sequence name="main">
        <assign name="startPoint">
            <copy>
                <from variable="variable1" />
                <to variable="variable2" />
            </copy>
        </assign>
        <forEach countername='n'>
            <startCounterValue>0</startCounterValue>
            <finalCounterValue>200</finalCounterValue>
            <sequence>
                <invoke name="ServiceDiscovery"
                partnerLink="getWSDL"
                operation="GET"
                outputVariable="mathcingResult"/>
            </sequence>
        </forEach>
        <assign name="endPoint">
            <copy>
                <from variable="variable1" />
                <to variable="variable2" />
            </copy>
        </assign>
    </sequence>
```

List 5.1 defines a workflow for WSDL service discovery. The `<forEach/>` defines number of repeat tasks based on the number defined in the `<finalCounterValue/>` tag. The `<invoke/>` defines an HTTP GET request that would fetch a WSDL from the external service defined in the `<partnerLink/>`. We implemented an external server that would return the following the WSDL (List 5.2) based on the request.

List 5.2 The WSDL defined in the external server

```
<message name="getTemperatureRequest">
  <part name="value" type="xs:string"/>
</message>
<message name="getTemperatureResponse">
  <part name="value" type="xs:string"/>
</message>
<portType name="Temperature">
  <operation name="getTemperature">
    <input message="getTemperatureRequest"/>
    <output message="getTemperatureResponse"/>
  </operation>
</portType>
<binding type="Temperature" name="b1">
   <soap:binding style="document"
   transport="http://schemas.xmlsoap.org/soap/http" />
```

```
   <operation>
     <soap:operation                                          soapAc-
tion="http://example.com/Temperature"/>
     <input><soap:body use="literal"/></input>
     <output><soap:body use="literal"/></output>
  </operation>
</binding>
```

List 5.3 Test case for CoAP service discovery

```
    <sequence name="main">
        <assign name="startPoint">
            <copy>
                <from variable="variable1" />
                <to variable="variable2" />
            </copy>
        </assign>
        <forEach countername='n'>
            <startCounterValue>0</startCounterValue>
            <finalCounterValue>200</finalCounterValue>
            <sequence>
                <invoke name="CoapServiceDiscovery"
                partnerLink="getWellKnowInCoap"
                operation=".well-known/core"
                outputVariable="coapServiceResponse"/>
            </sequence>
        </forEach>
        <assign name="endPoint">
            <copy>
                <from variable="variable1" />
                <to variable="variable2" />
            </copy>
        </assign>
    </sequence>
```

List 5.3 defines a workflow for CoAP service discovery. The `<forEach/>` defines the same behavior as previously described. The `<invoke/>` defines a CoAP `well-known` discovery to the service defined in the `partnerLink`.

The service provider was implemented in mobile device - LG G3 with Android version 5.0 under local wireless network in the University of Tartu. The client was implemented in mobile device - Google Nexus 5 running with Android OS 5.0.1 in the same local wireless network. The clients performed 10-200 parallel requests for testing. For each test case, we performed five times and calculated the average time.

2. Result

A. Throughput comparison:

Figure 5.1 shows the throughput comparison between the HTTP-based WSDL service provider and the UDP-based CoAP servicer provider. Because HTTP implemented on top of TCP, which guarantees reliable transmission of data. It maintains 100% success rate for the client invokes 10-200 parallel requests. However, the CoAP protocol was based on UDP in which packets are sent without guarantee of delivery. As we can see from the figure when the client invokes large than 100 parallel requests, the throughput of the service provider was not able to maintain 100% rate.

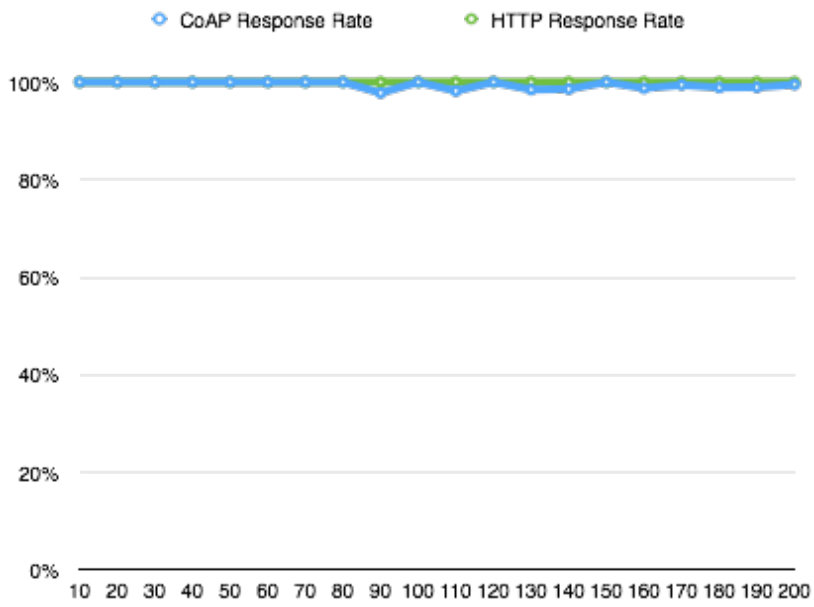

Figure 5.1 Line chart of the throughput

B. Response time comparison

Figure 5.2(a) shows the average response time for the parallel requests. As the parallel task request increased, the average response time for both CoAP and HTTP also increased. However, the average response time for CoAP is much less compares the HTTP request. Figure 5.2(b) shows the max response time for each parallel request. As we can see from the figure when the client sends more than 100 parallel requests, the max response time for HTTP has increased to almost 2 seconds. However, the max response time for CoAP remains almost the same level. The CoAP server is much more efficiency for processing the parallel request comparing to HTTP.

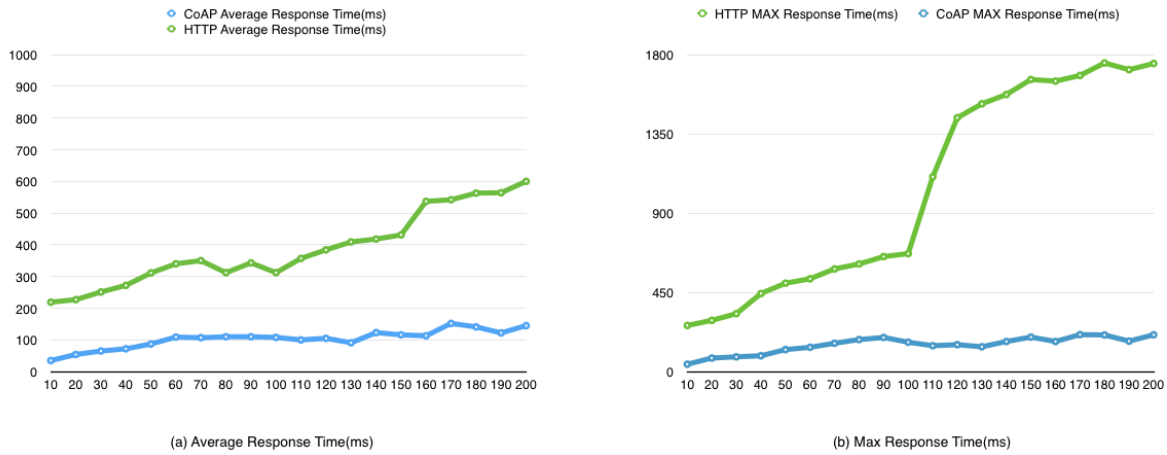

Figure 5.2 Line chart of the response time

C. Energy consumption comparison

The energy consumption was measured by simulating 200 parallel requests every 15 seconds in an hour. As Figure 5.3 shows, the battery drop 11% in traditional HTTP while the battery drop 9% in CoAP. The result shows that the framework supporting CoAP consumes less energy.

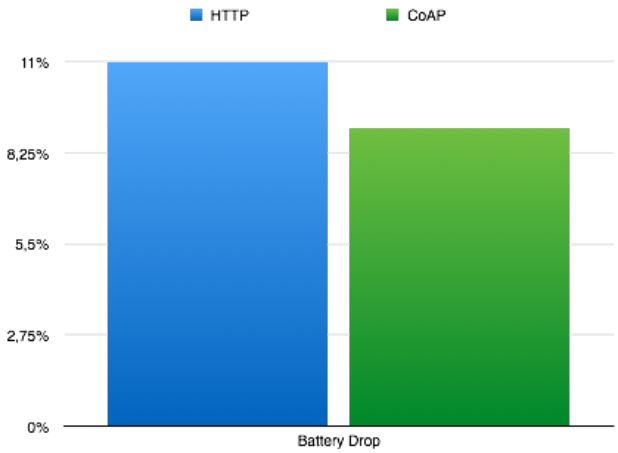

Figure 5.3 Bar chart for battery consumption

5.3 Scenario 2

The second scenario is the “content mashup scenario”. The scenario performed temperature mashup in the pervasive services environment. In 1.2.1 under Chapter 1, we described a scenario of finding parking lot information. In our test case 2, we defined exactly the

same workflow but finding temperature information instead of finding parking lot information. The challenge is that potentially there are many service providers, it is difficult and time-consuming for a single device to explore and filter the desired service in the pervasive services environment without collaborating with other devices. We proposed a resource-aware offloading scheme in our framework. We evaluated the framework under two test cases. In the first test case, we measured the workflow execution time and compared the execution time difference between partition equal and partition using our proposed algorithm. In the second test case, we measured the energy consumption between running the workflow locally and partition the workflow to the collaborative devices.

1. **Setting**

The workflow begins with the service discovery. For each service, it performs service invocation and service filter. We performed the scenario under the CoAP implementation. Therefore, the workflow contains CoAP discovery, CoAP `well-know` invoke, CoAP ontology matching based on the resource type (See List 5.4).

List 5.4 Workflow for scenario 2

```
<process name="ExecuteWorkflow"
    xmlns="http://docs.oasis-
open.org/wsbpel/2.0/process/executable"
xsi:noNamespaceSchemaLocation="datatype.xsd">
    <partnerLinks>
        <partnerLink name="BLEScanning"
            partnerLink-
Type="tns:GetData">coap://localhost/coapIP</partnerLink>
        <partnerLink name="getWellKnowInCoap"
            partnerLink-
Type="tns:GetData">$scanCoapResultUsingBLE</partnerLink>
        <partnerLink name="matchingCoap"
            partnerLink-
Type="tns:GetData">coap://localhost:5684/temperatureMatching
</partnerLink>

    </partnerLinks>
    <variables>
        <!--
         Reference to the message that will be returned to
the requester
         -->
        <variable name="scanCoapResultUsingBLE"
            messageType="tns:List"/>
        <variable name="coapServiceResponse"
```

```
            messageType="tns:String"/>
        <variable name="postData"
            messageType="tns:String"/>
        <variable name="mathcingResult"
            messageType="tns:String"/>
    </variables>
    <sequence name="main">
        <assign name="startPoint">
            <copy>
                <from variable="variable1" />
                <to variable="variable2" />
            </copy>
        </assign>
        <forEach countername='n'>
            <startCounterValue>0</startCounterValue>
            <finalCounterValue>20</finalCounterValue>
            <sequence>
                <invoke name="CoapServiceDiscovery"
                    partnerLink="getWellKnowInCoap"
                    operation=".well-known/core"
                    outputVariable="coapServiceResponse"/>
                <assign name="assign">
                    <copy>
                        <from  variable="coapServiceResponse"
/>
                        <to variable="postData" />
                    </copy>
                </assign>
                <invoke name="invokeMatchingCoap"
                    partnerLink="matchingCoap"
                    operation="POST"
                    inputVariable="postData"
                    outputVariable="mathcingResult"/>
            </sequence>
        </forEach>
        <assign name="endPoint">
            <copy>
                <from variable="variable1" />
                <to variable="variable2" />
            </copy>
        </assign>
    </sequence></process>
```

For each `partnerlink`, it means a different service running on the android phone. For example, the `partnerLink` "`getWellKnowInCoap`" is the service running on android to performing the CoAP `well-known` discovery and get the ontology defined on the resource type. The `partnerLink` "`matchingCoap`" is another service running on android to perform the ontology matching (See Figure 5.4).

```
private static boolean MatchTemperatureProperty(com.hp.hpl.jena.rdf.model.Resource resource, Property property){
    HashMap<String,Boolean> temperatureOntologyMap =InitTemperatureOntologyMap();

    StmtIterator iterator = resource.listProperties();
    while(iterator.hasNext()){
        Statement statement = iterator.nextStatement();
        String key = statement.getObject().toString();
        if(temperatureOntologyMap.containsKey(key)){
            temperatureOntologyMap.put(key, true);
        }
    }

    for(Map.Entry<String, Boolean> entry: temperatureOntologyMap.entrySet()){
        if(!entry.getValue()){
            return false;
        }
    }

    return true;
}

private static HashMap<String,Boolean>  InitTemperatureOntologyMap(){
    HashMap<String,Boolean>  temperatureOntologyMap = new HashMap<>();
    temperatureOntologyMap.put("http://ontology/scale" , false);
    temperatureOntologyMap.put("http://ontology/degree" , false);
    return temperatureOntologyMap;

}
```

Figure 5.4 Temperature ontology matching

We run this workflow in LG G3 running Android version 5.0 under local wireless network in the University of Tartu. Three collaborative devices participate the workflow offloading. Two devices are Google Nexus 5 running on Android 5.0.1 and one device is Google Nexus 7 running on Android 4.4.4. We run the parallel tasks from 10 – 70 tasks and measure the time consumption using the algorithm to perform task partition and equal task partition. For each test case, we performed five times and calculated the average time.

2. Result

A. Time comparison

As Figure 5.5 shows, when the parallel tasks less than 20 tasks, there is no much difference when using the algorithm to enable the tasks partition or equal tasks partition. Because the total number of the tasks is small, the task assigned for each collaborative almost the same as the equal tasks partition even using the algorithm. However, when there are many parallel tasks, using the algorithm to enable the task partition takes less execution time compare just partition the tasks equally.

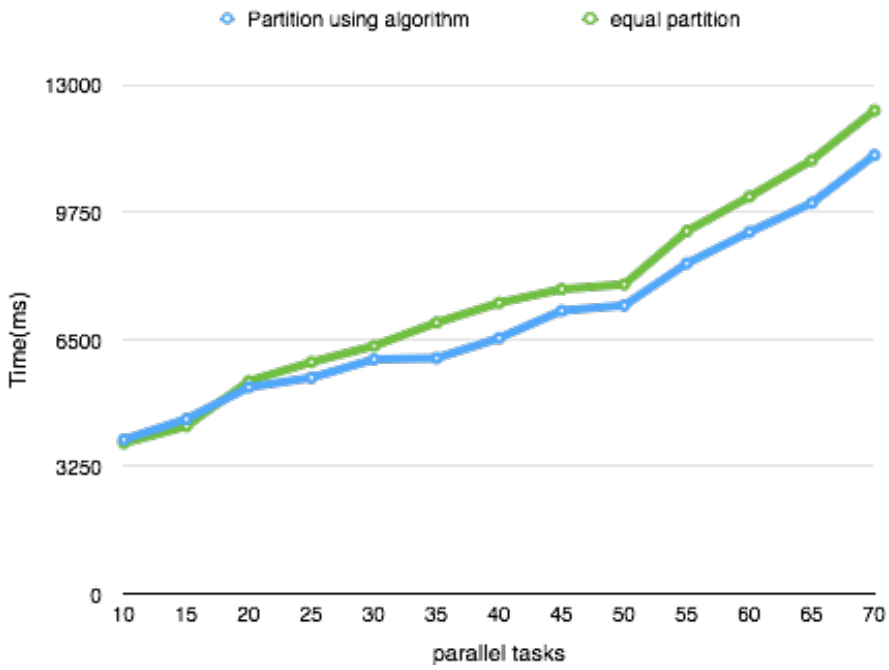

Figure 5.5 Scenario 2 time comparison

B. Energy consumption comparison

We run two test cases to compare the energy consumption. One test case is running the entire workflow locally. The other test case is offloading the task to collaborative devices using partition algorithm. The testing was repeatedly running 24 parallel tasks defined in the workflow (List 5.4) every 20 seconds for one hour and measuring the battery consumption. As Figure 5.6 shows, the battery computation saved nearly half when offloading the workflow to collaborative devices than running the workflow locally. The result proved that the offloading scheme consumes much less energy than running the workflow locally.

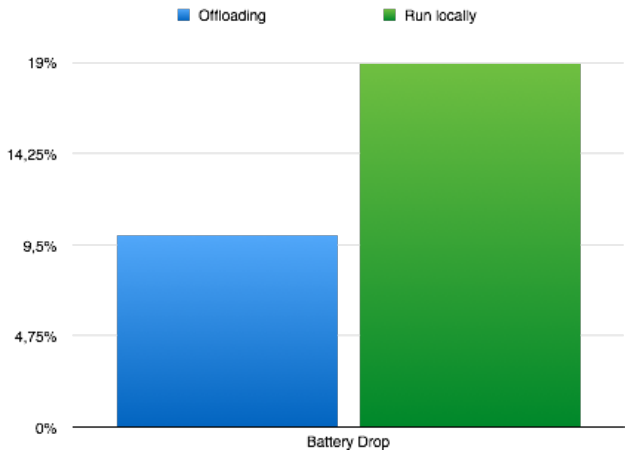

Figure 5.6 Battery consumption for running workflow locally and partition the workflow to collaborative devices

5.4 Discussion

In general, the test cases results were as we expected. Figure 5.2 shows that the service discovery using CoAP takes less time compares the traditional HTTP WSDL service discovery. Figure 5.3 shows that using CoAP consumes less energy in terms of battery consumption. It proves that the framework is lightweight and CoAP supported workflow engine. Figure 5.5 shows that it is efficient using our ranking algorithm to decide the portion of the offloading tasks instead of partition the task equally. Figure 5.6 shows that offloading the tasks to collaborative devices can save a significant amount of battery than running all the tasks locally. The framework supports task offloading in content mashup in a collaborative environment. The result shows that the framework achieves the objective defined in Chapter 1.

6 Conclusions

6.1 Research Summary and Contributions

This thesis intends to investigate an approach towards resolving Collaborative Content Mashup with Pervasive Services. We implement a framework that supports content-aware service discovery, decentralisation, and a collaborative task-offloading scheme by enabling service-oriented service composition among mobile resources. The framework supports BPEL workflow to achieve the content mashup. In order to resolve resource-constrained issues in mobile devices, the framework supports CoAP service interaction and task offloading scheme. Based on the device real-time condition (CPU, RAM, Battery and Bandwidth), we implement a fuzzy logic for deciding whether to offload tasks to collaborative devices. We proposed two offloading scheme for how to partition and offload the activities in sequence and parallel tasks. We proposed a ranking algorithm for collaborative devices to decide the portion of the offloading tasks. The framework can dynamic generate new workflows to collaborative devices and modify the original workflow based on the decision was made at run-time. The first test case shows the advantage of the framework supporting the CoAP protocol in service discovery phase in terms of energy and time consumption. The second test case shows the advantage of the ranking algorithm to decide the portion of the offloading tasks instead of partition the task equally. It also proves that offloading the tasks to collaborative devices can save a significant amount of battery than running all the tasks locally. The evaluation results have shown that the framework supports collaborative task-offloading scheme that reduces the resource usage of mobile devices.

6.2 Future Research Directions

Currently, we continue validating the framework and the possible improvement could be

- The framework is capable of processing basic BPEL workflow, `<sequence>`, `<flow>`, `<invoke>`, `<assign>` and `<forEach>`. However, the more advance workflow, `<faultHandlers>`, `<catch>` and `<validate>` is not supported yet.

- When deciding the portion of offloading tasks to collaborative devices, we only consider the available CPU, RAM, Bandwidth, and Battery usage in our ranking algorithm. We are still studying and trying to find a better benchmarking algorithm to compare different device capabilities.

Besides the possible improvement of our framework, there are still many unsolved challenges. We list our future research directions as follows

- The framework does not address how to establish a high-quality long-live communication with collaborative devices in an unstable network communication environment. When the framework decides to offload certain tasks to the collaborative device, due to the high dynamic change of collaborative device, the connection between the initiator device with the collaborative device could be lost during runtime. Even the collaborative device finished the assigned tasks. It could not be able to send the result back to the initiator device.
- The potential collaborative network could consist of 1000 or more mobile devices. It is too expensive to explore all the collaborative devices and to calculate the ranking score for each collaborative device at run-time for each workflow activity. It requires further investigation to find a proper solution.
- The framework does not address the trust and security with the collaborative devices to enable task delegation. Untrusted user from the public could assign malicious tasks to the device, which is not the friend of the untrusted user. A trustworthy collaborative environment requires further investigation.

References:

Bakhouya, M., & Gaber, J. (n.d.). Service composition approaches for ubiquitous and pervasive computing: A survey.

Berardi, D. (n.d.). Automatic composition of process-based Web services: a challenge.

Berners-Lee, T., Hendler, J., Lassila, O., & others. (2001). The semantic Web.

Bossi, L., Braghin, S., & Trombetta, A. (2014). Multidimensional Reputation Network for Service Composition in the Internet of Things. In *Services Computing (SCC), 2014 IEEE International Conference on* (pp. 685–692).

Bottaro, A., Gérodolle, A., & Lalanda, P. (2007). Pervasive service composition in the home network. In *Advanced Information Networking and Applications, 2007. AINA'07. 21st International Conference on* (pp. 596–603).

Chang, C. (2013). *Service-oriented mobile social network in proximity*. Monash University. Faculty of Information Technology. Caulfield School of Information Technology.

Chang, C., Srirama, S. N., & Ling, S. (2014). SPiCa: A Social Private Cloud Computing Application Framework. In *Proceedings of the 13th International Conference on Mobile and Ubiquitous Multimedia* (pp. 30–39). New York, NY, USA: ACM. http://doi.org/10.1145/2677972.2677979

Chen, L., Shadbolt, N. R., Goble, C., Tao, F., Cox, S. J., Puleston, C., & Smart, P. R. (2003). Towards a knowledge-based approach to semantic service composition. In *The Semantic Web-ISWC 2003* (pp. 319–334). Springer.

Cingolani, P., & Alcalá-Fdez, J. (2013). jFuzzyLogic: a java library to design fuzzy logic controllers according to the standard for fuzzy control programming. *International Journal of Computational Intelligence Systems*, *6*(sup1), 61–75.

CoAP, A. P. (2012). Coap: An application protocol for billions of tiny internet nodes.

Cubo, J., Brogi, A., & Pimentel, E. (2012). Towards behaviour-aware compositions of things in the future internet. In *Proceedings of the 2nd International Workshop on Adaptive Services for the Future Internet and 6th International Workshop on Web APIs and Service Mashups* (pp. 28–35).

Dar, K., Taherkordi, A., Baraki, H., Eliassen, F., & Geihs, K. (2014). A resource oriented integration architecture for the Internet of Things: A business process perspective. *Pervasive and Mobile Computing*.

Dornemann, T., Juhnke, E., & Freisleben, B. (2009). On-demand resource provisioning for BPEL workflows using Amazon's elastic compute cloud. In *Cluster Computing and the Grid, 2009. CCGRID'09. 9th IEEE/ACM International Symposium on* (pp. 140–147).

Fielding, R. T. (2000). *Architectural styles and the design of network-based software architectures*. University of California, Irvine.

Gama, K., Touseau, L., & Donsez, D. (2012). Combining heterogeneous service technologies for building an Internet of Things middleware. *Computer Communications*, *35*(4), 405–417.

Guinard, D. (2010). *Mashing up your Web-enabled home*. Springer.

Guinard, D., & Trifa, V. (2009). Towards the Web of things: Web mashups for embedded devices. In *Workshop on Mashups, Enterprise Mashups and Lightweight Composition on the Web (MEM 2009), in proceedings of WWW (International World Wide Web Conferences), Madrid, Spain* (p. 15).

Guinard, D., Trifa, V., Pham, T., & Liechti, O. (2009). Towards physical mashups in the Web of things. In *Networked Sensing Systems (INSS), 2009 Sixth International Conference on* (pp. 1–4).

Hirsch, F., Kemp, J., & Ilkka, J. (2007). *Mobile Web services: Architecture and implementation*. John Wiley & Sons.

Kim, S. D., Lee, J. Y., Kim, D. Y., Park, C. W., & La, H. J. (2014). Modeling BPEL-Based Collaborations with Heterogeneous IoT Devices. In *Dependable, Autonomic and Secure Computing (DASC), 2014 IEEE 12th International Conference on* (pp. 289–294).

Klein, N. (2008). China's all-seeing eye. *Rolling Stone*, *28*.

Klir, G., & Yuan, B. (1995). *Fuzzy sets and fuzzy logic* (Vol. 4). Prentice Hall New Jersey.

Kopecky, J., Vitvar, T., Bournez, C., & Farrell, J. (2007). Sawsdl: Semantic annotations for wsdl and xml schema. *Internet Computing, IEEE*, *11*(6), 60–67.

Kortuem, G., Kawsar, F., Fitton, D., & Sundramoorthy, V. (2010). Smart objects as building blocks for the internet of things. *Internet Computing, IEEE*, *14*(1), 44–51.

Lanter, M. (2013). Scalability for IoT Cloud Services.

O'reilly, T. (2007). What is Web 2.0: Design patterns and business models for the next generation of software. *Communications & Strategies*, (1), 17.

Ouyang, C., Dumas, M., Van Der Aalst, W. M. P., Ter Hofstede, A. H. M., & Mendling, J. (2009). From business process models to process-oriented software systems. *ACM Transactions on Software Engineering and Methodology (TOSEM)*, *19*(1), 2.

Palattella, M. R., Accettura, N., Vilajosana, X., Watteyne, T., Grieco, L. A., Boggia, G., & Dohler, M. (2013). Standardized protocol stack for the internet of (important) things. *Communications Surveys & Tutorials, IEEE*, *15*(3), 1389–1406.

Peltz, C. (2003). Web services orchestration and choreography. *Computer*, *36*(10), 46–52.

Perfilieva, I., & Močko\vr, J. (1999). *Mathematical principles of fuzzy logic*. Springer Science & Business Media.

Runkler, T. A. (1997). Selection of appropriate defuzzification methods using application specific properties. *Fuzzy Systems, IEEE Transactions on*, *5*(1), 72–79.

Shelby, Z. (2012). Constrained RESTful Environments (CoRE) Link Format.

Shelby, Z., Hartke, K., & Bormann, C. (2014). The Constrained Application Protocol (CoAP).

Sheng, Q., & others. (2006). *Composite Web services provisioning in dynamic environments*. University of New South Wales.

Spiess, P., Karnouskos, S., Guinard, D., Savio, D., Baecker, O., Souza, L., & Trifa, V. (2009). SOA-based integration of the internet of things in enterprise services. In *Web Services, 2009. ICWS 2009. IEEE International Conference on* (pp. 968–975).

Srirama, S. N., Jarke, M., & Prinz, W. (2006). Mobile Web service provisioning. In *Telecommunications, 2006. AICT-ICIW'06. International Conference on Internet and Web Applications and Services/Advanced International Conference on* (p. 120).

White, S. A. (2004). Introduction to BPMN. 2004. *IBM Corporation*.

Zadeh, L. A. (1965). Fuzzy sets. *Information and Control*, *8*(3), 338–353.

Zhang, X., Song, P., & Qin, X. (2008). An analysis of tourism economic growth point: an empirical study based on the effects of "golden week." *Tourism Tribune*, *23*(10), 16–22.

I. License

Non-exclusive licence to reproduce thesis and make thesis public

I, **Wei Ding** (date of birth: 23.03.1991),

(*author's name*)

1. herewith grant the University of Tartu a free permit (non-exclusive licence) to:
   1.1. reproduce, for the purpose of preservation and making available to the public, including for addition to the DSpace digital archives until expiry of the term of validity of the copyright, and
   1.2. make available to the public via the Web environment of the University of Tartu, including via the DSpace digital archives until expiry of the term of validity of the copyright,

of my thesis

A Framework for Collaborative Content Mashup with Pervasive Services,

(title of thesis)

supervised by Chii Chang, Satish Narayana Srirama

(supervisor's name)

2. I am aware of the fact that the author retains these rights.

3. I certify that granting the non-exclusive licence does not infringe the intellectual property rights or rights arising from the Personal Data Protection Act.

Tartu, **27.05.2015**