UNIVERSITY OF TARTU

Institute of Computer Science
Software Engineering Curriculum

Jakob Mass

An Adaptive Mediation Framework for Workflow Management in the Internet of Things

Master's Thesis (30 ECTS)

Supervisor:	Chii Chang, PhD
Supervisor:	Satish Narayana Srirama, PhD

Tartu 2016

Kohanemisel põhinev vahendusraamistik võimaldamaks töövoohaldust värkvõrgus

Lühikokkuvõte: Tärkavad värkvõrksüsteemid koosnevad arvukast hulgast heterogeensetest füüsilistest seadmetest mis ühenduvad Internetiga. Need seadmed suudavad pidevalt ümbritseva keskkonnaga suhelda ja osana lõppkasutaja rakendusestest edendada valdkondi nagu tark kodu, e-tervis, logistika jne. Selleks, et integreerida füüsilisi seadmeid värkvärgu haldussüssteemidega, on *töövoo haldussüsteemid* kerkinud esile sobiva lahendusena. Ent töövoo haldussüsteemide rakendamine värkvõrku toob kaasa reaalajas teenuste komponeerimise väljakutseid nagu pidev teenusavastus ja -käivitus. Lisaks kehtib küsimus, kuidas piiratud ressurssidega värkvõrgu seadmeid töövoo haldussüsteemidega integreerida ning kuidas töövooge värkvõrgu seadmetel käitada. Tööülesanded nagu pidev seadmeavastus võivad värkvõrgus osalevatele piiratud arvutusjõudluse ja akukestvusega seadmetele nagu nutitelefonid koormavaks osutuda. Siinkohal on võimalikuks lahenduseks töö delegeerimine pilve. Käesolev magistritöö esitleb kontekstipõhist raamistikku tööülesannete vahendamiseks värkvõrgurakendustes. Antud raamistikus modelleeritakse ning käitatakse tööülesandeid kasutades töövoogusid. Raamistiku prototüübiga läbi viidud uurimus näitas, et raamistik on võimeline tuvastama, millal seadmeavastusülesannete pilve delegeerimine on kuluefektiivsem. Vahel aga pole töövoo käitamistarkvara paigaldamine värkvõrgu seadmetele soovitav, arvestades energiasäästlikkust ning käituskiirust. Käesolev töö võrdles kaht tüüpi töövookäitust: a) töövoo mudeli käitamine käitusmootoriga ning b) töövoo mudelist tõlgitud programmikoodi käitamine. Lähtudes katsetest päris seadmetega, võrreldi nimetatud kahte meetodit pidades silmas süsteemiressursside- ning energiakasutust.

Võtmesõnad: Värkvõrk, Töövoohaldussüsteemid, teenuskompositsioon, Teenusavastus, SOA, Teenusorienteeritud arhitektuur, Mobiili- ja pilvearvutus, Edge network, Töövookäitus

CERCS: Arvutiteadus, arvutusmeetodid, süsteemid, juhtimine (automaatjuhtimisteooria) (P170)

An Adaptive Mediation Framework for Workflow Management in the Internet of Things

Abstract: Emerging Internet of Things (IoT) systems consist of great numbers of heterogeneous physical entities that are interconnected via the Internet. These devices can continuously interact with the surrounding environment and be used for user applications that benefit human life in domains such as assisted living, e-health, transportation etc. In order to integrate the frontend physical things with IoT management systems, Workflow Management Systems (WfMS) have gained attention as a viable option. However, applying WfMS in IoT faces real-time service composition challenges such as continuous service discovery and invocation. Another question is how to integrate resource-contained IoT devices with the WfMS and execute workflows on the IoT devices. Tasks such as continuous device discovery can be taxing for IoT-involved devices with limited processing power and battery life such as smartphones. In order to overcome this, some tasks can be delegated to a utility Cloud instance. This thesis proposes a context-based framework for task mediation in Internet of Things applications. In the framework, tasks are modelled and executed as workflows. A case study carried out with a prototype of the framework showed that the proposed framework is able to decide when it is more cost-efficient to delegate discovery tasks to the cloud. However, sometimes embedding a workflow engine in an IoT device is not beneficial considering agility and energy conservation. This thesis compared two types of workflow execution: a) execution of workflow models using an embedded workflow engine and b) execution of program code translations based on the workflow models. Based on experiments with real devices, the two methods were compared in terms of system resource and energy usage.

Keywords: Internet of Things, Workflow Management Systems, Service Composition, Service discovery, Service-Oriented Architecture, Mobile Cloud, Edge network, Workflow Execution, Workflow Translation

CERCS: Computer science, numerical analysis, systems, control (P170)

Contents

1 Introduction **8**

1.1 Preamble . . . 8

1.2 Motivation . . . 8

1.2.1 Example Scenario . . . 9

1.2.2 Challenges . . . 9

1.3 Research Objectives and Contribution . . . 11

1.4 Thesis Outline . . . 11

1.5 Publications . . . 11

2 Literature Review **13**

2.1 Internet of Things . . . 13

2.1.1 Connectivity . . . 13

2.1.2 The Features of an IoT sytem . . . 14

2.2 Service Oriented Architecture . . . 15

2.3 Service-Oriented IoT . . . 16

2.3.1 Virtualisation of Things . . . 16

2.3.2 Middleware Architecture . . . 17

2.3.3 Discovery . . . 18

2.3.4 Service Description for IoT . . . 18

2.4 Mobile Cloud Computing . . . 21

2.4.1 Task Delegation and Code Offloading . . . 21

2.5 Workflow Management Systems . . . 22

2.5.1 Workflow Languages . . . 22

2.5.2 Business Process Management Life Cycle . . . 23

2.5.3 Orchestration and Choreography . . . 24

2.6 Mobile Workflow Management Systems for IoT . . . 24

2.6.1 Modelling IoT for WfMS . . . 24

2.6.2 Implementing and executing workflows in IoT systems . . . 25

3 System Overview **26**

3.1 Scenario . . . 26

3.2 SCORPII framework design . . . 27

3.2.1 SCORPII - Mobile Host Side (ScoMH) . . . 28

3.2.2 SCORPII - Utility Cloud Side (ScoUC) . . . 29

3.3 Cost-efficient and Context-aware Workflow Approach Selection Model 30

3.3.1 Preliminary . . . 30

3.3.2 Time-span of Approach . . . 33

3.3.3 Raw Cost Elements and Context Parameters of Approach . 34

3.3.4 Cost-Performance Index-based Approach Selection . . . 34

3.3.5 Context-aware weight calculation of the cost elements 35
3.4 SCORPII Prototype 36

4 Adaptive Workflow Execution in the Internet of Things 37
4.1 Overview 37
4.2 Workflow Translator implementation 39
4.2.1 Code Generation 39
4.2.2 Groovy language example 39
4.2.3 Workflow Engine 40
4.2.4 Test Scenario 40

5 Evaluation 43
5.1 Dynamic Configuration Use Case 43
5.2 Dynamic Configuration Evaluation 44
5.2.1 Context-aware Cost Element Weighing 46
5.3 CPI Scores and Approach Selection 49
5.4 Workflow Execution Approach Comparison 51
5.4.1 Bootstrapping 51
5.4.2 Process Execution Time 51
5.4.3 System Resource and Power usage 53
5.5 Bootstrapping and Resource Consumption of Deployment 56
5.6 Discussion 56

6 Conclusion 57
6.1 Future Research Direction 57

List of Figures

1 Example environment with smart objects 10
2 The general SOA model 15
3 A general service-oriented IoT system architecture 17
4 Example of sensor data represented using JSON-LD 19
5 Example of Air Temperature sensor and its value represented using SensorML 2.0 20
6 Simple BPMN 2.0 process model example 23
7 Service discovery scenario. 26
8 Architecture of SCORPII middleware framework. 28
9 Workflow patterns. 33
10 System design overview 37
11 BPMN 2.0 Business process model of parallel tasks 41
12 BPMN 2.0 BP model with no parallel tasks 41
13 Default service workflow (simplified) 43
14 Bandwidth measurement comparison 44
15 CPU usage comparison 45
16 Time-span comparison 45
17 CPI comparison of the two approaches in the test scenario with *Context A* 48
18 CPI comparison of the two approaches in the test scenario with *Context B* 50
19 Tomcat startup CPU usage 52
20 Tomcat startup Memory usage 52
21 Tomcat startup power consumption 52
22 CPU usage during execution (Parallel BP) 54
23 Memory usage during execution (Parallel BP) 54
24 Power consumption during execution (Parallel BP) 54
25 CPU usage during execution (Non-Parallel BP) 55
26 Memory usage during execution (Non-Parallel BP) 55
27 Power consumption during execution (Non-Parallel BP) 55

List of Tables

1 Context parameters used in the regression training dataset 46
2 Fuzzy representation of the context parameters and weights 47
3 Weight formulas with regression coefficients plugged in 48
4 Final weights of the cost elements with *Context A* 49
5 Final weights of the cost elements with *Context B* 50
6 Parallel BP execution times 51
7 Non-Parallel BP execution times 53

1 Introduction

1.1 Preamble

New, emerging technologies in the fields of mobile and ubiquitous computing promote the phenomena where heterogeneous physical entities connect with the Internet via various embedded services and lightweight wireless communication protocols. This development is called the Internet of Things (IoT). Cisco predicts that by 2020, the number of *things* connected to the Internet will reach 50 billion [Eva11].

IoT bears the potential of greatly influencing (and improving) the way we work and live. In our daily lives, IoT may enhance assisted living, domotics, e-health, transportation and so forth. Businesses are also expected to benefit greatly, e.g. via enhanced logistics, industrial automation, etc [AIM10].

However, introducing various types of things into the same network brings along challenges such as overcoming the heterogeneous nature of the things in terms of varying communication protocols, operating systems and software stack. Furthermore, the framework which integrates information systems with physical things should offer an easy-to-use interface for designing the tasks and processes assigned to the things. The framework should be able to re-configure and optimize how the things function together without the need of performing low-level configuration on each individual thing.

One solution for overcoming this issue of management and composition of *things* is to use Workflow Management Systems (WfMS). A workflow is a sequence of tasks, events and decisions. Workflow management is the field of designing, executing and observing work sequences and providing methods for improving and managing the work efficiently.

WfMS can benefit IoT by providing a language to define arbitrary processes involving the *things* and the functionalities they offer. Secondly, WfMS provide the methodology for monitoring, interpreting and improving how these processes are run. By using workflow modelling languages such as Web Services Business Process Execution Language (WS-BPEL) or BPMN 2.0 [bpm], analysts can focus on workflow design, while developers can focus on providing software for integrating the *things* with WfMS.

1.2 Motivation

WfMS for IoT has applications such as in logistics, where cargo could be monitored by a management system in real-time, triggering events immediately after something happens (e.g. the temperature of the goods exceeds a threshold) [GEPF11]. In the smart building domain, a building's heating, ventilating, and air conditioning (HVAC) systems can benefit from a WfMS. For instance, a hotel could keep track of electricity and heating usage per room, allowing the system to bill the customer according to the usage instead of charging a fixed rate [TSD+12]. The healthcare domain is another very promising target for WfMS IoT applications. In healthcare scenarios, a person is often equipped with wearable sensor devices such as a heart rate and temperature monitor. The sensor data is collected by a broker device and may be forwarded to a remote system, such as a hospital information system, where a doctor is able to use it to analyse patient status [DTRE11].

1.2.1 Example Scenario

Consider the following scenario: *Alice* is travelling through urban space inhabited by a large number of smart objects (see Fig. 1). These include smartphones, sensors, media content providers, such as for example text and video content associated with a historic statue near Alice (object A in Fig. 1). Alice is equipped with a smartphone, which gathers data from nearby *things* while she is walking. For example, her smartphone may collect noise level, temperature and light levels of various locations in the current area. The smartphone combines these functions of the proximal smart objects with additional services from the web, such as news reports, social media or weather broadcasts. The composition of all this content results in a mobile application for aiding Alice in her travels. However, in some cases, Alice's smartphone does not support the communication standards used by certain smart objects (e.g. device C in Fig. 1) . In such cases, Alice's smartphone uses another type of smart object nearby (object B in Fig. 1), a gateway, which gathers the data from sensor objects and provides them to Alice's smartphone.

This thesis aims to conceive a middleware platform for realizing such scenarios. The proposed **S**ervice-oriented **C**omposer for **O**rchestrating **R**eal-time **P**roximity-based **I**ndustrial **I**nternet of Things (SCORPII), as we call it, manages the work (sensor data gathering, service composition) and objects involved in the scenario by means of using a service-oriented WfMS. As such, it is assumed that the smart objects are capable of executing workflows or being invoked as part of workflows.

1.2.2 Challenges

Service Discovery In the above scenario however, before the smartphone can retrieve useful data from nearby smart objects, the services (functionality) offered by them must be discovered. The services provided by the smart objects are described in Service description metadata (SDM), which can be obtained in 2 ways:

1. Using direct peer-to-peer communication (such as Bluetooth, WiFi Direct, etc.), to retrieve the SDM directly from the device [Rag15].

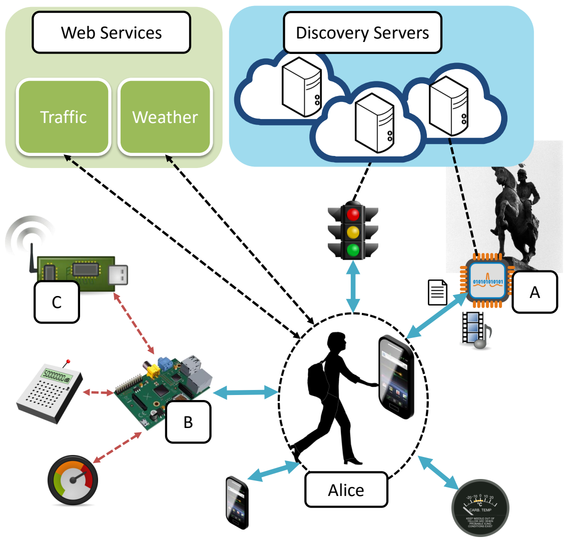


Figure 1: Example environment with smart objects

2. The smart object provides (e.g. via Radio-frequency identification [RFID]) an address to a cloud discovery server corresponding to the smart object [Goo] (see Fig. 1). This cloud service then provides the SDM. This option is useful when the smart object is a more constrained device in terms of communication capabilities.

Alice's smartphone then processes the SDMs of the nearby devices to filter out the ones which provide the desired services that fit the current workflow requirements. However, as the number of nearby devices can be large, processing all these SDMs can be cumbersome for the smartphone. To remedy this, the task of filtering SDMs could be offloaded to the cloud. On the other hand, if the number of devices near Alice is small, processing SDMs locally on smartphone may be preferred to improve responsiveness of the user application.

This raises two questions:

- *How does Alice's smartphone decide which processes require computational task offloading and when?*
- *How does Alice's smartphone perform the configuration dynamically at run-time?*

Execution approaches A second matter of interest in the previously described IoT scenario is the question of *how to execute workflows on smart objects?* One option is to embed the smart object with a workflow engine capable of directly executing workflow models [DTB+15]. The alternative is to translate the workflow model into executable code and execute the program code on the smart object [CK11b, GEPF11, CDD+12]. While both approaches have been previously explored, a direct comparison of the two is missing to the best of the authors knowledge.

1.3 Research Objectives and Contribution

This thesis aims to reach two goals.

1. To investigate, develop and validate an adaptive middleware framework for proximity based service composition in IoT.
2. Secondly, to develop and validate a testing environment for providing a guideline for developers in the domain of workflow execution on resource-constrained devices. More precisely, the guideline focuses on comparing executing a workflow using as input:
   (a) a model of the workflow
   (b) executable code corresponding to the model.

1.4 Thesis Outline

The rest of this thesis is structured as follows. Section 2 provides a background of technologies and related works. Section 3 details the design and implementation of the SCORPII framework and section 4 describes the workflow execution extension for resource-constrained devices. Section 5 compares the performance of the decision-making mechanism in SCORPII and provides a comparison of workflow execution approaches. Finally, Section 6 concludes and summarizes the work done in this thesis and provides future research directions.

1.5 Publications

Publications involved in this thesis are listed as follows.

1. C. Chang, S. N. Srirama, J. Mass: *A Middleware for Discovering Proximity-based Service-Oriented Industrial Internet of Things*, 12th IEEE International Conference on Services Computing (SCC 2015), June 27 - July 2, 2015, pp. 130-137. IEEE.
2. C. Chang, S. N. Srirama, J. Mass: *Adaptive Mobile Cloud Workflow Management System for Service Discovery in Proximity-Based Internet of Things*,

International Journal of Services Computing (IJSC), ISSN: 2330-4472, 3(1):44-56, 2015. The Services Society (SS).

3. J. Mass, C.Chang, S. N. Srirama: *Workflow Model Distribution or Code Distribution? Ideal Approach for Service Composition of the Internet of Things*, 13th IEEE International Conference on Services Computing (SCC 2016) (*Accepted for publication*)

2 Literature Review

This section explains terms, concepts and previous research relevant to this thesis, including the Internet of Things, Service Oriented Architecture, Mobile Cloud Computing and Workflow Management Systems.

2.1 Internet of Things

The term Internet of Things (IoT) was coined by Kevin Ashton in 1999 [Ash09], but has since been used to describe various visions, resulting in different definitions. The IEEE Internet Initiative has published an entire report [MBR15] aimed at defining IoT. For instance, Vermesan et al. define:

> "The goal of the Internet of Things is to enable things to be connected anytime, anyplace, with anything and anyone ideally using any path/network and any service." [VFG$^+$15]

In short, IoT marks the rapid spread of devices with communicating, sensing and actuating capabilities [GBMP13]. Communication is usually achieved using the Internet, but can also involve proximal communication such as Bluetooth. The sensing and actuating abilities of the devices bring together the daily environment of humans and the connected devices. Using sensors, smart objects can capture information from their surroundings, e.g. gather temperature or light readings. With actuator capabilities, smart objects can influence their surroundings, for example toggle switches, open doors.

2.1.1 Connectivity

As the name suggests, one of the fundamental ideas of IoT is the fact that all things which make up an IoT system are connected to the Internet. This is supported by the Internet Protocol (IP) standard, the latest iteration of which, IPv6, theoretically allows for 2^{128} unique addresses. This means that using IPv6 virtually all the objects of our daily lives can be uniquely addressed.

Thanks to advances in wireless technology, the physical size of modern wireless radio modules is minuscule, thus extending even very small objects with wireless capability comes with insignificant overheard in terms of physical size.

In addition to being connected, the IoT objects interact with their surrounding world in various forms. They may collect information from their surroundings (using sensors) or they interact with the physical world by either using actuators or forwarding commands to devices which manipulate the physical world, such as a light.

In the first IoT conceptualizations, the things were simply represented by Radio-Frequency IDentification (RFID) tags. Between 1999 and 2003 institutions Auto-ID Labs [aut] and EPCGlobal [TAB+15] introduced the concept of attaching low-cost RFID tags on any products so that they could be tracked. The information about the products could be retrieved up from databases via the Internet [MBR15]. Later, the IoT vision evolved beyond only RFID-enhanced objects, including devices with embedded computing capabilities and using other technologies instead of RFID such as Near Field Communication (NFC) or QR-codes (Quick Response Code).

2.1.2 The Features of an IoT sytem

A IEEE Internet Initiative report [MBR15] highlights some distinct features of an IoT system:

- **Interconnection of Things**: the things are able to communicate with one another.
- **Connection of Things to the Internet**: the things are connected to the Internet, thus an Intranet is not qualified as an IoT network.
- **Uniquely Identifiable Things**: each of the things within the IoT system are uniquely identifiable
- **Ubiquity**: this feature refers to the "anytime, anyplace" portion of the definition mentioned previously. Ubiquity represents the idea that the IoT system is available for use whenever necessary for the application scenario, and that IoT scenarios can be found in any application field (smart home, assisted living, healthcare, logistics, etc.).
- **Sensing/Actuation capability**: the physical objects are enhanced by sensor / actuator devices, this adds smart behaviour to the objects.
- **Embedded Intelligence**: the smart objects contain intelligence and knowledge capabilities which allow them to become tools for aiding human life
- **Interoperable Communication Capability**: the communication of the Things is based on established standards and protocols.
- **Self-configurability**: in the heterogeneous environment of IoT, the things have self-management capabilities such as service discovery and network organization. [CHK+12]
- **Programmability**: behaviour of the thing can be modified without physical changes.

2.2 Service Oriented Architecture

Service-Oriented Architecture (SOA), also knowns as Service-Oriented Computing (SOC), is a computing paradigm in which the fundamental building blocks for software solutions are software services.

Software services are interfaces that enable devices and their hosted applications to provide their functionalities while being platform-independent and self-contained [Pap03, GP08]. Self-containment here means that the service provider manages its state independently of clients using the software service.

Loose coupling is a feature where each component of a software system utilizes little knowledge about other components in the same system. Loose coupling is fundamental to SOA systems, allowing entities involved in the system to be interacted with using a higher level of abstraction.

Additionally, SOA services support dynamic service discovery and composition. Dynamic service discovery allows clients to query some third-party discovery service to identify which service(s) can meet their requirements.

Service composition allows a system to create a service that utilises the mechanisms derived from other providers [GP08]. For example, a location-based service can integrate Google Maps[1] and Foursquare[2] to provide point-of-interest information in a map-review based application.

When a service provides its features using application layer Internet standards, such as Hypertext Transfer Protocol (HTTP), XML or REST, it is called a *Web service*. Web services are well compatible with the fundamental SOA aspects such as platform independence.

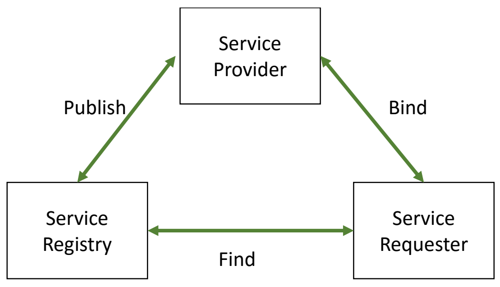


Figure 2: The general SOA model

The model of SOA consists of three types of participants [IGH+11, Pap03]:

[1]https://developers.google.com/maps/
[2]https://developer.foursquare.com/

1. Service providers, the endpoints of the services. These are the devices which provide the functionality contained in the service.

2. Service requesters, which are the clients who use the functionalities provided by the services.

3. Service registries, who provide information about services providers so that service requesters can discover them.

The interaction between these three has been illustrated in Figure 2. Service Providers notify Service Registries of their existence and their capabilities, after which the Service Registry makes this information (the service descriptor) discoverable to others. Using the Service Registry, Service Requesters can discover Service Providers that match their needs and get the necessary metadata to enable service invocation between the Requester and Provider.

2.3 Service-Oriented IoT

SOA has become an important paradigm for realizing IoT middlewares. A middleware is a software layer (or layers) standing between two levels of a software solution. An IoT middleware allows application developers to focus on their domain (e.g. smart health) without concerning themselves with details about other lower layer technologies such as network protocols.

IoT middleware architectures proposed in the recent years often follow the SOA approach [AIM10, THIG11]. Following the SOA model enables structuring large, complex systems in a way where the individual components are well-defined and can be easily composed and re-composed.

2.3.1 Virtualisation of Things

As an IoT system consists of a large group of heterogeneous objects using different communication methods and protocols, it is desirable to have a layer which abstracts the different technologies so that they can be used with a common language [AIM10]. Using SOA, the heterogeneity and isolation issues of IoT systems can be addressed. By abstracting the IoT devices, *things* can be utilised as atomic services or they can be used to form a composite service. In practice, Web services have often been used for this [GTW10, DGV09, SJP06].

Web service standards such as SOAP, WSDL etc. reduce the need for gateway devices and translations between the components [GTK$^+$10]. They also allow orchestration of the services with higher-level Enterprise Resource Planning (ERP) applications. Generally, Web service-based IoT systems are implemented using

Web Service (WS) protocols such as Device Profiles for Web Services (DPWS), Constrained Application Protocol (CoAP) or REST.

For connecting the resource-constrained smart objects to the Internet, conventional networking standards are not sufficient. A key concept, *IPv6 over low-power wireless area networks* (6LoWPAN), aims to realize IPv6 usage on the low-power smart objects with limited processing capability. 6LoWPAN simplifies IPv6 functionality, by restricting header formats to very compact sizes. At the same time, the established benefits of Internet protocols are preserved, such as re-using existing network infrastructure, existing standards (such as TCP, UDP, HTTP, CoAP, MQTT), APIs and tools for managing and using IP-networks. [SB11]

In this way, IoT devices act as the service providers in SOA.

2.3.2 Middleware Architecture

This section describes the common architecture used in existing SOA-based IoT systems, such as MOSDEN [PJZ+14] (part of European Commision project "OpenIoT" [3]) and Mobile Digcovery [JLF+14].

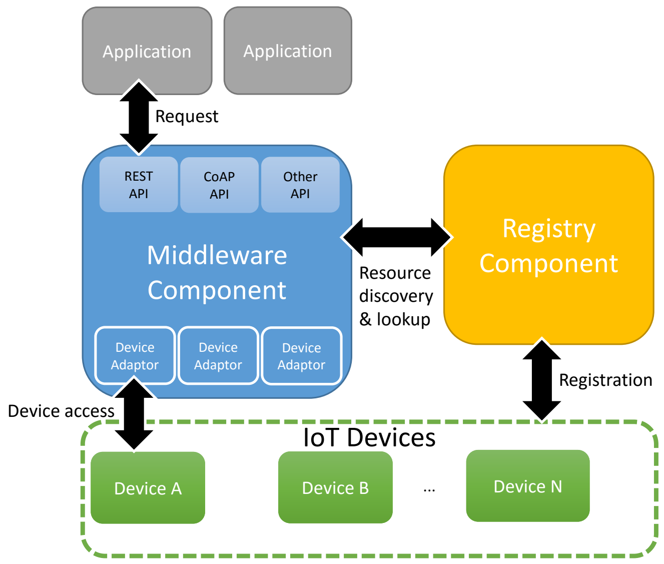


Figure 3: A general service-oriented IoT system architecture

As presented in Figure 3, end user applications (such as Web or mobile applications) make requests to a middleware component. This request does not need

[3]http://cordis.europa.eu/project/rcn/101534_en.html

to specify which protocol or technology is needed to communicate with the IoT devices because the process is handled by the middleware.

The middleware uses a registry component to query for devices which can be used to create a composite service for fulfilling the requirements of the request.

The Registry component is capable of providing the information of the registered things (e.g. their supported functionalities) to the middleware. After the middleware component has used the Registry component to look up appropriate devices to fulfil the request, the middleware can access the devices to execute the subtasks which compose the original application request. Access to the devices is provided by adaptor components within the middleware. These adaptor components allow accessing the heterogenous devices which may be using different protocols and technologies.

Finally after having accessed the device, the middleware can process the results and respond to the user application with the final request response.

2.3.3 Discovery

Technologies enabling the discovery of *things* and their details can be divided into two: a) *global network*-based and b) *edge network*-based approaches.

Global network based approaches In this approach, a global registry, accessible anywhere via the Internet, is used to manage a listing of devices and their metadata. There, devices and details regarding their supported features and protocols can be queried. Examples of this are [PJZ$^+$14, JLF$^+$14].

Edge network based approaches In this approach, the smart object being discovered itself serves as the entry-point for obtaining information about the device. The smart object may provide information about itself using technologies such as Bluetooth Low Energy[4], Wi-Fi Aware[5] or ZigBee[6]. However, to maintain energy-efficiency for the smart object, the device may provide an address to an external cloud server which provides the details about the smart object. This has been done for example by: Google Physical Web [Goo] or Apple iBeacon [Ap].

2.3.4 Service Description for IoT

To be able to advertise smart objects and their capabilities in a formal way, several standards have emerged. These standards enable discovering the heterogenous objects in a structured and machine-readable fashion. By adding semantic data

[4]`https://www.bluetooth.com/`
[5]`http://www.wi-fi.org/discover-wi-fi/wi-fi-aware`
[6]`http://www.zigbee.org/`

to the service description of the IoT nodes, the system can support autonomous service invocation.

While metadata description has been well covered in the domain of Semantic Web, the standards therein such as Resource Description Framework (RDF)[G+04] do not fit the constrained resource requirements set by IoT [SRN+15]. Here we list some standards for IoT device metadata representation.

Sensor Markup Language (SenML) [JAS12] is a data model for connecting sensors and actuators to the Internet by providing device parameters and measurements. The format is designed to be able to transport sensor data using devices that are very limited in terms of hardware specification (computing power). SenML can be serialized using JSON, XML and Efficient XML Interchange (EXI) and can be extended with custom attributes.

JSON for Linked Data (JSON-LD) is a format which aims to represent the RDF metadata model in a JSON representation. JSON-LD is endorsed by the World Wide Web Consortium as a promising attempt among the several approaches to capturing RDF semantics using JSON [SRN+15]. Figure 4 shows some sensor data represented in the JSON-LD format.

```
{
  "@context":
  {
  "i": "http://iot.fi/o#",
  "accX": "i:accX", "accY": "i:accY", "accZ": "i:accZ",
  "magX": "i:magX", "magY": "i:magY", "magZ": "i:magZ",
  "temp": "i:temp", "timeStamp": "i:timeStamp"
  },
  "@id": "i:accmagSensor01",
  "@type": "i:Sensor",
  "accX": "618", "accY": "319", "accZ": "671",
  "magX": "123", "magY": "234", "temp": "22.5",
  "timeStamp": "2012-05-18T12:00:00"
}
```

Figure 4: Example of sensor data represented using JSON-LD [SRN+15]

Sensor Model Language (SensorML) [BRGW14] is a standard for modelling and encoding arbitrary sensor-related processes. In SensorML, processes include hardware devices such as sensors and actuators. Processes are defined by their input, output, methods and parameters. As such, SensorML processes are discoverable and executable.

An example of a temperature sensors reading formatted using SensorML 2.0 is presented in figure 5

```
<?xml version="1.0" encoding="UTF-8"?>
<sml:PhysicalComponent gml:id="MY_SENSOR"
xmlns:sml="http://www.opengis.net/sensorml/2.0"
xmlns:swe="http://www.opengis.net/swe/2.0"
xmlns:gml="http://www.opengis.net/gml/3.2"
xmlns:gmd="http://www.isotc211.org/2005/gmd"
xmlns:gco="http://www.isotc211.org/2005/gco"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xmlns:xlink="http://www.w3.org/1999/xlink"
xsi:schemaLocation="http://www.opengis.net/sensorml/2.0␣http://schemas.opengis.
    net/sensorml/2.0/sensorML.xsd">
   <!-- ================================================= -->
   <!-- System Description -->
   <!-- ================================================= -->
   <gml:description> Temperature sensor on my window </gml:description>
   <gml:identifier codeSpace="uid">myCompany.com.63547</gml:identifier>
   <!-- ================================================= -->
   <!-- Observed Property = Output -->
   <!-- ================================================= -->
   <sml:outputs>
      <sml:OutputList>
         <sml:output name="temp">
            <swe:Quantity definition="http://sweet.jpl.nasa.gov/2.2/
                quanTemperature.owl#Temperature">
               <swe:label>Air Temperature</swe:label>
               <swe:uom code="Cel"/>
            </swe:Quantity>
         </sml:output>
      </sml:OutputList>
   </sml:outputs>
   <!-- ================================================= -->
   <!-- Sensor Location -->
   <!-- ================================================= -->
   <sml:position>
      <gml:Point gml:id="stationLocation" srsName="http://www.opengis.net/def/
          crs/EPSG/0/4326">
         <gml:coordinates>47.8 88.56</gml:coordinates>
      </gml:Point>
   </sml:position>
</sml:PhysicalComponent>
```

Figure 5: Example of Air Temperature sensor and its value represented using SensorML 2.0
[Bot]

Device Profiles for Web Services (DPWS) is a set of Web service specifications for enabling secure messaging, discovery, description and eventing that is suitable for automation systems.

2.4 Mobile Cloud Computing

As described by [BYV$^+$09], Cloud Computing (CC) is a paradigm in which "*computing is being transformed to a model consisting of services that are commoditized and delivered in a manner similar to traditional utilities such as water, electricity, gas, and telephony.*"

In cloud computing, users have at their disposal the exact amount of computing power they need and exactly as long as they need it, available for use anywhere. In other words, CC provides users with infrastructure (such as servers, networks and storage), software (middleware services and platforms) in an on-demand fashion, i.e. the resources can be used elastically [DLNW13].

The major providers of Cloud computing applications and platforms today are Amazon, Google, Salesforce, Microsoft. They provide various configurations that are suitable for integration with different devices, including mobile phones (smartphones).

The significant emergence of smartphones within the last decade together with the maturing of Cloud Computing has motivated Mobile Cloud Computing (MCC). In short, MCC is an infrastructure where data processing (and data storage) takes place outside of the mobile device.

Via MCC, the mobile environment inherits the features of CC such as elasticity and on-demand usage [DLNW13]. As modern smartphones provide applications and services which aggregate several data sources, involved computational tasks can be quite resource intensive. For example, processing high resolution images or processing natural language on smartphones can be both time- and energy consuming. By utilising MCC, the resource intensive tasks can be offloaded to external computational resources in order to improve the overall performance of the process [FLR13a].

In general, computational offloading in MCC can be performed in two approaches: *task delegation* and *mobile code offloading* [FS14].

2.4.1 Task Delegation and Code Offloading

Task delegation represents a computational process, where a process that was originally to be performed on the mobile device, is offloaded to a cloud service which has an equivalent mechanism to execute the process. Task delegation involves mobile network communication, which in some cases can cause extra latency. Hence, when a system involves task delegation, the overall cost and performance of the process need to be considered.

On the other hand, according to the **offloading** model, portions of software may be detected to require too much computational effort and will be accordingly executed either locally or remotely. The remote execution commonly uses an approach similar to traditional Remote Procedure Call (RPC) methods [FLR13b]. When using this model, the questions of what, when, where and how to offload must be answered. To determine which parts of code could be offloaded, code profilers or manual code annotations are often used [FHT+15]. To determine when the system should try to save energy by offloading, system profilers can be used [FLR13b].

2.5 Workflow Management Systems

Workflow Managements Systems (WfMS) are a suiting approach for service composition [SQV+14, RS04]. A *workflow* is a sequence of tasks, events and decisions. Some of these elements of the sequence may happen in parallel. Alternatively, a workflow may be called a business process (BP) [VDAVH04].

Workflow management is the field of observing and designing workflows and providing methods and software tools for managing and improving the work being carried out.

Workflow Management Systems are generic software packages for managing business processes [VDAVH04]. WfMS distribute work to to executors (actors) based on a process model [DLRMR13]. Because the system automates the execution and distribution of tasks, it is easy to introduce changes to the workflow model, e.g. change the order of tasks done.

However, this process-centric perspective of information systems lacks a general agreement (standard) on fundamental concepts [vdA13]. As a result, a large number of different languages for modelling workflows exist. As described by Dumas et al. [DLRMR13], a workflow modelling language generally involves two basic types of nodes: activity nodes and control nodes. Activity nodes represent work that may be executed by a human, software agent or a combination of them. Control nodes represent the flow of execution between activities (the arrangement of work within the sequence). Additionally, event nodes are often also included in a workflow modelling language. Event nodes represent events which may happen within the workflow or execution environment, to which the workflow can then react to. For example, upon receiving a message (an event), the user may be asked to respond to it (an activity).

2.5.1 Workflow Languages

In this subsection, the major workflow modelling languages are summarised. One of the most well known workflow languages is Business Process Model and Notation

(BPMN 2.0) [bpm], a standard introduced by the Object Management Group (OMG) in 2011. BPMN is a graphical notation. BPMN targets to allow business analysts and system architects to describe processes, resulting in a higher level of abstraction which means that direct execution of BPMN models is not usually existent in BPMN software tools [ODTHVdA06]. Figure 6 presents a simplistic BPMN 2.0 process model, including parallel execution of tasks and a decision point.

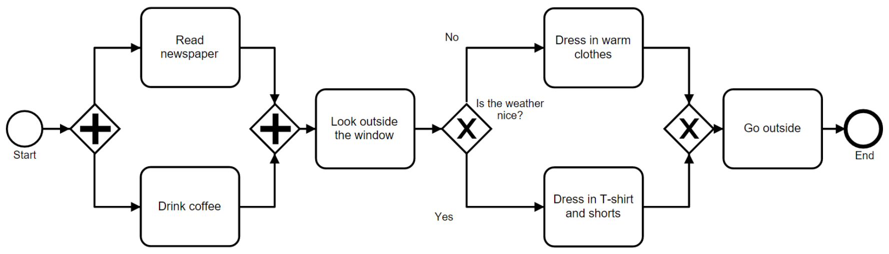


Figure 6: Simple BPMN 2.0 process model example

Web Services Business Process Execution Language (WS-BPEL) [AAA$^+$06], a standard of the Organization introduced by the Organization for the Advancement of Structured Information Standards (OASIS), provides a more formal and strict way to describe business processes, integrated with Web service technologies. Because WS-BPEL describes precise semantics for process execution, WS-BPEL based software tools can directly execute WS-BPEL models.

Introduced by the Workflow Management Coalition (WfMC), the XML Process Definition Language (XPDL) [Int01] is a format that aims to be platform-independent, meaning compatibility across different modelling and management tools. In essence, XPDL is an exchange format for process definitions between different workflow languages, it may be considered as the serialization Format for BPMN [xpd].

Comparing these languages, BPMN's main focus is visualization and communication of workflows, WS-BPEL focuses on execution with the integration of Web Services and XPDL aims at serialization and documentation [LW13].

2.5.2 Business Process Management Life Cycle

The workflow life cycle can be broken down into three distinct phases: *(re)design phase*, *implement/configure phase* and the *run and adjust* phase [vdA13].

In the *(re)design* phase, the process model is designed. Considering IoT-based WfMS, this involves the question how the smart objects of IoT are represented in the model. The *implement/configure* phase involves implementing a mapping so that the designed model can be enacted by a workflow engine. The *run and* *adjust* phase corresponds to how the WfMS executes the process. In this phase, the process model itself is not redesigned, but configuration adjustments do take place. Executing the process results in log traces which depict how the execution went. This can then be analysed to further improve the process model. As such, the life-cycle begins a new, as the (re)design phase is entered again.

2.5.3 Orchestration and Choreography

WfMS can generally be divided into two architectures: **orchestration** and **choreography**. The idea of orchestration follows a centralized architecture, where the entire business process is managed by a single management system. Choreography follows a de-centralized architecture, where the process model (or parts of it) are handled by a several external information systems. While orchestration has been the dominant approach in existing IoT-related WfMS, recently, the importance of using choreography for IoT has been recognized. Considering the IoT environment in which participating devices require agile reaction, a centralized, orchestration approach may be insufficient [DTRE11]

2.6 Mobile Workflow Management Systems for IoT

As mentioned in section 2.3, Service Oriented IoT can abstract the heterogeneous IoT environment into a more uniform set of services. Using WfMS allows for service composition to support any kind of work sequences that are also easy to monitor and manage.

2.6.1 Modelling IoT for WfMS

Bringing IoT objects into WfMS is not a straightforward task, as the objects typically involve different communication protocols, network topologies and hardware specifications. There are two main approaches to modelling IoT objects for usage in WfMS:

1. **Things as services.** In this approach, the smart objects are expressed as network services that can be invoked by following the request-response model. The network service embedded in the smart object can be implemented e.g. as a SOAP service [PRS$^+$13] or a REST-based service [DTB$^+$15]. Hence, existing modelling standards such as BPMN are left unmodified.

2. **Defining new IoT elements.** Conventional WfMS assume that all devices involved in the system provide the capability to be directly invoked and automated by the system. However, this assumption may not always be

applicable in IoT systems [MRH15]. IoT objects that have different capabilities may be connected to the WfMS in different ways. Instead of direct IP network connectivity, some things may be connected via multiple network layers or routings. For example, the management system may interact with a gateway service which is connected to multiple sensor nodes.

Another example is continuous tasks, such as sensor data streaming. Existing standard-based BPM modellings tools and WfMS cannot properly handle such processes [MRH15].

As a solution, extensions to languages such as BPMN can be created, which capture the intrinsics of IoT elements and are different from the common workflow nodes.

2.6.2 Implementing and executing workflows in IoT systems

As mentioned previously, the *implement/configure phase* involves mapping the abstract workflow model to a machine-executable software program. Existing common tools for business process execution, such as Activiti[7], Camunda[8], BonitaBPM[9] or Apache ODE[10] however lack support for many of the protocols used in IoT devices such as CoAP and MQTT.

IoT objects can take part in a business process not only as a service, but also by executing an entire business process model themselves. To execute a workflow on a device, there are two approaches: 1) Execute the process model in a workflow engine which is running on the thing [DTB$^+$15, GCFP10] and 2) translate the process model into executable code and run the code on the thing [GEPF11, CDD$^+$12].

[7]`http://activiti.org/`
[8]`https://camunda.org/`
[9]`http://www.bonitasoft.com/`
[10]`http://ode.apache.org/`

3 System Overview

This chapter describes the Service-oriented Composer for Orchestrating Real-time Proximity-based Industrial Internet of Things (SCORPII) middleware framework. SCORPII is a service-oriented middleware for real time service composition from heterogeneous proximal pervasive resources. The framework utilizes dynamic elastic Cloud computing and workflow automation to optimize the task allocation among localhost services and Cloud services based on resource availability and a cost-performance index model.

3.1 Scenario

Here we consider the scenario introduced in section 1.2.1, in which Alice is travelling in an urban IoT environment with her smartphone. Alice's device is providing a real time content mashup service to Alice to serve her purpose. Since the content mashup process requires continuous proximal discovery of surrounding, it can consume a lot of computing power of the mobile device if the number of smart objects is large. As Figure 7 shows, in the first timestamp, the mobile device (SCORPII Mobile Host) has only discovered a small number of smart objects, which includes 3 smart objects that have registered to two different discovery servers, and one smart object (n_1) that is available for direct P2P communication.

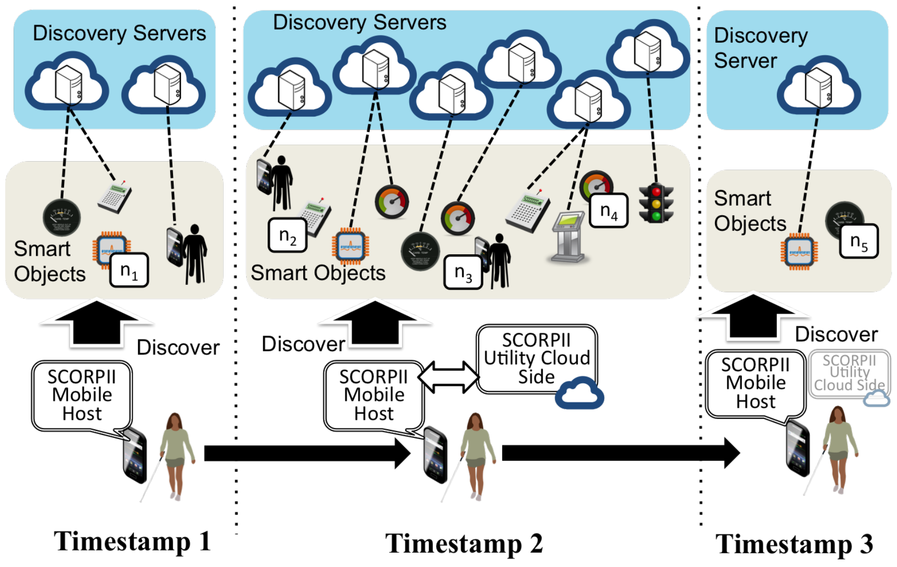


Figure 7: Service discovery scenario.

While Alice is moving, at timestamp 2, the number of smart objects discovered by Alice's device has greatly increased. The large number of discovery processes involve the discovery servers from which the mobile device needs to retrieve service description metadata and also process them. Using SCORPII's analysis component, SCORPII has decided to launch a Utility Cloud service instance to assist certain computational tasks. The tasks which involve interacting and processing the data between SCORPII and external discovery servers have been partitioned as separate workflows and have been executed on the Utility Cloud side. Note that in this environment, n_1, n_2, n_3, n_4 and n_5 are direct communicable smart objects that do not require further communication with discovery servers. The interaction between Alice's device and those smart objects can be performed in the P2P manner.

Afterwards, at timestamp 3, Alice has moved to an area that consists of a minor number of smart objects. Since the data transmission and processes have decreased, SCORPII's analysis component is able to determine that mobile device-based processing is more cost-efficient. Thus, SCORPII terminates the Utility Cloud side components to reduce the cost. This scenario raises the following questions:

- *How does SCORPII decide when and which processes require computational task offloading?*
- *How does SCORPII perform the configuration dynamically at runtime?*

3.2 SCORPII framework design

The framework design is based on the Enterprise Service Bus (ESB) architecture [VdA98]. ESB is a software infrastructure that can easily connect resources by combining and assembling services to achieve a Service Oriented Architecture. Figure 8 illustrates the architecture and the main components of SCORPII. The architecture consists of three parts that are described as follows.

General Internet consists of general Web services such as the discovery servers of the proximal smart objects or third party services that can be utilized to assist certain tasks such as semantic ontology processes. They are reachable through proper Unique Resource Identifiers (URIs) and standard communication technologies such as HTTP or CoAP.

Proximal P2P environment consists of various smart objects. SCORPII can utilize common standard protocols to discover and interact with the proximal smart objects to retrieve data or to retrieve the description of their discovery servers that can provide further information or data regarding the smart object.

SCORPII is a dual workflow controlled middleware service. It consists of Mobile Host side (ScoMH) and Utility Cloud side (ScoUC). ScoMH is the main controller of the entire middleware and ScoUC is mainly used for resource intensive tasks. The components of each side are described as follows.

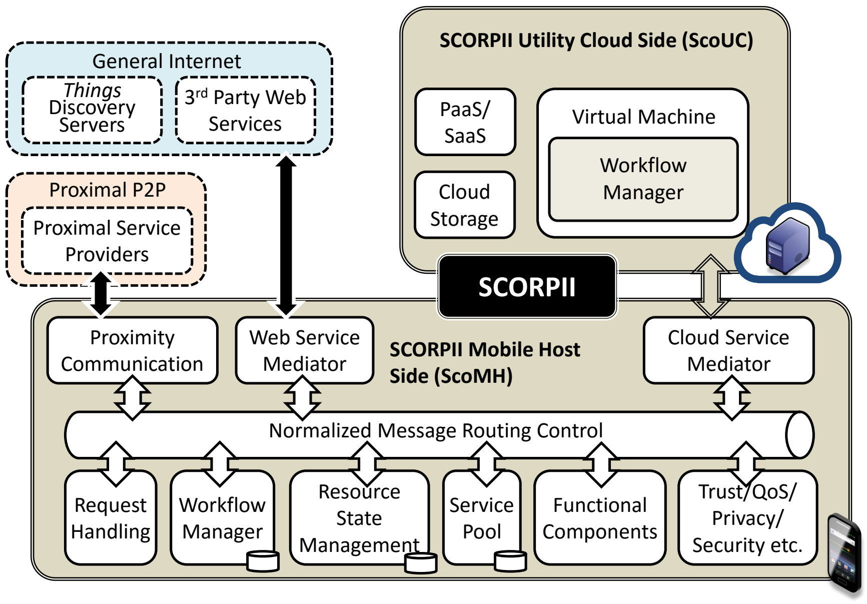


Figure 8: Architecture of SCORPII middleware framework.

3.2.1 SCORPII - Mobile Host Side (ScoMH)

ScoMH consists of the following main components.

Proximity Communication consists of a number of components that enable short range networked resource discovery and interaction using communication protocols such as, Bluetooth, Wi-Fi direct, ZigBee etc.

Web Service Mediator allows ScoMH to interact with global networked services such as the smart object discovery servers using standard communication protocols such as HTTP or CoAP.

Cloud Service Mediator can dynamically configure and launch virtual machine instances and set up the needed components to enable ScoUC.

Normalized Message Routing Control component handles incoming and outgoing messages. It processes the message to meet the required format for the receivers of the message.

Request Handling (RH_{MH}) component receives request messages from other applications (e.g., a User Application, which provides user interface for users to access SCORPII) and forward the request message to corresponding components via the Normalized Message Routing Control component.

Workflow Manager consists of following mechanisms.

- Manages, monitors, and executes workflow. It creates a number of Task

Manager components to analyse the status of tasks.

- Manages a collection of pre-defined abstract workflow models and approaches. When it receives a request that contains the goal of the process, a corresponding abstract workflow model will be executed. A flow relation pattern defines the structure of a set of workflow nodes. The definition of abstract workflow model and approach will be described in Section 3.3.1.

- Handles runtime workflow configuration processes. For example, after a workflow is executed, Task Managers will monitor the cost-efficiency of the tasks. If certain tasks are more cost-efficient to be replaced by different tasks, the corresponding Task Manager will request the controller of the workflow to perform re-configuration.

The *Resource State Management* component is responsible for continuously monitoring the resource usages such as CPU usage, network bandwidth usage, cloud utility service usage, etc. These resource usages are cost intensive, and are the main elements influencing the decision making of the adaptation scheme that is described in the next section.

The *Service Pool* is responsible for managing information on internal services, Utility Cloud services, and services provided by external service providers. It contains a collection of the service descriptions of external service providers, the service descriptions of each internal service and each accessible utility Cloud service.

Functional Components are miscellaneous utility components such as semantic metadata matchmaking component, message parsing and calculation component, for calculating the Cost Performance Index (CPI) value described in the next section.

Trust/QoS/Privacy/Security, etc. are additional components needed to provide trustworthy service discovery processes and to improve the quality of service and security requirements. They are not within the scope of this thesis. They are considered as a future research direction.

3.2.2 SCORPII - Utility Cloud Side (ScoUC)

Different to the common design which assumes the cloud middleware to be always deployed and always available, SCORPII utility cloud – ScoUC is launched on-demand. Same as the other local services, if ScoUC is no longer needed, it will be terminated. One drawback of the on-demand utility cloud component is that it takes time to launch the instance and prepare the cloud platform. For simple processes (e.g., only one single request), ScoUC may not be cost-efficient and will not be needed. The instance can be stored as a snapshot in Cloud storage to reduce the need of uploading the file directly from mobile host. The main advantage of this design is to fully achieve the pay-per-use concept of Utility Cloud services.

Workflow Manager. The Workflow Manager (WM) takes care of both workflow execution and also handling requests related to processes. The Workflow Manager includes a Workflow Engine (WE_{UC}) which is a workflow execution service that handles the workflow passed from the ScoMH.

Secondly, the Workflow Manager processes and analyses the request messages in order to perform corresponding actions. For example, a workflow that is sent from ScoMH will be passed to WE_{UC} after the payload of the request message has been analysed.

Paas/SaaS denotes other Web services that have been pre-deployed by SCORPII user on Platform as a Service (PaaS)-based Cloud service or the known Software as a Service (SaaS)-based Cloud service that can be used as the substitution of the self-managed Web application in the virtual machine.

Cloud Storage. SCORPII can also utilize Cloud storage services to store files that are needed for ScoUC. E.g., customized Web application packages.

3.3 Cost-efficient and Context-aware Workflow Approach Selection Model

This section describes the cost-performance index-based scheme for selecting the configuration of workflow processes at runtime.

3.3.1 Preliminary

Definition 1 (Cost element set—E). *E is a finite set. Each $e \in E$ is a cost element defined as a tuple (id, v) where:*

- *id is a unique identification of e.*
- *v is the cost value of e.*

A cost element can be CPU usage, RAM, bandwidth, cost of Cloud service and so on.

Definition 2 (Context parameter set— $\mathcal{X}$). *$\mathcal{X}$ is a finite set. Each $x \in \mathcal{X}$ is a context parameter defined as a tuple (id, v) where:*

- *id is a unique identification of x.*
- *v is the cost value of x.*

A context parameter can be the remaining battery life, the connection type (3G/WiFi), the mobile data plan cost etc.

Definition 3 (Abstract workflow model). *An abstract workflow model is defined as a tuple (N, F, τ), where:*

- *N is a finite set of nodes.*
- *$F \subseteq N \times N$ is a set of flow relation rules.*
- *$\tau : N \to \Upsilon$ maps nodes to node types. A node type can be parallel gateway, start event, end event, task, sub-process etc.*

Let $n \in N$ be a node: $\bullet n = \{m|(m, n) \in F\}$ is a preset of n, and $n\bullet = \{m|(n, m) \in F\}$ is a post-set of n.

An *abstract workflow model* describes the structure of a process in the abstraction level. Each request received by SCORPII will trigger a corresponding workflow execution process, which consists of a set of sequences and parallel tasks.

Definition 4 (Task Type—*tType*). *tType is a type of node that represents a main task which needs to be accomplished in the workflow. It is defined as a tuple (ID, IN, OUT) where:*

- *ID is the identification of tType.*
- *IN is the input message type.*
- *OUT is the output message type.*

A *tType* node can be substituted by another *tType* node or it can be substituted by another workflow as a sub-process as long as the substitution matches the *IN* and the *OUT* of the original *tType* node defined in the *abstract workflow model.*

Here, we refer to the terms described in [VdA98]: a task that is to be accomplished is called a *work item*. A *work item* is executed by a *resource*. A *resource*, in our case, is a localhost component/service or a Web application that has been deployed in the Cloud. When a *resource* is executing a *work item*, it is called an *activity*.

Additionally, we define the following term:

Definition 5 (Approach—*h*). *An approach is defined as a tuple $(N, F, \tau, \alpha, \varrho, c, \varpi, \rho)$, where:*

- *(N, F, τ) corresponding to the descriptions in **Definition 3**.*

- $\alpha : N \rightarrow \mathcal{A}$ *maps nodes to activities.*
- $\varrho : \mathcal{A} \rightarrow \mathcal{R}$ *maps activities to resources.*
- $c : \mathcal{A} \rightarrow \mathcal{E}$ *maps activities to cost elements.*
- $\varpi : \ddot{\mathcal{E}} \rightarrow \mathcal{X}$ *maps cost elements to context parameters.*
- $\rho : \mathcal{A} \rightarrow \mathcal{T}$ *maps activities to time-span.*

Each element of $\alpha, \varrho, c, \varpi, \rho$ *is defined as a tuple (key, value).*

An *approach* can consist of one or multiple activities. In other words, an *approach* can be seen as an individual workflow (or a sub-process in BPMN). A *tType* node in an *abstract workflow model* can be accomplished by different *approaches* as long as the *approach* satisfies the IN and the OUT of the *tType* node. The *approach* of a *tType* node can be dynamically changed at runtime of workflow execution. However, the activities in an *approach* are static unchangeable when the *approach* is chosen, because the elements $(\alpha, \varrho, c, \varpi, \rho)$ of the *approach* have already defined the configuration.

An activity can be performed by a pre-defined *approach* or it can be performed by a dynamic defined *approach* at runtime. A dynamic defined *approach* is based on composing a number of pre-defined *approaches* to achieve the same output.

Example 1 (Dynamic Approach Pattern). Suppose a sequence workflow consists of 3 tasks: $task_1 \rightarrow task_2 \rightarrow task_3$. The arrows correspond to the process flow. $task_2$, which has the IN type as “X" and the OUT type as “Y". Assume at runtime, $task_2$ cannot be accomplished by its original defined *approach* due to the resources of the activity are suddenly unavailable, and a substitute *approach* that matches to the IN and the OUT types could not be found in the pre-defined *approach* patterns. Hence, the Workflow Manager attempts to define a substitution based on a composite *approach*. By searching the *approach* patterns, Workflow Manager discovered the following 3 patterns:

Pattern 1 consists of 2 nodes with structure as below:

- $n_1 : IN = \text{“}X\text{”}, \quad OUT = \text{“}A1\text{”}, \quad n_1\bullet = \{n_2\}.$
- $n_2 : IN = \text{“}A1\text{”}, \quad OUT = \text{“}B1\text{”}.$

Pattern 2 consists of 6 nodes with structure as below:

- $n_1 : IN = \text{“}B1\text{”}, \quad OUT = \text{“}O1\text{”}, \quad n_1\bullet = \{n_2\}.$
- $n_2 : IN = \text{“}O1\text{”}, \quad OUT = \text{“}P1\text{”} or \text{“}P2\text{”}, n_2\bullet = \{n_3, n_4\}.$

– $n_3 : IN =$ “$P1$”, $OUT =$ “$Q1$”, $n_3\bullet = \{n_5\}$.

– $n_4 : IN =$ “$P2$”, $OUT =$ “$Q2$”, $n_4\bullet = \{n_5\}$.

– $n_5 : IN =$ “$Q1$”or“$Q2$”, $OUT =$ “$L1$”, $n_5\bullet = \{n_6\}$.

– $n_6 : IN =$ “$L1$”, $OUT =$ “$G1$”.

Pattern 3 consists of 2 nodes with structure as below:

– $n_1 : IN =$ “$G1$”, $OUT =$ “$F1$”, $n_1\bullet = \{n_2\}$.

– $n_2 : IN =$ “$F1$”, $OUT =$ “Y”.

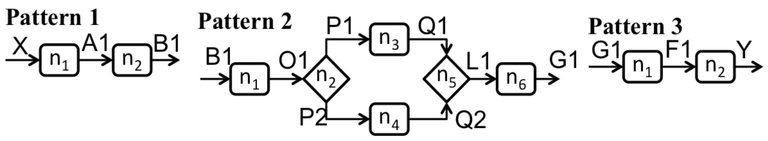


Figure 9: Workflow patterns.

Figure 9 illustrates the conceptual approach patterns described above. By composing the three patterns, an approach with the $IN =$ “X" and the $OUT =$ “Y" can be generated for accomplishing the $task_2$. For more information regarding to pattern similarity searching scheme can be found in [DDGB09].

3.3.2 Time-span of Approach

In this thesis, we consider time-span as the performance of an *approach*. The time-span of an approach is influenced by the involved tasks and the resources. The formal method to measure the time-span of approaches is omitted in this thesis. The related approach has been introduced in [DB07]. In this prototype, we measured the time-span by manually recording the time-span of each test case using customized methods.

Based on the flow relation rules of a given approach, a number of routes (process flow) can be discovered. Each route may consume different time-span depending on the activities involved. Let h be an *approach*, and N_h denotes the nodes in *approach h*. Let O_h be a set of routes that can occur in h. $o \in O_h$ denotes one of the routes. $N_o^{tType} \subseteq N_h$ denotes the set of *tType* nodes involved in o route. $\mathcal{T}_o = \{t_i | 1 \leq i \leq \mathbb{N}\}$ corresponding to the time-spans of each activity involved in N_o^{tType}. E.g., t_1 denotes the time-span of the activity that is executed for the task node—$n_1 \in N_o^{tType}$, which has been defined in ρ map (see **Definition 5**). The time-span of o (denoted by TS_o) is computed by:

$$TS_o = \sum_{i \in |N_o^{tType}|} ts_i \tag{1}$$

where: ts_i is time-span value of $n_i \in N_o^{tType}$.

Based on the calculation above, if the *approach* h has more than one route, its time-span value will be presented with $[TS_h^{min}, TS_h^{max}]$ where $TS_h^{min} = \min\limits_{o \in O_h}\{TS_o\}$ denotes the minimum time-span value and $TS_h^{max} = \max\limits_{o \in O_h}\{TS_o\}$ denotes the maximum time-span value.

Fundamentally, an *approach* that has a shorter time-span is considered as having better performance. However, the shortest time-span may not always mean that the *approach* is the most efficient selection. Hence, we utilize the cost-performance index (CPI) scheme to optimize the *approach* selection. The scheme combines fuzzy set [Zad65] and the weight of context [HKZG08]. The reason we use fuzzy set is to compare the performance and cost between *approaches* instead of using static values.

3.3.3 Raw Cost Elements and Context Parameters of Approach

First, we describe how the raw cost element and raw context parameter sets are generated before we explain the cost-performance index model.

Similar to the measurement of time-span, the cost of an *approach* is also influenced by the number of routes defined in the *approach*. For each route $o \in O_h$ of *approach* h, let $\ddot{E}_o$ denote the cost elements of o defined in $\mathcal{E}$ map of *approach* (see **Definition 5**). Each element $\ddot{e} \in \ddot{E}_o$ denotes the sum of the cost element—e's value from all the *tType* nodes involved in o. The cost elements of o will be within $[\ddot{E}_o^{min}, \ddot{E}_o^{max}]$ where $\ddot{E}_o^{min} = \min\limits_{o \in O_h}\{\ddot{e}_o\}$ corresponds to the minimum cost elements, and $\ddot{E}_o^{max} = \max\limits_{o \in O_h}\{\ddot{e}_o\}$ corresponds to the maximum cost elements.

3.3.4 Cost-Performance Index-based Approach Selection

Let D be a finite set of time-span values of the selective *approaches*—H, $H = \{h_l | 1 \leq l \leq \mathbb{N}\}$, where $|D| = |H|$, $D = \{d_l | 1 \leq l \leq |H|\}$, in which d_l is the maximum time-span of h_l by measurement.

Let ST be the shortest time-span in D, where $ST = \min\{d_l \in D\}$. The performance ranking value of an *approach*—h_y (denoted by PR_{h_y}) is computed by below:

$$PR_{h_y} = \left(1 + \frac{ST}{\sum\limits_{l \in |D|} v_{d_l}}\right) - \frac{v_{d_y}}{\sum\limits_{l \in |D|} v_{d_l}} \tag{2}$$

where v_{d_l} is the value of d_l, and v_{d_y} is the value of d_y.

The formula (2) can generate the ranking value for *approaches* in which the lower time-span the *approach* has, the higher ranking value it has.

We need the normalized fuzzy number of the ranking value. Hence, the fuzzy number of an *approach*'s ranking value (denoted by $\widetilde{PP}_{h_y}$) is:

$$\widetilde{PP}_{h_y} = \frac{PR_{h_y}}{\sum\limits_{l \in |H|} PR_{h_l}} \tag{3}$$

where PR_{h_y} is the performance ranking value of h_y derived from (2), and $\widetilde{PP}_{h_y}$ is the normalized fuzzy number of the performance ranking value of h_y, in which $0 \leq \widetilde{PP}_{h_y} \leq 1$.

The cost element set (**Definition 1**) must be comparable between different related *approaches*. If cost elements of *approach* h_1—$\ddot{E}_{h_1}$ contains the value of 'battery cost', then the *approach* h_2—$\ddot{E}_{h_2}$ must also contain such a value. Accordingly, the overall CPI between different *approaches* can be computed.

3.3.5 Context-aware weight calculation of the cost elements

Since we are comparing the cost elements between different *approaches*, the normalized value of a cost element—$\tilde{v}_{\ddot{e}_y}$ is computed as below.

$$\tilde{v}_{\ddot{e}_y} = \frac{v_{\ddot{e}_y}}{\sum\limits_{\eta \in |\ddot{E}_{h_l}|} v_{\ddot{e}_\eta}} \times \frac{w_{\ddot{e}_y}}{|W|} \tag{4}$$

where $w_{\ddot{e}_y}$ is the importance weight of the cost element, and W denotes all the weight values of cost elements. It is considered because a cost element is different for different users. For example, when the device battery-life remains 50%, the user may consider that saving the battery life of his/her mobile device is more important than spending money on using cloud services for computational needs. In this case, the weight of the battery life cost element will be higher than the weight of the bandwidth cost of the cloud service. The weight of the cost element for the approach is dependent on the context parameters. The context of a cost element h_l (denoted by $\ddot{e}_\eta^{h_l}$) is retrieved by using function — $\varpi(\ddot{e}_\eta^{h_l})$ (see **Definition 5**) where

$\mathcal{K}^{\ddot{e}_\eta^{h_l}} = \{k_1^{\ddot{e}_\eta^{h_l}}, k_2^{\ddot{e}_\eta^{h_l}}, \cdots, k_n^{\ddot{e}_\eta^{h_l}}\}, n \in \mathbb{N}$. Hence, the weights can be defined as:

$$w_{\ddot{e}_y} = \Theta_0^{\ddot{e}_y^{h_l}} + \Theta_1^{\ddot{e}_y^{h_l}} k_1^{\ddot{e}_\eta^{h_l}} + \Theta_2^{\ddot{e}_y^{h_l}} k_2^{\ddot{e}_\eta^{h_l}} + \cdots + \Theta_n^{\ddot{e}_y^{h_l}} k_n^{\ddot{e}_\eta^{h_l}}, n \in \mathbb{N} \tag{5}$$

Where $\Theta^{\ddot{e}_y^{h_l}}$ are the regression coefficients for a given cost element $\ddot{e}_y \in \ddot{E}_{h_y}$ and $\ddot{e}_y$ is influenced by context parameters $\mathcal{K}^{\ddot{e}_\eta^{h_l}}$. $\Theta^{\ddot{e}_y^{h_l}}$ is a $(n+1)$-dimensional vector.

The average value of the total cost of an *approach* (denoted by CE_{h_l}) is computed as:

$$CE_{h_l} = \frac{\sum\limits_{\eta \in |\ddot{E}_{h_l}|} \tilde{v}_{\ddot{e}_\eta}}{|\ddot{E}_{h_l}|} \tag{6}$$

By applying the basic CPI model, the cost-performance value of an *approach* (denoted by CPV_{h_l}) is:

$$CPV_{h_l} = \frac{\widetilde{PR}_{h_l}}{CE_{h_l}} \tag{7}$$

3.4 SCORPII Prototype

A prototype of SCORPII has been implemented for real Android-based mobile devices together with certain components hosted on Amazon Web service. The prototype consists of two parts:

ScoMH components (i.e. *Workflow Manager, Resource State Manager, Functional Components,* etc.) have been implemented in Google Nexus 5 mobile device that runs on Android OS 5.0. The Workflow Manager has been implemented as a localhost service that can process BPEL [MIS+03] documents. The current version of ScoMH's BPEL engine can support sequential and parallel tasks.

ScoUC, which is managed by ScoMH, is a dynamically launched instance on Amazon EC2 (t2.medium) with a Ubuntu OS. ScoUC components are Web applications operated in Apache Tomcat 7 and the workflow engine of ScoUC is managed by Apache ODE, which is hosted using Apache Tomcat.

4 Adaptive Workflow Execution in the Internet of Things

4.1 Overview

The SCORPII framework design is extended with an additional component, in which the workflow manager is extended with the ability to allow IoT devices to execute either a model representation of a business process or a piece of program code corresponding to the workflow.

In Fig. 10 we present how this extension is applied to SCORPII. Most importantly, the Workflow Manager of SCORPII Utility Cloud side has been extended with a Workflow Translator component.

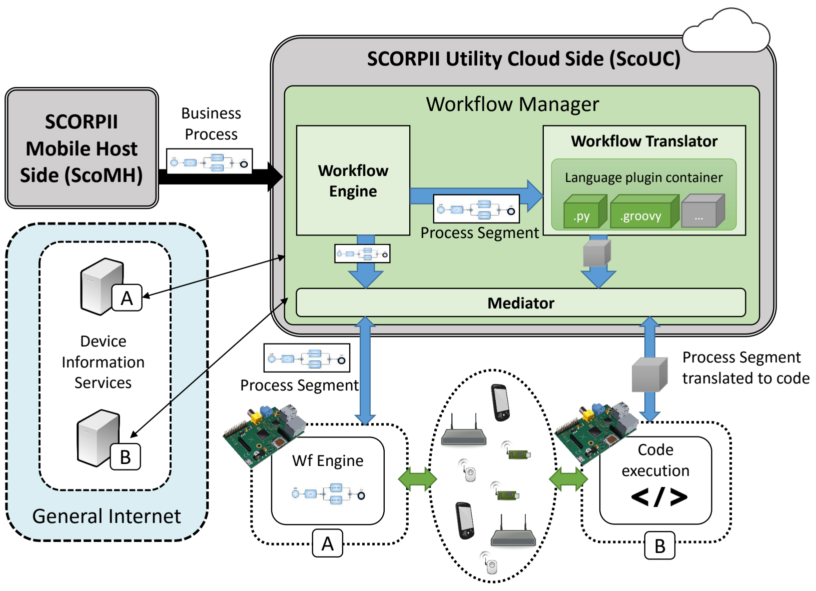


Figure 10: System design overview

The Utility Cloud Side (ScoUC) provides a façade for executing workflows in the IoT edge network/environment. The Mobile Host sends their business process to the Cloud Side, where the process model is then parsed. Using a decision making mechanism (as described in section 3.3, ScoUC may execute the process workflow entirely in the Cloud, invoking IoT nodes as part of *Workflow Tasks* in the conventional SOA style, alternatively it may partition the workflow into segments that are to be executed on the IoT nodes.

Information about an IoT device in the edge network can be inquired using the corresponding Device Information Service (DIS). A DIS provides information regarding the device capabilities such as whether the device has a workflow engine or a code execution environment, what kind network protocols are supported, which functions (e.g. sensors, programming languages) are supported.

With this information, a user application which uses the SCORPII Mobile Host Side middleware may configure a composite service involving a number of edge nodes, and submit it to ScoUC for execution.

In our illustrated example, the Cloud Side has determined that the Business Process provided by the ScoMH should be divided into two segments (subprocesses) to be executed by two IoT devices, A and B.

Device A supports model execution as it is running its own workflow engine, while device B supports code execution only.

For device A, the matching segment of workflow is transferred (e.g. in the BPMN format) to the device. However, for device B, ScoUC invokes the Workflow Translator component to convert the workflow model document to executable code in a programming language supported by B.

The Workflow Translator uses a plugin-based design which allows developers to easily add translators from workflow languages to programming languages. The translated business process is sent to the device and executed there.

Communication between the ScoUC and IoT devices is carried out using Internet standards such as HTTP (Hypertext Transfer Protocol), CoAP (Constrained Application Protocol) or MQTT (formerly MQ Telemetry Transport). Additionally, executing the business process on the IoT device may involve Machine-to-Machine communication with other nearby IoT entities (e.g. via Bluetooth Low Energy, WiFi).

The above description illustrates the conceptual approach of workflow execution and service composition at the edge nodes in an IoT environment. Code execution is commonly considered as a lightweight approach that is more feasible to the resource constrained IoT devices [CK11a, GEPF11, CDD+12]. However, there is no comprehensive comparison between the two approaches in previous research works.

> *The main question is whether the code execution approach is able to outperform the workflow engine approach decidedly or can the workflow engine approach be applied equally efficiently for edge network BP execution?*

4.2 Workflow Translator implementation

This section describes the implementation of the workflow translator component of SCORPII Cloud-Side. The translator prototype consists of two components: (a) the software which transforms BPMN processes into executable code (translated BP execution) and (b) the resource constrained IoT device which executes the workflow.

To run experiments regarding different approaches of BP execution, we used a Raspberry Pi B+ [11], running Raspbian GNU/Linux 8. The device has a single core CPU, configured to allow a maximum CPU clock speed of 900MHz and has 512MB SDRAM. The Java JDK installed is Oracle JDK 8.

4.2.1 Code Generation

The main mechanism of Workflow Translator is to convert the input BPMN 2.0 model (i.e. `.bpmn` file) to executable program code. The converted code depends on which programming language is supported by the code executor.

General architecture In order to be able to map the process into a given programming language, a developer should provide modules which define how elements of the BPMN language are to be represented in that language. For example, XOR gateways are denoted by the syntax of an if-else statement in that language, or parallel gateways are denoted by syntax and constructs for parallel threaded execution.

In addition to transforming the flow and gateway elements of a BPMN process model, the tasks and events must also be transformed to code representation. This is also language-specific and described by the language module (plugin). The following section provides an example based on the case where the supported language is Groovy.

4.2.2 Groovy language example

We created a Groovy language module for generating Groovy scripts from `.bpmn` files. Groovy was chosen as it is one of the two scripting languages included in the prepackaged Camunda distributions. To create a Groovy representation for a task, the developer should include a folder with the name of the corresponding task, containing a `.groovy` file which depicts the groovy code for that task, and an `imports.groovy` file which includes the necessary dependency statements.

[11] `https://www.raspberrypi.org/products/model-b-plus/`

Note that the above folder-based approach could easily be switched with a different Groovy module implementation where per-task implementations are looked up from a remote service.

In our prototype implementation, we have also developed a Python-based module. We will consider the comparison of different modules in the future.

4.2.3 Workflow Engine

For BP model execution on the edge node, we chose the free (community) edition of the Camunda BPM workflow engine. Camunda BPM supports scripting with JSR-223 compatible script engines [Cam] and comes with a Groovy script engine included. While the Groovy environment included with Camunda BPM is version 2.4.5, we installed Groovy 2.4.6 on the Raspberry Pi.

We used the pre-packaged Tomcat Camunda distribution (released November 30, 2015), which contains the Camunda BPM platform v.7.4.0 as a set of Tomcat 8.0.24 web applications. The sample web applications and workflows provided with this distribution were removed to improve the start time of the platform.

In order to deploy (and run) a business process on the Camunda BPM platform, a Java process application for the process must be created.

4.2.4 Test Scenario

To conduct experiments regarding the different workflow execution approach, we conceived the following test scenarios and corresponding business processes.

The first business process used in our experiments is a simple IoT scenario where tasks are executed in parallel (see Fig. 11).

The workflow consists of two flows of parallel tasks. The first flow consist of two sequential tasks—"Read File" and "Sending using MQTT". "Read File" represents the retrieval of sensory data collected by the device that have been stored in its local memory. Once the data is retrieved, it is sent to a specific remote client via MQTT protocol. The second flow also consists of two sequential tasks—"Get CPU Temperature" and "Send using HTTP", which represent the device invokes an internal service to retrieve its current CPU temperature data and afterwards sending the data via HTTP to the remote management system which is monitoring the activities of the device.

Each of the tasks in the test process model has a corresponding Groovy implementation in the Groovy module of the workflow translator.

The `.bpmn` file itself was created with the Camunda Process modeler [12], all tasks are of type script task, their implementations described in the Groovy language, equivalent to the ones provided to the *workflow translator*.

[12] `https://camunda.org/bpmn/tool/`

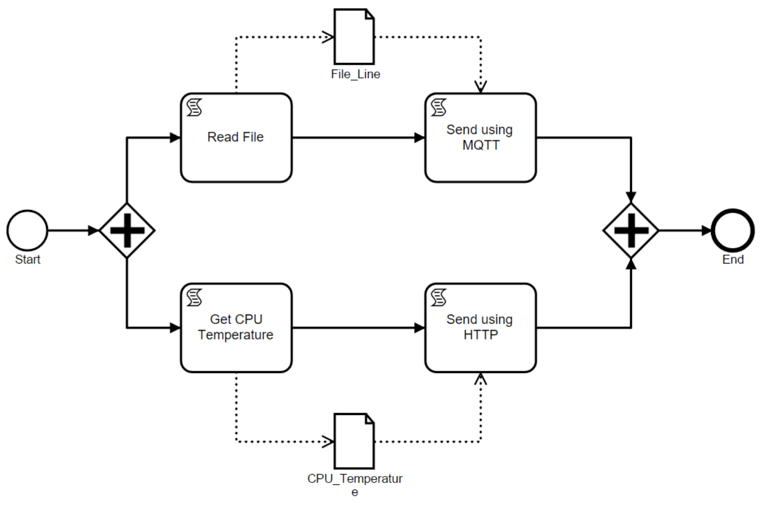


Figure 11: BPMN 2.0 Business process model of parallel tasks

We also included a second, simplified version of the business process, in which no parallel tasks are present. The second business process is presented in Fig. 12.

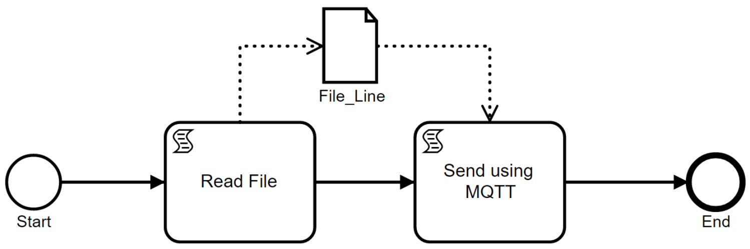


Figure 12: BPMN 2.0 BP model with no parallel tasks

Camunda BPM execution To start an instance of the business process, the Java Web application containing the BP must first be deployed to the Tomcat container. We did this by placing a `.war` file in the "`webapps`" folder of Tomcat.

To benchmark the performance of the process execution on Camunda BPM, our approach was the following. First, the process is deployed onto the server. Then, a new process instance is started using the REST API of Camunda BPM, this is considered as the starting point of the workflow execution. The framework does not respond to the REST call until the workflow execution has been finished. The moment when the REST response is received, it is considered the end of execution.

Groovy script translation execution The Groovy translation of the BP is executed using the command line Groovy invocation command. The starting point of execution is considered the moment the command is executed. The execution end is considered as the moment the Groovy program has finished, the Groovy environment has exited and returned control to the command line.

5 Evaluation

The SCORPII framework prototype has been tested based on case studies in order to evaluate: 1) the runtime dynamic workflow configuration which is based on the cost-performance index scheme, 2) the different workflow execution approaches on resource-constrained devices.

5.1 Dynamic Configuration Use Case

We used a case study to evaluate our prototype based on the scenario described in Section 3.1. The aim is to validate the CPI-based approach selection and workflow reconfiguration. The basic setting is as follows. Fig. 13 illustrates the default workflow of the use case. A localhost application (*reqApp*) in ScoMH can browse smart objects in proximity and retrieve the name or ID of the device from their advertisement.

For each timestamp (5 seconds), device names or IDs retrieved by the *reqApp*, will be sent to the Workflow Manager of ScoMH.

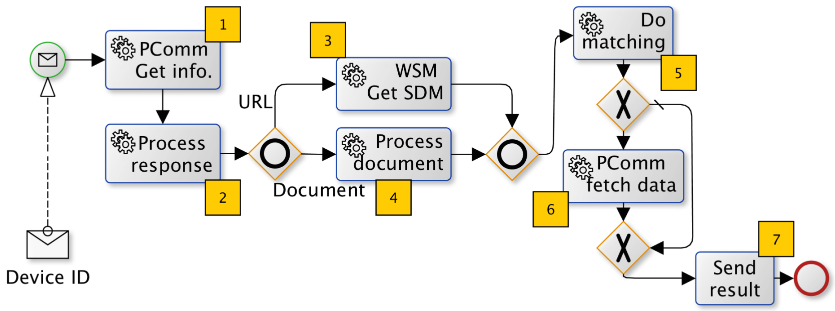


Figure 13: Default service workflow (simplified)

As Figure 13 shows, each device ID is the input parameter of the workflow process, the Device ID will be forwarded to proximity communication service (PComm) and PComm will perform a simple GET request to the corresponding device (step mark 1). After PComm retrieves the response, the response message will be forwarded to another component to analyze what type of message it is (mark 2). If it is an URL address, ScoMH will use the Web Service Mediator (WSM) to GET the service description metadata (SDM) from the discovery server based on the URL (mark 3). On the other hand, if the message contains a document, ScoMH will use a corresponding functional component to process the document to analyze if the document is a SDM (mark 4). Afterwards, the SDM, which is either retrieved from either step 3 or step 4, will be forwarded to another local host service to perform the service matchmaking (mark 5). If the service described in SDM matches the requested type (in this case study, the type is "current noise level"), the workflow manager will use PComm to retrieve the data from the corresponding device based on the URI described in SDM (mark 6), otherwise, a simple "no match" response will be generated as a response. Afterwards, the workflow service will send the result back to the *reqApp* (mark 7).

At the beginning, mobile requester averagely discovers 50 service providers per timestamp. After 10 timestamps, the discovery frequency has increased from 50 to 500. Hence, the Workflow Manager performed re-configuration based on the two selectable approaches:

- *Approach 1 (A1)* : The activities are unchanged as the workflow of figure 13 described. All the SDM retrieval and matchmaking processes were done in ScoMH.
- *Approach 2 (A2)* : The SDM retrieval and matchmaking processes, which involved discovery servers, are distributed to ScoUC.

The context that influences the decision-making is based on the *context parameters* defined for each approach, plus the frequency of discovering new service providers. The context parameters we used in our test scenario are explained later. The costs considered in the use case are: CPU usage of the mobile device, bandwidth of mobile device and bandwidth of Cloud service. CPU and bandwidth usages are the major influencers of the energy consumption of mobile devices. Note that in reality, there are more cost elements which need to be considered.

We have been studying cloud cost models and auto-scaling models for a while now [SO14] and these models will be introduced into SCORPII at later stages.

5.2 Dynamic Configuration Evaluation

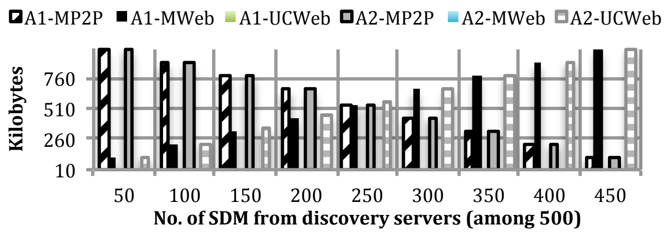


Figure 14: Bandwidth measurement comparison

Figure 14 illustrates the bandwidth measurement comparison between the two approaches (A1 and A2) recorded by SCORPII for the CPI calculation. The measurement of each approach consists of three cost elements: (1) MP2P, corresponding to the bandwidth cost of retrieving SDMs directly from proximal nodes; (2)

MWeb denotes the bandwidth cost of the mobile Internet connection; (3) UCWeb denotes the bandwidth cost of ScoUC for retrieving SDMs from discovery servers. Since A1 did not use Cloud at all. Hence, A1-UCWeb is always consuming 0 bandwidth. Although A2 used ScoUC to retrieve SDMs, it also consumed the mobile Internet bandwidth because the communication between ScoMH and ScoUC consumed a minor bandwidth, in which the number is around 9.5 kilobytes and it is too little to be shown on the figure.

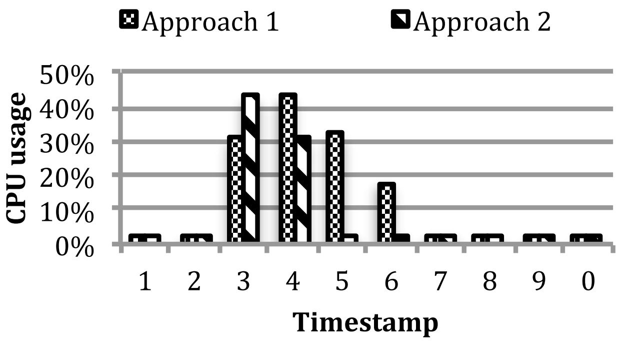


Figure 15: CPU usage comparison

Figure 15 illustrates the CPU usage comparison between A1 and A2 when 200 (among 500 nodes; the rest SDM were already retrieved directly from the smart objects) service discovery processes involved retrieving and processing SDMs from discovery servers. As the figure shows, A1 consumed CPU over 10% for 4 timestamps, and A2 consumed CPU over 10% for 2 timestamps because A2 distributed its tasks to ScoUC. The CPU cost element in our test case was based on how long the CPU usage retains over 10%.

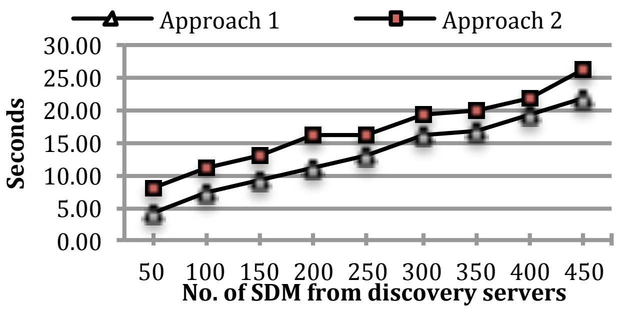


Figure 16: Time-span comparison

5.2.1 Context-aware Cost Element Weighing

To attribute weights to the mentioned cost elements, the regression coefficients in equation (5) must be found. To obtain these, we used multiple linear regression, using the context parameters as explanatory variables and the weight as the dependent variable, as already explained in section 3.3.

Table 1: Context parameters used in the regression training dataset

Cost element	**Context parameters addressed in A1**	**Context parameters addressed in A2**
CPU cost	CPU load, Battery level, Connection type	CPU load, instance type
Mobile internet cost	Data plan cost, Connection type, Signal strength	Data plan cost, Connection type, Signal strength
Cloud instance cost	$\varnothing$	Cloud infrastructure load, Cloud endpoint distance

For multiple linear regression, we need a dataset of observed weight values, then fitting the linear equation optimally to that dataset gives us the coefficients. However, we were unable to obtain a dataset based on real-life data traces and as such, we decided to generate the dataset, using certain user preferences based on expert knowledge to define the behaviour how the weight values vary in response to context values.

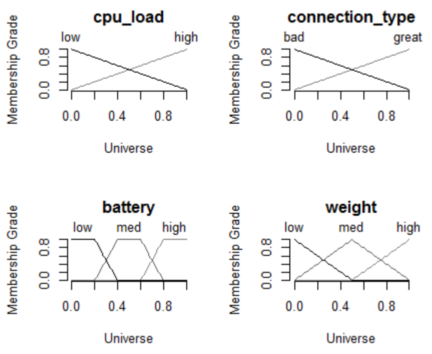
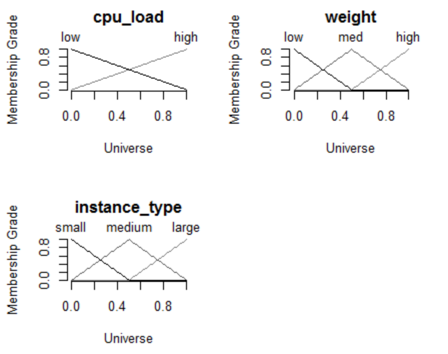
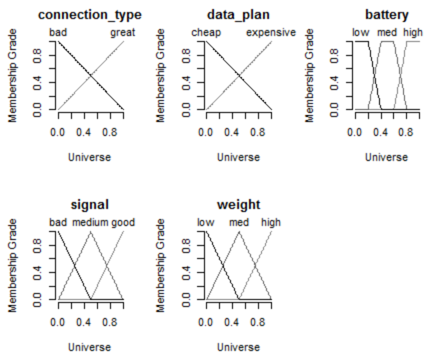
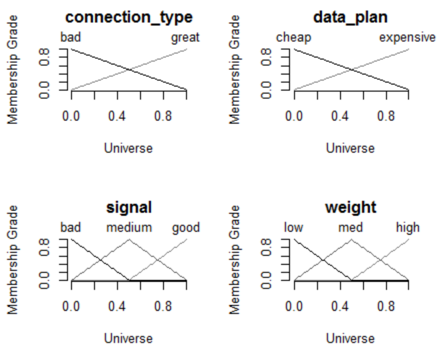
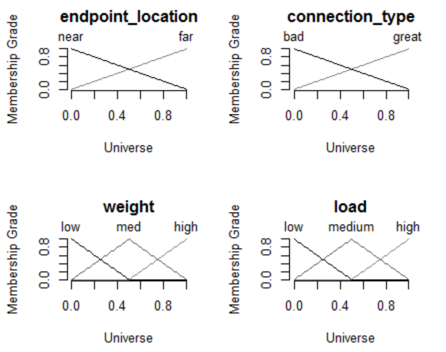
First, we fixed a set of context parameters for each cost element, as shown in Table 1. The context parameters for a given cost element may vary per approach. In some cases (such as with *Cloud instance cost for A1*), the context parameter set may be empty. In this case, the weight defaults to the value 1.

To obtain a meaningful dataset, in which the context parameter values affect the weights in a realistic manner, we used fuzzy rules and fuzzy inference. In order to do fuzzy inference, we first had to define the context parameters as fuzzy variables.

The values of the context parameters have been normalized to the interval $[0.0, 1.0]$. Then, a fuzzy representation for the context parameter could be created. In addition, a fuzzy representation of the weight itself is also required in order to do fuzzy inference. An overview of the created fuzzy variables for our test scenario can be seen in Table 2.

In order to embed various desired qualities into the dataset, based on expert advice, fuzzy rules were defined. An example rule is "If the battery is high and the signal strength is low, then cloud instance cost weight is high" or "If battery is medium and data plan cost is high, then the CPU cost weight is low".

Table 2: Fuzzy representation of the context parameters and weights

Cost Element	**A1 fuzzy context variables**	**A2 fuzzy context variables**
CPU Cost		
Mobile internet cost		
Cloud cost	∅	

With such rules and fuzzy variables in place, we used Mamdani fuzzy inference [MA75] to infer weight values according to the fuzzy rules and a randomly generated set of raw fuzzified context parameter values. The inference process combines the distributions of the fuzzy rules influencing the weight into a single fuzzy output distribution. Taking the centroid of this resulting distribution will give us an exact numeric value for the weight.

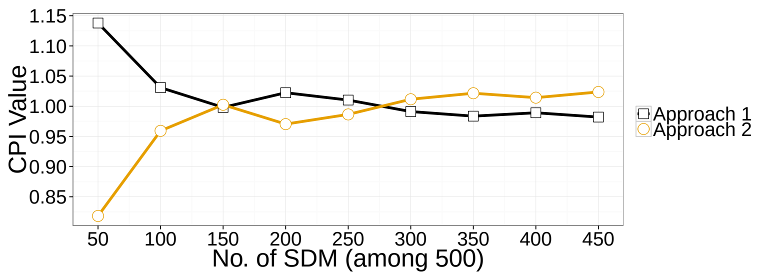


Figure 17: CPI comparison of the two approaches in the test scenario with *Context A*

These inferred weight values were then used with their corresponding context parameters values as training examples to fit the multiple linear regression lines. The resulting values for our experiment can be seen in Table 3.

Table 3: Weight formulas with regression coefficients plugged in

Cost element	**Weight formula with our test case regression coefficients**
A1 CPU	$0.435 + 0.1158 \times \text{CpuLoad} + 0.1301 \times \text{ConnectionType} - 0.1196 \times \text{Battery}$
A2 CPU	$0.4935 + 0.2265 \times \text{CpuLoad} - 0.23 \times \text{InstanceType}$
A1 Mobile bandwidth	$0.558 - 0.121 \times \text{CpuLoad} + 0.125 \times \text{DataPlan} - 0.117 \times \text{Battery} + 0.006 \times \text{Signal}$
A2 Mobile bandwidth	$0.507 - 0.239 \times \text{ConnectionType} + 0.23 \times \text{DataPlan} - 0.004 \times \text{Signal}$
A2 Cloud bandwidth	$0.462 + 0.117 \times \text{EndpointDistance} - 0.122 \times \text{ConnectionType} + 0.08 \times \text{CloudLoad}$

Table 4: Final weights of the cost elements with *Context A*

Cost element	**A1 weight**	**A2 weight**
CPU	0.61	0.316
Mobile Bandwidth	0.428	0.2898
Cloud Bandwidth	1	0.405

5.3 CPI Scores and Approach Selection

In the test case, we considered two context scenarios, *Context A*, in which the mobile device's resources are exhausted and *Context B*, where the mobile device is in a "normal state", with abundant resources. Weights attributed to the various costs based on Context A can be seen in Table 4. The context parameter values considered for *Context A* are the following:

1. Connection type : Excellent (WiFi)
2. Battery: low (20 %)
3. Data plan: cheap (price per kb is low)
4. Signal strength: Medium (50%)
5. Mobile phone CPU load: 65%
6. Cloud CPU load: 1%
7. Cloud instance type: large
8. Cloud Endpoint geographic distance: medium

Figure 17 illustrates the CPI computation records (Equation 7) for *Context A* when a different number of service discovery processes are involved in the discovery service.

Such a context implies that the mobile phones resources are constrained as it is already low on battery and the CPU is already under considerable load. Meanwhile, the phones connection type and mobile data plan benefit mobile cloud offloading.

For the second set of context parameters examined, *Context B*, we changed context parameters which affected the device itself (such as *CPU load, battery, signal* etc.), while the cloud-related context (e.g. *instance type*) remained the same as it was in *Context A*. The corresponding CPI computation for *Context B* can be seen in Figure 18. Weights attributed to the costs in case of Context B can be seen in Table 5.

The values used were the following.

1. Connection type : Excellent (WiFi)
2. Battery: High (75 %)
3. Data plan: cheap (price per kb is low)
4. Signal strength: Good (90%)
5. Mobile phone CPU load: 1%
6. Cloud CPU load: 1%
7. Cloud instance type: large
8. Cloud Endpoint geographic distance: medium

Table 5: Final weights of the cost elements with *Context B*

Cost element	**A1 weight**	**A2 weight**
CPU	0.48	0.316
Mobile Bandwidth	0.366	0.2898
Cloud Bandwidth	1	0.405

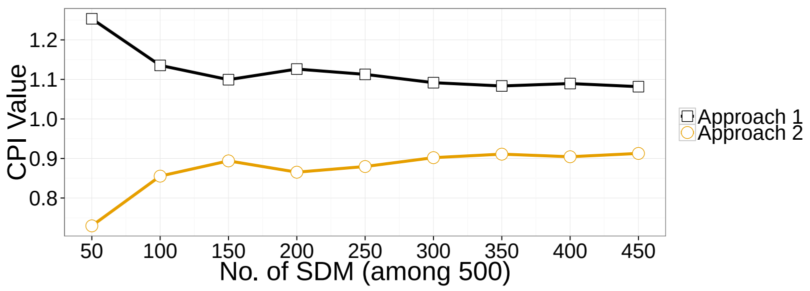


Figure 18: CPI comparison of the two approaches in the test scenario with *Context B*

With *Context A*, Approach 1 initially outscores Approach 2, but as the number of actual Service descriptor metadata received from nearby devices grows, Approach 2 starts outperforming Approach 1, as parsing the SDMs on the cloud becomes increasingly more cost-efficient for the entire approach.

However, in the case of *Context B*, the Workflow Manager decided to launch Approach 1 instead of Approach 2 in all cases, as the mobile device is in an idle state and off-loading to the cloud is more costly.

5.4 Workflow Execution Approach Comparison

In this section, we compare two workflow execution methods:

- Executing the translated BP Model and
- Executing BP Model on a workflow engine on the device

We monitored the following characteristics. The CPU and Memory usage was recorded using the sysstat[13] performance monitoring software tools. Power consumption was found based on current measurements done with a PeakTech 3430 Digital-Multimeter.

5.4.1 Bootstrapping

For approach (a), the mandatory preceding steps are booting the Tomcat server itself and then deploying the BP Java project onto the Tomcat server. With our scenario, the server launch took on average 290 seconds and the process deployment took 35 seconds. Fig. 19–21 show the CPU, memory and power usage during the Tomcat launch.

Table 6: Parallel BP execution times

	Groovy Translation of BP	BP model deployed to Camunda BPM
Cold Execution Time	1m09s	1m16s
Warm Execution Time	1021ms	807ms

The server startup is tasking for the CPU, using up the full capability of the CPU for essentially the entire server launch time.

5.4.2 Process Execution Time

In our case, executing an instance of the BP involves initializing the Groovy runtime, which itself is time consuming (on average 54s). This is necessary for both

[13]`http://sebastien.godard.pagesperso-orange.fr/`

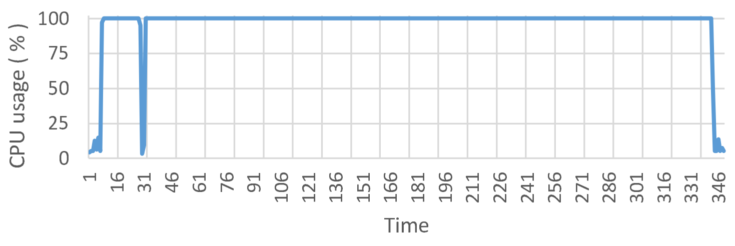


Figure 19: Tomcat startup CPU usage

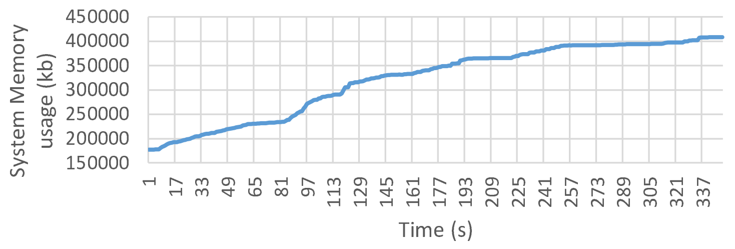


Figure 20: Tomcat startup Memory usage

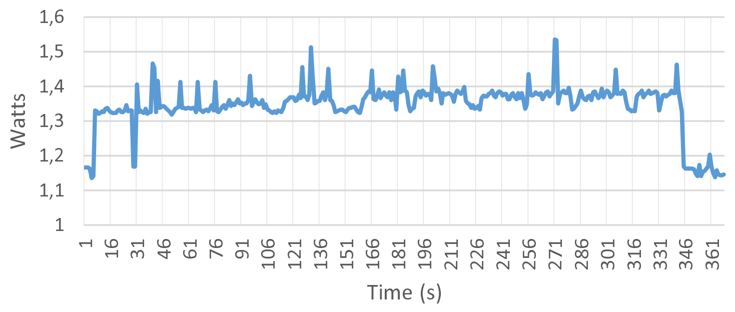


Figure 21: Tomcat startup power consumption

approaches and we denote execution which involves the runtime initialization as a "*cold execution*".

However, an important feature of the Camunda BPM engine is the fact that the script language runtime is not stopped after finishing execution of a workflow instance. This results in much faster subsequent execution times after a language runtime has been already started for workflows using the same script language. We denote these faster, subsequent execution types as "*warm executions*".

On the other hand, when running a Groovy translation of the workflow from the command line, the runtime environment is started and stopped after each execution ("cold executions"). In order to compare methods (a) and (b) in the case of a *warm execution*, we iterated the Groovy translation within the Groovy runtime.

Parallel task BP An overview of *cold execution* times, *warm execution* times and deployment times for Business Processes with a parallel gateway introduced in the previous section (see Fig. 11) can be seen in Table 6.

The execution of the translated version outperforms workflow engine execution by a few seconds in the case of cold execution, yet the Worfklow engine execution outperforms the translation execution in the case of *warm execution.* The better performance of Camunda BPM can attributed to caching mechanisms within the Camunda platform.

Simple BP In the case of the second workflow model with no parallel tasks (see Fig. 12), the execution time was smaller for both approaches, but still the translated execution outperformed the workflow engine in case of *cold execution.* Table 7 presents the average execution times for the simpler BP model.

Table 7: Non-Parallel BP execution times

	Groovy Translation of BP	BP model deployed to Camunda BPM
Cold Execution Time	45s	56s
Warm Execution Time	1003ms	966ms

5.4.3 System Resource and Power usage

We examined the usage of power consumption and system resource usage for both workflow models.

Parallel Task Business Process Memory Usage. Speaking of memory consumption, executing the translated code consumes ~2 times less memory than executing the BP Model in the Workflow Engine, as the Software stack required for execution on the Camunda Platform consists of several web applications, as described in section 4.2.3.

Secondly, as can be seen in Fig. 23, which depicts memory usage during execution of the BP using both methods, when the execution ends, the translated version execution immediately releases used memory, while Camunda BPM keeps the loaded runtime in memory, resulting in a continuous, high usage of memory.

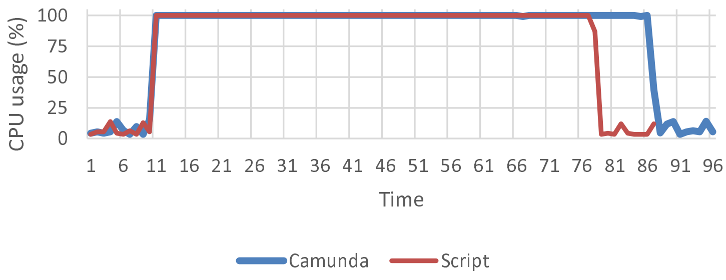


Figure 22: CPU usage during execution (Parallel BP)

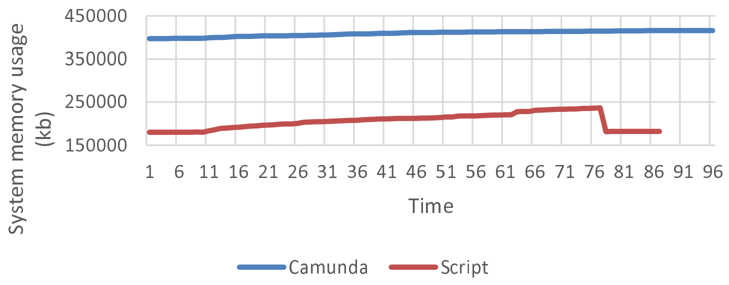


Figure 23: Memory usage during execution (Parallel BP)

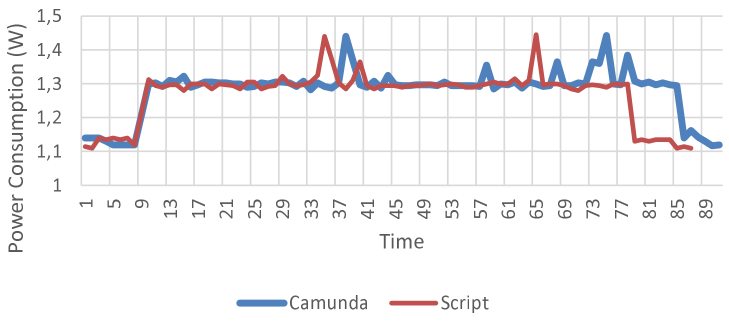


Figure 24: Power consumption during execution (Parallel BP)

Parallel Task Business Process CPU & Energy consumption. Power consumption and CPU usage measurements showed largely similar results for both approaches during execution time (see Fig. 22, Fig. 24) but at the end time of execution, higher power consumption is evident for the workflow engine approach. This is likely due to additional callbacks which take place within the Camunda Platform after a process execution finishes.

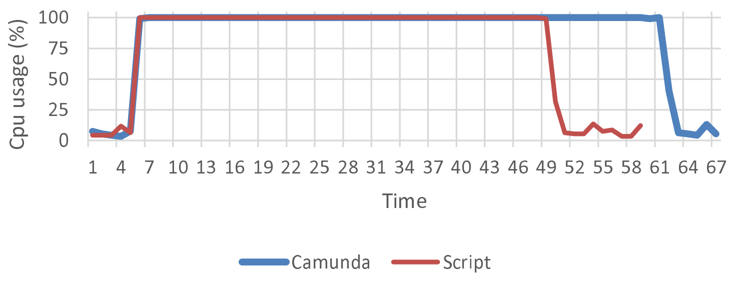


Figure 25: CPU usage during execution (Non-Parallel BP)

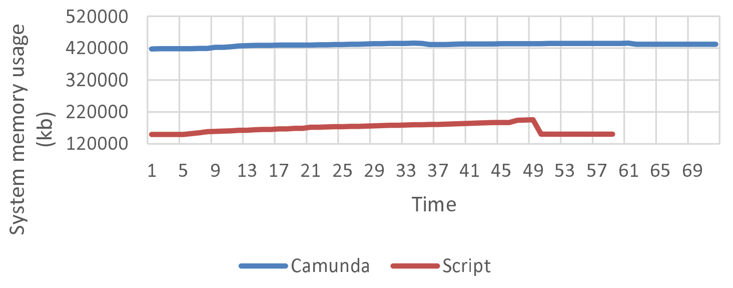


Figure 26: Memory usage during execution (Non-Parallel BP)

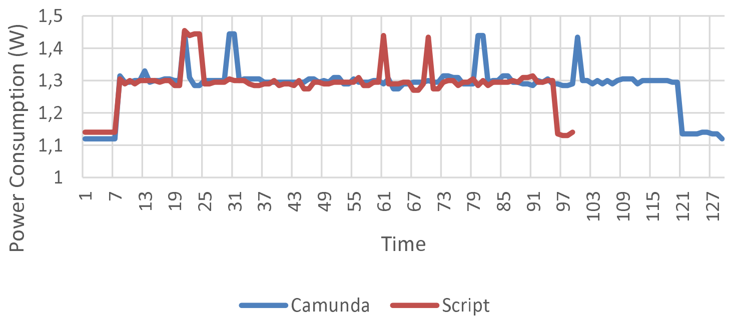


Figure 27: Power consumption during execution (Non-Parallel BP)

Non-parallel BP model Memory Usage. The memory footprint is lower for code translation in comparison to the parallel gateway workflow (see Fig. 26. This is due to the smaller amount of dependencies used for the simpler workflow.

Non-parallel CPU & Energy consumption. CPU usage in the case of the non-parallel workflow shows continuous high usage throughout execution (Fig.

25). The power consumption(Fig. 27) shows a similar pattern, with the workflow engine approach consuming more power in total as the execution time is larger.

5.5 Bootstrapping and Resource Consumption of Deployment

When we consider the CPU consumption during the bootstrapping of the Camunda Platform, this was continuous near-100% CPU usage throughout the 290 seconds (Fig. 19) that it takes to launch the Tomcat Server on the Raspberry Pi.

Similarly, each deployment of a workflow resulted in continuous high CPU utilization. This is a serious drawback in terms of energy and CPU usage in cases where workflow deployments occur often or when the workflow engine must be re-launched.

Meanwhile, deployment of a translated BP is not necessary, the only cost is transferring the translated BP to the executing edge node, which we excluded from our evaluation.

5.6 Discussion

Experiments showed that running a workflow engine for model execution comes at the price of noticeably larger memory consumption, and when workflows are deployed and run on the engine only once, the execution time is slightly higher in comparison to executing code which was translated from the BP model.

Additionally, first-time launching of the workflow engine and deployment of new BPs to the workflow engine platform are tasking in terms of processor usage.

However, if a single workflow deployment is reused multiple times in the workflow engine, the larger memory consumption pays off as subsequent runs of the workflow reuse necessary components for execution that had been cached.

Thus, for one-off workflow execution where the business process is not re-used, translated model execution is beneficial in terms of both execution time and energy consumption. In cases where the same BP is executed multiple times and memory can be spared, running a workflow engine on the edge node is very viable.

6 Conclusion

This thesis aimed to develop and validate an IoT service composition middleware framework. Secondly, the thesis aimed to provide a guideline for developers in the domain of workflow execution on resource-constrained devices. Namely, the guideline should focus on comparing executing a workflow using as input:

1. a model of the workflow
2. executable code corresponding to the model.

In this thesis, we introduced a service-oriented Internet of Things middleware framework—SCORPII. SCORPII utilizes a hybrid service-oriented workflow system to achieve the autonomous task reconfiguration at runtime by partitioning and allocating tasks among the mobile terminal and the Utility Cloud. Furthermore, the framework is capable of automatically deciding task allocation based on the context influence (e.g., the number of interacting nodes), hardware and network resource availability, and the importance weight of the resource usage based on user preference.

Additionally, a component which enables workflow execution on resource-constrained devices where embedding a workflow engine is not plausible, was proposed. The component is able to translate workflow models into program code that can be directly executed on the resource-constrained device. The component uses a design which allows for easy development of plugins to support translation into additional languages.

Experiments were conducted based on a use case and the results have shown that the proposed SCORPII framework is capable of providing a cost-efficient workflow task configuration dynamically. The success of the approach also paves way for utilizing similar adaptive mediation frameworks in solving several domain specific issues.

A direct comparison of two execution approaches for Business Processes on resource-constrained IoT edge nodes was also carried out: executing the BP model on a workflow engine deployed to the node and executing a translated BP model in a programming language supported by the node. According to the comparison experiments, executing BPs on a workflow engine cannot be justified in cases where system memory is sparse and the model is not reused. Instead, for one-off executions of business processes, translated workflow execution is beneficial.

6.1 Future Research Directions

- **Application of an Adaptive Mediation Framework in Internet of Things Scenarios.** We plan to evaluate the SCORPII framework in different IoT scenarios, such as smart office or smart home.

- **Quality of Service for Internet of Things Middleware.** We also plan to extend the framework with additional mechanisms that enable Quality of Service (QoS)-aware dynamic service composition with resource constrained IoT service.
- **Study of Translated Workflow Execution on Resource-constrained Devices.** Additionally we aim to compare additional programming language translations. We are also interested in repeating the translated workflow execution experiments on Android compatible smart phones and Arduino kits [ard].

References

[Ap] Apple Inc. . ibeacon for developers. https://developer.apple.com/ibeacon/ [Accessed: 13.05.2016].

[AAA+06] Alexandre Alves, Assaf Arkin, Sid Askary, Ben Bloch, Francisco Curbera, Yaron Goland, Neelakantan Kartha, Dieter König, Vinkesh Mehta, Satish Thatte, et al. Web services business process execution language version 2.0. 2006.

[AIM10] Luigi Atzori, Antonio Iera, and Giacomo Morabito. The internet of things: A survey. *Computer networks*, 54(15):2787–2805, 2010.

[ard] Arduino. https://www.arduino.cc/ [Accessed: 28.04.2015].

[Ash09] Kevin Ashton. That 'internet of things' thing. RFID Journal, 2009. http://www.rfidjournal.com/articles/view?4986.

[aut] Auto-id labs. http://autoidlabs.org, Accessed: 06.04.2016.

[Bot] Botts Innovative Research Inc. Sensorml 2.0 examples. http://www.sensorml.com/sensorML-2.0/examples/ [Accessed: 18.05.2016].

[bpm] Business process model and notation 2.0.

[BRGW14] M Botts, A Robin, J Greenwood, and D Wesloh. Ogc® sensorml: Model and xml encoding standard. *Technical Standard*, 2, 2014.

[BYV+09] Rajkumar Buyya, Chee Shin Yeo, Srikumar Venugopal, James Broberg, and Ivona Brandic. Cloud computing and emerging it platforms: Vision, hype, and reality for delivering computing as the 5th utility. *Future Generation computer systems*, 25(6):599–616, 2009.

[Cam] Camunda Services GmbH. Script task. Docs.camunda.org. https://docs.camunda.org/manual/7.4/reference/bpmn20/tasks/scripttask/ [Accessed: 21-Mar-2016].

[CDD+12] Fabio Casati, Florian Daniel, Guenadi Dantchev, Joakim Eriksson, Niclas Finne, Stamatis Karnouskos, Patricio Moreno Montera, Luca Mottola, Felix Jonathan Oppermann, Gian Pietro Picco, et al. Towards business processes orchestrating the physical enterprise with wireless sensor networks. In *Software Engineering*

(ICSE), 2012 34th International Conference on, pages 1357–1360. IEEE, 2012.

[CHK$^+$12] Ioannis Chatzigiannakis, Henning Hasemann, Marcel Karnstedt, Oliver Kleine, Alexander Kroller, Myriam Leggieri, Dennis Pfisterer, Kay Romer, and Connie Truong. True self-configuration for the iot. In *Internet of Things (IOT), 2012 3rd International Conference on the*, pages 9–15. IEEE, 2012.

[CK11a] Alexandru Caracaş and Thorsten Kramp. *Business Process Model and Notation: Third International Workshop, BPMN 2011, Lucerne, Switzerland, November 21-22, 2011. Proceedings*, chapter On the Expressiveness of BPMN for Modeling Wireless Sensor Networks Applications, pages 16–30. Springer Berlin Heidelberg, Berlin, Heidelberg, 2011.

[CK11b] Alexandru Caracaş and Thorsten Kramp. On the expressiveness of bpmn for modeling wireless sensor networks applications. In *Business Process Model and Notation*, pages 16–30. Springer, 2011.

[DB07] Andrea D'Ambrogio and Paolo Bocciarelli. A model-driven approach to describe and predict the performance of composite services. In *Proceedings of the 6th international workshop on Software and performance*, pages 78–89. ACM, 2007.

[DDGB09] Remco Dijkman, Marlon Dumas, and Luciano García-Bañuelos. Graph matching algorithms for business process model similarity search. In *Business Process Management*, pages 48–63. Springer, 2009.

[DGV09] Simon Duquennoy, Gilles Grimaud, and Jean-Jacques Vandewalle. The web of things: interconnecting devices with high usability and performance. In *Embedded Software and Systems, 2009. ICESS'09. International Conference on*, pages 323–330. IEEE, 2009.

[DLNW13] Hoang T Dinh, Chonho Lee, Dusit Niyato, and Ping Wang. A survey of mobile cloud computing: architecture, applications, and approaches. *Wireless communications and mobile computing*, 13(18):1587–1611, 2013.

[DLRMR13] Marlon Dumas, Marcello La Rosa, Jan Mendling, and Hajo A Reijers. *Fundamentals of business process management.* Springer, 2013.

[DTB+15] Kashif Dar, Amir Taherkordi, Harun Baraki, Frank Eliassen, and Kurt Geihs. A resource oriented integration architecture for the internet of things: A business process perspective. *Pervasive and Mobile Computing*, 20:145–159, 2015.

[DTRE11] Kashif Dar, Amirhosein Taherkordi, Romain Rouvoy, and Frank Eliassen. Adaptable service composition for very-large-scale internet of things systems. In *Proceedings of the 8th Middleware Doctoral Symposium*, page 2. ACM, 2011.

[Eva11] Dave Evans. The internet of things: How the next evolution of the internet is changing everything. *CISCO white paper*, 1:1–11, 2011.

[FHT+15] Huber Flores, Pan Hui, Sasu Tarkoma, Yong Li, Satish Srirama, and Rajkumar Buyya. Mobile code offloading: from concept to practice and beyond. *Communications Magazine, IEEE*, 53(3):80–88, 2015.

[FLR13a] Niroshinie Fernando, Seng W Loke, and Wenny Rahayu. Mobile cloud computing: A survey. *Future Generation Computer Systems*, 29(1):84–106, 2013.

[FLR13b] Niroshinie Fernando, Seng W Loke, and Wenny Rahayu. Mobile cloud computing: A survey. *Future Generation Computer Systems*, 29(1):84–106, 2013.

[FS14] Huber Flores and Satish Narayana Srirama. Mobile cloud middleware. *Journal of Systems and Software*, 92:82–94, 2014.

[G+04] RDF Working Group et al. Resource description framework (rdf). *W3C–Semantic Web*, 1, 2004.

[GBMP13] Jayavardhana Gubbi, Rajkumar Buyya, Slaven Marusic, and Marimuthu Palaniswami. Internet of things (iot): A vision, architectural elements, and future directions. *Future Generation Computer Systems*, 29(7):1645–1660, 2013.

[GCFP10] Pau Giner, Carlos Cetina, Joan Fons, and Vicente Pelechano. Developing mobile workflow support in the internet of things. *IEEE Pervasive Computing*, (2):18–26, 2010.

[GEPF11] Nils Glombitza, Sebastian Ebers, Dennis Pfisterer, and Stefan Fischer. Using bpel to realize business processes for an internet of things. In *Ad-hoc, Mobile, and Wireless Networks*, pages 294–307. Springer, 2011.

[Goo] Google Inc. The physical web cookbook. the physical web, http://google.github.io/physical-web/cookbook/. `http://google.github.io/physical-web/cookbook/` [Accessed: 29.04.2016].

[GP08] Dimitrios Georgakopoulos and Michael P Papazoglou. *Service-oriented computing.* The MIT Press, 2008.

[GTK$^+$10] Dominique Guinard, Vlad Trifa, Stamatis Karnouskos, Patrik Spiess, and Domnic Savio. Interacting with the soa-based internet of things: Discovery, query, selection, and on-demand provisioning of web services. *Services Computing, IEEE Transactions on*, 3(3):223–235, 2010.

[GTW10] Dominique Guinard, Vlad Trifa, and Erik Wilde. A resource oriented architecture for the web of things. In *Internet of Things (IOT), 2010*, pages 1–8. IEEE, 2010.

[HKZG08] Pari Delir Haghighi, Shonali Krishnaswamy, Arkady Zaslavsky, and Mohamed Medhat Gaber. Reasoning about context in uncertain pervasive computing environments. In *Smart Sensing and Context*, pages 112–125. Springer, 2008.

[IGH$^+$11] Valérie Issarny, Nikolaos Georgantas, Sara Hachem, Apostolos Zarras, Panos Vassiliadist, Marco Autili, Marco Aurelio Gerosa, and Amira Ben Hamida. Service-oriented middleware for the future internet: state of the art and research directions. *Journal of Internet Services and Applications*, 2(1):23–45, 2011.

[Int01] Workflow Process Definition Interface. Workflow management coalition workflow standard workflow process definition interface–xml process definition language. 2001.

[JAS12] Cullen Jennings, Jari Arkko, and Zach Shelby. Media types for sensor markup language (senml). 2012.

[JLF$^+$14] Antonio J Jara, Pablo Lopez, David Fernandez, Jose F Castillo, Miguel A Zamora, and Antonio F Skarmeta. Mobile digcovery: discovering and interacting with the world through the internet of things. *Personal and ubiquitous computing*, 18(2):323–338, 2014.

[LW13] Jorg Lenhard and Guido Wirtz. Measuring the portability of executable service-oriented processes. In *Enterprise Distributed Object Computing Conference (EDOC), 2013 17th IEEE International*, pages 117–126. IEEE, 2013.

[MA75] Ebrahim H Mamdani and Sedrak Assilian. An experiment in linguistic synthesis with a fuzzy logic controller. *International journal of man-machine studies*, 7(1):1–13, 1975.

[MBR15] Roberto Minerva, Abyi Biru, and Domenico Rotondi. Towards a definition of the internet of things (iot). IEEE Internet of Things: "Define IoT" Document, May 2015. `http://iot.ieee.org/definition`.

[MIS$^+$03] Microsoft, IBM, Siebel, BEA, and SAP. Business Process Execution Language for Web Services Version 1.1, May 2003.

[MRH15] Sonja Meyer, Andreas Ruppen, and Lorenz Hilty. The things of the internet of things in bpmn. In *Advanced Information Systems Engineering Workshops*, pages 285–297. Springer, 2015.

[ODTHVdA06] C Ouvans, Marlon Dumas, Arthur HM Ter Hofstede, and Wil MP Van der Aalst. From bpmn process models to bpel web services. In *Web Services, 2006. ICWS'06. International Conference on*, pages 285–292. IEEE, 2006.

[Pap03] Mike P Papazoglou. Service-oriented computing: Concepts, characteristics and directions. In *Web Information Systems Engineering, 2003. WISE 2003. Proceedings of the Fourth International Conference on*, pages 3–12. IEEE, 2003.

[PJZ$^+$14] Charith Perera, Prem Prakash Jayaraman, Arkady Zaslavsky, Dimitrios Georgakopoulos, and Peter Christen. Mosden: An internet of things middleware for resource constrained mobile devices. In *System Sciences (HICSS), 2014 47th Hawaii International Conference on*, pages 1053–1062. IEEE, 2014.

[PRS$^+$13] Tao Peng, Marco Ronchetti, Jovan Stevovic, Annamaria Chiasera, and Giampaolo Armellin. Business process assignment and execution from cloud to mobile. In *Business Process Management Workshops*, pages 264–276. Springer, 2013.

[Rag15] Dave Raggett. An introduction to the web of things framework, May 2015. `https://www.w3.org/2015/05/wot-framework.pdf` [Accessed: 29.04.2016].

[RS04] Jinghai Rao and Xiaomeng Su. A survey of automated web service composition methods. In *Semantic Web Services and Web Process Composition*, pages 43–54. Springer, 2004.

[SB11] Zach Shelby and Carsten Bormann. *6LoWPAN: The wireless embedded Internet*, volume 43. John Wiley & Sons, 2011.

[SJP06] Satish Narayana Srirama, Matthias Jarke, and Wolfgang Prinz. Mobile web service provisioning. In *null*, page 120. IEEE, 2006.

[SO14] Satish Narayana Srirama and Alireza Ostovar. Optimal resource provisioning for scaling enterprise applications on the cloud. In *Cloud Computing Technology and Science (CloudCom), 2014 IEEE 6th International Conference on*, pages 262–271. IEEE, 2014.

[SQV$^+$14] Quan Z Sheng, Xiaoqiang Qiao, Athanasios V Vasilakos, Claudia Szabo, Scott Bourne, and Xiaofei Xu. Web services composition: A decade's overview. *Information Sciences*, 280:218–238, 2014.

[SRN$^+$15] Xiang Su, Jukka Riekki, Jukka K Nurminen, Johanna Nieminen, and Markus Koskimies. Adding semantics to internet of things. *Concurrency and Computation: Practice and Experience*, 27(8):1844–1860, 2015.

[TAB$^+$15] Ken Traub, Felice Armenio, Henri Barthel, Paul Dietrich, John Duker, Christian Floerkemeier, John Garrett, Mark Harrison, Bernie Hogan, Jin Mitsugi, Josef Preishuber-Pfluegl, Oleg Ryaboy, Sanjay Sarma, and KK Suen. The epcglobal architecture framework, April 2015. `http://www.gs1.org/id-keys-epcrfid-epcis/epc-rfid-architecture-framework/1-6` Accessed: 06.04.2016.

[THIG11] Thiago Teixeira, Sara Hachem, Valérie Issarny, and Nikolaos Georgantas. Service oriented middleware for the internet of things: a perspective. In *Towards a Service-Based Internet*, pages 220–229. Springer, 2011.

[TSD$^+$12] Stefano Tranquillini, Patrik Spieß, Florian Daniel, Stamatis Karnouskos, Fabio Casati, Nina Oertel, Luca Mottola, Felix Jonathan Oppermann, Gian Pietro Picco, Kay Römer, et al. Process-based design and integration of wireless sensor network applications. In *Business Process Management*, pages 134–149. Springer, 2012.

[VdA98] Wil MP Van der Aalst. The application of petri nets to workflow management. *Journal of circuits, systems, and computers*, 8(01):21–66, 1998.

[vdA13] Wil MP van der Aalst. Business process management: A comprehensive survey. *ISRN Software Engineering*, 2013, 2013.

[VDAVH04] Wil Van Der Aalst and Kees Max Van Hee. *Workflow management: models, methods, and systems.* MIT press, 2004.

[VFG$^+$15] Ovidiu Vermesan, Peter Friess, Patrick Guillemin, Raffaele Giaffreda, Hanne Grindvoll, Markus Eisenhauer, Martin Serrano, Klaus Moessner, Maurizio Spirito, Lars-Cyril Blystad, and Elias Z. Tragos. *Internet of Things beyond the Hype: Research, Innovation and Deployment.* European Research Cluster on the Internet of Things, 2015.

[xpd] Xml process definition language (xpdl). `http://www.xpdl.org/` Accessed: April 14, 2016.

[Zad65] Lotfi A Zadeh. Fuzzy sets. *Information and control*, 8(3):338–353, 1965.