DISSERTATIONES
INFORMATICAE
UNIVERSITATIS
TARTUENSIS
37

JAKOB MASS

Process Management for Internet of Mobile Things

DISSERTATIONES INFORMATICAE UNIVERSITATIS TARTUENSIS
37

DISSERTATIONES INFORMATICAE UNIVERSITATIS TARTUENSIS
37

JAKOB MASS

Process Management for Internet of Mobile Things

UNIVERSITY OF TARTU
Press
1632

Institute of Computer Science, Faculty of Science and Technology, University of Tartu, Estonia.

Dissertation has been accepted for the commencement of the degree of Doctor of Philosophy (PhD) in Computer Science on 21st October, 2022 by the Council of the Institute of Computer Science, University of Tartu.

Supervisors

Prof. Dr. Satish Narayana Srirama
Visiting Professor, University of Tartu, Estonia
Associate Professor, University of Hyderabad, India

Dr. Chii Chang
Apromore Pty Ltd
Melbourne, Australia

Opponents

Prof. Antonio Brogi
Department of Computer Science
University of Pisa, Italy

Prof. Andrea Marrella
Department of Computer, Control and Management Engineering
Sapienza University of Rome, Italy

The public defense will take place on December 9th, 2022 at 10:15 in Narva mnt 18 - 2048 and via Zoom.

The publication of this dissertation was financed by the Institute of Computer Science, University of Tartu.

ISSN 2613-5906 (print)
ISSN 2806-2345 (PDF)
ISBN 978-9916-27-076-9 (print)
ISBN 978-9916-27-077-6 (PDF)

University of Tartu Press
http://www.tyk.ee/

There is nothing like looking, if you want to find something (or so Thorin said to the young dwarves). You certainly usually find something, if you look, but it is not always quite the something you were after.

– J. R. R. Tolkien, *The Hobbit*

ABSTRACT

The Internet of Things (IoT) vision promises to connect and capture devices and their data from any environment at an unprecedented scale. The rise of IoT drives interest in how it can be joined with other technologies, such as Business Process Management (BPM) systems. BPM has seen a lot of success in improving processes that are automated with conventional enterprise information systems, which largely follow the Service-Oriented Architecture (SOA) principles. The service-oriented perspective is important for IoT as well, as it largely addresses one of its core challenges - interoperability between heterogeneous devices and their services.

While BPM technologies are already generally well-aligned with SOA, making it directly applicable for IoT as well, it is not clear however, how exactly can BPM be applied to IoT applications involving new computing paradigms such as Edge Computing and Fog Computing. These paradigms shift the control, management, and execution of processing, networking and storage tasks from the previously dominant centralized data-centers to the edge network, onto devices such as connected vehicles, smartphones, routers and local gateways, resulting in a more de-centralised system. Communication in the edge network is often reliant on device-to-device or local wireless area networks. This makes accounting for the mobility of devices an especially challenging aspect, as movement into and out of radio coverage areas means intermittent connectivity. We refer to IoT scenarios, which involve such mobility as the *Internet of Mobile Things* (IoMT).

Secondly, Fog computing systems include the concept of general-purpose fog servers deployed near the edge network. This creates the opportunity to offload some computational tasks from the IoMT devices to these nearby servers. However, the optimal offloading decisions need to consider the performance, energy and delay trade-offs.

This thesis studies how Business Process execution can be realized in the Edge by embedding process engines on Mobile nodes, resulting in so-called Mobile Process Hosts (MPH), capable of autonomously executing processes in the edge network without relying on a centralised system. In our approach, we adopt existing BPM standards and technology, such as the BPMN 2.0 modelling language, which increases the direct applicability of the results to real systems. We show how different Internet of Mobile Things applications can be modelled and executed with the BPM standards.

We present two distinct features to support autonomous, continuous, adaptive process execution on the mobile nodes: device-to-device migration of process instances during runtime and a scheduling algorithm for deciding which nearby fog servers to involve in the edge process execution.

Device-to-device migration of business processes allows a process execution continue in a specific edge location even if the set of devices in that location are changing, and one single MPH is not guaranteed to be available in the loca-

tion. The scheduling algorithm takes into account the locations of the fog servers, movement trajectory of the MPH, and characteristics of the task being processed.

We demonstrate with experiments on smartphones that the migration feature is applicable to real-life mobility scenarios such as those in logistics and transportation. Based on discrete event simulation, we analysed the scheduling algorithm and showed how it improves energy-efficiency and success rates of processes involving offloading.

Based on insights from the conducted studies and to support further research into edge-fog process management with a focus towards mobility-oriented scenarios, we established the STEP-ONE testbed. STEP-ONE is a set of tools developed to simulate mobility-oriented IoT systems where the management and composition of applications is handled with BPM standards.

We demonstrate STEP-ONE with a smart-city scenario, where an application modelled as a choreography of processes is executed between different resources - cloud, fog servers and MPHs. We show how STEP-ONE supports the construction, simulated execution and analysis of such scenarios and how it can be extended with algorithms and decision-mechanisms to adaptively drive the execution of such processes.

CONTENTS

1. Introduction **14**
1.1. Motivating Scenario 17
1.2. Problem Statement & Research Goals 19
1.3. Methodology & Contributions 20
1.4. Thesis Structure 21

2. State of the Art **23**
2.1. Internet of Things 23
2.1.1. Five Layer Conceptual IoT Model 24
2.1.2. Cisco IoT Reference model 25
2.1.3. Discussion 26
2.2. Interoperability and Communication in Internet of Things 27
2.2.1. Interoperability 27
2.2.2. Service Oriented IoT Frameworks 28
2.2.3. Communication Models & Protocols 30
2.2.4. Delay-Tolerant & Opportunistic Networking 32
2.3. Computing Paradigms for Internet of Things 32
2.3.1. Cloud Computing 33
2.3.2. Edge Computing 34
2.3.3. Cloudlets, Fog Computing 35
2.3.4. Practical Fog Computing - Standards, Technology 36
2.3.5. Key Challenges for Fog Computing 37
2.3.6. Fog applications - overview 38
2.4. Workflow & Process Management 39
2.4.1. Business Process Management Lifecycle 40
2.4.2. Modelling and Analysis of Business Processes 41
2.4.3. Process-Aware Information Systems, Process Execution . . 43
2.4.4. Flowable Java API 45
2.5. Integration of IoT and BPM 46
2.5.1. Modelling IoT processes 47
2.5.2. Execution of IoT Business Processes 50
2.5.3. Dynamic service binding 51
2.5.4. Control-flow based Dynamic Fault handling 52
2.5.5. Context-aware process driving 54
2.5.6. Adaptive Process Execution Platforms 54
2.5.7. Discussion 56
2.6. Adaptive Edge Process Management 56
2.6.1. Edge-Hosted Execution Engine 57
2.6.2. Task Scheduling in Edge and Fog 59
2.6.3. Mobility-Aware Offloading 59

2.6.4. Process-Oriented Fog Computing 60
2.6.5. Simulation of Edge Process Management 62
2.7. Summary 65

3. Process Execution & Migration In The Edge **66**
3.1. Scenario 67
3.1.1. Challenges 69
3.2. Architecture 69
3.2.1. Process Owner 71
3.2.2. Process Executor 71
3.3. Prototype and Experiments 73
3.3.1. Experiment Description 73
3.3.2. Results 74
3.4. Discussion 75

4. Context-aware bottom-up scheduling of Fog-related process tasks **77**
4.1. System Design 79
4.1.1. Fog Server 80
4.1.2. Mobile Thing Host 82
4.1.3. Central Management Cloud 83
4.1.4. Adaptive Task Execution Scheduling Scheme 84
4.2. Experimental Case Study 85
4.2.1. System implementation in Simulated Testbed 85
4.2.2. Scenarios 87
4.2.3. Isolated scenarios with single mobile host 89
4.2.4. Scenario with Multiple Mobile Thing Hosts 92
4.2.5. Discussion 94
4.3. Summary 94

5. Simulating Mobility-Oriented Edge-Fog Processes **96**
5.1. STEP-ONE Software Requirements & Design 97
5.1.1. Existing ONE simulator features 98
5.1.2. Requirements for STEP-ONE 98
5.1.3. Process Engine Application 98
5.1.4. Discussion 101
5.2. Implementation 101
5.2.1. Messaging implementation 101
5.2.2. Events & Signals implementation 103
5.2.3. Simulated Work Task & Cost Management 104
5.2.4. Other features 106
5.2.5. Modelling Processes 108
5.3. Case Study 109
5.3.1. Implementing scenarios in STEP-ONE 110
5.3.2. Establishing the simulation world map 111

5.3.3. Describing Processes Definitions 112
5.3.4. Process Implementation in STEP-ONE 114
5.3.5. Configuring Networking and Reporting 115
5.4. Evaluation 115
5.4.1. Experiment Setup 115
5.4.2. Case Study 116
5.4.3. Process allocation 117
5.4.4. Process Performance - Duration 117
5.4.5. Discussion 118

6. Conclusion **120**
6.1. Future Research Directions 121

Bibliography **123**

Acknowledgements **140**

Sisukokkuvõte (Summary in Estonian) **141**

Curriculum Vitae **144**

Elulookirjeldus (Curriculum Vitae in Estonian) **145**

List of original publications **146**

LIST OF FIGURES

1. Motivating scenario for usage of Mobile Process Hosts and BPM across the Edge-Fog-Cloud continuum 18
2. High-level architectural models of Internet of Things 25
3. Device-to-Device direction communication 30
4. Cloud-related IoT communication 31
5. Approaches to application initiation in Fog: Top-down and Bottom-up 39
6. Phases of the Business Process Management lifecycle 40
7. Example of business process visually modelled using the Business Process Management and Notation 2.0 (BPMN 2.0) standard. . . . 42
8. High-level architectural view of a typical business process management system 43
9. Example of IoT-annotated BPMN process model 49
10. A selected sequence of events from the WiseWare goods monitoring scenario 67
11. Movement of data between Process Executors and Process Owners. 69
12. Architectural Overview of WiseWare platform 70
13. Business process used in the experiments. 74
14. Time consumption of process migration. 74
15. Run-time factors that influence the cost-performance efficiency of EPM in mobile IoT-based disaster scenario. 78
16. Overview of System Architecture 80
17. BPMN 2.0 Process Models used in the case study 88
18. Simulation settings, *s0* and *s1* are the fog servers, *p2* is the Mobile Thing Host 89
19. *Case 1* distributed task time- and energy consumption for Mobile Thing Host 90
20. *Case 2* - distributed task time- and energy consumption for Mobile Thing Host 91
21. *Case 3* - Distributed task time- and energy consumption for Mobile Thing Host 92
22. Case 5 Simulation Map 92
23. Mean Distributed Task completion for MoTHs 93
24. Case 5 Energy usage 93
25. Distribution of tasks among servers with message size = 5M . . . 94
26. Comparison of STEP-ONE with related simulators in the level of detail addressing OSI network layers, mobility, energy and processes modeling 97
27. Overview of STEP-ONE architectural Components 99
28. Overview of how Process Messages are mapped to ONE messages and vice versa. 102

29. ONE Simulator with the Process Info Extension of STEP-ONE visible. 107
30. STEP-ONE version of Flowable Modeller 109
31. Single Road Segment Analysis Process Overview 110
32. Simulation World Map Preparation 111
33. Smart City Road Monitoring Process Collaboration Diagram . . . 113
34. No. of different Processes run across experiments 116
35. Processes run in different layers 117
36. Process distribution among fog hosts 117
37. Median Durations of Different Processes 118
38. ECDFs of Master Road Analysis Process 118
39. No. of messages and bandwidth transferred by message type . . . 119

LIST OF TABLES

1. Comparison of networking and mobility-oriented simulation tools 65
2. Migration file sizes and total time consumption with different numbers of instances. 75
3. Simulated wireless interface & energy model configuration 86
4. Case 5 Simulation parameters 93
5. BPMN 2.0 Message Task parameters in STEP-ONE 103
6. Event & signal constructs in STEP-ONE 104
7. Evaluation parameters and their values 116

1. INTRODUCTION

The ability to manufacture microcontrollers and CPUs of smaller physical dimensions at reduced cost has made it feasible to embed these devices alongside various "everyday" objects at a massive scale - home appliances, industrial equipment, farm animals and street lights, to name a few examples. Combining computing capacity with sensors, actuators and a communications module such as a WiFi radio, allows interfacing the physical objects with digital systems. Such a *smart device* could for example enable sensing the CO2 level of a room or toggling a light switch, and when linked to a network, relay its current state and allow remote control.

Usage of *smart devices* as part of different domain-specific applications and control systems - industrial manufacturing, healthcare, logistics, etc. improves the quality and efficiency of these services, as the software can incorporate detailed real-time monitoring, tracking, analysis and control enabled by the smart devices. The adoption has led each sector to develop their specialised, domain-specific technologies. As an example, consider domain-specific communication protocols: OPC-UA for industrial control systems, BACNET and ModBus for building automation, M-BUS for power metering and Controller Area Network (CAN bus) for vehicles.

The Internet of Things (IoT) is an idea which takes this a step further: if individual *smart device*-using, domain-specific applications are bridged by domain-independent intermediary services, networks and unified standards, then the resulting network(s) allows interaction, cooperation and data-sharing between the applications and devices at an unprecedented scale [Al-+15]. The IoT vision brings new opportunities for automation, business innovation, collaboration and data analysis, but it posits two general challenges. Firstly, how to realise this bridging? How to adapt and mediate existing protocols so that the already-deployed devices' communication can be translated, and what should the future technologies be like to avoid translation and foster interoperability inherently? This bridging is successful if a holistic method for modelling and describing the set of devices, their capabilities and metadata is used, enabling discovery and querying of IoT resources. The second challenge is scalability - as the interoperability improves, fostering the growth of adoption, the number of connected devices and the volume of data produced by them increases. This raises challenges in coordinating the devices' functions and transporting and processing the large volume of data efficiently.

For the typical early adopters of smart devices and IoT (industrial manufacturing, logistics and healthcare), the tools and practices of Business Process Management (BPM) play an important role [LMM19]. BPM [Dum+18] is a research area concerned with observing, discovering, designing, analysing, automating and improving business processes. A business process (BP) is a (specific) sequence or collection of tasks, events and decisions aiming to achieve a specific goal within

one or across several organizations. As such, BPM marks the continuous effort to enact and improve BPs. In practice, it is often supported by software for automating the execution and monitoring of processes and analyzing historical traces of process execution. Such software are referred to as Business Process Management Systems (BPMS).

Integrating the IoT and BPM domains benefits them both - IoT data allows for finer-grained automated monitoring, analysis and control during Business Process execution. At the same time, the complexity of large-scale planning and scheduling of IoT devices' functions, dataflows and collaboration between individual applications and organizations can be enhanced by BPM [Jan+20]. Motivation of jointly using IoT and BPM is also driven by concepts such as Industry 4.0 [Gre+18], which in addition to the increased smart-device usage also prescribes a factor of high individual customizability of every manufactured product, something that BPMS can help effectively capture.

BPM has mainly been focused on supporting enterprise IT systems and provides comparatively less visibility/integration w.r.t. operational technology as found in industrial domains [KS14], for example, existing BPMS are generally not readily interfaced with the previously mentioned domain-specific protocols. On the other hand, Web technologies are generally well-supported in BPMS. The usage of Web services, Web technologies and Service-Oriented Architecture (SOA) to link different organizations and software has been a key aspect responsible for the success of using BPM in enterprise software systems. Web technology support is also important for IoT, as it is seen as a possible solution to the bridging problem and the lack of standardized communication protocols [Al-+15]. By following frameworks such as Web of Things [W3C20], which prescribes web technology usage for IoT, the integration of BPM and IoT already becomes feasible [HM11] - IoT devices expose and describe their functionality as web services, and the business process execution software invokes and composes these services.

Generally, IoT environments are much more volatile compared to the relatively static enterprise servers, causing the availability of IoT devices and services to be intermittent. For instance, energy-related downtime may occur for devices relying on batteries or energy sources such as solar panels. Outages due to physical damage are also more likely if devices are deployed in the wild: forests, fields, oceans, mining sites, etc. This has led to a body of research introducing adaptiveness, flexibility and self-healing capabilities to process-aware information systems [Wie+15; MMS16; SHA17], where the main goal is for the BPMS to compensate or avoid situations where an IoT resource used within the execution of a BP becomes unavailable.

This thesis focuses on another such source of volatility which is mobility. *Mobile IoT* (MIoT) represents scenarios where movement of devices in the IoT network is typical: smartphone-equipped pedestrians, (autonomous) connected vehicles and drones, for example. In MIoT, issues with wireless signal quality and connection availability are common - devices are constantly entering and exiting

the radio coverage areas of other IoT devices, wireless access points or cellular base stations.

Another concern is the centralized design of the conventional BPMS architecture, where the software platform is hosted at the cloud/data-center servers. The increasing count of IoT devices and data streams participating in processes managed by a central BPMS make it prone to become a bottleneck [Shi+16]. Secondly, in specific scenarios, this can also raise privacy concerns. An emerging IoT trend which aims to remedy this Cloud-side bottleneck problem is *Edge Computing* [Sat17].

Edge computing is the idea of using devices in the edge network to perform as many of the processing, networking and storage tasks as possible. Edge computing moves the data-handling from remote data-centers closer to the data source, near the IoT devices. While large-scale analysis and storage of historical IoT data in data-center-hosted servers are essential to IoT solutions, with Edge Computing, sending pre-processed data instead of raw data reduces network usage (congestion) and avoids overloading the Cloud by distributing some of the processing to Edge Devices.

Alongside Edge computing, the idea of *Fog computing* [Bon+12] has also appeared. With Fog computing, some of the edge network devices act as general-purpose compute, storage and networking providers. These devices, sometimes also called *cloudlets* [Sat+09], provide a platform for hosting applications or services, where the deployment can happen on-demand, similar to Cloud computing. We distinguish Fog from Edge computing based the former's usage of virtualisation technologies typically found in Cloud. Secondly, Fog specifically relies on stand-alone general-purpose devices, whereas in Edge computing, the IoT device that collects sensor data may itself also perform the computing. Thus, we see Edge as a superset concept of Fog.

These trends have motivated research to try and create BPMS that follow the Edge computing paradigm by distributing its components to the edge network devices [SHA17; Mas+18; RML12; Gre+18]. Industry players such as CISCO [KS14] have also highlighted interest in this, but the overall adoption of Fog or Edge Computing in popular BPMS software such as Camunda[1] is rare or non-existent. The degree to which process execution is shifted to the edge in a system can vary, depending on the requirements. For instance, a central system may still initiate and provide a global view of the process execution, but certain (sub-)processes or tasks can selectively be delegated to edge devices [DMC15; DFB14]. Alternatively, the most extreme case is deploying a full BPMS software on an edge device such as a smartphone or a vehicle's on-board computer. Works such as ROME4EU [RML12] and [Sch+16] have shown the feasibility of implementing such an approach, which we term *Mobile Process Host* (MPH) - denoting an edge network device with a BPMS that may also move.

[1] `https://camunda.com/`

The MPH approach eliminates reliance on a back-end server, the execution state is autonomously governed by the mobile device, and it interacts with nearby smart devices directly in a device-to-device manner. This suits scenarios where infrastructure is unavailable - disaster scenarios, Humanitarian Assistance and Disaster Relief (HADR), remote field work - or where privacy is critical (e.g. healthcare). Usage of remote services is still possible based on the availability of Internet connectivity.

While the previously-mentioned MPH-s can be used if an organization purchases and configures such devices for process management, Fog computing opens the opportunity to perform process management in the edge network on-demand. Fog computing can participate in process execution in 2 ways: A) as the performer of some compute-intensive tasks contained in a BP executed by the edge device, such as video processing, or B) as an on-demand host for deploying BPMS in the edge. Compared to integrating IoT and BPM, the usage of Fog computing in the context of BPMS has been researched less; the literature which goes in this direction usually focuses more on Cloud-based solutions [MVJ18; Wai+21; Bou+17]. These technologies (containerization, PaaS) are applicable to Fog, but the mentioned works do not explicitly mention Fog computing.

1.1. Motivating Scenario

To illustrate and concretize how MPH-s can be used in process-based Mobile IoT applications alongside Fog computing, we present a scenario case that motivates the research goals of this thesis, the scenario is also depicted on Fig. 1.

Walt the warehouse worker uses the logistics companies' smartphone-based MPH to organize and help automate Walt's daily tasks. Walt receives a new assignment: deliver some fragile goods packaged in sensor-equipped parcels to the parking lot, where a truck will pick them up. As he loads the parcels onto his forklift, he QR-scans each of them. This initiates a process instance **(P1)** per each parcel on the BPMS, which constantly monitors the sensors, making sure that the sensor levels are within normal. Otherwise, alerts and instructions on how to handle the situation are shown.

The exact process instance varies depending on the product, the BPMS fosters the management of the different processes. Further, Walt works in a remote area, cellular and WiFi are not available in all parts of the warehouse. To this end, the BP execution is done by the stand-alone BPMS embedded in the MPH, ensuring continuous, autonomous monitoring. When connectivity is available, events or alarms which occurred during the process are broadcast to the central system, where a historic trace of all parcels' transport is collected.

Walt brings the goods to the truck. As he hands them over, the P1 process instance also needs to be migrated over to the truck's on-board computer MPH, since the parcels will now be out of the wireless reach of Walt's MPH. Walt initiates a procedure to transfer P1 from his smartphone to the truck's MPH, making

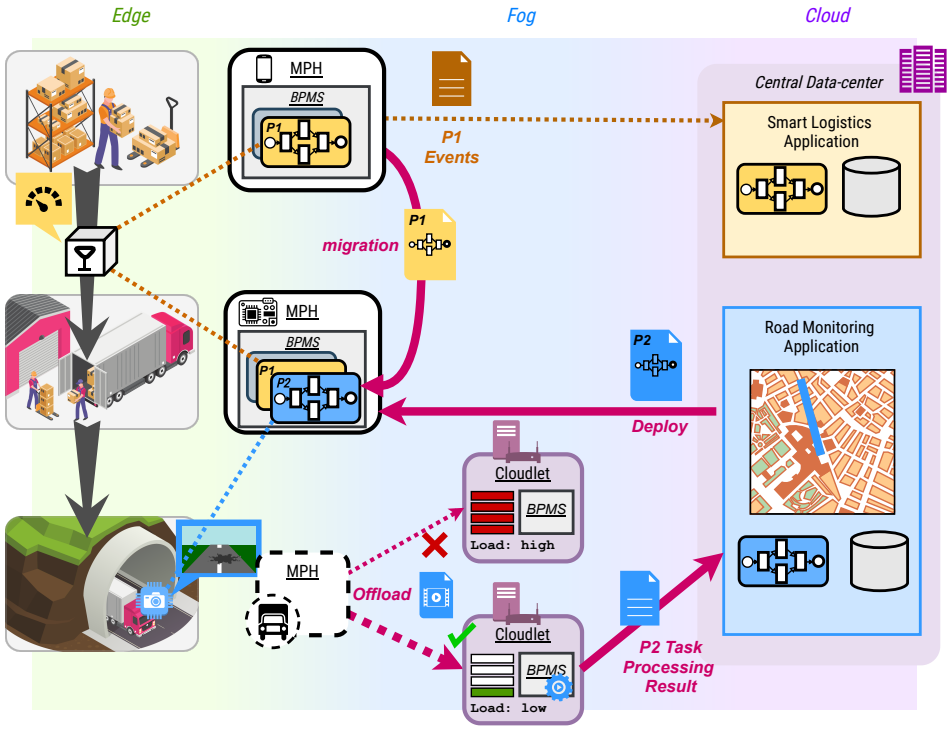


Figure 1: Motivating scenario for usage of Mobile Process Hosts and BPM across the Edge-Fog-Cloud continuum

the latter now the sole responsible executor of the P1 instance.

The logistics company is partnered with a road monitoring company's system, which performs daily monitoring of roads to identify pot-holes, debris, etc. in a crowdsourced fashion. Participants need to capture video of an assigned road segment and transfer it to one of the cloudlets deployed throughout the region by the local telco. The road monitoring company has deployed a video analysis service onto the cloudlets.

The trucker is assigned one such road segment and a respective process instance **(P2)** is started on the truck's MPH. After capturing the video, the 1st cloudlet which the truck passes is overloaded due to the large number of other clients already in the area. The BPMS opts to avoid transferring the data to this cloudlet, as it is aware of another one on the route. The 2nd cloudlet is idle, resulting in a smooth data offloading.

The cloudlet processes the captured video and sends the analysis results to the central system. As a result, the heavy-bandwidth video data is processed in the Edge and Fog networks, without a need to transfer large payloads to the central core network where the Cloud services are located, which may become a bottleneck in such a system.

1.2. Problem Statement & Research Goals

In the above scenario, the usage of a MPH allows processes to be executed in the Edge network in situations where constant reliance on a central system is not feasible - in the warehouse at a remote location or when driving through tunnels, for example. Moving the process execution engine from the relatively stable and static data-centers to the edge means that the process execution now must adapt to not only the changing context of surrounding IoT devices, but also its own mobility.

Existing works for process execution on Edge devices have addressed the need for adapting to faults, context changes, and provide more flexibility to process execution [SRT18] [HZL12] [Sei+15]. Device-to-device collaboration and migration-like functions with BP in the Edge are shown in [SHA17] and [Dar+15], however, these works mainly capture the context of external devices and not the internal factors of the edge BPMS host, such as its movement.

Mobility is an important source of uncertainty; it affects which devices, for how long of a duration and at what signal quality are available. Considering this, in the following, we identify problems of interest to this thesis and specify the respective research goals.

- Assuming that process execution is handled entirely by edge MPH-s, the movement of a host creates an issue if the process execution involves an IoT device whose trajectory is not identical to the MPH-s trajectory. When these trajectories are about to separate, one solution to continue the execution is to migrate the process to another MPH whose location or trajectory continues to satisfy the constraint. For example, consider processes that monitor IoT-equipped physical objects as in the described logistics company scenario (further detailed in section 3.1). If the cargo is handed over from person-to-person, organization-to-organization, the process execution can simultaneously be migrated between the MPH devices carried by the persons. Migration of process instances has been studied before [Bar+12], but not on resource-constrained devices such as smartphones. This motivates the 1st research question addressed in this thesis:

  Research Question 1: *How to realise execution of processes on edge network devices such as smartphones, with the possibility of device-to-device process migration during runtime?*

  To answer this question, our goal is to design and implement a prototype based on Android smartphones that can perform this migration using wireless technologies and assess the performance and technical complexity of such an approach.

- With Fog computing, processes are interested in making use of the fog infrastructure in certain tasks such as compute-heavy processing of data, such as the road video processing on cloudlets in the above scenario.

  When invoking a Fog server, several factors should be considered. Fog

nodes' locations and the MPH-s trajectory influence the availability and signal quality of the communication, affecting both energy consumption and data transfer rate. Fog node's load and hardware configuration affects the wait time and performance of processing the task. The task execution may be subject to and quality-of-service (QoS) constraints such as temporal deadlines and an energy budget. Thus, a decision has to be made which fog node to offload to or whether the offloading is even feasible - perhaps the task should be handled locally by the MPH itself. Ideally, this type of adaptive decision-making behaviour should also be configurable at the process modelling level so that business process designers can define the behaviour.

Research Question 2: *How to schedule the execution of process tasks involving nearby device-to-device communication, such as offloading to cloudlets, in an energy-aware and quality-of-service-aware manner while taking into account the mobility of MPH, the workloads and locations of the cloudlets?*

- While mobility plays a vital role in the edge network, performing mobility-involving experiments with real world-devices at a larger scale is expensive. Consider the complexity of distributed road and street condition monitoring as mentioned in the scenario (described further for a Smart City use case in section 5.3). Such a scenario may involve hundreds or thousands of MPH devices and road segments. Simulation tools could help in the initial assessment of process-based solutions involving Fog computing and mobility - to identify bottlenecks, analyse performance and scalability, assess costs. While adapting BPM to the edge may help avoid the congestion issues associated with a centralized process execution approach, it comes with its own cost, as fog infrastructure needs to be purchased and deployed; and the quality of service may be lower due to the network volatility. This shows the need to compare the cost and performance of edge process management vs centralized process management, simulation can help in this comparison.

  Research Question 3: *How to simulate scenarios with realistic mobility and Edge-Fog computing requirements, modelled and executed using existing BPM technology such as BPMN 2.0?*

1.3. Methodology & Contributions

In light of the raised gaps, this thesis proposes a technical framework to realise what we term Edge Process Management (EPM) with a heightened focus towards Mobile IoT application scenarios.

- We present the system design and prototype for a BPMS on the Android platform, where process instance execution can be migrated from one MPH to another during execution runtime. It allows for "roaming", continuously

executing processes, whose execution can be transferred from device to device without reliance on external services, increasing the resilience of applications such as sensor-based goods monitoring in cases where there is movement of goods and handover of goods (and sensors) between persons/organizations. The temporal performance and scalability of the migration are evaluated on real devices.

- Secondly, a heuristic decision mechanism for scheduling the execution of BP tasks that rely on nearby cloudlets is presented. The mechanism takes into account the movement trajectory and energy state of the BP executor, the locations and current load of nearby cloudlets, metadata about the task's complexity and the user's QoS preference to make a cost-efficient decision about which cloudlet to involve in the task execution or whether to resort to local/offline execution instead. The mechanism is demonstrated in discrete event simulation-based experiments using the Opportunistic Network Environment simulator (ONE) [KOK09].
- Finally, to support the development and research of process-based scenarios and models for EPM in environments such as smart cities, where mobility aspects are common, we present STEP-ONE: a Simulated Testbed for Edge-Fog Processes based on the Opportunistic Network Environment Simulator. STEP-ONE is an extension to the ONE simulator, adding process execution features based on the Flowable[2] BPMS. This allows discrete event simulation of scenarios, where the application logic is defined as BPMN 2.0 processes. Process definition is supported by a set of ready-to-use components, including events and signals for time, mobility, and connectivity-related situations in the simulator. Tooling supports visual design of process models and configuration options for varying the executed processes, their inputs and process engine parameters when executing a batch of simulations. STEP-ONE includes functionality for simulating compute tasks with different hardware configurations, which help realise Fog-computing scenarios. We apply STEP-ONE for a case study of EPM-based street condition monitoring in a smart city.

1.4. Thesis Structure

The rest of the thesis is structured as follows. Chapter 2 gives the state of the art on topics crucial to the thesis, such as business process management, internet of things and describes existing research addressing the overlap of the two in the context of edge-, fog computing and mobility. Chapter 3 tackles the 1st research question, focusing on technical aspects of executing processes in the edge using Android smartphones and is based on the work [MCS16a]. Chapter 4 addresses the 2nd research question, proposing a context-aware heuristic decision-

[2] `https://flowable.com`

mechanism, based on the work [MCS19]. Chapter 5 addresses the 3rd research question, describing the STEP-ONE testbed. The chapter is based on the work from [MSC20]. Finally, chapter 6 is the concluding chapter, including possible future research directions.

2. STATE OF THE ART

This chapter gives an overview of the fundamental background in IoT, BPM and Edge Process Management. For IoT, the chapter covers conceptual layered models of IoT, how interoperability is achieved with web services, how communication is organized, the computing paradigms for IoT such as Edge, Cloud and Fog computing. In addition to the fundamental concept of BPM and how BPMS software works, this chapter covers the existing research directions regarding usage of IoT with BPM such as process modelling or adaptiveness. The last part of the chapter focuses on execution of IoT processes in the Edge and Fog networks, including mobile execution.

2.1. Internet of Things

The term Internet of Things has been credited to Kevin Ashton, who used it when working on linking Radio Frequency Identification (RFID) technologies with the Internet in the context of supply-chain info-systems around 1999 [Ash09]. The overview of Internet of Things in this subsection is in part influenced by the book written by Sunyaev [Sun20].

At the most basic level, IoT represents the transformation of physical objects (things) and environments into "smart things" and "smart environments", whose status can be queried and state can be controlled from anywhere and at any time over the network. It is a reflection of and reaction to the trend of an ever-growing number of devices connected to the Internet and in particular, the growing count of Internet-connected devices per-person [Sha+19].

The building blocks of IoT consist of existing technology such as embedded devices, communication technologies, sensor networks, Internet protocols and applications. IoT can be seen as a continuation of previous research fields such as ubiquitous- and pervasive computing, cyber-physical systems. Significant effort has previously been put into forming a definition and characterizing the concept of IoT. Several organizations have developed documents in which they define or characterize the Internet of Things phenomenon, such as IEEE [MBR15], CISCO [Cis14], the International Telecommunications Union (ITU) [ITU12] and European Research Cluster on the Internet of Things [Ver+11] amongst others. For instance, IEEE produced a document extensively covering the state of the art of IoT and proposed two variants of a definition for IoT, distinguishing smaller scale, "small environment" and large-scale complex "large environment" scenarios [MBR15]. The latter is defined as:

> "Internet of Things envisions a self-configuring, adaptive, complex network that interconnects 'things' to the Internet through the use of standard communication protocols. The interconnected things have physical or virtual representation in the digital world, sensing/actuation capability, a programmability feature and are uniquely identi-

fiable. The representation contains information including the thing's identity, status, location or any other business, social or privately relevant information. The things offer services, with or without human intervention, through the exploitation of unique identification, data capture and communication, and actuation capability. The service is exploited through the use of intelligent interfaces and is made available anywhere, anytime, and for anything taking security into consideration." [MBR15]

Such a network can be seen as a system of systems, a better understanding of their nature can be examined using high-level architectural models. Several such architecture models have been proposed for IoT [Sun20], their aim is to conceptually organize the problems an IoT system manages. Next, we discuss 2 common examples of such IoT models: a five-layer and a seven-layer model.

2.1.1. Five Layer Conceptual IoT Model

The Five layer model [SS17], popular for its simplicity, consists of the *Perception*, *Transport*, *Processing*, *Application* and *Business* layers, shown on Fig. 2.

Perception The bottom-most Perception layer represents the sensor- (and actuator)-equipped devices (smart objects) which gather information about the physical environment or objects therein. Devices generally have a small form-factor and low energy consumption and may be battery-powered. As a result, they are relatively constrained in terms of computing resources (no. of CPU cores, clock speed, amount of memory). Devices are often based on embedded systems with power-efficient microprocessors (e.g. ARM architecture CPUs) or microcontrollers.

Transport This layer interconnects the Perception layer with the Processing layer, delivering the collected sensor data to the upper layers and vice versa, to relay processing results and actuation commands to the devices in the Perception layer. Internet infrastructure and wireless technology such as 5G, 4G-LTE, WiFi, ZigBee, Bluetooth Low Energy, Z-Wave, LoRaWAN, NB-IoT, SigFox are typically used to realize the Transport layer. In addition to transporting data, this layer allows smart devices to connect and interact with other smart devices, servers.

Processing This layer consists of a large set of services for storing, accessing, analysing, transforming the data incoming from the lower Transport layer. The services act as middleware to the other layers, typically containing database modules, big data- and stream processing; and are hosted with cloud computing technology due to the large data volumes.

Application Here use-case/domain-specific applications based on the IoT data and middleware services are delivered to users. They typically include a graphical user-interface through a web- or smartphone application.

Business The final, top layer manages the entire IoT system, defining rules governing the applications and services in the lower layers, driving e.g. user Quality-of-Service experience policies, privacy and security policies, business and profit models.

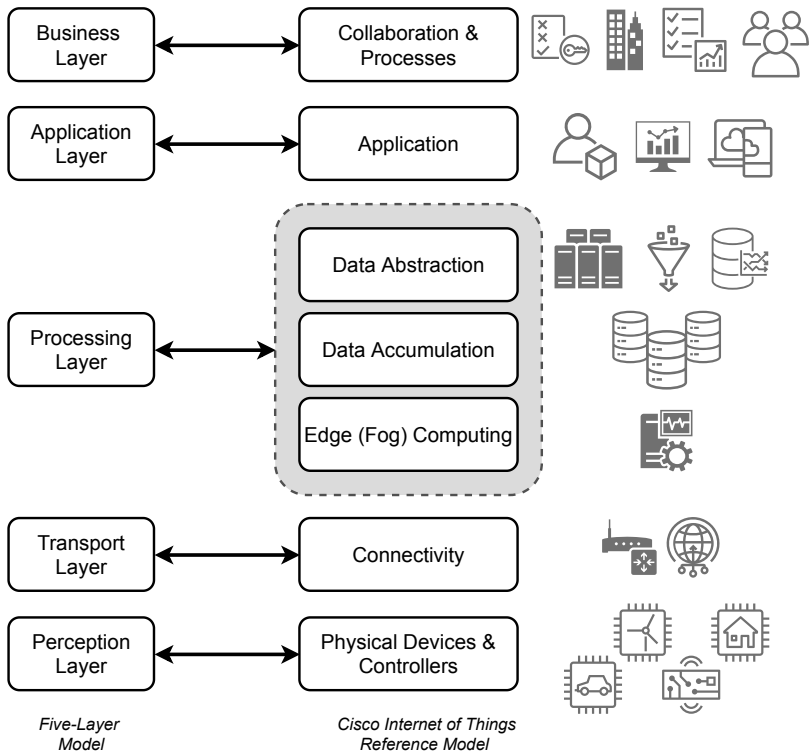


Figure 2: High-level architectural models of Internet of Things. Left - Five-layer model. Right - Cisco IoT Reference Model. Adapted from [SS17] and [Cis14]

2.1.2. Cisco IoT Reference model

Another noteworthy IoT reference architecture was presented by the IoT World Forum Architecture Committee, led by Cisco [Cis14]. It was developed in collaboration with companies from telecommunications and other sectors, amongst them were Oracle, Intel, Samsung, General Electric, General Motors. The reference architecture features 7 layers which in large part can be mapped to the layers of the 5-layer model as depicted on Fig. 2. The lower layers - *Physical Devices & Controllers*, *Connectivity* correspond to the *Perception* and *Transport* layers of the 5-layer model respectively. The top-most *Collaboration & Processes*, *Application* layers correspond to the *Business* and *Application* layers of the 5-layer model.

However, instead of a single Processing layer, the 7-layer architecture has split the data-processing related components into 3 distinct layers: *Edge (Fog) Computing*, *Data Accumulation* and *Data Abstraction*.

Edge (Fog) Computing This layer's main function is to transform the incoming flood of raw data into formats and events that are suitable for storage and higher-level analysis which takes place in the next layers (Data Accumulation, Abstraction). The layer embraces the principles of Fog computing: information processing should happen as soon possible and be performed as close to the data source in the edge network as possible. The concepts of Edge and Fog computing are explained further in section 2.3.2. Typical data processing activities include: detection of alerts/events based on thresholds for data values, re-formatting or re-structuring of the data, filtering, aggregation, reduction of the data to avoid redundancy and reduce network traffic in the upper layers.

Data Accumulation In the Edge (Fog) Computing layer and layers below, the data is said to be "in motion" and systems operating there work in an event-driven manner, reacting to the data as soon as it arrives over the network. At the Data Accumulation layer, the data "in motion" is captured and stored for non-real time access. Decisions are made whether or which data needs to be persisted (and for how long) and if necessary, the data are combined or aggregated with previously stored data. At this layer, the event-based data is converted for query-based processing, which is necessary for most applications in the higher layers. Further filtering or sampling of the data may happen at this layer and new events/alerts may be created based on analysis of the historical data trends.

Data Abstraction The Data Abstraction layer provides an abstracted data interface for applications in the upper layers. While data may be stored in different databases and different regions, this layer should combine the various data sources and provide a unified interface to it while maintaining consistent data semantics across the different sources. The interface should also be performant, prescribing indexing and normalizing of the data.

2.1.3. Discussion

The described two models give an idea of the collection of functions an IoT system must provide, however the physical organization of the layers/functions is not strongly defined by them. For instance, following the Edge Computing principle, this organization should maximize proximity of data processing to the data source in the bottom-most layer. However, an IoT system could also be built without any Edge processing at all, hosting all the layers except the *Physical Devices* and *Connectivity* in a cloud datacenter, for example.

Secondly, there is no consensus of any single IoT reference architecture, further overviews of large reference documents have been composed by Sunyaev in [Sun20] and by Weyricht and Eber [WE16], for instance.

In this thesis and for the rest of the state of the art chapter, primary focus goes towards the software-related aspects, which involve the top 3 layers (Busi-

ness, Application, Processing) more so than the Transport and Perception layers. Business & Application layer are largely captured by BPM, while Edge and Fog are concerned with the Processing layer. This thesis aims to enable processes in the Business & Application layers to drive the decision-making of using Fog infrastructure in the Processing layer with awareness of context in the Connectivity layer.

2.2. Interoperability and Communication in Internet of Things

This section discusses the importance of interoperability across the IoT layers and categorizes different types of interoperability. Then, we cover what protocols, standards are used on top of the physical data links - in the Processing and Application layers - that facilitate interoperability, and give special focus to frameworks which apply a service-oriented approach. Finally, we also discuss how IoT communication is organized physically in the Transport layer of IoT - which patterns are common in mediating information between devices, what (radio) technologies are commonly used for the physical level of communication.

2.2.1. Interoperability

Considering the layered IoT models, both intra- and interlayer interoperability is a crucial requirement for IoT. Intra-layer interoperability is based on the protocols, technologies, standards used by the individual devices and software within a layer to communicate between themselves. Interlayer interoperability allows software in one layer to interact with software in another layer. This involves larger-scale frameworks and standards, managing device- and service discovery, business-to-business cooperation, open APIs and public documentation, developer support etc.

A general lack of interoperability in the past has resulted in IoT applications developed for a narrow, domain-specialized use cases or within closed ecosystems. Such an approach makes the applications difficult to extend and maintain or integrate with other systems. To avoid this, we can consider interoperability from four different levels: device, network, syntactic and semantic [NAG19].

The devices in the *Perception layer* come with different hardware configurations, which sets constraints on their capabilities and interoperability - this is referred to as **device interoperability**. For example, virtualisation technology greatly simplifies portability of software. However, a low-end device, such as a microcontroller, may not support virtualisation features due to its hardware architecture, while more capable devices such as Raspberry Pi-s support it. Secondly, the communication hardware and protocols: WiFi, ZigBee, Bluetooth, Z-Wave, NFC and other similar radio modules embedded into the device distinguish whether a physical communication channel can be established and whether a new IoT device can be integrated to an existing IoT system.

In the *Transport layer*, **networking interoperability** determines whether end-to-end communication can take place, considering there may be multiple options (wireless/wired/delay-tolerant networking). For instance, a set of dispersed gateways may need to link their local networks containing end devices to a global IoT system available in the public Internet. Achieving a seamless message exchange in such a case can involve addressing, routing, QoS and security issues.

Once hardware-level communication issues and networking interoperability are ensured, **syntactic interoperability** is concerned with whether the data format, data structure and interfaces - e.g. Web Service Description Language (WSDL), RESTful API - used by IoT systems support integrating the systems with one another. Serialization and de-serialization of messages needs to follow the same formats (e.g. JSON or XML) or be able to translate between them. Otherwise, if the sender of a message does not use the same encoding standard as the receiver does for decoding, the message payload is lost.

Semantic interoperability - While syntactic interoperability ensures that the data is serialized in a known way, the contents of the data may still not be usable by different parties, since the representation of knowledge within a single data-structure (e.g. CSV or JSON) may differ. By following a standard for semantic interoperability, reasoning about data can be automated. Examples include SensorML [BR07] - a markup language for sensor data; Semantic Sensor Network Ontology, which is encoded in Web Ontology Language (OWL) and it's derivatives such as IoTLite [Ber+16]; or data structured according to a specific framework's semantics, such as FIWARE (explained below in 2.2.2).

While in the lowermost layer, the question is more about *how to securely directly link devices*, towards the upper layers, a challenging question is *how can the applications and processing components discover, query and optimally use only those data-sources which are of interest?* This requires abstraction of the underlying physical devices and annotation of them with meta-data describing their capabilities in terms of what IoT data they can supply, their context/environment and what standards they support. This is realized by middleware and IoT platforms which links a particular clients query/demand with a set of devices, using above-mentioned interoperability approaches to achieve it. However, one platform's abstractions and choice of standards are often not compatible with a different provider's platform - a lack of **platform interoperability**. The next subsection brings examples of some existing platforms.

2.2.2. Service Oriented IoT Frameworks

IoT frameworks aim to provide an answer to the interoperability challenges in the form of specifying a set of technological standards that address all the interoperability challenges in unison. Notably, the idea of Service-Oriented Architecture (SOA), which emerged in enterprise systems, has also become dominant within IoT frameworks [Spi+09]. SOA supports syntactical software interoper-

ability through standardized service interfaces based on Web Service (WS) technology, such as the previously mentioned WSDL, REST API, XML/JSON, OWL. Following SOA, IoT devices expose their functionality through service interfaces defined with these standards. While there are plenty of commercial frameworks, a few larger open-source projects also exist, such as FIWARE and Web of Things.

Fi-Ware. The European Commission-initiated FIWARE project[1] defines a set of APIs and open-source software to realise an interoperable platform for smart solutions, focusing on syntactic and semantic interoperability. The core of FIWARE is the *context broker* component, which mediates contextual IoT data between data producers (sensor, actuator devices, public info services) and consumers (applications, processing modules, etc.) [Cir+19]. Interaction is done through the FIWARE Next Generation Service Interface (NGSI). NGSI defines a data model for entities, their types, attributes, metadata, serializable as JSON. Further, it defines RESTful HTTPS interfaces for querying devices, data, including support for geographical context.

IoT data arrives to the context broker either directly from devices publishing data using NGSI API or alternatively, through a software agent which translates the device-specific interface to NGSI (e.g. from MQTT, SigFox, or OPC-UA), increasing device interoperability for FIWARE. Similar to the device-specific agents, FIWARE includes various optional modules which users of the platform can pick and choose. For example, this includes different domain-specific data models: Smart Agrifood, Smart Sensoring or Smart Cities.

More recently, NGSI is being superseded by the NGSI-LD standard, which adds linked data support and is an ETSI standard [FGB18]. NGSI-LD describes FIWAREs foundational data model concepts (Entitities, Relationships, Types) using RDF, OWL and JSON-LD, which are common standards for semantic web.

W3C Web of Things. Another framework for an interoperable IoT is Web of Things (WoT)[2], led by W3C WoT Working Group. Web of Things seeks to achieve an interoperable IoT architecture by using existing standardized Web technologies. WoT describes a set of building blocks and an abstract architecture defining the relationships between the blocks. In the following we give a quick overview of these building blocks.

The WoT *Thing Description* is the format for describing IoT smart object metadata and interfaces. The Thing Description is formatted as JSON-LD. WoT *Binding Templates* explains how protocols of a device (e.g. CoAP, MQTT, ModBus) should be translated to the WoT Thing Description abstractions and interfaces, and vice versa.

WoT Scripting API component allows interacting/composing Thing actions in Javascript without directly concerning the underlying protocols, they are abstracted away thanks to the WoT semantic and syntactic device-, device function

[1] `https://www.fiware.org/`

[2] `https://www.w3.org/WoT/`

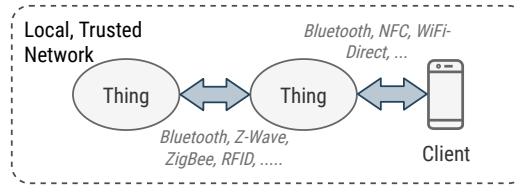


Figure 3: Device-to-Device direction communication

and interface descriptions.

The WoT project also includes a set of Security and Privacy Guidelines on how to securely implement and configure systems built based on WoT. WoT also proposes its own mechanism for device discovery, which can be based on DNS-Based Service Discovery.

Notably, the WoT project lists service mash-ups as one of the goals that the framework should help realise. For mash-ups, WoT supports orchestration using NodeRED.

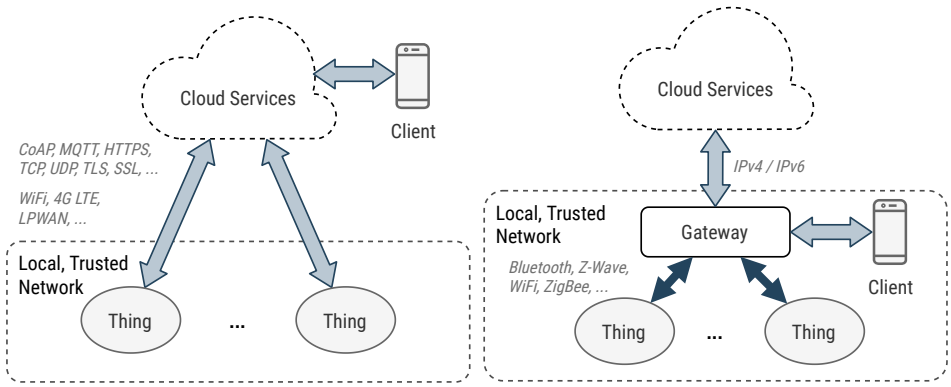
Commercial Frameworks. In addition to FIWARE and Web of Things, large software companies provide their commercial platforms to facilitate integration and interoperability of IoT solutions. Well-known examples include Amazon AWS IoT Core, Microsoft Azure IoT Suite, IBM Watson IoT. Besides these technology giants' platforms, numerous smaller companies specializing in IoT solutions offer their own platforms, such as Cumulocity, Ubidots, ThingWorx. In general, these frameworks offer similar interfaces based on service-oriented approaches and standards mentioned above. In case of platforms offered by Cloud providers such as Amazon, their solutions offer the benefit of seamlessly integrating with existing other Cloud services.

2.2.3. Communication Models & Protocols

Data movement within an IoT system can be organized in several ways, here we present 3 general IoT communication models which represent the basis of most IoT system designs [Yu+18].

For each one, we show how the smart objects (Things) in the Perception layer may share their collected data with other devices or a client, which is typically an application running on a smartphone / personal computer or a server running business logic.

Thing-to-Thing. Fig. 3 represents direct, device-to-device (D2D) communication between individual Things in the Perception layer (sometimes also called Machine-to-Machine). The communication does not rely on any external hardware device (such as a network master or router), the direct link is usually based on a radio-based solution such as Z-Wave, etc. A common example can be a smart lightbulb, which is controllable via Bluetooth using a Smartphone. The limitation of this model is that remote control/monitoring is not possible, but on the other hand the privacy and security management is reduced to only the local physical



(a) Thing-to-Cloud communication (b) Gateway-based communication

Figure 4: Cloud-related IoT communication

network. Further, devices which embed multiple radio capabilities (e.g. both WiFi-Direct and Bluetooth), may choose or combine the different technologies (hybrid approach) to achieve the best latency/QoS.

The devices themselves are responsible for interoperability - they embed software stacks that allow clients to discover device capabilities, query data or submit commands/operations (e.g. in case of Bluetooth, the GATT protocol). The drawback of this is that each link depends upon both devices supporting the same protocol (which often sets hardware-level requirements).

Thing-to-Cloud. To foster data sharing and easier, remote access to the devices, as depicted on Fig. 4a, the system relies on a remote server (typically cloud-hosted), to which the Things connect over an IP network. The cloud usually performs functions such as providing persistent storage for the (historic) IoT data and acting as a proxy to the IoT device, forwarding commands, queries to the device on behalf of a client. For this, the server provides a virtualized representation of the connected devices to external clients for data querying or command submission. From the device's perspective, this prescribes the usage of communication technology that is compatible with IP networks (WiFi, LTE, NB-IoT), involving cellular base stations or wireless access points.

Here, the Thing-to-Cloud link may be based on a protocol which minimizes data payload sizes and latency, such as MQTT. At the cloud, the small-format data is usually transformed to some other format which is easier to share at the interface for clients. The drawback is that embedding a TCP/IP stack on the Thing can hamper power-efficiency and drive up cost of hardware, while on the cloud-side, scaling to a large number of devices can become challenging considering the network bandwidth and data volume [You+19].

Thing-to-Gateway. In Fig. 4b, a **gateway** device is introduced into the local network as a mediator between the Things and the Cloud. The gateway acts as a unifying bridge, translating the different local network protocols used by the

Things into a single IP protocol for use by the Cloud-hosted services. Such an intermediary also allows moving certain application components from the remote server-side to the local gateway, for instance data compression, filtering, encryption, anonymization could be done at the gateway before leaving the local network into the (public) cloud.

From the perspective of client-side access to the IoT services, the gateway also creates the opportunity of hosting a local version of the client interface, decreasing latency and reducing traffic at the cloud. Typically, the cloud-side interface still remains an option even with the use of a local gateway, as it allows the system to be reachable anywhere over the Internet. A simple example of this structure is a wearable medical device which forwards data to a smartphone, which then publishes it to a cloud application.

Discussion. These 3 models serve only as the fundamental building blocks of network structures in IoT systems. Hybrid approaches are possible and for instance, a system may have an entire network of gateway-type devices both in the local and remote network serving the IoT system, as is the case with Fog computing, described in chapter 2.3.2.

2.2.4. Delay-Tolerant & Opportunistic Networking

The described communication models may be employed in various networking conditions. If we consider Thing-to-Thing and Thing-To-Gateway communication, the lack of a stable network connection is more common, especially if the Thing is moving. In this case, the communication in the edge must be based on brief, device-to-device wireless communications. The fields of Opportunistic Networking and Delay-Tolerant Networking have studied this problem for more than a decade [Tri+17].

In Delay-Tolerant Networking, to transfer messages to destination host that is not reachable by a direct device-to-device link, routing protocols based on a store-carry-forward scheme are used. Hosts exchange messages during encounters, for which the final recipients are not currently available, and as the hosts later move, these messages are further propagated to other encountered hosts, with the aim that the message will eventually reach its destination host.

Opportunistic networking aims to take into account contextual information such as mobility patterns (moving from home to work, bus lines), daily routines and social interests when routing messages in delay-tolerant networks or when forming mobile ad-hoc networks.

2.3. Computing Paradigms for Internet of Things

Large, scalable IoT systems rely on modern computing paradigms such as Cloud Computing. While Cloud usage has become common-place, it alone is not sufficient to realize the full potential of IoT, namely more de-centralised approaches

such as Edge and Fog Computing are also necessary, as we explain in the rest of this section.

2.3.1. Cloud Computing

Cloud Computing refers to the provisioning of (virtualized) compute, storage, networking resources on-demand, as a service [BVS13]. *Public* Cloud Computing service providers such as Amazon AWS, Microsoft Azure, Digital Ocean run datacenters with server clusters providing a wide array of different Cloud services. For instance, one may rent Virtual Machines (VM-s) and disk space with a desired configuration and manage the entire software stack within the OS by oneself or alternatively, directly deploy application code to a language/framework-specific runtime environment avoiding the configuration at OS-level. The former approach is known as Infrastructure-as-a-Service, the latter Platform-as-a-Service. In addition, Software-as-a-Service refers to usage of applications hosted in the cloud by thin clients (e.g. web browsers), removing the need to install software on the client device/manage its updates.

These different Cloud services have revolutionized the IT industry, drastically reducing the upfront capital expenses of establishing a software company and allowing Cloud users to quickly elastically scale their resources based on needs, on a pay-per-use basis. For IoT solutions, Cloud is often used for hosting user-facing applications, middleware, and for data storage (the Business-, Application- and Processing layers of the 5-layer IoT model).

In addition to typical use cases for Cloud, such as hosting web applications, the simultaneous rise of smartphone usage at the end of the first decade of the 21st century sparked interest in using Cloud resources to extend the limits of mobile devices. Utilization of Cloud resources by a mobile, for offloading, storage, or other purposes, is known as Mobile Cloud Computing [Din+13]. The motivation is to conserve the battery life or making processing more performant by moving computational tasks from the mobile CPUs to the Cloud, where processing, memory and storage resources are abundant. The transfer of computational tasks is referred to as computational offloading, or *offloading* in short [Kum+13].

Practical examples of mobile code offloading included mobile applications from large companies such as Google which executed tasks such as voice or image recognition on remote cloud servers on the input from the smartphone. In such solutions, the integration with the (Cloud) back-end is tightly coupled with the software product. In contrast, academic researchers were interested in creating general-purpose frameworks that leverage virtualisation, dynamic deployment for offloading of code from smartphones to (Cloud) servers at different levels of granularity. Another wide research trend is development of algorithms for decision-making whether offloading is necessary/beneficial from the mobile's perspective [Kum+13].

Cloud datacenters represent a relatively centralised approach from the network

structure perspective. For instance, Amazon Web Services (AWS), one of the leading Cloud providers, operates a total of 6 datacenters within Europe[3]. If the Application and Processing layer components from the 5-layer IoT model are hosted in the Cloud, the data flood of IoT converges at such data centers.

From a networking perspective, several researchers have noted that the centralised Cloud infrastructure is not well suited to handle the growing volume of data produced by the dispersed IoT devices [Zha+15; Bot+16; Bit+17; Sat17; APZ18]. The same applies for mobile offloading for low-latency interactive applications with high bandwidth, such as Augmented Reality or critical systems such as collision avoidance in autonomous vehicles [Bit+15]. The issue lies in network latency, involving two factors: 1) the mentioned geographic datacenter placement sets fundamental latency limits based on the users/devices location and speed of light, 2) the growing volume of network traffic, which leads to congestion and jitter. In response, the Edge Computing [Sat17] paradigm has emerged. Besides networking, another issue arises when regulation constrains the allowed locations for data processing or storage, but the Cloud operators datacenter locations do not comply with this [Mou+18].

2.3.2. Edge Computing

Edge Computing decentralizes the Cloud-centric IoT approach by using compute nodes in the immediate vicinity of the IoT devices to capture and process the data as close to where it is being produced as possible, instead of transmitting data to the datacenter. The processing (or storage) could be done by the data source device itself (e.g. a camera which has face detection/event detection capabilities), a nearby networking device such as a router with adequate capabilities or a stand-alone machine dedicated for the purpose, sometimes referred to as a *cloudlet* [Sat+09].

The origins of Edge Computing can be seen in content delivery networks (CDN's) in the end of 1990s [DPW04]. CDN-s were conceived to speed up performance of web requests by caching content at nodes close to end users. CDN's not only serve static cached content, but could involve some dynamically composed content (e.g. advertisements on a website that are based on the user's location).

Satyanarayanan [Sat17] highlights 4 key benefits of Edge Computing:

1. Minimal latency - processing in the edge network ensures the lowest possible communication latency and least network hops, this allows reacting to the data quickly and realizing applications with real-time latency requirements such as augmented reality.
2. Scalability of systems increases, as the stream of frequent, raw data gets consumed at the edge network and only already processed, extracted data

[3]As of July 2021 Frankfurt, Ireland , London, Milan, Paris, Stockholm according to https://aws.amazon.com/about-aws/global-infrastructure/

are transmitted to nodes further away (such as the ones handling functions of the upper layers in the discussed IoT models).

3. Privacy. The proximal edge compute nodes can enforce privacy policies early in the edge network, before the data traverses to the cloud over Wide Area Networks.
4. Masking cloud outages. By hosting fall-back functionality on cloudlets, failures and outages of the cloud infrastructure (e.g. due to network failure or cyberattacks) can be temporarily masked.

Although the above latency-argument has become debatable, as recent studies [Moh+20] show that Cloud round-trip-times in areas like Europe or North America from experimental probes are under 20ms for 80% of probes in optimistic scenarios, we can still recognize the utility of Edge based on the combination of the mentioned benefits.

2.3.3. Cloudlets, Fog Computing

In this thesis, we use Edge Computing as an umbrella term for several similar concepts, including *cloudlets*, mobile edge computing, microclouds and Fog Computing, Mist Computing [You+19]. This subsection aims to give an overview and comparison of these similar terms.

In 2009 Satyanarayanan et al. presented the idea of *cloudlets* - "a trusted, resource-rich computer or cluster of computers, well-connected to the Internet and available for use by nearby mobile devices" [Sat+09]. Cloudlets provide a VM-based platform to which mobile clients can deploy and subsequently use software packaged as a VM on-demand (e.g. a speech recognition service). The mobile client may alternatively choose to use a distant cloud server or its own resources in cases where cloudlet infrastructure is not available. The vision of cloudlets prescribes their decentralized and widely dispersed deployment, similar to how Wi-Fi access points have been deployed.

The idea of using devices dedicated for general-purpose / utility computing located close to the clients was driven further by the Fog computing term proposed by Cisco in 2012 [Bon+12]. The core idea is highly similar to Edge and cloudlets - providing a (virtualized) platform for hosting compute, network, storage, usually at the edge of the network. However, Fog Computing represents a larger, more general system - a network of nodes spanning from the cloud to core- and metropolitan networks and to the fog nodes in the edge network, as opposed to the self-contained cloudlets. Another way to look at the concept of Fog computing is to see it as a set of cloudlets, who may rely on other cloudlets or the Cloud when necessary (multi-layer architecture).

Compared to the original Edge computing idea (Sec. 2.3.2), there is no strict restriction for placement of the Fog nodes / Fog services, the services may or may not be close to the data source. Fog Computing prescribes usage of virtualization, while Edge Computing can also take place without it. Secondly, Edge Comput-

ing does not explicitly require dedicated nodes for the processing, it merely states that data should be processed near the data source, possibly even on the same device that generated the data, whereas Fog Computing and cloudlets are described as separate devices from the data producers. Due to the autonomous nature of cloudlets, they need little management compared to the network of Fog computing nodes, which need management and federation of services and providers.

We shall refer to the computing resource providing devices, such as cloudlets or the nodes in Fog computing as *Fog Nodes*.

Another significant Edge initiative is the specification of Mobile Edge Computing (MEC) [Hu+15] by the European Telecommunications Standards Institute (ETSI) in 2014, later renamed to Multi-Access Edge Computing. MEC takes the principles of Edge Computing and focuses on realizing them in mobile cellular networks - using the cellular base stations and/or Central Offices as the nodes providing the edge computing platform. A driving force behind MEC is the trend of virtualising the network components in mobile network infrastructure, known as Network Function Virtualisation (NFV). The dedicated networking hardware is replaced with general-purpose computers, which results in the telcos effectively operating small data-centers which run various virtualised network function services. This creates the opportunity to open the Radio Access Network (RAN) edge (e.g. Core base stations) to trusted, authorized 3rd parties and provides means to flexibly and quickly deploy applications and services to the RAN, where they are hosted near to the mobile users. An interesting concern for MEC is the focus on client mobility - the edge services need to follow the user as they move, for this RAN data can be beneficial, as for example 5G networks bring enhanced user mobility tracking features. An example commercial solution based on MEC is Amazon AWS Wavelength [4], which allows deployment of AWS compute and storage services within the mobile network infrastructure of telco providers, avoiding the latency of traffic having to leave the telecommunications network.

2.3.4. Practical Fog Computing - Standards, Technology

The biggest international consortium in Industrial IoT and Fog/Edge Computing is the Industrial Internet Consortium (IIC). While IIC-s focus was originally accelerating the secure adoption of IoT technology for businesses, industry and society in general, in 2016, IIC was joined by the OpenFog Consortium, expanding the scope and partnerships more towards Fog networks and Edge computing. The OpenFog Consortium was founded in 2015 by Arm, Cisco, Dell, Intel, Microsoft, and Princeton University [Ope17] and in 2017, OpenFog Consortium published the reference architecture for Fog Computing [Ope17]. The document intends to aid business leaders, software developers, silicon architects, and system designers create and maintain the hardware, software and system elements necessary for fog computing. The document describes how Fog systems can be assembled and

[4] https://aws.amazon.com/wavelength/

function in terms of necessary technical features related to networking, security, virtualization and describes several use-case scenarios which show the necessity of a Fog Computing framework (instead of a Cloud-based solution). The OpenFog reference architecture was adopted as an IEEE standard in 2018 [Ass18].

On the hardware side, the majority of existing solutions are proof-of-concepts published as academic research. Few commercial products do exist, such as Cisco's routers for the Cisco IOx platform. Such routers support application deployment based on VM and containerisation technology. Containerisation is a crucially important virtualisation technique for Fog computing, being a more lightweight alternative to VM-s and provides the opportunity of sharing a base software stack layer among multiple applications packaged as containers.

2.3.5. Key Challenges for Fog Computing

To summarize, Fog and Edge computing refer to distributed systems spanning the continuum from cloud datacenters to devices in the edge network. The systems offer services for storage, processing, networking purposes in an on-demand, elastic fashion à la Cloud Computing. The need for these services is for achieving low-latency, data-privacy, and distribution of networking/computational load in the whole network. In this subsection we highlight some key remarks regarding Fog Computing and its applications, challenges interesting to this thesis.

Mobility and Location-Awareness. An important challenge for Fog Computing is enabling mobility: usually movement of client devices, but also movement of the Fog nodes themselves. For a client device on the move, such as smartphone worn by a pedestrian, vehicle, drone, the set of available Fog nodes is constantly changing as the client traverses between local-area networks. The Fog network needs to adapt to the mobility of the clients: services and applications have to be migrated between nodes to ensure proximity to the clients and intermittent connectivity due to changing signal strengths must be taken into account. An abundant geographical distribution of fog nodes also creates opportunities for new location-based services (LBS) [Gua+18]: clients can use LBS without sharing their location data with the Cloud - this is kept private at the Fog node. The fog-hosted LBS can only receive requests from nearby devices, so explicit location data need not be shared to maintain location-awareness.

Load-balancing and scheduling. The resource allocation within the Fog network must be efficiently managed, optimally balancing low-latency requirements with capabilities and current workloads of the nodes in the network. In Fog Computing, the balancing and task placement also includes the Cloud resources, not just edge network nodes. This is challenging due to the heterogeneity of Fog network nodes - differing hardware capabilities, workloads and latencies need to be accounted for in the decision-making [Sun20]. For mobility-oriented scenarios, the optimal resource balancing further needs to take into account the mobility context in addition to the workloads and hardware configuration.

2.3.6. Fog applications - overview

In this subsection we analyse software applications for Fog from 2 perspectives. Firstly, which characteristics of an application make it most suitable for Fog/Edge instead of Cloud. Secondly, which party (client or central system) is the main invoker of services.

Low-latency, high-bandwidth applications. While a wide array of applications could be adapted for Fog, the largest benefit is reaped by those with low network latency and high local bandwidth requirements. High bandwidth applications either involve: 1) large amounts of data generated by single entities (e.g. 4K video camera), industrial equipment/automotive equipped with a large no. of high-frequency sensors; or alternatively 2) the single entities' data production is relatively small, but the number of total devices whose data needs to be transported is large (e.g. thousands of traffic sensors deployed over across a smart city). Fog nodes located as close as a single network hop from the client devices offer low latency. While in ideal conditions Cloud-hosted applications can still provide quite low latency (10-100 ms depending on region [Moh+20]), they are still prone to suffer from jitter, making Fog the best candidate for latency-sensitive applications. These are applications which need to provide feedback/react fast enough so that the experience is seamless for humans - AR/VR applications, gaming, health monitoring or industrial scenarios where the intervention/reaction delay must be minimal.

Top-down and bottom-up approaches. We categorize the applications also based on who initiates and manages the application and services, i.e. how the control logic is distributed within the network [Man+19]. The two approaches -*top-down* and *bottom-up* [MRB21]- are visualized on Fig. 5.

In case of a **top-down approach**, a central controller (often Cloud-hosted) deploys necessary resources to other nodes in the geo-distributed network (e.g. services S1, S2 on Fig. 5). This is based on information about the entire Fog topology managed by the Cloud. Devices carry out instructions received by the controller. This centralised view allows for very complex decision-making for resource allocation, considering user/node loads across the Cloud-Fog continuum, etc. This assumes a centrally federated network. Top-down approach can achieve efficient load-balancing and realize applications which "follow" the user as they travel. For example on Fig. 5 S1 can be deployed to those Fog nodes which are predicted to be near Client A during different moments of time (*t0*, *t1*). The top-down approach is suitable for services which are shared between a large no. of clients, e.g. a road-side unit at a traffic junction, exchanging data between passing vehicles and a central system. Top-down approach enables optimization for large numbers of users: consider CDN-s where the data is cached according to knowledge about which content is requested in which regions.

Alternatively, in the **bottom-up approach**, the resource deployment and management is initiated by the client devices and/or Fog gateways. They try to query

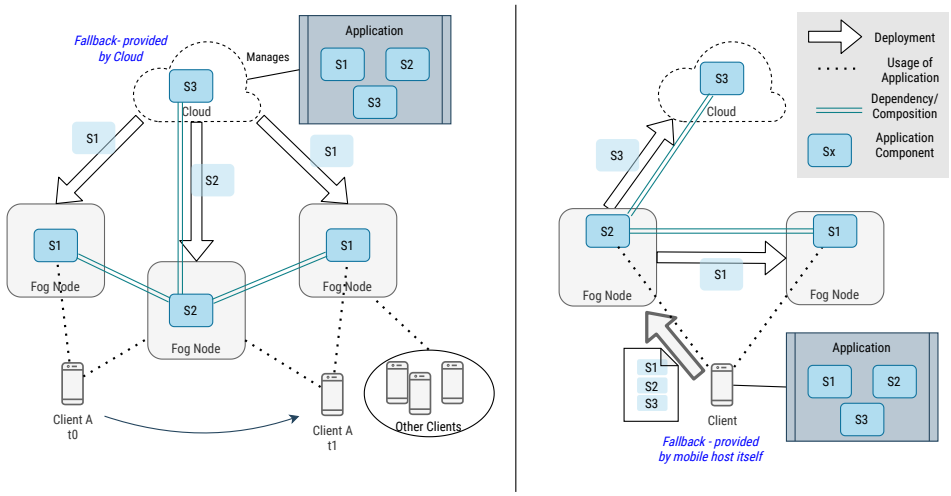


Figure 5: Approaches to application initiation in Fog: Top-down (left) and Bottom-up (right)

the network for availability of nodes which could be used for the deployment. For example, on Fig. 5, the Client distributes S1-S3 to a nearby Fog node, which in turn deploys S1 and S3 to another peer Fog node and the Cloud, respectively. Here, usage of nodes by different vendors (public, private clouds, public/private fog nodes) is more common. The bottom-up approach is thus more de-centralised and autonomous, the resources are used opportunistically based on their current availability, the Fog gateway can influence the entire Fog network less compared to the top-down Cloud perspective. Examples of this approach can be seen more in sensor network research, crowdsensing, mobile code offloading, etc. Generally, the applications support a single client at a time.

Both of these solutions should also be capable of adapting to cases where additional resources are not available at all, and the application needs to have a fallback option, to work based solely on the client and controller.

2.4. Workflow & Process Management

In this section we explain the fundamental concept of BPM and then show the current situation of applying BPM in the IoT context.

Business Process Management is the practice of designing, monitoring, analysing, improving and automating business processes [Dum+18]. Business processes themselves are the fundamental set of actions that businesses and organizations enact to deliver a product or service. They involve a sequence of actions, events and decisions and BPM aims to manage these processes as a whole, not just focusing on improving some individual sub-actions.

An example of a process could be a *Procure-to-pay* process - where an organization needs to purchase a product or service, they go through steps such as

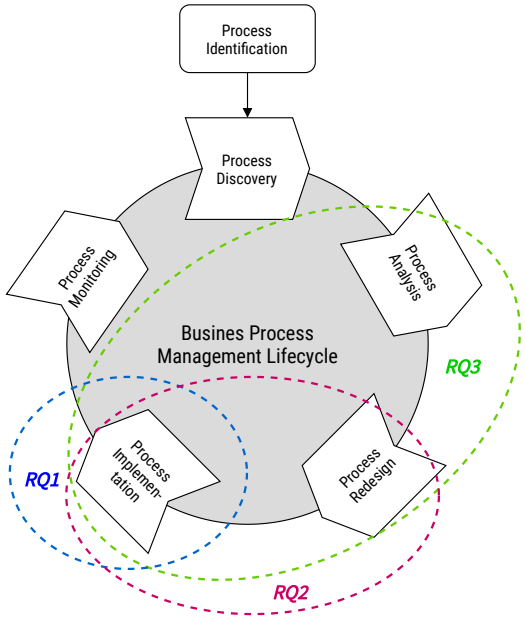


Figure 6: Phases of the Business Process Management lifecycle. Adapted from [Dum+18]

obtaining quotes, selecting a supplier/vendor, issuing a purchase order, consuming the service once delivered and paying the invoice. This type of process could take place when a client requests computing resources from a set of Fog Computing providers and decides on a single provider. A continuous application of the different BPM activities is referred to as the BPM lifecycle.

2.4.1. Business Process Management Lifecycle

The so-called BPM lifecycle describes the components of BPM and how they relate to one another, forming a cycle. The following explanation is adapted from the book by Dumas et al. [Dum+18], where the main lifecycle components considered are: Process *Identification*, *Discovery*, *Analysis*, *(Re)-design*, *Implementation* and *Monitoring*, as illustrated on Fig. 6.

To begin applying BPM, an organization must first look at it's existing processes and identify which of these are of interest for a particular business problem at hand. This phase, referred to as *Process identification*, results in a selection of processes of the organization to be managed throughout the rest of the lifecycle.

Next, in the *Process discovery* phase, the identified existing processes are documented in detail. The goal is to accurately capture the existing processes as-is. This can be done with modelling tools such as BPMN 2.0. While not required, Process Discovery may be supported with automated tools and techniques such as Process Mining. Process Mining refers to analysing historical log traces and gain-

ing insights from them, including automatic generation of process models based on them. To this end, specialized process mining software such as ProM [Van+09] exist.

Issues related to the as-is processes are identified and documented in the *Process Analysis* phase. Quantitative performance metrics may be used to achieve this. As a result, a prioritized collection of issues with the existing processes is produced.

Based on this, changes to the processes are proposed, analysed and eventually chosen to create a re-designed process. This phase is termed "*Process redesign*" and results in an improved, "To-be" process model. At this stage, various analysis techniques may be employed to determine which of the changes to choose for acting on, such as using Simulation to play out various versions of a process and their performance, identify bottlenecks, usage of resources. An example of such a simulation tool is QBP simulator[5].

In the *"Process implementation"* phase, the as-is process(es) in the organization are then transformed to the to-be process(es). The changes needed to achieve this include both organizational change management and automation aspects. The latter concerns development, deployment and operation of IT systems which support enacting the to-be process. The organizational change management concerns the participants involved in the process and changes to the work culture, skillsets, etc.

During enactment of the deployed re-designed process, relevant metrics data are collected to further analyse how the process is performing. The aim is to identify bottlenecks, recurrent issues, misalignment between the modelled process and actual execution of the process. The result of this (called Process Monitoring) is a new collection of issues with the process(es) to be improved. Finally, this closes the cycle, as the newly identified issues from the Process Monitoring lead to a new iteration of the cycle - again identifying the existing process, prioritizing the flaws and improving them.

The research questions addressed by this thesis are mainly concerned with the Process implementation phase, as RQ1 and RQ2 directly deal with how to implement and execute processes. RQ2 also touches upon how the adaptive behaviour can be captured in the process models. RQ3 is concerned with Implementation phase, but also the process Analysis and Redesign phases, as a simulation tool allows for analysing and experimenting with different process designs. The mapping between the research questions and BPM lifecycle is also depicted on Fig. 6.

2.4.2. Modelling and Analysis of Business Processes

The common approach to modelling business processes is using standards such as BPMN 2.0, CMMN and others. These standards establish semantics for describ-

[5] `https://www.qbp-simulator.com/`

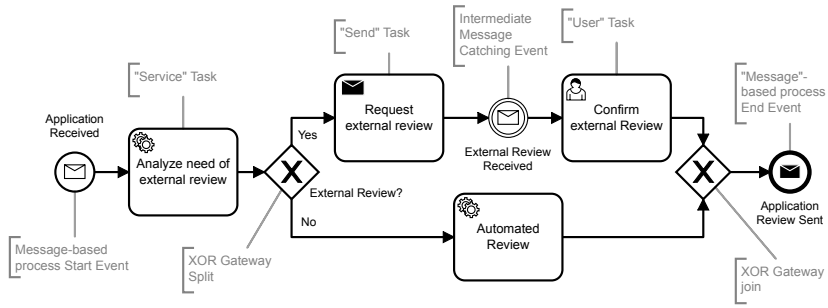


Figure 7: Example of business process visually modelled using the Business Process Management and Notation 2.0 (BPMN 2.0) standard.

ing Business Processes - the activities, resources involved, decisions, pathways and events. The most prevalent modelling language here is the Business Process Model And Notation Specification Version 2.0 (BPMN 2.0, or just BPMN). It defines an XML-based schema for defining business process diagrams in flowchart-like fashion, and it also includes visual specification of how certain messages, activities (e.g. human or non-human tasks) should be represented.

Thanks to the XML-backed format, BPMN models can be interpreted by software for analysis or automation, supporting the implement phase. While BPMN itself is not tailored for a particular execution or implementation environment, the standard is easily extensible, so an information system implementing or automating its processes with BPMN can define its own extension elements to the model according to its needs.

A visual example of a BPMN diagram can be seen on Fig. 7. It represents a process for reviewing applications (e.g. for a loan), and based on the information provided by the applicant, the application may need external review from a 3rd party that then has to be checked by a human or can be automatically reviewed by the companies' info-system if it is determined to be a non-exceptional case. After a review has been conducted, the applicant is sent a message with the review, finishing this process. Looking at the example model left-to-right, it starts using a "Message-receive"-type *Start Event*, which indicates that an instance of this process is started upon receiving a particular kind of message. Subsequently, the flow of this process includes different types of tasks (Service Task, Message Send Task, User Task) and involves a split of the control flow, which is represented by an XOR gateway. Different task types reflect some differences in how they are enacted. User tasks involve some input or operations from the user, e.g. through some form-based graphical user interface, while Service tasks involve invoking some software service, e.g. performing an HTTPS request to a Web Service. Message-related tasks and events involve sending and receiving of messages, the messaging medium depends on the implementation/use-case: e.g. e-mail, instant messaging, web requests, SMS, etc.

In cases when more formal analysis is required, methods such as Petri Nets

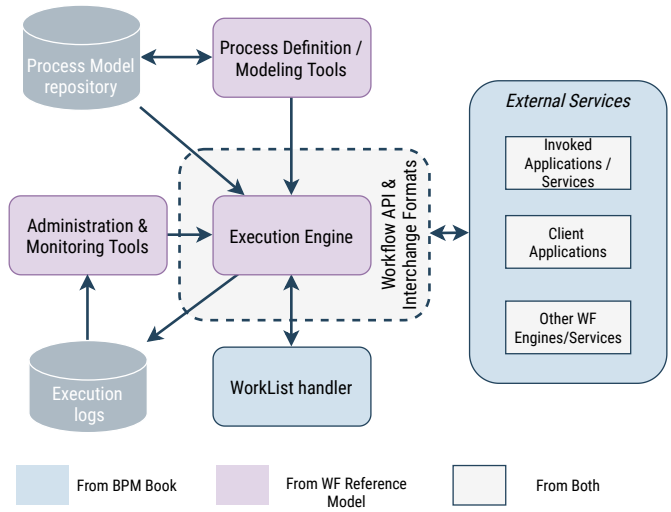


Figure 8: High-level architectural view of a typical business process management system

or graph-theory-based methods may be used to model the processes. Petri Nets and its variants such as Timed Petri Nets are mathematical modelling languages, which allow using the existing theories and proofs surrounding them to perform rigorous analysis of processes. For instance, Petri Nets allow verification of process models (checking for deadlocks, checking for the reachability of a desired state from a given current state) or with Time Petri Nets, consider temporal constraints.

2.4.3. Process-Aware Information Systems, Process Execution

An array of open-source and commercial software aimed at supporting the BPM lifecycle stages exist. These systems, called Business Process Management Systems (BPMS) or Business Process Management Suites or sometimes Process-Aware Information Systems (PAIS), each combine several tools, the typical components of such a system are the following:

- **Process designer (modeller)** - a visual modeller for creating BPMN (and other) models. The tool allows defining the executable processes for the BPMS. This includes defining the control flow, data flow, and other constraints, such as assigning which resources can be assigned to which activities (e.g. based on business roles).
- **Execution engine** for the implement phase, which drives the automated enactment of process definitions in software - managing process instance state, involved resources, etc. It orchestrates the invocation of required services / other components of the BPMS according to a process's state. As a side-effect of process execution, log traces are written to an *Execution log*.
- **Worklist handler** Based on the active tasks within running process in-

stances and available resources, the worklist handler maps which tasks may be completed by which resources. It provides an "inbox"-like functionality, where users/clients can query which work items (tasks) are currently available to them. When one resource assigns/accepts a work item, it then becomes locked and cannot be assigned to other resources.

- **Administration & Monitoring tools** These provide overview of the available processes, resources, running instances. Among administrative tasks is managing the resources such as the roles of human users in the system (e.g. junior or senior level employees have different roles in the system, Worklist handler assigns them different tasks based on this). Monitoring, analysis and insight tools facilitate monitoring the state, statistics of currently running instances as well as historic data. This part may include advanced, business process intelligence functions which allow analysing performance of process execution, detect outliers and even suggestions for improvement.
- **Workflow clients and External Services** - BPMS are often designed to work with a kind of Client Application, often HTTPS / Web interface based or some stand-alone client software (e.g. desktop client or mobile app), to interact with the BPMS. In case of user-facing web-based clients, the BPMS provides libraries and standards to generate UI-s (e.g. input forms) for tasks based on the process models, to support process execution, saving development time of creating GUI-s for the tasks. Another common feature is bundling various integration modules and libraries - e.g. to easily configure BPMN tasks to interface with other web services, process engines integrate HTTP client libraries.

The above components are shown also on Fig. 8, which represents an architectural view of a typical BPMS. The BPMS architecture on Fig. 8 is based on combining 2 literature sources: the *workflow reference model* [Hol95] by Workflow management coalition and the BPM book [Dum+18]. We found this combination to be a useful representation of a typical BPMS, since the literature review of BPMS architectures by Pourmirza et al. [Pou+17] highlighted a lack of a modern reference architectures for BPMS. Namely, as of 2017, 25 out of 41 architecture-related studies considered in their review did not base their architecture on any existing reference architecture. The most prominent reference architecture according to the study was the workflow reference model [Hol95], published in 1995.

An influential open-source BPMS has been Activiti, a Java-based workflow engine for BPMN. Activiti was first released in 2010, later, several projects based on forks of Activiti have also seen success: Camunda and Flowable. In addition to the Activiti and its forks, other open-source alternatives (jBPM, Bonitasoft) exist. The common model for these projects is to also have a company associated that provides a commercial version of the open-source core project with support for customers. There are of course other, fully closed-source commercial solutions.

Well-known proprietary BPMS examples are IBM Blueworks, Oracle Business Process Management Suite[6] and Genexus[7].

For the *Process Implementation* phase, which is the main focus of this thesis, the crucial parts are creating *executable* process models and the modules for executing them. This involves managing a repository of process definitions deployed to the system; managing the execution state of any running processes; managing an interface which allows resources (software agents, human resources using BPMS clients) to acquire tasks for execution and indicate that a task has been finished with certain results. We refer to the component which handles these functions as the *execution engine* of the BPMS.

2.4.4. Flowable Java API

Flowable is a relatively recent fork of Activiti, which includes modern features such as support for event-based systems (Kafka, RabbitMQ, ..) and enhanced asynchronous execution abilities. It can be embedded into an application as a singe library, revealing the Java API of Flowable, or it can also be deployed as a RESTful service, where other system components interact with it over the Flowable REST API. We give an overview of how Flowable's execution engine can be used in practice by software developers.

Listing 2.1 shows a basic example of how the Java API can be used to deploy process definitions, start process instances, and complete tasks within a running process instance. In the listing we see usage of a `RepositoryService`, which provides means to interact with the Process Model Repository described in section 2.4.3, the `RuntimeService` is used to start and interact with running process instances, while `TaskService` is used to query for tasks and their status plus updating their status (such as marking them completed), as such `TaskService` involves some functions of the WorkList handler from section 2.4.3.

The code example also shows how process variables can be attached to processes and their tasks (lines 13-18). These process variables are a key enabler of automating business logic within the workflow. While often some variables are set during the instantiation of the process instance, during execution of the process, variables may be updated or new variables may set as well.

```
1 // Main modules of the Engine:
2 RepositoryService repositoryService;
3 RuntimeService runtimeService;
4 TaskService taskService;
5
6 ...  // initialization of the Engine and Modules
7
8 repositoryService.createDeployment()
9   .addClasspathResource("process_a.bpmn20.xml")
```

[6] https://docs.oracle.com/cd/E14571_01/doc.1111/e15176/intro_bpm_suite.htm

[7] https://www.genexus.com

```
  .deploy();

// Start process instance with variables
Map<String, Object> variables = Map.of(
  "employee", "John McClane",
  "nrOfHolidays", "5",
  "description", "Description of John's Holiday");

runtimeService
  .startProcessInstanceByKey("holidayRequest", variables);

// Query for a task and it's attached variables
Task task = taskService.createTaskQuery()
    .taskCandidateGroup("managers")
    .singleResult();

Map<String, Object> processVariables = taskService
  .getVariables(task.getId());

int days = (Integer) processVariables.get("nrOfHolidays");

// Complete the task based on business logic
if (noOfHolidaysRequested < 7){
  variables = Map.of("approved", true);
  taskService.complete(task.getId(), variables);
} else {
  // ...
}
```

Listing 2.1: Example of Flowable Java API usage

Flowable's REST API is organized similarly to the Java API: the `/service/repository` endpoint is used to interact with the `RepositoryService`, while `/service/runtime/` corresponds to the `RuntimeService`, and so forth. Additional parameters such as setting of process variables takes place based on JSON-formatted payloads.

The more detailed description of the Flowable Java API can be found at documentation website[8].

2.5. Integration of IoT and BPM

As mentioned in the introductory chapter, the integration of IoT and BPM technologies can be seen from dual perspectives - based on the benefits this integration brings to either side. For IoT, BPM helps capture the agendas and routine activities of smart things with process-based modelling and enactment. This simplifies orchestration and efficient planning of resources [Jan+20]. This is especially important for Fog computing, where service placement is a core challenge. Further, BPM can be used as an intermediary layer between raw event data and higher-level (business) knowledge and events [Jan+20].

[8] `https://www.flowable.com/open-source/docs/bpmn/ch04-API/`

On the flip side, IoT devices and data enhance the digitization of the real-world context that a business process is concerned with. Thanks to this, detection of starting and ending of process activities in the physical world can be automated, something which would otherwise rely on manual human input for their digitization. The result is more timely and accurate data. IoT data can also be used in the decision-making of business process control-flows [Jan+20].

The core theme of integrating IoT technologies within BPM execution is the conflict between: 1) the prescribed stability of the process model and services implementing the BP execution in existing BPM solutions; and 2) the dynamic, volatile behaviour of IoT environments [LMM19; HKS09].

The survey of BPMS usage for IoT by Chang et al. [CSB16] also identified the following challenges in light of implementing IoT with the static conventional SOA-architecture based BPMS: the existing BPMS have no way of directly communicating or influencing the edge devices or their network, they are dependent on the abstraction and mediation middleware for this, which limits the possibility of defining fault-tolerance or dynamic behaviour at the process level that takes the edge topology into account. For instance, if two physically co-located devices need to interact with one another in IoT BP execution, in a SOA-based system this communication may need to go through multiple upper layers, despite the device's co-location, decreasing performance.

In the following subsections, we focus on how existing research addresses the following *BPMS for IoT* issues:

- How to model IoT devices and processes and bring the IoT data, context to the processes?
- How to support flexibility, dynamicity and adaptiveness in IoT process enactment?
- How to transform the existing centralised BPMS architecture to more distributed designs, including to the mobile/edge network?

However, we note that the full scope of described challenges in the literature is even wider [Jan+20; LMM19; HA19], including amongst others: cooperation and incentive models for *things* to participate in processes, privacy and ethics-related issues, and automatic detection of processes from IoT data (process mining) and adaptation [LMM19], and how to effectively but conveniently notify humans in processes where IoT and human resources coexist [HA19].

2.5.1. Modelling IoT processes

BPMN has been noted to be the most suitable business process modelling approach for IoT compared with WS-BPEL, UML or ePC [Mey+11]. The same is reflected in the systematic literature survey by Compagnucci et al. [Com+20] - only 4 out of 48 considered research works used non-BPMN-related notations. Despite this, the question of how to represent IoT devices and their events within BPMN models is not straight-forward. One issue is whether to represent IoT de-

vices and their services explicitly as process participants (resources) or process tasks, or to abstract them away from the models. The former means choosing which BPMN constructs to use and possibly extending the BPMN specification with new constructs accordingly, while the latter means that the modelling can be based more on conventional BPMN 2.0 approaches, but the BPMS or some middleware needs to provide the abstraction layer that transforms between the raw IoT devices / data and more abstract business events.

Various proposals to extend BPMN 2.0 for explicit representation of IoT in the process model are described in the IBM research report by Brouns et al. [Bro+18] and also above-mentioned systematic literature review [Com+20]. Brouns et al. identify the following categories of sub-aspects, how to model and represent (which primitives to use):

- Control flow and activities, tasks
- the *things* and devices of IoT - agents participating in the process
- events, how the (often real-time) IoT data can be brought to processes through BPMN 2.0 event primitives
- location - the geographic context and mobility of IoT participants
- temporal constraints and specification - which are important for real-time, low-latency IoT applications

As revealed by the systematic literature review of Compagnucci et al. [Com+20], one of the most significant efforts of extending BPMN for IoT has been presented by Meyer et al. The works stem from the EU project IoT-A and are concerned with how to represent *things* (real-world objects which IoT digitizes), the smart devices, their interrelationships and involvement with Task/Activity primitives of BPMN 2.0. They proposed to capture the IoT devices and individual native services offered by IoT devices as new kinds of process resources with their own metamodel. Their extension to the BPMN 2.0 standard uses *lanes* to indicate IoT devices and new Task extensions such as "Sensing Task" for native services of IoT devices. The metamodels of these extensions specify possibility of various parameters for the resources, which the BPM engine can then evaluate and adapt the execution based on, e.g. sensor accuracy for use in BPM business rules.

The *things* are represented as a separate *Pool*-like construct, whose custom implementation is based on the *Participant* construct of BPMN Things [MRH15]. An example IoT process based on this approach is shown on Fig. 9.

The *uBPMN* BPMN 2.0 extension also proposes how to model IoT devices and things [You+16]. While Meyer split the physical thing and smart device as separate artifacts, uBPMN merges them under the *Smart Object* element, based on the BPMN 2.0 *Data Object*. uBPMN provides a more varied set of Activity/Task-based extension: *Sensor Task*, *Reader Task*, *Audio Task*, *Image Task*, *Collector Task*. Similarly, *Sensor Events*, *Reader Events*, *Image Events*, *Audio Events* and *Collector Events* are included. They also present typical context-aware modelling patterns using their extension in [You+18]. Thus, uBPMN is concerned with the

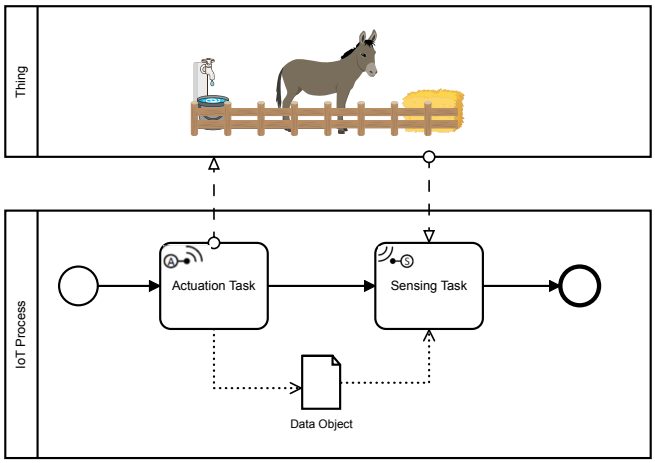


Figure 9: Example of IoT-annotated BPMN process model (based on Meyer et al. [MRH15])

different types of data from ubiquitous computing scenarios, and does not cover the relationship between things/devices.

Events. uBPMN and IOT-A project extensions focus more on representing Devices, Things, and Activities. If we consider BPMN Events, we see that standard BPMN 2.0 Events such as *Signal*, *Message* or *Timer* events are already useful for IoT, to represent communication between devices, temporal limits or scheduling and so forth. *Error*, *Cancel* and *Compensation* events can even be useful for describing fault-tolerance. For applications with mobility and location-awareness, however, introducing new location-related events can simplify expressing situations such as reaching a destination (e.g. arriving at a Fog server, a Point-of-Interest) [Bro+18; Koz10].

Data. Alongside Events, BPMN *Data Object*-s are general-purpose and thus already fit also a lot of IoT purposes without any extensions necessary. Existing works regarding Data representation in BPMN have covered amongst others data quality [MSM11] and security [SZ15]. IoT data are often revealed as a stream of data or stream of events. The above-mentioned BPMN 2.0 Events however represent singular events, not streams. Data streams are not well captured by standard BPMN 2.0, the system relies on some other component to map the event stream into (business) events. In this light, Appel et al. [App+14] proposed an extension for *Stream Processing Unit* Tasks.

It should be pointed out that standard BPMN 2.0 without extensions can also be successfully used for IoT, the drawback being that in this case, the modeller may need to provide more annotations to improve the readability of the models.

Besides BPMN, another BPM-related standard which has been gaining attention recently [Man+19] as a suitable way of modelling IoT is the Case Management Model and Notation (CMMN) standard. Compared to BPMN, CMMN has less rigid control flow structure constraints - they consist of a set of "mini"-

processes or process fragments, each of which can be worked on independently. As such it supports unpredictable, knowledge-intensive processes with less clear structure.

2.5.2. Execution of IoT Business Processes

The above-mentioned extensions define the machine-readable XML Schema Definitions (XSD), they are still not sufficient to automate the process execution fully. The BPMS can automate the control and data-flow, but invocation of devices usually relies also on service discovery, deciding which service to bind the activity to and negotiating the interface on which to communicate.

The involvement of IoT devices in BP execution (corresponding to Implementation phase of the BPM lifecycle) can be realized in 3 main approaches, as explained by Chang et al. [CSB16]:

- "Centralized - Device as BP participant" - The devices are invoked directly by the centrally hosted BPMS, in a *Thing-to-Cloud* manner. This is based on either the device directly hosting some invokable (web) services, opening a socket for direct communication; or through some Adapter component, which the BPMS relies on to interact with IoT devices. This is the most common approach in previously described modelling works and SOA BPM. If the IoT system exposes all of its devices' functionality as web services, existing BPMS generally already provide the necessary HTTPS client implementations and parsers to interpret the responses of devices.
- "De-centralized - Device as executor of compiled BPM code fragments" - To reduce the centralization of the previous option, some of the BP logic is moved to the IoT edge devices, to increase autonomy. More specifically, in this case fragments of the process model are centrally transformed into executable code, in a language supported by the edge device. These executable fragments are deployed on the IoT devices, allowing them to interact with the central BP. This allows even resource-constrained devices to participate in centrally modelled abstract processes, but the approach relies on the central system having a component for translating the BP model to executable code. Examples of this approach are [Tra+12; MCS16b; MD17].
- "De-centralized - Device as BPMS host". In case an IoT device's hardware and software constraints allow it, a full standards-compliant BP engine could be hosted on the device allowing it to execute and manage entire BPMN processes itself. Thus, the highest level of flexibility is provided - the device may orchestrate a process by itself locally, or participate in process choreography with other central or local processes. Previous works demonstrating this are for instance [RML12; Sch+16]. This thesis's interest lies in the case where the IoT device hosting the BPMS is a moving device, i.e. a **Mobile Process Host (MPH)**.

The mentioned approaches help capture some of the IoT behaviour at the mod-

elling level and how to organize the execution in terms of (de-)centralization. However, to manually describe and handle all the potential contingencies of IoT processes is laborious and bloats up the process control flow with technical details. IoT process execution should dynamically be able to adapt to the current process state and the context of the IoT environment - availability, capabilities & status of devices. Optimal decisions here minimize likelihood of failures and inefficiencies.

Regardless, even with such context-aware planning, the BPMS should still provide fault-tolerant mechanisms to overcome failures if they do occur. In following subsections we outline different approaches in the existing research towards fault-management and adaptive, context-aware BPM execution for IoT.

2.5.3. Dynamic service binding

Instead of relying on fixed bindings of activities (e.g. service tasks) to services that are defined during design-time, as is the case with conventional enterprise processes, IoT BPMS often feature dynamic discovery of suitable service(s) during run-time. A prerequisite to realize dynamic **Service Discovery** is a semantic model/ontology to describe the IoT devices and their capabilities, services. These descriptions would be managed by a discovery server, which allows clients (the BPMS) to query for known devices and their service descriptions. Secondly, the task definition of a dynamically bound service task in the process model needs to define criteria which the service descriptions must match in order to be chosen during the dynamic binding (e.g. sensor type, data accuracy, location, supported protocols, etc.). Based on task criteria, the BPMS can query the discovery server and attempt to find a matching service to be bound and invoked. Depending on the degree of (de-)centralization in the system, discovery could be based on a global central repository system [Jar+14; Dar+15]; a federated discovery network based on an edge network [TVM18]; or direct device-to-device based local discovery (Bluetooth, WiFi-Direct, LTE-Advanced D2D, NFC).

The devices and services could be described with existing Web-Service standards (WSDL) or Semantic Web (RDF, JSON-LD, OWL) technologies or IoT-specific description languages (as mentioned in section 2.2.1). For instance, in [Hub+16] IoT devices are described using the DogOnt domain-specific ontology, which is based on OWL, and dynamically bound process steps describe criteria for matching services using SPARQL queries based on contextual data such as location.

The problem of deciding which discovered service is the optimal match for the given process or task - i.e. optimal resource allocation - is recognized as an important research challenge for the intersection of BPM and IoT [HA19; Jan+20]. If we leave this problem unsolved/delegate it to the domain expert process designer, we see another use of dynamic service discovery - using it as an assistance tool for process designers

In [Dar+15], the process designer defines service requirements for the process task and the modelling tool then lists suitable discovered services. The final binding in this case is still manual, but the assisted service discovery can be helpful to quickly identify appropriate services in systems with a huge number of devices. When adding the IoT service to the process, the service description is also retained in the process definition, which allows later (run-time) fault-management or replacement of the service.

Goal-based refinement of abstract tasks. In the above described dynamic service binding approach, the process task is defined by the criteria which will be dynamically matched with a single service. A possible expansion of this is for the task to describe the expected outcomes/changes to the process state that a service or a composition of services should produce after invoking. In other words, the dynamic matching is no longer only about finding atomic suitable services, but also considers compositions of services that would produce the desired effect on the process state. Some works include pre-defined compositions (process fragments) [Kép+16] while others are able even to perform the composition completely dynamically [Buc+12].

In order to realize this, the semantic model needs to involve more details - the necessary pre-conditions (e.g. input data and process state, context) and effects of invoking a given atomic service or process fragment. Bucchiarone et al. present a framework as part of the ASTRO project [Buc+12], where actors (businesses, devices) involved in a process each offer a selection of discoverable process fragments they support. Processes and process-fragments are modelled using *Adaptable Pervasive Flows* based on Blite, a process calculus for WS-BPEL. In their approach, abstract tasks are annotated with an expected goal in terms of the expected state of context variables. The system tries to realize the goal during runtime using automated planning. Based on the goal and system state, a planning domain and problem are formulated, and using automated planning techniques, a suitable composition of process fragments is generated.

In [Kép+16], process models are analyzed to identify process sub-parts for which there exist equivalent fragments in a repository of process fragments, based on structural and semantic matching of the process activities. The original process is transformed so that the regions for which matching alternative fragments were identified are replaced with placeholder activities which invoke a situation-aware middleware. The latter dynamically chooses a suitable process fragment that best matches the current context during runtime.

2.5.4. Control-flow based Dynamic Fault handling

The description of expected effects of executing a task is also useful for identifying when the expected state and actual state mismatch, indicating a fault has occurred. In such a case, a compensation or repair mechanism needs to align the real state with the expected state before the rest of the process execution can

continue. This re-aligning typically means that the process structure is modified during runtime. In [Buc+12], the same automated planning-based approach described above in Sec. 2.5.3 is also applied to deal with such exceptions. Automated planning is also used by Marella et al. in the SmartPM [MM17] system for automatically adapting the process and creating a dynamic substitute service composition in-case of failures. Based on a domain theory, the tasks in the process model and a task repository are described in terms of their pre-conditions and effects. IndiGolog and LPG-td are used to realize the automated-planning based adaptation.

Seiger et al. [Sei+19] use the MAPE-K feedback loop principle to detect faults and apply dynamic adaptation to them. IoT-related process steps are annotated with Goals and Objectives based on the authors proposed meta-model. The context of the environment and IoT-related process state is modelled based on DogOnt ontology. Every time a potentially adaptive process task is executed, a *Feedback Service* is invoked in parallel with the execution, which given the goal and context information, will return if the original task has fulfilled its goal criteria, or apply some extra self-healing mechanisms to bring the state to match the goal. The compensations are based on graph queries to find suitable services in the IoT environment to fulfil the goals.

The PROtEUS execution system by Seiger et al. [SHS18] combines the mentioned MAPE-K feedback loop with dynamic service invocation. PROtEUS is a model-based workflow execution engine which can execute process models defined in a YAWL-compatible meta-model variant. The system supports complex event processing, dynamic service invocation and is demonstrated through 2 scenarios: health monitoring processes and smart home automation. It features self-healing functionality via feedback loops, where the system adapts to inconsistent states between the process and actual physical world. Similar to BPMN, PROtEUS supports human tasks and invoking services through web standards. PROtEUS uses the SPARQL-based Semantic Access Layer [Hub+16] to dynamically invoke services based on context and requirements.

While approaches like [Buc+12; SHS18] rely on a custom PAIS for the adaptive behaviour, in [SHA19] Seiger et al. discuss how existing PAIS can also be retrofitted with similar dynamic, runtime goal-based process structure changes.

In this thesis, we do not employ a dynamic healing/adaptation based on device capabilities or process-task goals, expected effects of a task at the PAIS level. Instead, our adaptive behaviour is based on the implementation logic of individual tasks and the process design, without runtime process structure changes. We have included this discussion to showcase another research direction that is definitely also useful for *mobile* IoT use-cases, where the process may need to adapt to an unexpected change in the route or simply due to connectivity issues related to movement. Considering mobility, one distinction is whether the adaptation/self-healing process can influence the movement of agents or not. In case of autonomous vehicles or robots (such as in [MM17]), movement can be

part of the generated adaptation. But in other scenarios, such as a person riding a bus, the movement pattern usually cannot be influenced, and the adaptations to the process control flow can only use the fixed movement (trajectory, speed) into as extra context when planning its adaptation.

2.5.5. Context-aware process driving

Another kind of adaptive behaviour in IoT-oriented BPM is the dynamic driving and constraining of the execution engine or worklist handler components of the BPMS. For the execution engine, the state of tasks and whether or not they can be executed may be bound to context-based information, such as location. For instance, [Zhu+16] use a coloured Petri Net-based approach that allows defining the process control-flow with geospatial information-aware constraints and rules. As a result, certain tasks may only be finished if the associated resources are currently in a certain location, for example.

Context information can also be used by a BPMS Worklist handler to prioritize and filter the list of available tasks for resources involved in processes. Smartphones provide several opportunities to capture the user's context for human process resources. Giner et al. [Gin+10] use tagging technology such as QR-codes or RFID to present dynamic task lists to the process participants. Stach et al. [Sta+18] use various mobile context information, such as battery levels, connectivity and Bluetooth beacon-based location proximity to automatically prioritize work items for resources.

The aim of these systems is to stream-line interactions with the PAIS, minimizing the steps a user must perform to find the work items most relevant to them and avoiding tasks which involve some waste (e.g. changing location to complete them). These systems also include notification aspects, to avoid missing tasks which are relevant to the user and easy to complete in their current context.

2.5.6. Adaptive Process Execution Platforms

Adaptive behaviour can also be introduced on the level of the execution host and software, infrastructure providing the execution environment. At this level, techniques such as replication or distribution of PAIS or their components, dynamic migration/delegation of process instances between hosts can be found.

A logical fragmentation of the process was presented by Zaplata et al. [Zap+10a] - i.e. migrating the process instance between different execution hosts, where each host takes care of executing particular logical fragments. This increases flexibility as the process can be migrated between hosts during runtime based on different criteria such as privacy-constraints but also resource-efficiency, predictions about stability of hosts, etc. Two approaches for the necessary information for run-time migration are discussed by the authors: 1) altering the original process model to explicitly include the migration-related data and tasks (e.g. state of the process instance, applicable target process engines) and sending this data in a message or 2)

store this information in some additional annotation or document object, keeping the original process unaltered, but requiring support for interpreting them from the engine. The former has the benefit of being applicable even with existing BP modelling standards, but requires extra effort during design-time and alters the process, whereas the latter relies on novel components and middleware in the engine, but allows the migration functions to be handled more dynamically during run-time, as they are not fixed in the process control flow.

Another version of migration is used in [Dar+15]. Here, the goal is to execute the process always near the user and as the user moves around, the process is migrated to another device currently near the user (e.g. from a home set-top-box to the user's smartphone as they leave the home). However, due to heterogeneity of the different platforms, for each device, a different variant of the process has been modelled, to better support the particularities of that device. While it increases customisability for particular device, this requires extra effort during design-time, plus it requires a mechanism of transforming the state of the process between different process variants. In this case, due to having different (device-specific) variants representing the same process, we call this approach *process delegation*.

Seiger et al. [SHA17] have extended the previously mentioned PROtEUS engine to also distribute fragments of a main process from a *leader* process engine to peer engines, managed by the leader. This increases resiliency of the system - the fragment executing peer can enact autonomously and be less dependent on network conditions, etc.

Besides migration and delegation, another way to increase fault-tolerance is to replicate the execution platform (or parts of it) on many hosts. This increases resilience to host failure, as clients can immediately continue execution based on the other replicas of the engine. This approach is followed by Zhang et al. [Zha+18] where a multi-engine replicated workflow architecture for mobile ad-hoc networks is presented. In their design, a *master/replicas* approach is taken, where clients interact with the master instance of the engine, while simultaneously the process instance states are synchronized between the master and all replicas. Should the master become unavailable, a new master is elected and clients are transferred to interact with the new master.

While [Zha+18] follow a simple trigger and heart-beat based approach to synchronization and fault detection, a more complex replication scheme is presented by [SRT18], where compensation and rollbacks are also taken into account in case of replicated engines.

As the synchronization of the entire BPMS and all its process states is challenging, another possibility is to replicate sub-parts of the engine and process instances. For instance, only the worklist handler and process tasks relevant to a particular resource are replicated in [Sta+18]. The worklist handler component and currently active work items are replicated on the smartphone-based client, which can perform decisions-making related to the task even if connectivity to the master process engine is lost. Regardless, there is still a periodic synchronization

of state.

2.5.7. Discussion

It is common practice to capture the adaptive behaviour in the form of an abstract, proxy task, which when executed, then realizes the adaptive behaviour as necessary. The run-time adaptive enactment relies on extra knowledge - about goals, constraints for the tasks as well as the context of the environment. This knowledge can be captured with standard process model annotations or by customised workflow execution engines that have context-aware modules or additional goal-modelling capabilities. Striking a balance between these approaches is challenging - annotations encumber the process modeller and bloat the process definition; whereas a complex, custom PAIS is highly context-aware, but limits influence of modellers as they are rather opaque incompatible with existing standards.

From the perspective of handling faults, the discussed mechanisms can be divided into 2: *pro-active* and *reactive* approaches. Pro-active approaches attempt to minimize likelihood of runtime failure by taking into account current context of the system and requirements of the process model when:

- binding the process activities to resources and services
- managing, constraining the execution state of the process
- deciding which work items are currently available for a resource

Reactive approaches attempt to identify mismatches in the expected and actual states of the system and environment, and attempt to re-align the mismatching states through approaches such as automated planning.

Challenges and gaps. From the modelling perspective, the SLR of Compagnucci et al. identified some IoT challenges which have gained less attention by the existing works, they are: Scalability, Availability, **Mobility**, Fault tolerance and Uncertainty of Information.

2.6. Adaptive Edge Process Management

Compared to the integration of IoT and BPM, relatively fewer research addresses usage of Edge / Fog Computing and BPM in unison. This section provides the overview of that perspective in the literature.

Considering the goals of this thesis, we are mainly interested in aspects where devices and hosts in the system participate as infrastructure / platform hosters/providers. That is, if an edge device is invoked from a central service as part of a process, we see this as IoT device usage in processes, not Edge Computing. But if the edge device can take care of executing (sub-)processes by hosting a BPMS or sub-modules of it, we see it as Edge Process Management.

The following subsections describe related works from the following themes: embedding process engines into edge devices and other de-centralised edge process execution schemes; how mobility is taken into account in a) fog/edge of-

floading and b) workflow management; and what tools exist that could be used to simulate scenarios of this sort.

2.6.1. Edge-Hosted Execution Engine

In this subsection, we focus on PAIS or process execution engines which are on devices in or near the edge network, focusing mainly on smartphone-type devices and Fog servers as the engine host. The problem here is that a typical edge device is either not using hardware platforms like x86 or ARM (on which respective software is usually already supported) or the OS is a lightweight or modified version of the desktop/server counterpart, which disallows running standard commercial off-the-shelf process engines, as libraries may be missing, and available memory, storage, etc. capabilities may not be sufficient.

As a result, either a custom light-weight process engine tailored to the platform needs to be developed or existing BPMS need to be modified to fit the available software stack. For example on platforms such as smartphones (iOS, Android), which run custom OS kernels, a custom engine must be built or the COTS engines must be adapted to fit the available software stack.

Since the emergence of devices such as PDAs or smartphones, the possibility of automated execution of processes on these devices has been studied, allowing direct involvement of business activities and information at the mobile device. In earlier custom, mobile hosted BPMS, such as *Sliver* [Hac+06], the dominant standard for defining workflows was WS-BPEL. Sliver was built based on Mobile Information Device Profile (MIDP) Java specification for embedded devices from early 2000s.

More recently, as sensor-enriched mobile devices have become commonplace, mobile process engines have begun integrating sensor support at the process-level. For example, Schoebel et al. built a custom [Sch+16] Android-based mobile process engine developed for the QuestionSys questionnaire system project, which can execute models defined in the non-standard ADEPT2 specification and considers integration of additional sensors in its architecture. An iPad prototype of a custom engine for XML-based non-standard processes is presented in MEDo [Pry+15], an engine for supporting medical ward round processes.

Custom-built engines for the BPMN standard on Android, such as [Pen+14] tend to have limited functionality and are in prototype stage, as implementing a full-featured engine is a resource-intensive effort. The alternative is to adapt an existing engine, as done by Dar et al. in [Dar+15], where the open-source Activiti engine has been adapted to utilize those software packages available on the Android platform: e.g. the XML parsers used in the original source code needed to be replaced with Android-specific XML libraries.

Compared to mobiles, hosting the process engine on a Fog node is generally simpler, as Fog architecture prescribes some kind of virtualized platform, in which COTS can be deployed. For this purpose container-based virtualization (Docker,

Kubernetes, LXC containers), or Virtual-Machine based virtualization are both suitable and directly applicable [Wai+21].

In case of Android, up until recently, normal virtualization has not been an option. However, virtualisation support has been very recently introduced Android version 13 with the addition of *protected kernel-based virtual machine* (pKVM) [9], plus another possibility is to run COTS software within a terminal emulator, such as *Termux*[10]. Termux provides a Linux environment and its own package manager, where a large set of software has been patched and cross-compiled for Android, including programming language interpreters and compilers. If the PAIS is based on software and dependencies available in the Termux package repository, e.g. Java in case of Activiti/Camunda, it can be installed and run within Termux or one may cross-compile the necessary software.

The above approaches enable execution of processes on the device, which allows performing them offline, irrelevant of connectivity issues. This is fine for processes where the tasks only involve activities local to the device/user. More complex processes may involve several devices, external sensors or a choreography of processes hosted by different devices. Here, execution frameworks with collaboration and choreography features have arisen, where the process engine is distributed among multiple Edge hosts, in a mobile ad-hoc network (MANET). A MANET-based PAIS was used in the WORKPAD EU project [Cat+11], which also combined dynamic fault handling adaptation techniques similar to those mentioned in 2.5.4. In WORKPAD, one Edge device acts as the orchestrating leader engine, and assigns tasks to peers who host specific sub-services.

An extension of Sliver is presented in [SRG08], where the process is decomposed into sub-processes and distributed among hosts. Another example is [Zha+18], where MANET devices may host Wf engines in order to enable Wf-based technologies in situations with missing communication infrastructure (e.g. natural disasters). Their architecture features a multi-engine scheme, where in case of failing nodes in the MANET, another node quickly replaces the failed node acting as an engine. As discussed in Section 2.5.6, these MANET-based distributed engines can increase fault-tolerance, but also increases the complexity of managing the state, synchronization of the engine and processes.

Discussion. In general, standard-based process engines are not readily available for popular platforms such as Android, due to the difference in library support. Custom-built mobile engines tend to be based on proprietary/custom process modelling languages, which decreases interoperability. On the other hand, the custom modelling languages often include adaptive features support not available in the standards such as BPMN. The edge hosted engines may feature various adaptive functions, such as process instance migration or delegation [SHA17] and others discussed in section 2.5.6. These execution engines do not feature compu-

[9] `https://source.android.com/docs/core/virtualization/architecture`

[10] `https://termux.com/`

tational offloading of individual process tasks to Fog nodes.

2.6.2. Task Scheduling in Edge and Fog

Considering the *bottom-up* Fog applications (refer to section 2.3.6), an important use-case for mobile process engines is the offloading/delegation of tasks from the mobile to the fog server(s), such as the video processing task offloading presented in the introductory chapter. In an edge environment, offloading may be desirable to achieve better latency or to save energy, but if possible, the system should also be capable of falling back to local computation, if there is no offloading target available.

First, let us consider factors other than mobility which affect scheduling of task offloading from mobiles to proximal Fog or Edge nodes. In [Liu+16], the authors create a delay minimizing task scheduling policy based on Markov decision processes that takes into account power constraints, communication channel status, task queues. Similarly, Zeng et. al [Zen+16] consider the task scheduling between embedded devices and computation servers in the Fog context and use non-linear programming-based algorithm to minimize the task execution time. Like [Liu+16], the algorithm also involves deciding whether to execute locally or to offload. These works focus on information about workloads, task queues and energy usage for the decision-making.

2.6.3. Mobility-Aware Offloading

To the best of our knowledge, few workflow-based works address mobility-related details, in the following we give an overview of how mobility is addressed in other edge- and fog computing research. Fog computing works addressing the placement question of task offloading (*which fog server to offload to?*)[ATH16; CTC17] usually consider the geographic locations of the servers and system resource utilization. More precisely, Alam et al. [ATH16] manage mobility in fog computing offloading in terms of migrating the already offloaded tasks (basic blocks) within the fog network and/or neighbouring networks to support mobility of the user. That is, the task migration minimizes the distance and latency between the fog server and moving user.

The placement question addressed also in the framework proposed by [CTC17]. Their Software-Defined Network-backed fog platform takes into account the geographic locations of the mobile and fog server, but does not consider the trajectory of the mobile.

Lee et al. [LS13] create a Markov model from user smartphone mobility patterns based on records of encountered networks. Based on such trained mobility models, the future network condition is predicted, which serves as the basis for the fog task distribution decisions.

Mobility-awareness for allocating computing resources in fog and cloud based on number of active users is addressed in [Bit+17]. Here, the mobility of sev-

eral users causes congestion for certain fog servers, and thus policy-based load-balancing between the fog and cloud is used to alleviate it.

The ENDA architecture [Li+13] collects smartphone GPS traces of user movement routes, based on these traces, current user movement is predicted. This prediction is used for WiFi access point (AP) selection during offloading. The AP selection mechanism also considers the AP locations, bandwidth and latency capabilities.

Alongside the above-described offloading cases, mobility-awareness has been studied also in other similar applications, such as data-fetching, or vehicular communications, which we describe next.

Using predictions about user mobility trajectory and processing tasks, authors in [Hon+13] pre-emptively fetch and process necessary data to reduce latency for spatio-temporal Event Processing in a mobile situation awareness system.

A scheduling scheme for wireless communication in Vehicle-to-Vehicle networks is proposed in [CWB17], where the markov decision process is based on a stochastic game theory approach.

The authors in [Ott+14] proposed an adaptive mobile Complex Event Processing-based vehicular warning system where based on user location, the event processing, which is modelled using operator graphs, is dynamically updated to reduce latency and increase energy efficiency.

Another example is *marion* [Sch+13], where the agricultural harvesting process which involves cooperating machines is dynamically optimized. The system solves the planning problem of involved machine routes given the set of spatio-temporal constraints using robotics algorithms.

Mobility awareness involves performing geo-location fix acquisition operations. However, continuously tracking the location involves additional energy cost. In [Pha14], the system tries to perform location acquisition tasks sparsely, at the appropriate moment to conserve energy. The acquisitions are scheduled based on user activity recognition, e.g. when the user's transportation mode changes (e.g. from *driving car* to *on foot*).

Discussion. A common theme in the above works is reliance on models trained on historic user movement datasets, which may be difficult to obtain. Instead, some knowledge could already be predicted based on the path-finding algorithms and map data (e.g. as in offline car navigation systems). Secondly, none of the described works have integrated the scheduling models with workflow or process management systems. In contrast, the proposed contribution in this thesis has integrated with WfMS and it utilises run-time context factors without additional historical data.

2.6.4. Process-Oriented Fog Computing

While fog computing is not directly dependent on process-based management, we highlight some of the fundamental research challenges of fog computing [MSW18]

that could effectively be implemented with process-aware information systems:

- how to distribute computational tasks/applications within the network
- how to offload tasks (e.g. which tasks to offload, to where), while assuring SLA (Service-level Agreement), QoS (Quality of Service) and QoE (Quality of Experience) [Mah+19]) such as latency (Rodrigues et al., 2017), resource utilization [Du+18], task priority [AMS19], energy efficiency [Yan+18], profit-awareness [Mah+20], etc.
- how to manage the deployment of tasks and execution in heterogeneous infrastructure
- how to migrate computation within Fog nodes

From the processes perspective, this means distributing execution of some of the (sub)processes to Fog nodes situated in proximity to the end clients, by hosting the process engine in the Fog infrastructure, this is feasible using light-weight BPMS such as Flowable, that can be deployed within a second [AG19]. Since decision-making and exception handling are integral to business processes, the algorithms and models for these issues (such as SLA and QoS control of Fog applications) can be integrated into process engines and be directly invoked at the process abstraction level. A lot of fog applications, which deal largely with messaging and deployment tasks can comfortably be modelled as processes. Since the definition of processes is standardized, deployment across devices is simplified. More recent works have begun exploring this process-based fog computing direction as well. Cheng et al. [Che+18b] describe an IoT execution environment which hosts process engines on the smartphone, while Fog nodes manage service discovery, deployment and event management.

To the best of our knowledge, few workflow-based works address mobility-related details, in the previous subsection we gave an overview how mobility is taken into account in other, non-workflow-oriented edge- and fog computing research. Let us now look at works which use non-standard workflow methods in the context of Fog/Edge.

Numerous existing works in mobile-cloud computing, robotics, IoT and related domains model the application workflow as a directed acyclic graph (DAG) [STB17; Che+18a; Xu+19; Rah+16] and capture workflow characteristics (e.g. input/output requirements of individual Wf tasks), network and hardware configuration of the system (e.g. the mobile nodes, cloudlets, cloud VMs) in order to optimize the offloading of workflow tasks to other nodes in terms of energy-efficiency or to meet deadline constraints, for example.

While mobile phones are often considered as participants in the workflow, they do not take the mobile's movement into account. As an example, the multi-objective workflow-based computation offloading framework proposed by Xu et al. [Xu+19] uses genetic algorithms to simultaneously consider energy usage and deadline constraints when offloading to cloudlets, however the user mobility and related network changes were left as part of future work.

The robotics framework by Rahman et al. [Rah+16] includes mobility when modelling the cost of offloading workflow tasks. Here, mobility influences two aspects: (1) movement energy consumed by the robot and (2) the bandwidth of the network in the robot's current location. However, their simulation considered 3 location zones, each one with a distinct, fixed bandwidth value, the proximity to the signal source and its influence on link-speed is not considered.

A more detailed consideration of mobility is in [Qi+16], where in addition to the geographic location of the fixed compute resources (cloudlets), the availability of mobile resources is determined based on a headlight model, capturing a zone along the direction of movement, similar to the headlight of a vehicle. The model also takes into account the mean encounter time between a mobile and a cloudlet network, based on statistics about user historic movement data.

Alternatively, analysis of workflow-based systems may also be done using Petri Nets. Ni et al. [Ni+17] model tasks in fog computing using Priced Timed Petri Nets and give an algorithm to predict the time consumption of executing these tasks. Based on predicted execution times and other factors such as fog node credibility rating, they devised a dynamic resource allocation scheme that maximizes resource utilization and ensures QoS for users.

Discussion. Above-mentioned works do not consider the implementation aspect of a workflow-based system, they focus rather on the theoretical model of the decision-making. The approach of how the devices handle execution and offloading of workflow tasks in terms of standards and software stack are omitted.

2.6.5. Simulation of Edge Process Management

Considering that IoT and edge systems are to involve huge numbers of devices, the evaluation of most existing related works is done at a scale of up to a few devices (e.g. 2 robots and 1 stationary system in [SHA17]). Indeed, large-scale real-world experiments are costly to perform, so the common approach is to simulate the behaviour [Ded+18].

Simulating mobility behaviour is usually based on discrete-event simulation, where the state changes of the modelled world are caused by events occurring at discrete points in time. As opposed to continuous simulation, a discrete event simulator (DES) jumps from one event to the next, the system state between these jumps is not captured. In this subsection we outline the better-known networking and mobility simulators relevant to edge-fog computing.

Ns-3. Ns-3[11] is a general-purpose network DES. It includes simulation models that closely reflect the Linux networking stack and as a result can be used as a real-time network emulator, enabling plugging it into real-world networks and integrating with real-world protocol implementations.

Ns-3 primarily targets IP networks at its core, but also includes simulation models for wireless technology such as Wi-Fi, LTE, WiMax. The ns-3 project has

[11] `https://www.nsnam.org/`

a large community and consortium behind it, contributing development of various modules, including an energy framework and mobility modelling.

The mobility module currently supports random-walk and random/non-random waypoint mobility models. While ns-3 does not support map-based movement models or schedule-based user movement out of the box, it can be integrated to mobility simulators such as SuMO, importing map-based mobility traces generated by these tools.

The energy framework of ns-3 takes into account several ISO/OSI model PHY layer properties in the case of WiFi.

As such, ns-3 offers multiple options for modelling physical layer properties of wireless links, includes a very detailed modelling of the ISO/OSI layers 2-4, and provides base classes for defining applications (layers 5-7).

OMNeT++. OMNeT++ [Var10] DES provides a core set of network modelling and simulation blocks which have been used in several specific extension frameworks for IP networks, vehicular networks, wireless networks, but also from domains such as sickness dissemination.

- **INET** is the OMNeT++ framework for IP-based networks and protocols at the layers OSI 2-4 in wired and wireless networks. It includes a rich set of implemented models for wireless technology such as IEEE 802.11 or 802.15.4. In addition to basic set of deterministic mobility models[12], INET provides trace-based and stochastic mobility models. Further, it has energy models[13] - both power generators and consumers.
- **VEINs** is an OMNeT++ based framework for inter-vehicular communication simulation, using SUMO simulator for road mobility modelling. It can import OpenStreetMap scenarios with buildings, speed limits and traffic light restrictions.

ONE Simulator. The Opportunistic Network Environment simulator (ONE) [KOK09] was developed for modelling behaviour of opportunistic networks and routing protocols there-in. The networking is captured at the link layer, so the PHY layer modelling is not very complex, as ONE focuses on store-carry-forward networking.

The radio links have communication range and bit-rate properties and do not include aspects such as signal attenuation. However, some of the provided models reflect interference and signal strength by updating the bit-rate based on the distances of hosts or by the number of other hosts nearby.

The set of movement models includes both random-based movement as well as map-based movement models, and for instance user behaviour modelling (e.g. workday model) or group-based behaviour, such as a group of pedestrians using public transport.

[12] `https://inet.omnetpp.org/docs/users-guide/ch-mobility.html`

[13] `https://inet.omnetpp.org/docs/users-guide/ch-power.html`

SUMO. Simulation of Urban Mobility (SUMO) [Beh+11] is a renowned simulation framework for road vehicles, pedestrians and their routing, accounting for aspects such as traffic lanes, intersections, traffic lights and multi-modal traffic. While SUMO does not provide any networking features, the rich mobility modelling of SUMO has been recognized by the above-mentioned networking simulators (OMNeT++, Ns-3), as they provide means to integrate SUMO-generated traces into them. SUMO supports importing maps from OpenStreetMap, VISUM, VISSIM and NavTeq.

iFogSim, EdgeCloudSim. Unlike the above-mentioned tools, iFogSim [Gup+17] specifically targets IoT, edge and fog computing environments. The project was developed for evaluating resource management policies with respect to latency, energy consumption, bandwidth usage and operational costs. iFogSim extends the CloudSim project, as such it inherits its features such as executing and migrating applications / VMs, and managing virtualization, resource management and scheduling functions for cloud datacenter.

However, iFogSim does not capture lower level networking properties such as interference. The infrastructure is defined as a hierarchy, where messages can be passed among child-parent pairs, in bottom-up direction (e.g. edge to fog). It does not support device-to-device communication, except for the cloud layer, where data centers can exchange messages.

iFogSim provides a set of entity classes for the main node types, such as FogDevice, Sensor and Actuator, and classes to model the Fog applications, which are a set of application modules that form a directed acyclic graph. FogDevice classes can be described by their memory, processor, storage size, uplink, and downlink bandwidths properties, while sensors, actuators have a latency property.

While iFogSim does not provide any mobility modelling features, another tool based on CloudSim, called EdgeCloudSim [SOE17], which specifically targets the edge computing environment, does include a nomadic mobility model. Further, it provides a networking module to reflect edge WLAN/WAN communications and an Edge Orchestrator module which performs decision-making for incoming requests, tasks.

Discussion. While tools such as ns-3 provide the most detailed networking simulation, the learning curve and set-up can be considerable. In novel areas such as Edge Process Management, quickly evaluating a selection of approaches at a higher abstraction level can be more important than the specific, realistically defined (and usually smaller scale) scenarios [CG03]. In fact, more detailed simulations can sometimes obscure effects of fundamental changes in the algorithms, since they involve managing a large amount of simulation parameters in addition to the algorithms themselves [CG03].

Meanwhile, the existing fog-oriented tools like iFogSim provide powerful datacenter/virtualization modelling features derived from CloudSim, however, the definition of application logic tends to be limited to certain types of applications. Table 1 summarizes this comparison of the related simulation tools.

Simulator	Wireless Networking	Mobility	Edge & Fog Computing	Learning Complexity	Languages
ns-3	Detailed	random, way-point based, map-based with SuMO traces	None	Complex	C++, Python
iFogSim	Basic	None	Resource Management policies	Medium	Java
EdgeCloudSim	Medium	Nomadic models	Edge Orchestration	Medium	Java
SUMO	None	Detailed map-based & random models, incl. lanes, traffic	None	Medium	C++
ONE	Medium	Map-based & random, OSM support	None	Medium	Java
OMNET++	Detailed (INET)	Trace-based, stochastic, OSM support with VEINs + SUMO	None	Complex	C++

Table 1: Comparison of networking and mobility-oriented simulation tools. Acronyms: OSM - OpenStreetMap

2.7. Summary

This chapter established the concepts of IoT and BPM. We explained the different kinds of typical communication styles of IoT (device-to-device, device-to-gateway, device-to-cloud), how SOA can alleviate the interoperability issues of IoT and covered new computing paradigms for IoT such as Edge- and Fog Computing, contrasting them against Cloud Computing. We showed how process execution is typically realized in technical terms, including specific examples from the Flowable BPMS. We covered the state of the art of integrating IoT and BPM: how processes are modelled, executed and made fault-tolerant and adaptive in the existing literature. We discussed the existing works specific to edge networks, de-centralised execution of processes, and processes involving computational offloading and mobility-awareness. Finally, we compared various simulation tools that may be helpful to study these systems.

3. PROCESS EXECUTION & MIGRATION IN THE EDGE

This chapter describes our effort to answer the 1st research question: *How to realize execution of processes on edge network devices such as smartphones, with the possibility of device-to-device process migration during runtime?*

In the 2nd Chapter (Sec. 2.6.1), we discussed how process execution software has previously been used on different Edge devices. However, our study of delegation and migration of processes between PAIS (Sec. 2.5.6) revealed that existing works proposing migration solutions either involve designing different, device-specific versions of the process that is being migrated [Dar+15] or did not directly consider migration between Edge devices specifically [Zap+10a]. Our research question is targeted at migrating processes in the Edge without altering them and directly considers the smartphone as a representative moving Edge device with limitations such as software portability (due to its CPU architecture).

Several reasons motivate D2D process migration - load-balancing of process execution between multiple hosts; energy management (e.g. a low-battery device may offload its processes to one with abundant energy) or other constraints such as business goals (e.g. geographic location/proximity) that dictate which host should currently execute a process. In this thesis, we focus on the latter, our study is framed from the perspective of a hypothetical application scenario in the domain of logistics and transportation, where process-based IoT is used to realize fine-grained monitoring of goods. Equipping workers with smart devices and parcels with sensors, the IoT system can recognize events occurring to individual parcels during transportation, notifying nearby workers to take action, while simultaneously collecting a unique, detailed history of all events that the parcel goes through during its journey within the logistics chain.

Generally, asset tracking is a typical IoT scenario which can be modelled and implemented using BPMN technologies, which is the dominant approach to modelling IoT business processes (as discussed in section 2.5.1). However, the goods monitoring should be continuous and take place even when Internet connectivity is not available (e.g. driving through a tunnel). Executing the process on an edge BPMS solves this issue - a device near the monitored goods, such as a warehouse worker's smartphone executes the process, querying the parcels' sensors for their status. Additionally, the edge BPMS should still propagate monitoring information to remote servers as early as possible in order for remote parties to react to events in an agile fashion.

Secondly, it is common that goods are handed over from one organization (company) to another during transportation and warehousing, which means no single device is always accompanying the goods. Thus, the challenge is how to realize continuous, process-based goods monitoring even if the monitored objects move through the boundaries of different business organizations and their devices'

coverage areas?

In the rest of this chapter, we detail this scenario further, to highlight its challenges and to illustrate the usefulness of D2D BP migration in the edge. We then propose a system architecture which enables decentralised Business Process execution in the edge network by MPH-s. Namely, throughout the lifetime of a single BP instance, the process execution is carried out by different devices (Mobile Process Hosts). This is realized by a process migration function in the BPMS of the MPH, that enables MPH-s to transfer running process instances to other nodes when appropriate. This allows BP execution in a way where during the lifetime of a single BP instance execution, it has been run by different process hosts belonging to different organizations, thus supporting the creation of long-running, delay-tolerant, pervasive business processes. For the mentioned scenario, this manifests in the goods monitoring BP-s being migrated from the MPH device of one company's worker to the MPH device of another company's worker dynamically, while the BP-s are being executed, to maintain a continuous monitoring of the goods.

We explain the technical details of implementing such MPH-s and their process migration functionality based on Android smartphones. We evaluate the scalability and temporal performance of D2D process migration and discuss the feasibility of this approach.

3.1. Scenario

Real-time monitoring of goods during transportation is necessary in case of products that bear damage if handled poorly (e.g. frozen foods exposed to unsuitable temperatures or fragile products experiencing rapid movement). Our motivating scenario is as follows.

A set of products is stored in Warehouse A. The warehouse uses an intelligent goods monitoring platform called "WiseWare", which keeps track of the status of compatible products stored at the warehouse. WiseWare-compatible products use parcels which have battery-powered, low-energy sensors attached to them. The sensors attached to the parcel vary by product (and manufacturer). During storage,

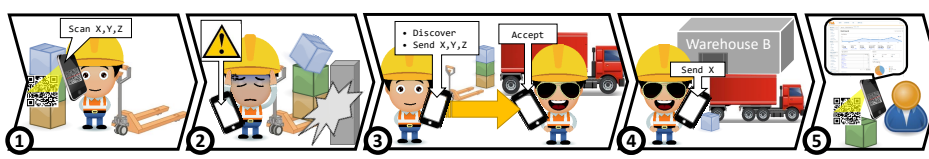


Figure 10: A selected sequence of events from the WiseWare goods monitoring scenario. From left to right: 1) scanning of products, 2) real-time reaction to events such as product orientation change, 3) transferring of products and their monitoring, 4) support for transferring (sub)selections of products, 5) web application based on the result of monitoring.

these parcel sensors are queried by monitoring devices attached to the warehouse shelves. The warehouse worker carries a personal digital assistant (PDA) with WiseWare software. The worker receives a notification that boxes X, Y and Z are to be delivered to the loading bay and handed over to a truck driver about to arrive and pick up these products. As the warehouse worker gathers the 3 items, each time (s)he picks up a parcel, (s)he scans the QR code on the parcel using the PDA to indicate that they are now handling the collected items (see Fig. 10 (1)). This triggers an event in the WiseWare system in which the task of monitoring the scanned parcels is handed over from the static, shelf-attached monitoring devices to the PDA of the worker. This is necessary because after picking them up, the parcel sensors are now out of wireless communication range of the static shelf devices.

Approaching the loading bay, the warehouse worker bumps into a corner with the pallet jack (s)he is using to transport the goods (Fig. 10, (2)). As a result, one parcel becomes tilted at an angle. The worker immediately receives a message on their PDA, stating that a parcel which needs to be held upright at all times has become dislocated. After taking care of the issue, the worker proceeds to the loading bay where the trucker is waiting.

When handing over the products, the warehouse worker uses the PDA to discover nearby WiseWare-compatible PDA devices. After choosing the trucker's PDA and the bundle of products, the trucker receives a notification on their smartphone that the warehouse worker would like to hand over the monitoring of goods X, Y, Z to the trucker. The trucker confirms this request and begins loading the boxes onto the truck, meanwhile the warehouse worker proceeds with their next task at hand (Fig. 10 (3)).

The trucker is fulfilling a delivery which involves two tasks: deliver product X to Warehouse B and deliver Y, Z to Store C. After driving to Warehouse B, the trucker similarly searches for nearby WiseWare-enabled PDAs, selects the appropriate one, but this time, only product X is selected. The warehouse worker at Warehouse B confirms the request and the product is handed over (Fig. 10 (4)).

While continuing the drive to Store C, the trucker receives a notification that the temperature of a parcel is becoming too high. The trucker adjusts the trucks' climate control and thus the issue is resolved. Finally, the products are handed over to Store C in previously described fashion.

After a few weeks, Store C has sold product Z to a retail customer. The customer is able to use the QR code on the product to open a web application which provides a history of events associated with the products' lifetime, including the slight increase in temperature at one point of time (Fig. 10 (5)).

Additionally, the manufacturer of product Z is able to look at the detailed log trace generated by using the WiseWare monitoring platform. Using this information, the manufacturer is able to assess which organizations are handling their products better and thus improve their business choices.

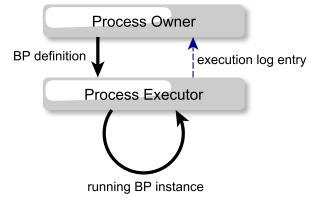


Figure 11: Movement of data between Process Executors and Process Owners.

3.1.1. Challenges

We focus on 3 challenges which the presented scenario raises. Firstly, the goods being monitored are in motion, posing the question: how to continue the monitoring, regardless of location? Using a conventional centralized approach is unsuitable, as it would require constant network connectivity, which may not be available during transportation (e.g. the truck is driving in a tunnel). Additionally, it would involve constant messaging between the WiseWare parcels and the central server, consuming bandwidth and increasing latency.

Secondly, how can the various devices which perform the monitoring adapt to the requirements of different goods dynamically? The warehouse worker might handle hundreds of different kinds of products per day, each one requiring a different kind of monitoring process, using different types of sensors, thresholds and activities for example.

Thirdly, the goods are being handled by different organizations (Warehouse A, truck company, Store C), so the WiseWare platform must support inter-organizational support on the fly.

3.2. Architecture

To address the raised challenges, we propose leveraging BPMS to realize goods monitoring in the presented scenario. When a product manufacturer decides to use WiseWare for goods monitoring, they model a Business Process for the monitoring of each of their products. As such, the BP is tailored to meet the product's monitoring requirements (e.g. temperature reading frequency) and the capabilities of the product's packaging (which sensors are bundled with the parcel).

By using Business Processes, devices can adapt to monitoring of arbitrary products, as the BP model provides the description of how the monitoring is carried out. Secondly, the use of standardized BP models such as BPMN 2.0, the potential for inter-organizational compatibility is increased.

However, if the goods are moved, e.g. handed over from a warehouse to a truck, the task of monitoring must also be handed over. For this, WiseWare is capable of process migration from device to device during runtime (explained in Section 3.3). This avoids constraining the monitoring only to a single device.

The system has two main categories of software agents: Process Owners and

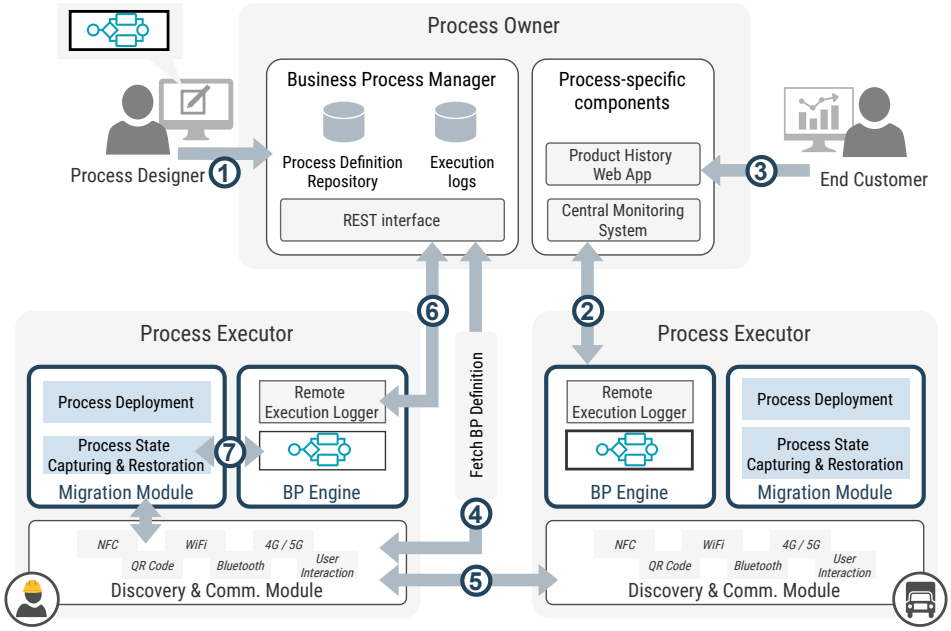


Figure 12: Architectural Overview of WiseWare platform

Process Executors. The Process Owner is interested in having their BPs executed to provide whatever service the process entails and so that they can gather log data regarding the execution of the process, which enables performing discovery or analysis-related activities of the BPM lifecycle (as discussed in section 2.4.1).

On the other hand, the Process Executor carries out the actual execution of processes. Process Executors can obtain BPs for execution in 2 ways: either instantiate a new BP acquired from a Process Owner or receive an already-running BP from another Process Executor via workflow instance migration (see Fig. 11).

In our WiseWare platform scenario, the product manufacturers who wish to enable goods monitoring and provide a "product history" service to their end-customers are the Process Owners. The various transportation entities (warehouses, truck drivers) who wish to provide an intelligent transportation service to manufacturers are the Process Executors. A single organization may participate in the system as a Process Owner and Process Executor simultaneously. For example, a manufacturer could also act as the Process Executor while the product is being stored at the factory warehouse.

Our proposed architecture (Fig. 12) adapts several of the common components of the BPMS reference architecture mentioned in section 2.4.3, distributing them between the Process Owner and Process Executors. Secondly, the conventional architecture is extended by modules necessary to perform device-to-device communication and migration. In the next subsections we detail each of the sub-components.

3.2.1. Process Owner

A Process Owner consists of a Business Process Manager and other, process-specific components which support the processes provided by the Process Owner (Fig. 12).

Business Process Manager. The Business Process Manager (BPMan) takes care of storing process definitions and allowing remote Process Executors to log their execution. BPMan exposes a REST interface to enable the invocation of these services by other parties.

In the WiseWare scenario, the product manufacturer's process designers create BP definitions that correspond to the company's products and submit them to the Process definition repository (see step 1 on Fig. 12). From this repository, Process Executors can retrieve process definitions to instantiate a process execution.

If a BP hosted by the Process Owner requires additional online services, these should also be deployed onto the Process Owner system. As an example, suppose that product X-s BP involves a BPMN *Send Task* which sends a set of recent temperature readings to the manufacturer (Fig. 12, step 2), where they are stored, so that a customer can later see them in a web application (step 3 of Fig. 12).

3.2.2. Process Executor

After the BP definition and relevant supporting components have been set up by the Process Owner, Process Executors may instantiate BPs by retrieving the definition from the Process Owner (Fig. 12, step 4). In our scenario, we assume that the manufacturer wishes to initiate the goods monitoring at the moment when the goods are handed over to Warehouse A. In this case, when the products arrive at Warehouse A, the warehouses Process Executor's Discovery & Communications module (DC module, explained further below) uses technology such as RFID scanners to identify the products and look up their corresponding process definitions. For each item, a corresponding BP instance is started. For example, if 100 units of product X arrived, then 100 instances of product X-s BP are started.

Discovery and Communications Module. The DC module provides interfaces for discovering and communicating with other entities and nodes within the platform. The communication may happen via proximity-based technologies such as Bluetooth Low Energy (BLE), NFC, RFID or via the Internet. Additionally, the DC module provides User Interaction (UI) functionality to notify the user about something or ask for user input. In our scenario, when Warehouse A-s worker wished to hand products over to the trucker, (s)he used the UI provided by the DC module to look for nearby WiseWare devices, selected the trucker's device, and was then presented a list of products whose BP-s were currently being executed by the worker's PDA. After selecting all products, the DC module sent the request to the truckers WiseWare device via BLE (step 5, Fig. 12). Upon receiving this request, the DC module of the trucker's device presented the request details, including information about the source of the request and the contents (list

of products) to the trucker. After the trucker confirmed the request, the warehouse worker's WiseWare device began the BP migration procedure, described in detail below (*MigrationModule subsection*).

Business Process Engine. Execution of BPs is carried out by the BP Engine. Unlike conventional BP engines such as jBPM or Activiti, the Process Executors BP Engine is extended with a Remote Execution Logger component. This component uploads execution log data to the Process Owner of the BP (step 6 of Fig. 12). If network connectivity is available, the Remote Execution Logger will upload log entries at a certain rate. When connectivity is unavailable, the Logger will buffer log entries so that they may be uploaded at a later time.

Migration Module. One of the critical enablers of the proposed system is the Migration Module (MM). MM is used to transfer a running BP instance from one Process Executor to another. To do this, MM halts the execution of the process instance, captures the process instance state and metadata using a migration model such as the one proposed by Barkhordarian et al. [Bar+12]. This metadata includes the execution state and attached process variables.

The migration module must include a strategy for deciding at what moment process instances can be halted. For example, should any currently pending jobs and activities be finished or instead should the process along with any actively running jobs (e.g. a BPMN Service Task reading a sensor value) also be suspended. The former approach is simpler, and avoids having to capture the internal state of already running jobs, but the trade-off is that the migration includes the additional delay of waiting for the active jobs to finish.

Once the process has been halted and the process state captured (step 7, Fig. 12), the model is serialized to a format such as JSON or XML. Now, the serialized migration data and process definition are sent to another Process Executor via the DC module (step 5, Fig 12).

When the process and its migration data are received at the destination device, the DC module forwards the data to the MM which then deploys the process onto the BP Engine and restores its state on the device. After succeeding, the original Executor is notified of a successful migration, and the process definition and any running instances are discarded.

As discussed by Barkhordarian et al. [Bar+12], the halting of a BP execution is not straight-forward, and can be especially complex if the BP execution is in a state of processing or waiting for (external) events. In this case, the unsubscribing and subscribing to the event source for the 2 Process Executors involved in the migration must be carried out.

The efficiency of the migration procedure determines the downtime of the process execution. For the goods monitoring scenario, this means that the faster the migration procedure is, the less monitoring time is lost.

3.3. Prototype and Experiments

We have implemented a prototype of the Process Executor component of our system as an Android application. The prototype is capable of executing and migrating BPs. For the BP Engine, we used the open-source Activiti BPM software version 5.21 (June 2016), which has been adapted to run on the Android OS (by referring to the source code from [Dar+15], which they used for an earlier version of Activiti BPM). Activiti uses a relational database to manage BP instance execution metadata and state. The Android version has been adapted to use SQLite as this database.

The Migration Module has been implemented as an extension to the Activiti engine. As described by [Bar+12], the Migration module listens to events broadcast by the Activiti Engine to determine the exact moment when migration can be triggered. Our prototype uses the strategy which waits for the currently active jobs to finish before initiating the migration. We distinguish two types of migration operations: **BP instance export** and **BP instance import**.

Instance Export denotes the process of suspending the execution of a BP instance, capturing the information relevant to the BP instance from the Activiti database (DB), and serializing this information into a format suitable for transferring to another Process Executor. In our prototype, we chose JSON as this format. After serializing into JSON, our prototype writes the data into a file.

Instance Import is the opposite of the previously described Export operation. Import involves deserializing the chosen BP state format, inserting the data into the Activiti DB and then activating the BP instance in the BP engine so that process execution may continue.

Our prototype presumes that when a BP is migrated from one node to another, then both nodes already have the BP definition deployed to their BP engine. Technically, dynamic deployment of the process definitions alongside the migration is feasible. The process definition .bpmn20 files have to be migrated and any additional libraries/classes which the process definition relies on need to be transferred and dynamically loaded into the Android Java Runtime, as described for example by Schobel et al. [Sch+16].

The source code of our migration-capable Activiti engine for Android is available at: `https://github.com/jaks6/WiseWare-BP-Engine`.

3.3.1. Experiment Description

In order to evaluate the scalability and performance of our prototype, we conducted a series of experiments with real devices. The experiment consisted of migrating different numbers of BPs concurrently from one Process Executor to another. The BP used in the experiment is a simple looping process which acquires two sensor values and acts on the result when appropriate, the BP model is presented in Fig. 13. Such a process represents a possible implementation of goods monitoring used in the WiseWare scenario. An instance of this process

runs for each product parcel, and when the parcels are handed over in the scenario, these processes are migrated.

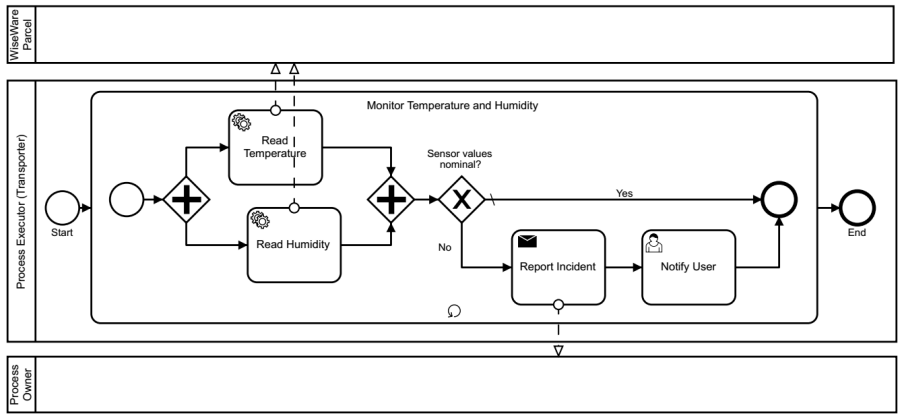


Figure 13: Business process used in the experiments.

Using LG Nexus 5 phones, we separately measured the execution time of both BP instance export and import. The focus of this contribution was to study the feasibility of the mobile Process Executor, our experiments did not cover D2D transfer performance of the serialized migration file. This D2D transfer is influenced by factors such as the type of technology used (BT, WiFi-Direct, for example) and environmental factors such as signal strength and device proximity. One can learn about the impact of the D2D transfer on the performance from works such as [Con+13].

3.3.2. Results

Execution time. Measuring the execution time for instance export and import yielded the results summarised in Fig. 14. As can be seen, exporting is considerably more time-consuming than importing. During export, the BP engine DB was actively being used by all the running BP instances, whereas during import, the engine DB was mostly idle as no BPs were running in our test scenario.

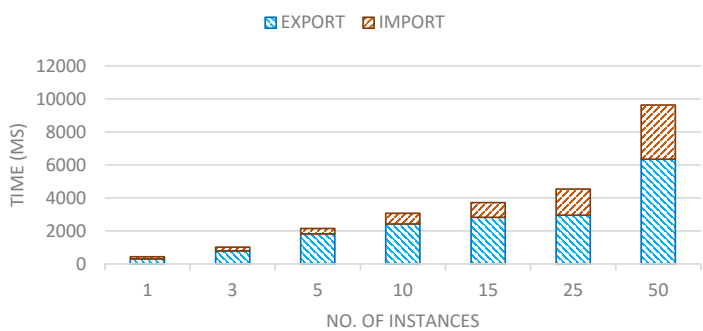


Figure 14: Time consumption of process migration.

No. of instances migrated	1	3	5	10	15	25	50
File size (kilobytes)	1.88	3.79	6.45	12.84	18.9	28.75	54.93
Time (s)	0.44	1.01	2.15	3.07	3.7	4.54	9.63

Table 2: Migration file sizes and total time consumption with different numbers of instances.

We have presented the time consumption of the migration process (adding both the import and export operations) in Table 2. In total, the migration would take slightly less than 10 seconds in case of migrating 50 instances. While this cannot be considered fast, as the D2D transmission is not included in this time, we find this to be still an acceptable result considering the WiseWare scenario, where the time waiting for BP migration is probably spent on cargo loading activities anyway. Additionally, we consider our prototype to be preliminary, so it can likely be significantly optimized.

Migration file size. Table 2 also presents the file size of the serialized BP instance state metadata. Considering the 54.93 kB migration file used when migrating 50 BP instances, a single instance took up around 1.1 kB of the file.

3.4. Discussion

This contribution proposes a system architecture for enabling continuous, delay-tolerant BP execution that brings the execution of processes to mobile process hosts as opposed to centralized BP execution. The system is capable of handing over the task of BP execution from one node to another in the form of process migration, enabling executed by different parties and thereby increasing the adoption of long-running edge processes.

We presented a motivating application scenario for this architecture from the domain of transportation and goods monitoring, showing how such a system can be applied to maintain detailed insight about product status throughout the process of transportation from the manufacturer to the end customer.

Through implementing a prototype of the application scenario and MPH, we showed in practice how edge process execution can be realized on Android smartphones using Activiti software, how to implement device-to-device process migration, which were the goals of the 1st research question. Previous works who considered Activit BPMN migration did not evaluate the feasibility on smartphones [Bar+12] or involved device-specific multiple variants of the process model, not direct migration [Dar+15] .

Evaluating the time consumption of our prototypes migration operations showed the performance being in the order of a few seconds when migrating up to 50 processes. Further investigation into optimizing the migration performance is nec-

essary to make this approach feasible for use-cases with larger parallel migration counts.

In the presented scenario, the decisions *whether* and *when* to migrate have been handled by the human users who manually initiate the migration procedure. This is reasonable for the described use-case, however, in other applications, it may be desirable to automatically initiate such procedures, as soon as the devices are within effective communication range for instance.

While this analysis did not include the effect of wireless signal quality on the D2D transfer, we have explored the impact of device proximity and signal strength on file transfers in the study [MCS18]. Our practical experiments using WiFi routers and smartphones investigated the communication performance in cases where the mobile device is moving. The results showed that the transfer times, which in optimal conditions took a few seconds, can double or triple depending on the exact scheduling of communications, showing that the timing of the communication is crucial.

In the next chapter we see how such D2D communication tasks can be scheduled automatically and explicitly modelled as part of an edge process.

4. CONTEXT-AWARE BOTTOM-UP SCHEDULING OF FOG-RELATED PROCESS TASKS

The Fog Computing trend [Bon+12] introduces cloudlets (fog servers) to the Edge network, where the IoT devices are located. This allows Mobile Process Hosts to execute business processes that make use of the nearby Fog infrastructure, such as offloading compute-heavy tasks to a nearby cloudlet.

Offloading of tasks can be desirable for energy-saving and performance, as a Fog server's computational capacity is likely larger than that of an edge device, such as smartphone or SBC. Further, compared to offloading to the Cloud, the Fog-based approach is more resilient to conditions where a constant core network connection is not available (such as disaster, military scenarios and other hostile environments) [Sat+13].

Making optimal decisions about scheduling interactions with the Fog is challenging for MPHs due to having to account for dynamic context factors of edge network environment, such as mobility and interrupted connections. Namely, the locations of the fog servers, their current load, task execution deadlines, offloaded task payload size and movement trajectory of the MPH influence this. A failure to take into account radio signal conditions in the coverage area of a Fog server or the Fog servers load can incur wasted energy and excessive time consumption for the MPH process.

In this chapter, we introduce the term *Edge Process Management* (EPM) to identify both the execution of IoT processes at the edge network, including the above-mentioned scheduling-related decision-making. This term follows the example of concepts such as Edge Computing [Sat17], Edge Analytics [Sat+15] and Edge Networking [Hua+16], which emphasise the activities at the edge networks in contrast to the core network, where the cloud or central management servers are located, as we explained in section 2.3.2.

Our discussion in section 2.6.1 revealed that the existing Edge-hosted execution engines do not consider using the Fog infrastructure as part of process execution. Additionally, section 2.6.4 concluded that while a number of BPMS for IoT (BPMS4IoT) have already integrated Fog- and Edge-related approaches in their architecture [CSB16], few of them address offloading decisions and mobility simultaneously as we will explain further in this subsection.

In order to clarify the problem statement, we describe an emergency response scenario for natural disasters. Fig. 15 illustrates a mobile IoT scenario in which the wearable device of a first responder, who is performing rescue activities, is continuously operating real-time sensing and context reasoning processes, e.g. gathering sensory data and video from nearby devices to map out people density or damaged infrastructure. These operations involve distributing some context reasoning and processing tasks to the fog when the device encounters fog servers. In order to identify the feasible schedule for executing the distributed task between the de-

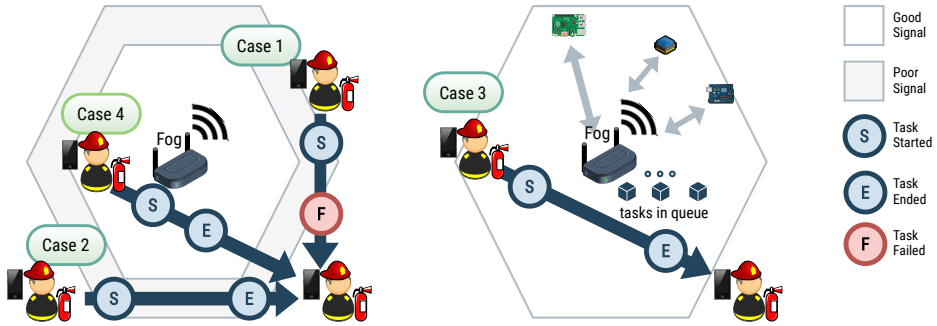


Figure 15: Run-time factors that influence the cost-performance efficiency of EPM in mobile IoT-based disaster scenario.

vice and fog server, common service-oriented systems would require the wearable device to continuously update its GPS location to the distant management server and to receive the best schedule from the management server. Such an approach however is fully reliant on the central server, plus involves extra energy costs due to using mobile Internet. Alternatively, the system could configure the wearable device such that the device autonomously executes the task whenever it enters a fog server's signal coverage. Ideally, such an approach can reduce significant resource usage for the wearable device. Yet mobility-related factors, identified as an important research challenge in mobile and fog systems [Guo+18; Mou+18; Reh+17] can result in poor performance and hence reduce the Quality-of-Service (QoS) for this approach.

For instance, as Case 1 of Fig. 15 shows, if the system executes the task while the person only stays in the signal coverage for a short period, the task can easily fail as the communication time for completing the task is insufficient; Case 2 shows that the system executes the task while the person enters the border of the fog's coverage, which has poor signal strength and consequently the timespan of the task execution is much longer than the ideal case (Case 4), which executes the task while the person is within the good signal coverage. Further, there are other factors which affect the cost-performance efficiency of distributed tasks in EPM. For example, in Case 3 (see Figure 15), the system executes the distributed task while the rescue worker is in the good signal coverage. However, there are other IoT devices simultaneously interacting with the fog server. Hence, the heavy workload of the fog server has increased the latency of the wearable device's distributed task and indirectly increased the energy consumption of the device because it has to maintain the network communication for a longer period.

Based on this scenario we highlight the 2nd research question:

How to schedule the execution of process tasks involving nearby device-to-device communication, such as offloading to cloudlets, in an energy-aware and quality-of-service-aware manner while taking into account mobility of MPH, the workloads and locations of the cloudlets?

Movement-induced connectivity disruptions can potentially be handled with the reactive approaches taken by [SHS18; MMS16; Zha+18] introduced in section 2.5, but these systems do not pre-emptively avoid connectivity issues which could be predicted based on knowledge about network coverage areas and movement trajectories. Additionally, the use cases presented in these papers do not explicitly focus on modelling system behaviour during such connectivity-related occurrences.

Dynamic workflows using opportunistically discovered services are managed in [Zou+16]. Here, the system uses the workflow template and therein described QoS requirements to schedule the services involved in the workflow on cloud servers fitting the requirements, considering available resources and geographic location. The system thus considers the user's current location to choose servers that are closer, but does not consider the user's mobility and future locations for future tasks within the workflow

A problem that has received significant attention in past literature is *whether* to distribute a task to a cloudlet or process it locally instead. Works such as [LS13; Li+13; Zha+14] give various mechanisms to make this decision. The common argument is that in case the offloading takes place, but the connection is interrupted, the user has to either re-offload or perform the process locally instead and as a result has consumed more resources than simply performing the task locally. On the other hand, in this chapter we emphasize *when* to offload, given that the decision *whether* to offload has already been made. As such, we focus on anticipating weak or interrupting signals, instead of reacting to exceptions with the adaptive schemes mentioned in [SHS18; MMS16].

Mobile-based works addressing the question "*which* cloudlet to offload to?" usually involve the geographic locations of the nodes and system resource utilization, as in [ATH16; CTC17]. Here, the distance between the mobile and cloudlet are taken into account, however, the effect of user mobility during the task and communication are not taken into account. The other common approach is to use historical recorded movement and connection events to train a model for prediction and decision-making, such as in [LS13; Li+13]. This kind of approach is limited in the sense it may not work for situations with new fog nodes or new mobile users for which there exist no historic data at all.

With respect to the above literature gaps and raised research question, we propose an adaptive task execution scheduling scheme for service-oriented EPM systems based on context-awareness. To validate the proposed scheme, we implemented and evaluated a proof-of-concept prototype in a simulated setting which consists of mobile devices and fog servers.

4.1. System Design

The Edge Process Management system we propose aims to fulfil the following high-level goals:

- Enable de-centralized process execution by hosting process engines on the edge devices
- Leverage mobility-related context such as user movement trajectories and fog infrastructure geographic locations.
- Manage the process task execution based on the mobility context, optimizing trade-offs between delays in task execution versus energy-efficiency and QoS
- Adapt existing Fog computing architecture approaches, such as virtualisation-based runtime environments

Based on these requirements, we propose a workflow-based fog system architecture, described in detail in the following subsections. The architectural overview of the system is shown in Fig. 16. It consists of 3 types of entities: fog servers, Mobile Thing Hosts (MoTH-s) and supporting Cloud Services (Central Management Cloud and Federated Fog Service Registries). These components are described further in detail.

4.1.1. Fog Server

The fog server is a *cloudlet*-type node, which provides compute, storage and networking services to MoTHs in proximity. It involves the following submodules (see *Fog Server* on Fig. 16).

Fog Application Runtime Environment. Following the fog computing paradigm, the fog server hosts a Fog Application Runtime Environment (FARE), which is

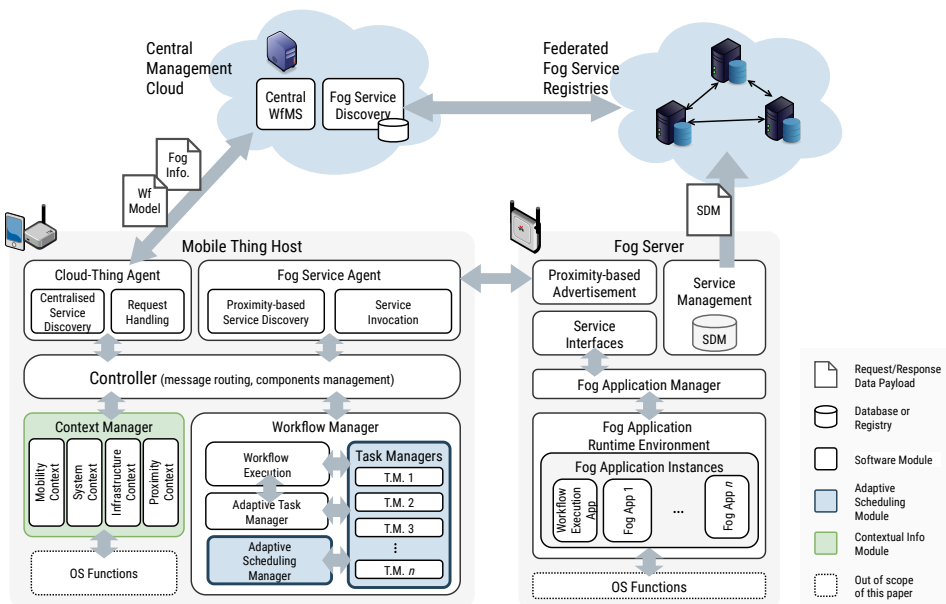


Figure 16: Overview of System Architecture

based on computing virtualisation technology (e.g. Virtual Machine or Containerization) and thereby provides an environment where isolated *Fog Applications* can be deployed and run. All of the services which a fog server provides to its nearby clients are packaged as Fog Applications, they can be seen a Software-as-a-Service style applications, which are deployed either by nearby MoTHs or from the Central Management Cloud.

The virtualised FARE allows applications to use the operating system functions of the host machine, including networking functions or system hardware status information. So, for example, if the FARE hosts an analytics server application (such as Apache Edgent[1]), then nearby MoTHs can use the functions provided by that application over the network.

The deployment of applications is based on either a central repository of applications (e.g. *Docker Hub* in case of Docker[2]-based containerization) or by direct device-to-device deployment (e.g. VM images) between Mobile Thing Hosts and Fog Servers.

In the context of this thesis, we focus at a particular Fog Application, the so-called *Workflow Execution App*. By using BPMS such as *Flowable*[3] as discussed in section 2.4.4, the fog server can execute processes and be used as part of edge process choreography, interacting with BP execution engines hosted on other devices.

Application Management and Service Interfaces. Application deployment and other life-cycle related functions of the FARE are handled by the Fog Application Manager. This component defines the supported operations for FARE, such as reserving an application to a particular client, allocating resources and priorities between the different applications, creating and deleting application instances.

The Fog Application Manager can be accessed through the Service Interfaces provided by the Fog Server. Service Interfaces are web standards-based interfaces (e.g. HTTPS RESTful API with JSON) that MoTHs can use to interact with the fog server. The exact technology may be motivated by the choice of a particular IoT framework uses (e.g. JSON-LD in case of FIWARE or Web of Things, as mentioned in section 2.2.2).

Service Management. In this architecture, each Fog Server is managed by some central organization (e.g. a regional government or a company). The Fog Server periodically reports to the corresponding central Federated Fog Service Registries, indicating its status, by sending Service Description Metadata (SDM). SDM includes characteristics such as hardware configuration (including rated wireless coverage range), server geographical location and logs of exceptional events or warnings. Providing SDM to the federated registries is the crucial requirement to enable discovery of fog servers based on their locations and capabilities (discussed further in 4.1.2, "Cloud-Thing-Agent"). The SDM updates can be

[1] `http://edgent.apache.org/`

[2] `https://www.docker.com/`

[3] `https://www.flowable.org/`

seldom, since hardware configuration and location are usually very static, thus the system can continue operating even when the central systems are not reachable (e.g. during a disaster).

Proximity-based Advertisement. Finally, to notify nearby clients of the existence and current real-time status of the Fog Server, the Proximity-based Advertisement component broadcasts the Fog Server's current state to nearby devices. This should be implemented with a lightweight technology that enables frequent periodic broadcasting, such as Bluetooth Low Energy. The advertised real-time status includes the resource utilization of the FARE, indicating whether the fog server is under significant load or idle.

4.1.2. Mobile Thing Host

The Mobile Thing Host is the moving edge device, which can communicate with nearby sensor devices and interact with Fog servers. In our system, the MoTH includes a Worklow Manager component, which enables it to execute edge processes or partake in them.

Controller. The inter-component messaging is performed through the Controller, which mediates requests and response messages from the different components listed in the following subsections, acting as a message bus.

Workflow Manager. The Workflow Manager of a MoTH provides means to execute workflows defined in a process modelling language (BPMN 2.0). The executed workflows involve the service composition and decision-making tasks involved in IoT scenarios. Workflow Execution component is the core of the Workflow Manager and in our system, we use the same Flowable-based solution that the FARE Workflow Execution App mentioned in 4.1.1 uses.

Adaptive Task Manager. If the IoT process involves distributed tasks interacting with proximal devices such as Fog Servers, then the task execution schedule must take into account energy or cost efficiency based on contextual factors. We call these types of tasks *Adaptive Tasks*. When workflow execution reaches such a task in the process, it uses the Adaptive Task Manager to manage its execution.

The Adaptive Task Manager creates a unique Task Manager (TM) instance (see Task Managers on Fig. 16) for each *Adaptive Task*. The TM instance contains the task requirement metadata, described in the workflow model, such as timeouts or QoS parameters and hardware requirements for the involved Fog Server.

Further, each TM instance uses the Adaptive Scheduling Manager to acquire the context-based execution schedule for that TM. Using the resulting schedule, TM signals the Workflow Execution of the chosen execution time and chosen Fog Server for the associated adaptive task.

Adaptive Scheduling Manager. This component performs the actual context reasoning and decision-making. It uses as input the Adaptive Task description from the TM and contextual information from the Context Manager to determine a fitting candidate Fog Server for the respective task. The contextual information

includes:

- Infrastructure Context, which contains the known fog server locations of the current area
- Mobility Context, which contains the current position and predicted movement trajectory of the Mobile Thing Host (based on GPS sensors, map data, user set destinations, etc.)
- System Context, which includes values such as battery level

Based on the above, an initial set of eligible Fog Servers are chosen. Whenever the Proximity-based Service Discovery component encounters a Fog Server nearby, additional information from the Fog Server advertisement message is used by the Adaptive Scheduling Manager to decide if the current Fog Server fits the current task.

Fog Service Agent. As mentioned above, the Fog Service Agent includes the Proximity-based Service Discovery for handling the advertisements of Fog Servers. Further, the Service Invocation component is used to call functions of the Fog Application Manager of the server, as explained in 4.1.1.

Cloud-Thing Agent. This component handles communication with the Central Management Cloud, and fulfils two functions: a) refreshing knowledge about Fog Servers and b) handling Workflow Manager requests from the cloud. For a), the Centralised Service Discovery queries the remote Fog Service Discovery repository. Here, the SDM of the Fog Servers, including their locations are returned. For b), the Request Handling module is capable of receiving workflow related operations from the Management Cloud, such as deployment of new Workflow Models or starting workflow execution instances (top-down approach as explained in 2.3.6).

Context Manager. As mentioned previously, the Context Manager accumulates contextual information such as fog server locations in the area, metadata about proximal devices and fog servers, mobility related information such as user's current moving trajectory and system data such as battery level.

4.1.3. Central Management Cloud (CMC)

This component is the initiator of tasks and workflows executed in the edge. On one hand, the CMC interacts with the Federated Fog Service Registries to stay up to date about the available Fog Infrastructure. On the other hand, the CMC Central WfMS can deploy workflow Models onto MoTHs, invoke execution of workflow instances on mobiles, or invoke workflow tasks on the mobile as part of cloud-edge workflow choreography.

To illustrate, consider the disaster recovery scenario mentioned earlier. The first responders on the scene each have a MoTH device, while the chief coordinators of the rescue operations use the CMC to deploy workflow models needed for the rescue to the MoTHs and Fog Servers in the region. Now the rescue workers

can operate the scene, while their devices perform the adaptive edge processes involving fog servers and workflow choreography.

4.1.4. Adaptive Task Execution Scheduling Scheme

In this subsection, we describe the heuristic scheme used by the MoTHs Adaptive Scheduling Manager for deriving adaptive task execution schedules. The scheme consists of a multi-criteria decision model which tries to decide which fog server to invoke in a distributed task.

The continuously running scheme has 2 phases: 1) candidate fog server set formation and 2) fine-grained decision-making.

Phase one. Let S be the set of known fog servers. Next, for each $s \in S$ we have some numeric parameters:

- $E(s)$ gives the estimated remaining energy (battery life) value of the MoTH when entering coverage area of s.
- $M(s)$ is meeting duration, the time spent in the coverage area of s by the MoTH .
- $D(s)$ is the average distance to s during movement in the coverage area.
- $B(s)$ is the maximum bandwidth capability of s.

Next, we normalize the parameters by scaling each value to $[0,1]$, and invert D, as lower values are seen as better and get the following set of parameters: $\mathscr{A}_s = \{e_s, m_s, d_s, b_s\}$, where for any $s \in S$:

$$e_s = \frac{E(s) - \min(E)}{\max(E) - \min(E)} \qquad m_s = \frac{M(s) - \min(M)}{\max(M) - \min(M)}$$

$$d_s = 1 - \frac{D(s) - \min(D)}{\max(D) - \min(D)} \qquad b_s = \frac{B(s) - \min(B)}{\max(B) - \min(B)}$$

Using this parameter set, $\forall s \in S$ we define the score β_s:

$$\beta_s := \sum_{a \in \mathscr{A}_s} a \times \omega, \tag{4.1}$$

Where ω is a vector $\omega \in \mathbb{R}^4$, that assigns a weight to each parameter in $\mathscr{A}$. This allows to fine-tune the model to emphasize a certain factor, such as battery life, for example.

The idea is to choose an s which maximizes β_s. Given some threshold γ, we can now omit some unfit fog servers. γ is set by the user or application as a QoS requirement. This concludes the first phase, and as a result we end up with the set of eligible candidate servers $S' = \{s \mid \beta_s \geq \gamma\}, S' \subseteq S$.

At this point, the task execution and Adaptive Scheduling Manager can go to sleep until the geographically closest $s \in S'$ is approached.

Phase two. Whenever the communication range of any of the servers in S' is entered by the MoTH, phase two is started. When this occurs, the MoTH fetches the status of the fog server. This introduces additional parameters:

- $Q(s)$ the work queue size of s, how much time it takes for tasks already queued on the server to finish.
- $J(s)$ the server's time consumption for processing the current task which the MoTH wants to distribute, including the transmission time.

Let λ be a QoS modifier, which amplifies the penalty of waiting for a task queue has, we define a weight w for every s:

$$w(s) = \begin{cases} 1 + \lambda(1 - \frac{Q(s)+J(s)}{M(s)}) & \text{iff } s \text{ status information is available,} \\ 1 & \text{otherwise} \end{cases}$$

Then, we get a weighted score β^*

$$\beta_s^* := w(s) \sum_{a \in \mathscr{A}_s} a, \tag{4.2}$$

β^* is used for final decision-making. Here, we can either choose the server s with $\max \beta_s^*$. or set another threshold γ^* to again filter out all suitable servers, $S'' = \{s \mid \beta(s) \geq \gamma^*\}$ and select an element of S'' based on a parameter of interest, e.g. choosing the closest of the fitting servers. However, if $S'' = \emptyset$, we can decide to either deem the task as failed or choose to process it locally.

4.2. Experimental Case Study

To see the impact of our proposed adaptive scheduling, we conducted experiments in a simulation test-bed that supports executing processes defined in BPMN 2.0, following the principles of the proposed system architecture.

4.2.1. System implementation in Simulated Testbed

Following the comparison of available simulation tools described in section 2.6.5, we based our prototype on the ONE simulator [KOK09], which is a discrete event simulation engine designed as a Delay-Tolerant Protocol testbed. The ONE supports various mobility models, including map-based movement models, it simulates communication interfaces, connections and messaging between nodes (called hosts). Further, the ONE includes an energy model based on the communication energy costs caused by using varying connection speeds. This makes it a suitable tool for simulating fog environments and implementing the fog servers and MoTHs in our architecture as ONE hosts.

Parameter	Value
Radio range	91 m
Linkspeed	1-54Mbps (distance-based)
Transmission Energy Cost	1 μJ/s
Scan Energy Cost	0.2 μJ
Scan Response Energy Cost	0.1 μJ

Table 3: Simulated wireless interface & energy model configuration

Workflow Manager & other Modules. While the Java-based ONE is easy to extend, the original software has no notion of workflow-based messaging and model-based process execution. To this end, we extended The ONE by implementing a Workflow Manager (WfM) (as described in section 4.1.2) for the hosts. The WfM is based on the Flowable Java process engine. Although in the 1st contribution the process engine used was Activiti, Flowable is a fork of the Activiti project. The switch to Flowable was made due to the Flowable projects active focus on optimizing the engine to be efficient and lightweight, allowing its usage in modern paradigms such as serverless[4], making it a better fit for Fog/Edge solutions. By integrating the ONE simulator and Flowable engine, the hosts involved in the simulation can execute BPMN 2.0-defined process models, and thereby participate in process choreography where simulated hosts can exchange messages, variables and do decision-making based on the defined workflows.

Other components presented in the system architecture, such as the Fog Service Agent, Adaptive Task Manager, Adaptive Scheduling Manager were also implemented for the simulator hosts. For instance, the mobility context is derived from the mobility models used by the simulator, while proximity-based advertisement and service discovery are implemented as ONE simulator messaging applications.

Technical Details. To keep the simulation performant, the Flowable engine used by the ONE hosts is configured to use an in-memory database, and features such as process diagram image rendering and process history have been disabled.

For the networking interfaces of hosts, we use the ONE's *DistanceCapacityInterface* implementation, which adjusts the link speed according to the distance to the other connected host. The interface is configured to resemble an IEEE WiFi 802.11g interface, with a maximum linkspeed of 54Mbps. Combined with ONE's energy model, weaker signals (longer distances) cause more energy consumption due to lower data rates. ONE's energy model[5] consists of an energy budget, which gets reduced by a fixed amount per second when transferring data or when scanning/responding to scan events. The numeric parameters of the wireless interface and energy model are listed in Table 3. The simulations were run on a PC with an

[4]https://www.flowable.com/blog/flowable-engine-as-a-serverless-function

[5]The model is defined in the Java class *routing.util.EnergyModel*

Intel i5-6200U processor and 16 GB of memory.

Discussion. The created testbed allows analyzing the influence of the dynamic fog environment on workflow-based process execution. On one hand, the processes can be designed at an abstract level through visual BPMN modelling tools. For instance, distributed task message sizes and QoS requirements have been set at the process modelling level, instead of configuring these at the lower level of the simulation engine itself. On the other hand, thanks to the ONE simulators map-based functionalities, the testbed enables more realistic process-based scenarios, by importing map and route data files from OpenStreetMap[6], useful for applications in the *Smart City* vision, for instance.

4.2.2. Scenarios

We studied two categories of scenarios: first, we examined the four isolated scenario cases described in the beginning of this chapter. These involve a simple setting with 1 MoTH and 2 fog servers and highlight the scheduling decisions of the individual MoTH. The second kind of scenario involves larger numbers of MoTHs, servers and a map more similar to real life streets, aiming to demonstrate the usage of the scheme on a larger scale.

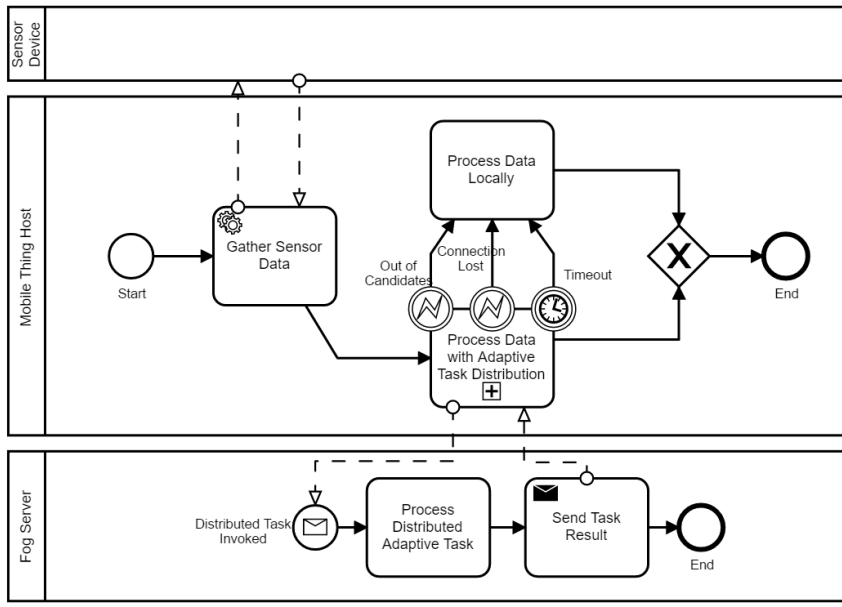
The respective workflows (described below) enacted by the MoTH and fog server are the same for all scenarios, but the configuration of server locations, MoTH movement speeds, message sizes, etc are varied.

Workflow models. The experiments use a choreography of 2 processes, shown in Fig. 17. The core idea is to perform distributed edge processing/analytics, where a MoTH gathers data from a nearby sensor device and forwards it to a proximal fog server where the data is processed, in other words, offloading the processing to the fog server. The processing results are directly sent back to the MoTH. If no fog servers are available or the connection is interrupted, the MoTH will fall back to processing the task locally.

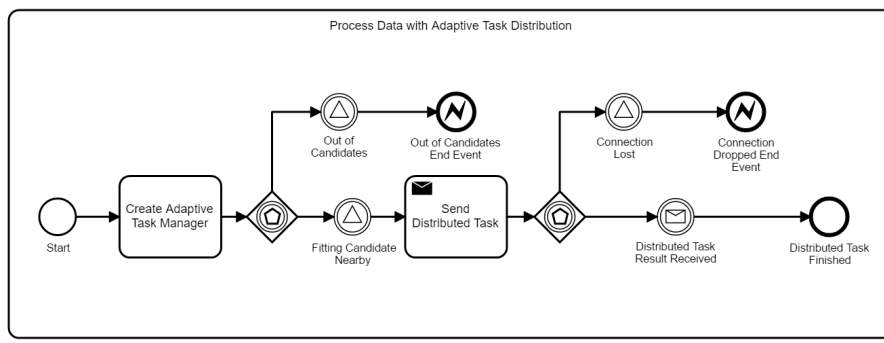
More precisely, in the MoTH process (middle pool of Fig. 17a), after gathering the sensor data, the Adaptive Distributed Task (ADT) subprocess is called, Fig. 17b. Here, as described in 4.1.2, first a new TM is created for the current ADT. Then, the process arrives at an event-based gateway, where the execution waits for the TM to raise one of the following events: a) the ADT timeout period (defined in the process model) is exceeded; b) the adaptive module has determined that no fog servers will be encountered within the given movement path or c) the Adaptive Scheduling Manager signals the process of a fitting nearby fog server. In the latter case, the execution proceeds: the gathered data is sent to the nearby fog server via a message (*Send Distributed Task* in the process diagram). After this, the process waits for the response message from the fog server which contains the analysis results.

The Fog Server process, at the bottom of Fig. 17a, is a reactive process, which

[6] `https://www.openstreetmap.org/`



(a) Case Study Process Choreography



(b) Adaptively Distributed Task expanded subprocess

Figure 17: BPMN 2.0 Process Models used in the case study

is started each time a new message indicating a distributed task is sent to the Workflow Execution Application. The first task performs the compute intensive processing of input data that was attached to the message and usually takes some time. The 2nd task sends the results of the processing as a message to the respective MoTH.

The described process choreography is used in all of the experiments. We wish to emphasize *"Create Adaptive Task Manager"* task, where the task includes some parameters for the adaptive model, such as the required QoS level, size of the distributed task and weight preferences. This way process modellers can configure the offloading behaviour and adaptive model to their application needs. While this process model definition is re-used for the different scenarios and for different

hosts, they are instantiated with varying parameter values in the experiments.

4.2.3. Isolated scenarios with single mobile host

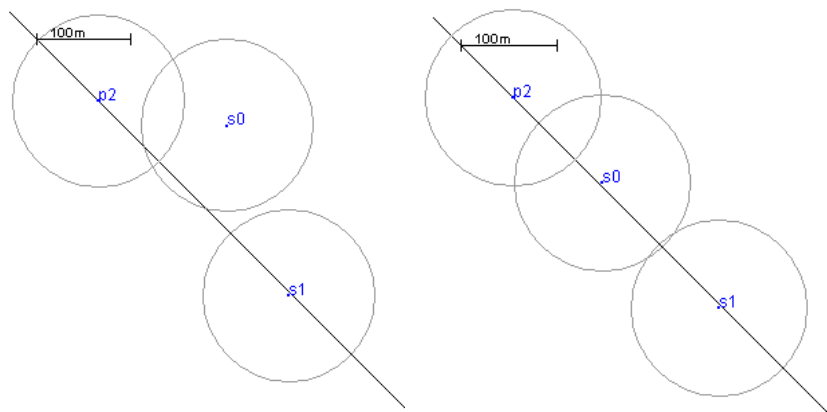
The simulated environment for the first 2 cases is shown in Fig. 18(a), where the circles depict coverage areas of fog servers (named *s0* and *s1* hereafter) and movement of the MoTH is shown as a line, beginning at the top-left.

We compared our adaptive scheduling model against a baseline naïve approach, where the MoTH always tries to distribute the task to the first fog server it encounters. This is actually also how our model will behave, if the QoS parameter γ is set to 0.0. We compare this naïve approach against our scheduling mechanism with $\gamma = 2.5$, observing the execution time and energy consumption of the distributed task. With these settings, the baseline always chooses the closer server *s0*, while our model tends to choose *s1* with the better signal coverage.

Case 1: Handling brief contact times. Here, the MoTH moves through the coverage area of *s0* only near the edge, where signal reception is poor, while *s1* is located exactly on the MoTH's trajectory.

The results are shown in Fig. 19, where the top row y-axis depicts total time (in seconds), measured starting from delegating the task to the fog server until receiving the task results, i.e. that includes both transmission and processing times. If server connection is dropped and the MoTH falls back to local task processing, then that time is also included. Please note that this time only includes time spent on the distributed task and not including movement. In other words, the time spent on approaching any server, which may also be located farther away, is not included.

The bottom row depicts the energy consumption of the process in Joules, as measured by ONE's energy model mentioned in section 4.2.1.



(a) Case 1, Case 2 (b) Case 3

Figure 18: Simulation settings, *s0* and *s1* are the fog servers, *p2* is the Mobile Thing Host

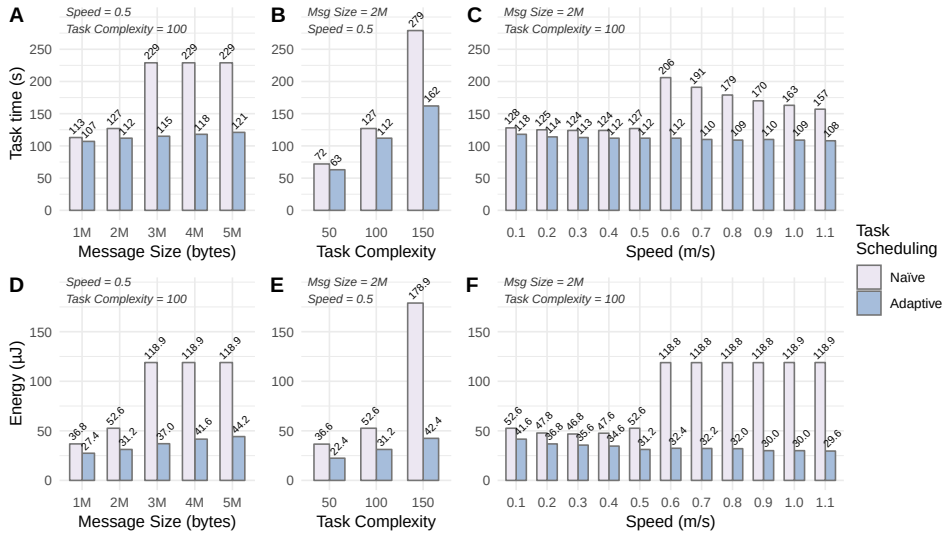


Figure 19: *Case 1* distributed task time- and energy consumption for Mobile Thing Host

Fig. 19(A) and 19(D) show the effect of the distributed task message size, which affects how long it takes to delegate the task, but does not affect the task processing itself. We see that for smaller messages, the result is similar for both approaches, with the baseline performing slightly worse due to the poorer signal quality.

However, a sharp increase in the processing time and energy consumption for the baseline case can be observed in all sub-figures of Fig. 19 (e.g. starting from speed 0.6 on sub-figures 19(C), 19(F)). This is due leaving the fog servers range before the distributed task finishes, causing the task to be processed locally. Meanwhile, our approach (QoS=2.5) always chooses server *s1*, where the distributed task can finish even with larger message sizes or task complexity. In the simulated implementation, task complexity is defined as how much simulation time it takes for the host to finish processing that task, excluding communicating the inputs or outputs. A single host can concurrently process 1 task.

High movement speed of the MoTH (Fig. 19 C,F) causes *s0*-s connection to drop earlier, causing the local processing to happen sooner in the naïve approach, thus reducing the total task time. Whereas for the adaptive approach, higher movement speeds mean the mobile reaches the best signal areas faster, improving performance.

Case 2: Handling poor signal contact. The 2nd case differs from the first one in that *s0* is located slightly closer to the MoTH-s movement path. This way, the coverage is large enough for the distributed task to finish, although the performance will be worse due to the weak signal compared to *s1*.

As can be seen from Fig. 20(A), with small message sizes, the time difference is minuscule. However, as the message size when communicating with the servers

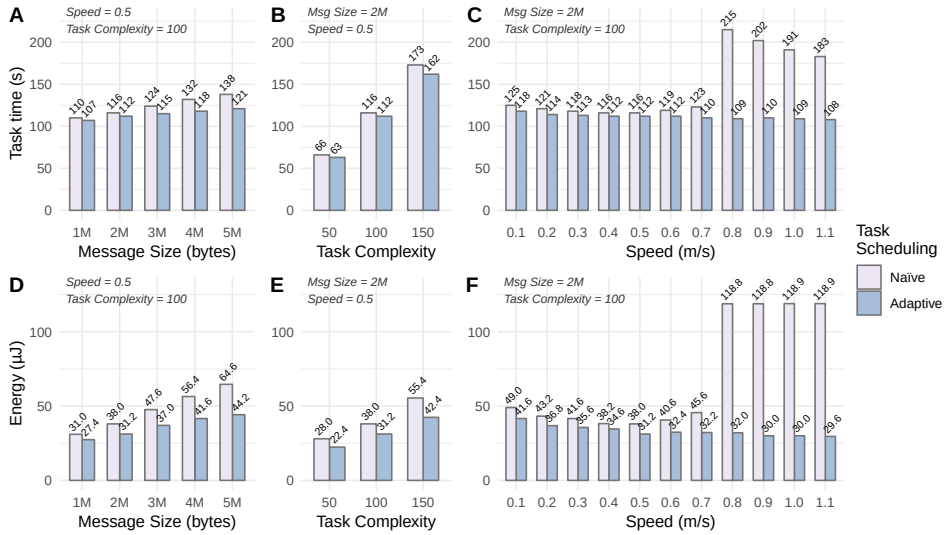


Figure 20: *Case 2* - distributed task time- and energy consumption for Mobile Thing Host

grow, the difference between the baseline and our approach increase, due to the signal strength influence which limits transmission data rates. The same can be observed for energy consumption (Fig. 20 (D)).

Sub-figures 20(B) and 20(E) show the effect of task complexity. Task complexity influences how long it takes for the task to be processed either on the server or the MoTH. As signal strength does not influence the processing itself, differing task complexities do not give our model an advantage if the server has idle load, however, with the visible results, our model still outperforms the baseline case due to the involved 2 megabyte communication.

Similar to Case 1, with higher speeds the mobile host either reaches better signal faster, increasing the performance up until a point where higher speeds cause the best reception area to be left sooner, again worsening performance. In this case, it can be seen that the optimal speed zone for the baseline case is narrower than for the adaptive model.

Case 3: Handling overloaded servers. In the final isolated case, server *s0* is already processing other tasks while the MoTH encounters it, meaning new distributed tasks are placed in a queue. Here, both servers are located equally close along the MoTH-s path (Fig. 18, b).

Fig. 21 shows the impact of different *s0* loads. The load of a server is the sum of task complexities of all tasks currently distributed to the server. If both servers are idle, both approaches opt for the closer server and result in the same task time. In case of 0 load, the energy consumption of the adaptive scheme is slightly higher because it involves communication to obtain the real-time fog server status information.

But as the load of *s0* increases, the baseline approach consumes more time and

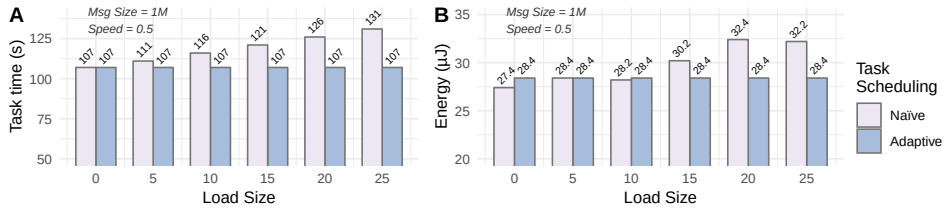


Figure 21: *Case 3* - Distributed task time- and energy consumption for Mobile Thing Host

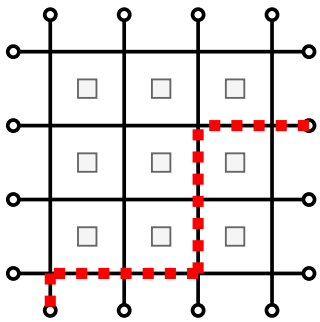


Figure 22: Case 5 Simulation Map

energy due to the added waiting time, while the adaptive model on the other hand chooses to use *s1*, which is idle, instead.

4.2.4. Scenario with Multiple Mobile Thing Hosts

The final examined scenario is more distinct, here, the simulation setting mimics Manhattan-style city streets (Fig. 22), there are 9 fog servers (grey squares) spread out on the map and 50 MoTHs (not depicted on the figure). The MoTHs randomly take trips from one of the marked points to the other, the red dotted line shows an example path. Each host start a new process instance per trip.

The idea is to see how the choices made by multiple MoTHs using our adaptive scheme influence the distribution of tasks among servers and performance for MoTHs. The simulation is run for 10 hours (simulation time), Table 4 lists the detailed configurations used in this simulation case.

Fig. 23 shows the mean success rate of tasks distributed by the MoTHS. In other words, how many distributed tasks were successfully finished (failure can occur due to the server being overloaded and the connection dropping before the server finishes processing the task). Recall that a QoS level of 0.0 means that all mobiles delegate the task to the first encountered server. As can be seen on the figure, the baseline approach can successfully finish less than 50% of distributed tasks. On the other hand, with our model, task completion improves significantly. With a setting of QoS=2.5, task completion reaches above 70%. Generally, raising the QoS constricts the decision-making to only include servers which are closer

Parameter	Value
Simulation time	10 hours
No. of MoTH	50
No. of Fog Servers	9
Median processes started per MoTH	40
λ	1.5
MoTH speed	0.6 (m/s)
Task Complexity	100 units

Table 4: Case 5 Simulation parameters

and more idle. As a result, setting too high values (e.g. with QoS=3.5 on Fig. 23) lead to under-utilized fog servers (in other words, MoTHs are looking for completely idle servers even though the server may be able to finish the task in time with a small load).

Secondly, the distribution of tasks among all the servers is shown in Fig. 25. The more uniform the distribution of tasks among servers, the higher the success rates for the participants, as overloaded servers are avoided. With QoS value of 0.0, the variation in balancing the load across servers is large, some servers are overloaded while others are utilized much less.

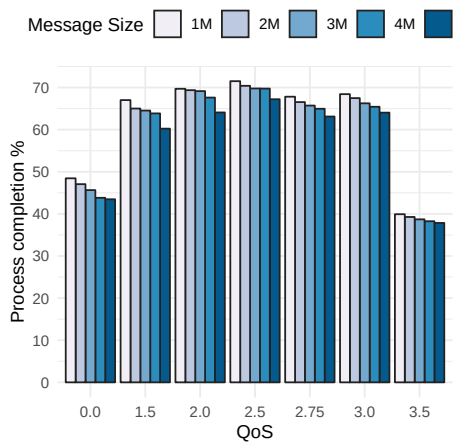


Figure 23: Mean Distributed Task completion for MoTHs

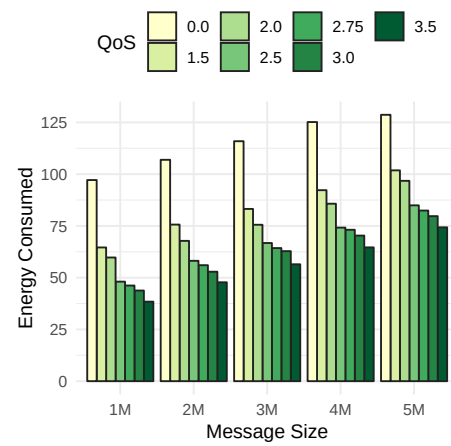


Figure 24: Case 5 Energy usage

Using our model, we see that the variation becomes smaller and more uniform up until a certain point. In this scenario, QoS of 2.5 showed best results, as higher values again constrained the decision-making too much and some servers were left under-utilized. This is especially evident with QoS 3.5, where some servers started 0 tasks, as was the case with QoS 0.0.

Fig. 25 also shows how setting different *lambda* values affect the performance. Recall that lambda affects how much the server load-related contextual data is

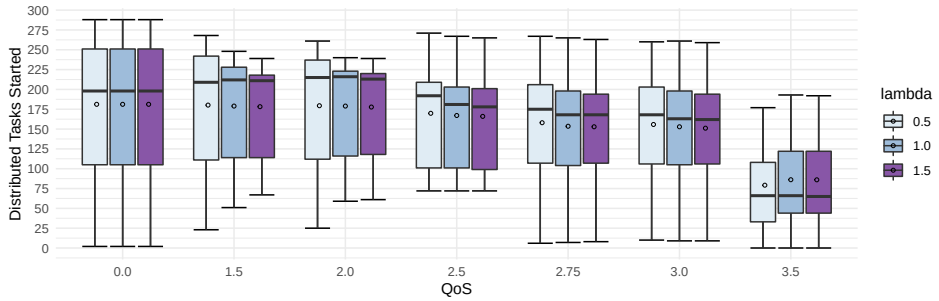


Figure 25: Distribution of tasks among servers with message size = 5M

taken into account. In this case, similar to QoS, raising the value has a constricting effect.

The energy consumption of MoTHs with different message sizes are shown in Fig. 24. We see the direct correlation between increased QoS and saved energy, additionally, the adaptive approach can outperform the baseline even when the former uses larger message sizes than the latter.

4.2.5. Discussion

Based on the previous results, we saw that using the adaptive scheme is beneficial in terms of energy consumption, task execution time, but also for balancing the load among multiple hosts and servers. The improvement is especially large when incorrect decision-making causes the task to fail and to be re-tried locally, which effectively doubles the processing time and energy consumption.

While the scenario in section 4.2.4 showed promising results for applying the model in real-world scenarios, it is important to note that choosing the appropriate QoS threshold and γ value affect the model's performance. The optimal values for these parameters may depend on the use case or user preference. For instance, setting very high QoS values may be imperative for more latency-sensitive scenarios or if the user prioritizes saving battery life over small delays.

Our adaptive scheme fits workflows and scenarios where some delay-tolerance is acceptable. But it is important to note that in some scenarios, choosing to distribute the task to a server which is encountered later may not be an option if the task is time-critical. In these scenarios, our model can revert to the local processing option based on the set time-out (as was shown in Fig. 17).

4.3. Summary

In this contribution, we presented an Edge Process Management system architecture which involves mobile devices that distribute certain tasks in their processes to nearby fog servers. Our system included a mobility- and context-aware mechanism for scheduling these distributed tasks to servers, based on the user's movement trajectory, server load, fog server location and configuration. The mecha-

nism includes QoS threshold parameters, which allow to constrain the decision-making to guarantee higher chance of success or fall back to local processing earlier, if it is estimated that no fog server can satisfy the constraint. Experiments with a prototype implementation of the system, conducted in a simulated testbed, showed how this adaptive scheme can avoid costly local task processing due to dropped connections and choose servers for efficient task offloading. The results showed how our approach raises success rate of distributed tasks from 48% to 71% and consumes up to 4 times less energy than greedy opportunistic approaches.

As part of future work, it is important to pursue more detailed modelling of wireless radio, taking into account physical factors such as radio and obstacle interference. To transform the simulation-based approach into a real-life deployment, introducing a trajectory prediction component may also necessary if the user is not willing to specify their destination and trajectory explicitly (e.g. when using a navigation software). The adaptive schedule can be fine-tuned with the per-parameter weights, however a means to easily infer the weight values for a given application's requirements is still needed.

Our decison model is very much focused on the bottom-up perspective of fog applications, the mobile device context and the currently proximal Fog server's information are the primary source of contextual data, with some limited info about the rest of the fog network. However there exists a lot of research on managing the fog network, with both centralised and decentralised architectures, exploring how fog servers can collaborate, perform load-balancing via replication or migration of application services [MRB21]. For example in [Bro+21], the fog network autonomously tries to balance resources based on decentralised collaboration between fog servers. Thus as future work, it is worth investigating how the MoTH could take into information not only about the currently proximal Fog server, but also it's neighbours, their scheduling, load-balancing policies. Outside of disaster scenarios, Cloud resources may still be available, and then the decision-making should include the trade-off between offloading ot the cloud or fog as for example in [AC21].

Finally, another important direction is to explore additional scenarios from domains such as smart transportation or smart city, using real-world city maps and bus route data in the simulation setting and how this information could be used in the decision-making. In the next chapter we present an advanced version of the testbed used in this chapter, which has been supplemented with modelling tools and more EPM-related simulation features, designed to be a suitable general-purpose toolset for such studies.

5. SIMULATING MOBILITY-ORIENTED EDGE-FOG PROCESSES

In the state of the art, adopting BPMS for execution on end-devices such as mobile phones has been researched several times [Hac+06; Dar+15; Sch+16], where a standards-compliant process engine is embedded on the device. However, while considering that IoT and edge systems are to involve huge numbers of devices, the evaluation of existing related works is typically done at a scale of up to a few devices (e.g. 2 moving robots and 1 stationary system in [SHA17]). Indeed, large-scale real-world experiments are costly to perform, so the common approach is to simulate the behaviour [Ded+18]. However, to the best of our knowledge, no existing simulation tool has adopted a process-oriented and mobility-aware perspective to IoT systems, this has motivated our 3rd research question and research goal.

To support development and research of process-oriented scenarios and models by simulating EPM environments where mobility aspects are common (such as smart cities), we propose **STEP-ONE**: a **S**imulated **T**estbed for **E**dge-Fog **P**rocesses based on the **O**pportunistic **N**etwork **E**nvironment Simulator. STEP-ONE is an extension to the Opportunistic Network Environment simulator (ONE) by Keränen et al. [KOK09], adding business process execution support on the hosts. This allows discrete event simulation of scenarios where the application logic is defined as BPMN 2.0 processes. STEP-ONE supports defining these processes with a set of ready-to-use process components, such as Message tasks for inter-process messaging across hosts, events and signals for time, mobility and connectivity-related situations and configuration options for varying the executed processes, their inputs and the process engine parameters when executing a batch of simulations. The testbed provides feedback in the ONE graphical user interface during execution and a visual modeller for creating processes. For results interpretation, we make use of the report generation features available in ONE to generate process execution traces. The process execution and visual modelling tools in STEP-ONE are based on the Flowable BPMS . STEP-ONE includes features modelling of computational tasks and computational capacity of the hosts in the simulation, to support studies of fog- and edge computing scenarios, where task delegation, migration and offloading play an important role.

We demonstrate STEP-ONE with a hypothetical Smart City scenario set in the city of Tartu, where the city uses an *edge process*-based approach to monitor the street and road conditions in a de-centralised fashion. This testbed enables enacting scenarios with large number of hosts in city map-based environments, thus allowing to explore the question of mobility, resource scheduling and placement effect on processes in the edge and fog.

5.1. STEP-ONE Software Requirements & Design

In the 2nd chapter, section 2.6.5, we gave an overview of different simulators such as NS-3, ONE, SUMO. Considering our review of these simulators, for this thesis, ONE emerged as the suitable selection for EPM. On one hand, its less-detailed lower-level networking favours rapid prototyping considering that finely detailed simulations may obscure effects of fundamental changes in algorithms, as a lot of attention goes to managing simulation parameters, in addition to the algorithms themselves [CG03].

Secondly, we considered the programming languages. Several industry-recognized process management software suites such as jBPM, Camunda, Flowable and Bizagi BPM are based on Java, as is ONE. This makes integrating initial results from the testbed into the real systems (and vice versa) easier than it would in the case of NS-3 or OMNeT++.

Based on its mobility, networking and communication features (see section 2.6.5), we have based the core of the testbed on the ONE simulator. Fig. 26 highlights how our contribution expands upon ONE while highlighting how our contribution is positioned with regard to the focuses of the other related simulation tools.

Before going into the details of how STEP-ONE expands on ONE in order to realize process-based scenarios, we first describe ONE's existing main concepts. Then we define requirements for STEP-ONE and proceed with a detailed explanation of how the tool fulfils the requirements.

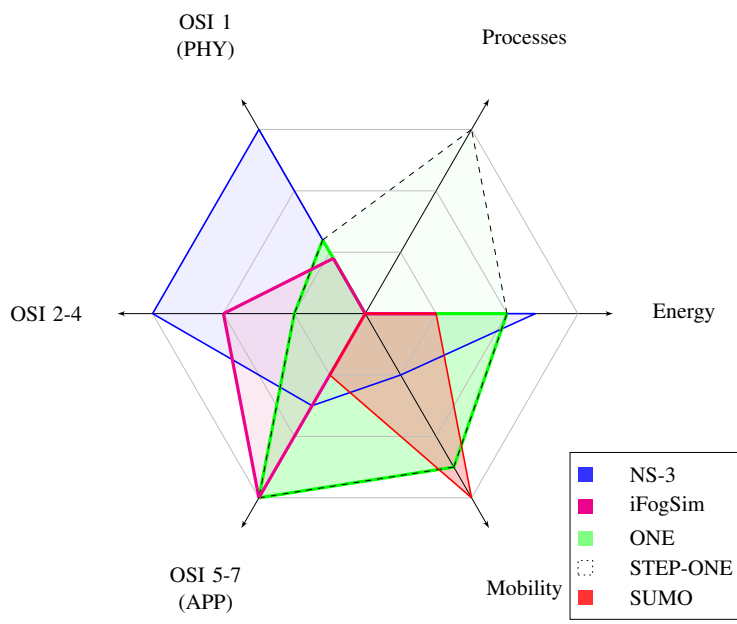


Figure 26: Comparison of STEP-ONE with related simulators in the level of detail addressing OSI network layers, mobility, energy and processes modeling

5.1.1. Existing ONE simulator features

Simulations in ONE are defined by configuration files that describe the hosts (nodes) involved in the simulation, their behaviour and capabilities in terms of communication interfaces, (delay-tolerant) routing mechanisms and movement. Given a configuration file, ONE simulations can be run: **a)** using a graphical user interface (GUI), which displays the movement of the nodes in the simulated world, connectivity and messaging events or **b)** in batch-mode, without a GUI, which is useful for running permutations of simulation configurations at higher performance. The simulation results are captured in report files, which can be selectively enabled. These reporting modules produce summaries of messaging statistics, e.g. delivery ratios, performance (latency), movement and connectivity information, such as contact times.

Hosts in ONE can run ***applications***: custom programmable modules, that handle incoming messages, create new messages, or run custom logic at every discrete time step. The implementation of applications in ONE is left to the developer. In addition to applications, messages can be produced from an external event generator.

5.1.2. Requirements for STEP-ONE

The above functions provide a foundation for the testbed, however, to achieve process management support, we define the following requirements. The testbed should:

- **REQ1:** support executing BPMN 2.0-defined processes on simulated hosts.
- **REQ2:** allow executed processes to interact with the simulated world and other simulated processes through messages, events and signals.
- **REQ3:** provide means to specify the process-related parameters for the simulation from a configuration file.
- **REQ4:** provide means to create process definitions and related messages using a visual editor.
- **REQ5:** allow specifying the computational complexity of certain process sub-tasks and specify the computational capabilities of host groups.
- **REQ6:** support generating reports based on the process execution-related logs.
- **REQ7:** be extendable with decision mechanisms and algorithms to study their effects on the scenario and process execution(s).

5.1.3. Process Engine Application

To address REQ1, we provide a specific ONE *application* implementation - a Process Engine Application (PEA) (see Fig. 27). During simulation runtime, the PEA handles deployment of BPMN 2.0 process definitions or starting execution of individual instances of deployed processes through respective messages from

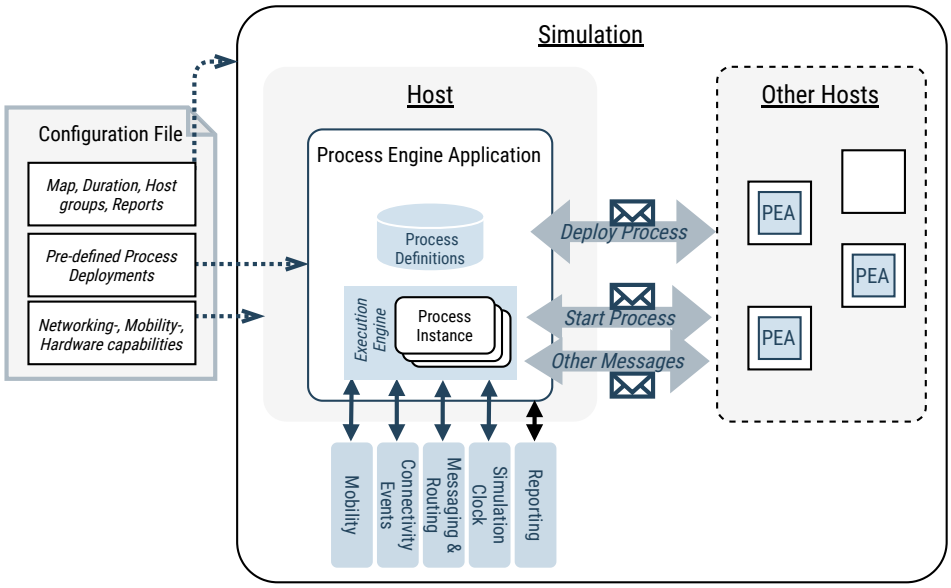


Figure 27: Overview of STEP-ONE architectural Components

other hosts (their processes). Alternatively, these actions can also be performed at initialization of the simulation based on the configuration file. The PEA enables various functionality for executing process instances based on REQ2, for example: connectivity events within ONE are mapped to events which processes can catch and *Message tasks* defined within a process will create a ONE message to be sent to other hosts.

Messaging. Messaging is the core functionality of both ONE and STEP-ONE. STEP-ONE processes exchange *Process Messages*, that are encapsulated within standard ONE messages. This allows the message routing to be handled by ONE networking capabilities.

ONE messages are defined by an ID, size, recipient host address and custom extension properties (key-value pairs). STEP-ONE uses these extensions to attach extra data which define a Process Message, for example: attaching process variables (optional) or *name* of the message (mandatory), processes which are interested in receiving messages have to subscribe to them by name.

STEP-ONE distinguishes four types of Process Messages, all of which can be sent from or received within a process execution, except "ONE Application Message", which can only be sent from processes.

- **Generic Process Message** - A generic message which originates from a "Send Message" task within a process. The receiving process engine forwards the message to the process instance which is subscribed to the message (if any).
- **Deploy Process Message** - when a process engine receives a "Deploy process" message, it attempts to deploy a process definition based on the re-

source file path attached to the message.

- **Start Process Message** - when a process engine receives a "Start process" message, it attempts to start a new process instance, given the identifying process key contained in the message. This assumes that a process definition with a start event corresponding to the message name has been deployed.
- **ONE Application Message** - this message can be sent from a process and is targeted at other ONE applications (that are not PEA-s). For example, invoking an atomic service from a central process engine would be done with this type of message.

Section 5.2.1 discusses the implementation details how these messages can be defined within a Process and how the process engine maps them to ONE messages and vice versa further.

Events, Signals. The process engine also handles events from the simulation world and notifies running process instances which are subscribed to related events. This is particularly useful for driving the process execution based on the events - waiting for a connectivity-related event before proceeding with a message sending task or interrupting a flow in reaction to an event. In the following, we list the events STEP-ONE processes can handle.

- **Connectivity Events** - When a network connection is established or broken between hosts, the events are translated into BPMN signals.
- **Timer Events** - Processes which contain timer-based events (e.g. timeouts or cyclic repeating behaviour) are notified of time elapsing based on the simulation clock.
- **Location Events** - When a moving host arrives at a location to which a running process instance is subscribed (based on coordinates), it is broadcast to the process engine as a BPMN signal.
- **Message Events** - This allows processes to wait for a given message to arrive before proceeding with the control flow.

Tasks & Activities. Processes are sequences of tasks. STEP-ONE provides a set of useful BPMN tasks which help quickly define simulated scenarios. STEP-ONE provided tasks are listed as follows:

- **Send Message Tasks** - for creating each of the above message types, a *Send Task* subvariant exists.
- **Simulated Work Tasks** - to simulate long-running, compute-intensive tasks (See REQ5 in 5.1.2), this task type is defined by a work size parameter, how large the task is in terms of millions of instructions (MI).
- **Generic Service Tasks** - Service tasks are generic tasks, whose implementation is up to the developer. This allows creating experiment-specific functionality, programmed in Java. For instance, a scheduling algorithm may be implemented as a Service task, which based on some process variables,

produces a scheduling decision and writes it into another variable, used in the rest of the process flow.

In general, the duration of tasks in simulation clock time is instantaneous, except for Simulated Work Tasks, whose duration depends on the node's hardware configuration and Send Message tasks configured not to finish execution before the message has been received at the destination.

5.1.4. Discussion

Since in addition to the above-mentioned STEP-ONE messages, tasks and events, BPMN 2.0 allows defining various control flow structures such as AND or (X)OR gateways, and custom logic can be attached through Service Tasks, execution listeners, etc., complex behaviour can be modelled and enacted.

As we will show in section 5.3, one class of applications to model as processes are Fog computing scenarios, where ONE-s networking abilities and STEP-ONE's messaging, simulated work tasks will be used.

5.2. Implementation

STEP-ONE is packaged as a Gradle-based Java project and is available as a public GitHub repository[1], with additional documentation and reference materials on how to use it.

The Process Engine Application embeds an instance of Flowable BPMS (version 6.4.2), which is a light-weight process software that supports a particular extension of the BPMN 2.0 standard - namely some Flowable-specific extension elements have to be included in the process model to enable automated execution.

To improve simulation performance, STEP-ONE uses Flowable with an in-memory database and features such as process history tracing can be turned off.

Implementation of the tasks mentioned in section 5.1.3 is based on the *Service Task* BPMN construct. Custom Service Tasks in Flowable must implement the `JavaDelegate` interface, which defines an `execute()` method called whenever the process execution flow reaches that task (Listing 5.2 shows an example). When describing a Service Task within a BPMN 2.0 process definition, an XML attribute specifies the Java class which implements the `JavaDelegate` interface. This was the basis of the Java implementations for the various task types.

5.2.1. Messaging implementation

STEP-ONE provides an implementation for each of the mentioned Message Tasks. When executed, the Java code constructs a ONE message and embeds a Process Message into it. After constructing the message object, it is added to an "outgoing messages" queue. The PEA queries the queue on each simulation update, and

[1] `https://github.com/jaks6/step-one`

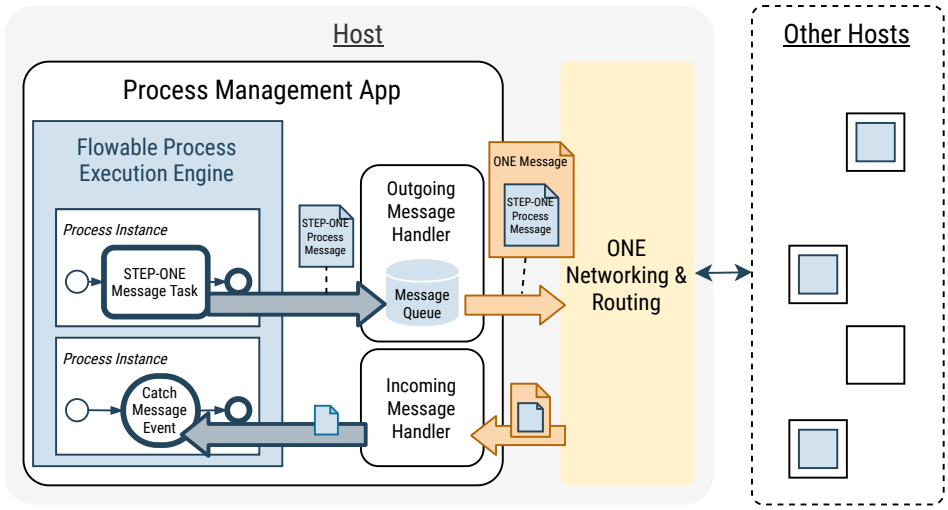


Figure 28: Overview of how Process Messages are mapped to ONE messages and vice versa.

hands them over to networking and routing modules of the ONE, which handle the message transfer to other hosts (Fig. 28).

When the target PEA receives a ONE message, the contents are parsed and if the message contains a STEP-ONE Process Message, the appropriate Flowable API (e.g. deployment, process start, message received) is invoked with the message contents and attachments.

Message Parameters. When defining a STEP-ONE Task, in addition to selecting the appropriate Java implementation, a number of task-specific parameters must be specified as BPMN *extensionElements*, more precisely Flowable's *extension fields*, which are parsed by the Java implementation.

The compulsory parameters for all types of message tasks are: Message Name, Message Size, and Destination Address.

Additional parameters vary by message task type. Both the compulsory and additional values are part of the Process Message object that is encapsulated in a ONE Message. The full list of parameters is shown in Table 5. These parameters can either be defined with static values, or alternatively using the *dynamically evaluated expressions* feature of Flowable. This allows for example inferring a message recipient address from a process variable during process runtime, instead of hardcoding it during design-time. Listing 5.1 shows an example of a BPMN 2.0-defined message task using the extension element parameters - the name and message size are defined with fixed values, while the destination address is a dynamic expression.

Parameter	Applies to	Req.	Description
Name	All Messages	+	String. Used to forward the message to all process instances that have subscribed to this name.
Size	All Messages	+	Bytes
Destination	All Messages	+	ONE host address
Target Execution ID	Generic Messages	-	Process Instance Execution ID. Allows the message to be forwarded to the exact corresponding process instance.
Included Process Variables	Generic Message, Start Process Message	-	Comma separated list of values attached to the message. The variables become available to the destination process instance
Resource Path	Deploy Process Message	+	Location of BPMN 2.0 process definition file (on simulation host machine classpath).
AppID	ONE App Message	+	ONE Application identifier - which ONE application this message targets

Table 5: BPMN 2.0 Message Task parameters in STEP-ONE. "Req." - Required

```
<serviceTask id="exampleMessageTask" name="Example Message"
    flowable:class="ee.mass.epm.sim.task.MessageTask"
    flowable:triggerable="true" flowable:type="stepone_msg">
  <extensionElements>
    <flowable:field name="msg_name">
      <flowable:string><![CDATA[Hello World]]></
    flowable:string>
    </flowable:field>
    <flowable:field name="msg_size">
      <flowable:string><![CDATA[1]]></flowable:string>
    </flowable:field>
    <flowable:field name="msg_destination">
      <!-- Using Dynamic expression evaluation -->
      <flowable:expression><![CDATA[${myDestinationVariable}]]
    ></flowable:expression>
    </flowable:field>
  </extensionElements>
</serviceTask>
```

Listing 5.1: Generic Process Message Task example

5.2.2. Events & Signals implementation

Movement- and connectivity-related events are handled via Flowable/BPMN *signals*. Like messages, signals have names and processes can subscribe to them based on their names. Further, signals can have process variables attached to

Name	Description	Defining in a process
Timer Event	Notifies host of time passing. Defined with ISO-8601, e.g. PT45S represents a time duration of 45 seconds	Define the Timer Event and specify the duration
New Connection Signal	When a connection is established, both hosts receive signal with name *Device Connected* and the other device's address attached as a process variable	Define a signal catching event for respective signal name
Disconnected Signal	When a connection is lost, both hosts receive signal with name *Disconnected* and the other device's address attached as a process variable	Define a signal catching event for respective signal name
Location Signal	When host position reaches a given coordinate, the host process engine receives the signal with name *New Coordinate*.	Define a signal catching event with extensionElement named "coordinate" giving the coordinate pair as value and specify a location execution listener[2].

Table 6: Event & signal constructs in STEP-ONE

them.

Time-based events on the other hand are handled with a modification to Flowable's time/calendar APIs that use ONE-s simulated time instead of the real world time. Time-based events are defined with an extension element, which specifies the value and whether it is a repeating time period, a single duration.

Table 6 lists the exact signal names, attached process variables and expected time format for Timer events.

5.2.3. Simulated Work Task & Cost Management

STEP-ONE provides functions for modelling of computational work for Simulated Work tasks mentioned in 5.1.3.

A host's computing capability is defined by: number of CPU cores: $NoOfCpus$, and speed per CPU: $CpuSpeed$ (in DMIPS, Dhrystone Million Instructions Per Seconds - a commonly used metric for performance evaluation [LKC17]). All CPU jobs, as defined by Simulated Work Tasks, are placed in a registered job queue. On every simulation update, the PEA manages the execution of job units in the queue using the algorithm shown in Alg. 1. While a single job can only be run by a single CPU at once, multiple jobs can be processed in parallel by different CPUs and a single CPU can work on several jobs sequentially within a single

[2]ee.mass.epm.sim.LocationSignalExecutionListener

simulation update iteration if the job sizes are sufficiently small.

```
On every simulation update: for cpu_i ← 1 to NoOfCpus do
    instructionsLeft ← CpuSpeed ;
    while instructionsLeft > 0 and not ( empty(activeJobs[cpu_i]) and
     empty(registeredJobs)) do
        if empty(activeJobs[cpu_i]) then
            activeJobs[cpu_i].add(registeredJobs.dequeue());
        end
        currentJob ← activeJobs[cpu_i].peek();
        currentJob.workLeft −= instructionsLeft ;
        if currentJob.workLeft ≤ 0 then
            activeJobs[cpu_i].dequeue() ;
            processEngine.notifyJobFinished(currentJob));
            instructionsLeft −= currentJob.workLeft ;
        else
            instructionLeft = 0;
        end
    end
end
```

Algorithm 1: Processor routine on every simulation update

Cost Estimation Helper. It is also possible to define cost parameters to a node's computational and networking resources. For compute resources, cost is defined per CPU usage at a granularity of either hours or seconds, (e.g. $0.02 per hour). For networking, cost is defined for each interface as cost of transmitting 1MB. If these values are specified, it is possible to use the *CostHelper* class, which provides methods:

1. getCostForTransmission(x, y) - returns the cost of transmitting *x* bytes over network interface *y*.
2. getCostForTask(z, w) - returns the cost of processing a task with size *z* (in MI) on host *w*. The time of processing the given task is rounded up to the specified cost granularity level.

The above methods can be used in Applications or Process Service tasks for making decisions based on costs and budgets. For instance, if a process involves deploying and starting another process including Simulated Work tasks to some external host, the *CostHelper* can be used to first perform some calculations to determine the most cost-efficient qualifying node. Code listing 5.2 shows an example usage of the *CostHelper* as part of a custom *Service Task*.

```
class CostAwareTask implements JavaDelegate {
  @Override
  public void execute(DelegateExecution execution) {
    DTNHost cloud;
    int taskWorkSize;
    int taskBandwidth;

    // Assign values to above, e.g. from process variables,
    // process field expressions or from simulation instance
    ...
    double cpuCpost = CostHelper.getCostForTask(taskWorkSize, cloud);
    double networkCost = CostHelper.getCostForTransmission(
    taskBandwidth, cloud.getInterface(1));

    // Make decision based on cost - e.g. whether to invoke cloud or
    not,
    // set process variables accordingly, etc.
    ...
    }
}
```

Listing 5.2: Java example of a Service Task implementation using the Cost-related functions in STEP-ONE

5.2.4. Other features

Full Duplex network Interface. Network interfaces provided by ONE simulator operate in half-duplex mode and the message routing is restricted to sending 1 message from a single host concurrently. This becomes an issue if a single host is communicating large messages with several hosts simultaneously, e.g. a cloudlet servicing multiple nearby MPHs. Thus, we implemented a full-duplex network interface `SharedBandwidthInterface`, which considers network speed of the upload and download links separately, and divides the bandwidth between all active transmissions. `SharedBandwidthInterface` is to be used in combination with STEP-ONE-s `DirectMultiTransferRouter`, a message router that enables transferring multiple messages at a time.

Reporting Tools & GUI additions. To complement the existing set of reporting facilities in ONE, we have added two additional modules: *BpmAppReporter* and *DetailedBpmReporter*.

The former gives aggregate statistics of processes run on all hosts during the simulation, such as total number of processes started/completed/cancelled, number of Process Messages sent/received, number of Process Activities started, completed, cancelled, and so forth. DetailedBpmReporter, on the other hand, records data about every individual executed process instance: the start and end timestamp of the process, which host executed it and various aggregate counts such as messages received/sent, activities started/completed as part of that process.

Alongside the reports, another way of getting feedback on process executions in STEP-ONE is using the GUI. We have modified ONE-s GUI, which provides routing & message-related details, to also show process states during runtime (see

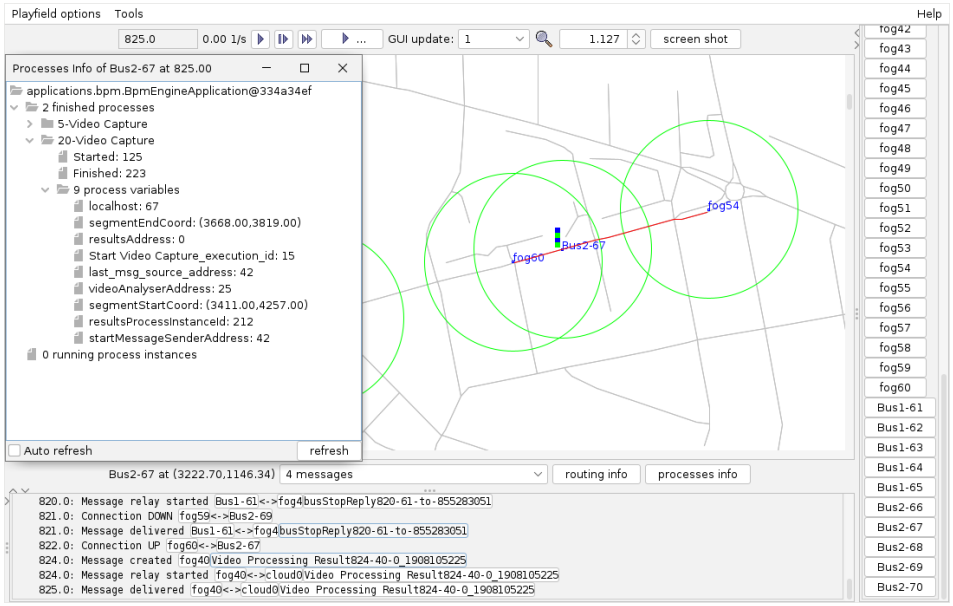


Figure 29: ONE Simulator with the Process Info Extension of STEP-ONE visible.

Fig. 29). The GUI panel captures running and finished process instances of a host, their process variables, process start time and currently executing sub-task (in case of running processes).

Configuration. STEP-ONE uses ONE-s configuration file approach to enable setting the parameters of its modules. STEP-ONE introduces the following new configuration options:

- The number of CPUs and CPU speed of hosts
- Compute and networking costs of hosts
- The Upload and Download speed of `SharedBandwidthInterface`
- Process definitions to be auto-deployed at beginning of simulation
- Processes that should be auto-started at beginning of simulation, including process variables that may be attached when starting them.
- Some of the features, such has generating signals for connectivity and movement events can be selectively disabled, as they have some effect on performance

An example of these configuration options is shown in Listing 5.3.

```
# Configuring the Process Engine Application
fogProcessEngine.type = bpm.BpmEngineApplication
fogProcessEngine.generateMovementSignals = false
fogProcessEngine.generateConnectionSignals = true

# Processes to run at simulation start:
fogProcessEngine.autoDeployedProcesses = ./bpmn/Road_Analysis_Process
    .bpmn20.xml
fogProcessEngine.autoStartedProcessKeys=roadanalysisprocessV2
fogProcessEngine.autoStartedProcessVars=segmentStartCoord
    =(12.00,23.00)&segmentEndCoord=(22.00,23.00)

# Network interface
wlanInterface.type = SharedBandwidthInterface
wlanInterface.downloadSpeed = 6750k
wlanInterface.uploadSpeed = 6750k
wlanInterface.transmitRange = 45

# Assign a group of hosts to use the above engine and interface
Group1.application1 = fogProcessEngine
Group1.interface1 = wlanInterface

# Setting compute capacity for the process engine
fogProcessEngine.noOfCpus = 4
fogProcessEngine.cpuSpeed = 31500

# Specify costs for compute and networking
fogProcessEngine.cpuCostGranularity = hour
fogProcessEngine.cpuCostPerTimeUnit = 0.017
fogProcessEngine.networkInterfacesCostsPerMb = 0.015
```

Listing 5.3: Example excerpt of a STEP-ONE configuration file

5.2.5. Modelling Processes

Manually writing the XML of BPMN 2.0 models including the STEP-ONE- or Flowable-specific extension elements and classpaths of STEP-ONE task implementations can be tedious. The easiest way to attain a compatible process model is to use the Flowable modeller, which is a web-based application for visually defining the process - its tasks and their parameters. Hence, STEP-ONE includes an extended version of the Flowable modeller, which adds the main STEP-ONE task types to the modellers palette and allows configuring their parameters in a fashion similar to the overall experience of Flowable modeller.

With the STEP-ONE modeller, the extension pre-fills the Java implementation values for the STEP-ONE tasks, while custom task behaviour can be defined by specifying execution listeners or specifying the Java class implementation.

Fig. 30 shows a screenshot of a STEP-ONE model designed with the modeller. As depicted on the bottom of the image, parameters such as the message name, size can be defined in text fields, note that these fields can also be specified as dynamic variables, e.g. the message Destination Address on the figure has been defined as the value of variable `videoAnalyserAddress`.

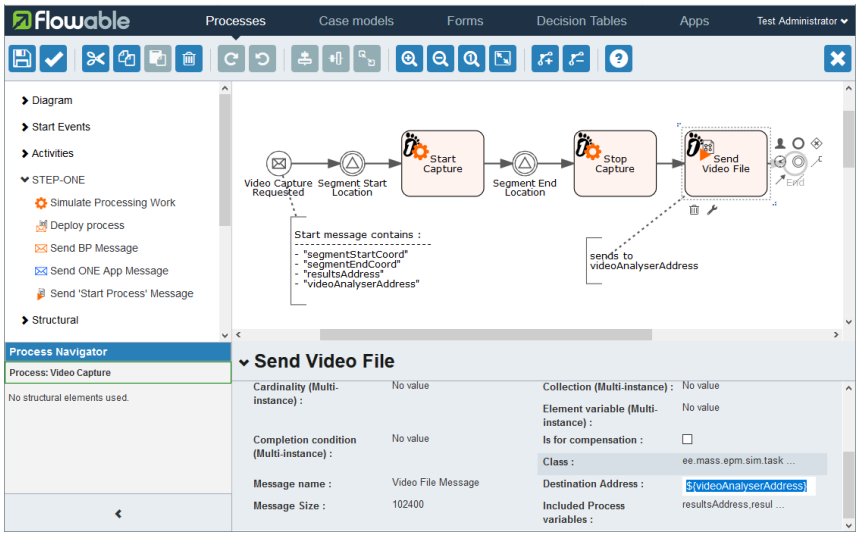


Figure 30: STEP-ONE version of Flowable Modeller

5.3. Case Study

We illustrate the usage of STEP-ONE with a hypothetical Smart City scenario:

The city of Tartu performs monitoring of street and road conditions along public transport routes using image processing techniques. Video footage captured by camera-equipped buses is analysed for objects such as potholes, cracks, snow and ice build-up using image processing techniques [ZNF17]. The analysis is done individually per road segment, each segment starting and ending at a bus stop. Based on the location, traffic frequency and weather conditions, the information system schedules analysis requests for individual road segments at different rates. A segment analysis request initiates video capturing from a vehicle whose route corresponds with the requested segment. Once the segment video is captured on the bus, it is forwarded to an external video processing service. The processing results are recorded in the monitoring systems database and are used to plan timely maintenance, repair, snow clearing etc. operations.

We show how this scenario can be realized as a process-based application in STEP-ONE as: 1) a 2-layer cloud-centric design and 2) a 3-layer fog computing design. In the scope of this case study, our process begins when a new request for analysing a road segment is received and ends when the video processing results are produced, we replace the dynamic scheduling of road segment analysis requests with some fixed request rates and leave out the management of the maintenance activities that occur based on the results.

The high-level process of a single segment analysis is shown on Fig. 31. It can be seen as a composite application with the following sub-components:

- Choosing a vehicle to perform video capture
- Capturing the video

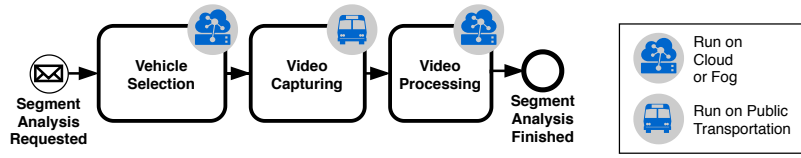


Figure 31: Single Road Segment Analysis Process Overview

- Performing video analysis

The subparts may run on different types of devices (bus, Fog server, cloud server). In the cloud approach, vehicle selection and video processing are run in the datacenter. However, in the Fog design, bus stops in the city are equipped with Fog servers. Vehicle selection is done by a Fog server located at the beginning of the road segment, while the Fog node at the segment end performs the video processing. In this case, the full raw video file is processed at the Fog, saving the bandwidth of transferring it from the bus to the cloud and also shifting more computational load to the Fog.

5.3.1. Implementing scenarios in STEP-ONE

The outline of steps needed to establish and simulate such scenarios is the following.

1. Establishing the simulation world map - the necessary .wkt format file can be attained by e.g. downloading a map from OpenStreetMap and converting it to `.wkt`. With software such as OpenJUMP, these files can be edited or created manually.
2. Preparing the process definitions - Using the Flowable modeller extension for STEP-ONE, the BPMN 2.0 files can be visually designed.
    - The process defines the tasks, message exchanges, signal & event listeners.
    - If the process includes custom code, these should be created accordingly in Java.
3. Configuring the ONE simulation
    - Network interfaces & routing algorithms on the nodes
    - Movement models of the nodes
    - Assigning applications for nodes, including `bpm.BpmEngineApplication` for STEP-ONE engine
    - Configuring STEP-ONE engine application
        - Auto-deployed processes for hosts (host groups)
        - Auto-started processes for hosts and their process variables
    - Choosing which reports to generate
4. Running the simulation(s)
5. Interpreting the results

- From GUI, including STEP-ONE-s process info pane, which shows running and finished processes, their execution states and variables
- From Generated Reports, including Process Reports

5.3.2. Establishing the simulation world map

For the Tartu Smart City scenario, we extracted a 50 km^2 area `.osm` file from *OpenStreetMap*, from which we extracted vehicle-navigable streets and converted the result to `.wkt` format, ending in a file with 20440 points and 3296 features (polylines). Then, we defined two bus routes using OpenJUMP by marking points on the map which correspond to a bus stop, this produced files with 38 and 51 points (stops) respectively. Finally, we also specified locations for fog servers as points, coinciding with the bus stops. The results are shown on Fig. 32, where the bus routes are shown in green and red, bus stops are marked as squares, and Fog servers are marked with blue circles.

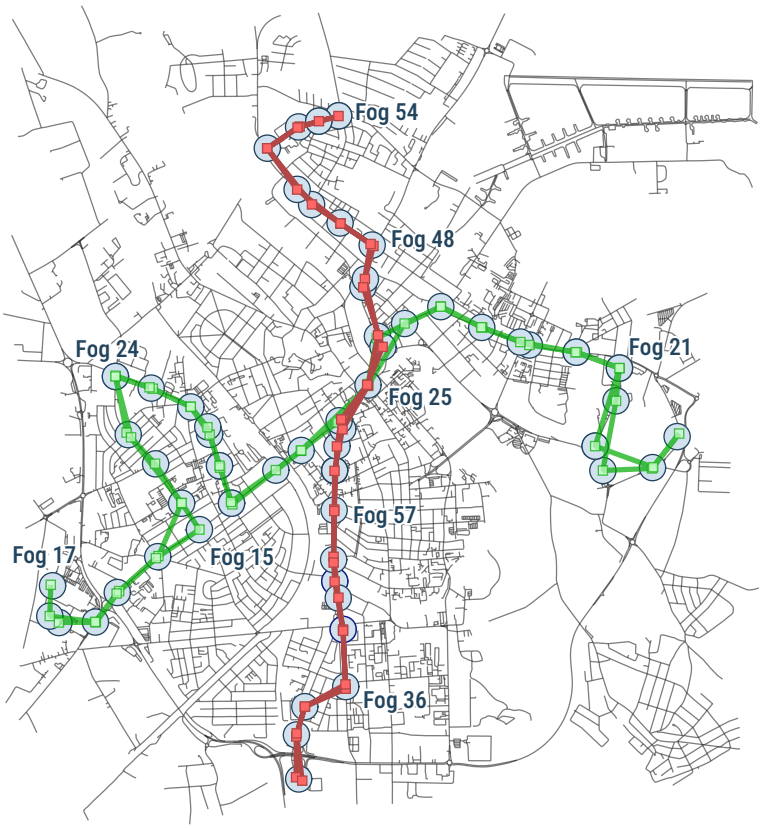


Figure 32: Simulation World Map Preparation: Tartu City streets, two bus lines (shown in green and red), bus stops shown as squares, fog servers shown as blue circles

5.3.3. Describing Processes Definitions

Before we discuss the technical implementation aspects of the processes in this scenario, we first introduce their behaviour - the tasks, messages, signals events and variables used. The visual BPMN process description used in our case study is presented on Fig. 33. It features several BPMN pools, the *Master Process* leads the orchestration and deployment of other processes, such as the ones depicted in the *Vehicle Selection* and *Video Processor* pools.

Master Process. The entire analysis is started by sending a Road Segment Analysis Request message, to which the Start Event of the Master Process is subscribed to. The request message includes an **Analysis Instance Descriptor** (AID), which is a set of process variables: the segment's start and end coordinates, the Master Process executor network address. Using information from these variables, the first task in the process, *Select Placement Strategy*, determines whether the components of the analysis (Vehicle Selector, Video Processing) should be placed on one of the fog nodes or be Cloud. As the output of this task, the process variables `Video Processor Address`, `Vehicle Selector Address` and `Placement Strategy` (Cloud or Fog) are added to the AID, these will be used to initiate resources during later stages of the process.

Next, the deployment and initiation of external processes takes place - first, the Video Processor is deployed to the previously assigned address and secondly, if applicable, the Fog version of the Vehicle Selection process is deployed at the road segment start Fog server. The Cloud variant is assumed to be pre-deployed.

After the external processes have been deployed, the Vehicle Selection Process is initiated through a Start Process Message Task. From the Master Process perspective, what follows are simple receive message events (*Recording Started*, *Recording Finished*), which reflect the progress of the segment analysis. These messages are sent from the external processes which the Master Process has initiated by this stage, described below.

Vehicle Selection Processes. The *Vehicle Selection Process* has the goal of choosing a bus for deploying the "Video Capture Process" and invoking it. The message starting this process instance includes process variables from the AID.

Cloud variant of Vehicle Selection. In the Cloud variant of this process, the bus located closest to the segment start and whose route includes the road segment is selected, based on AID object and queries to the Transport Information System. Then, the process is deployed and started.

Fog variant of Vehicle Selection. The Fog variant of the process catches new connection (WiFi) events, and in each case, queries the other device (bus) what its' next movement destination (stop) is. If the response message indicates that the next stop of the bus matches the requested road segment, the video capture process is deployed and started on that device via messages. Finally, the Video Selection Process sends a notification message to the Master Process about its progress.

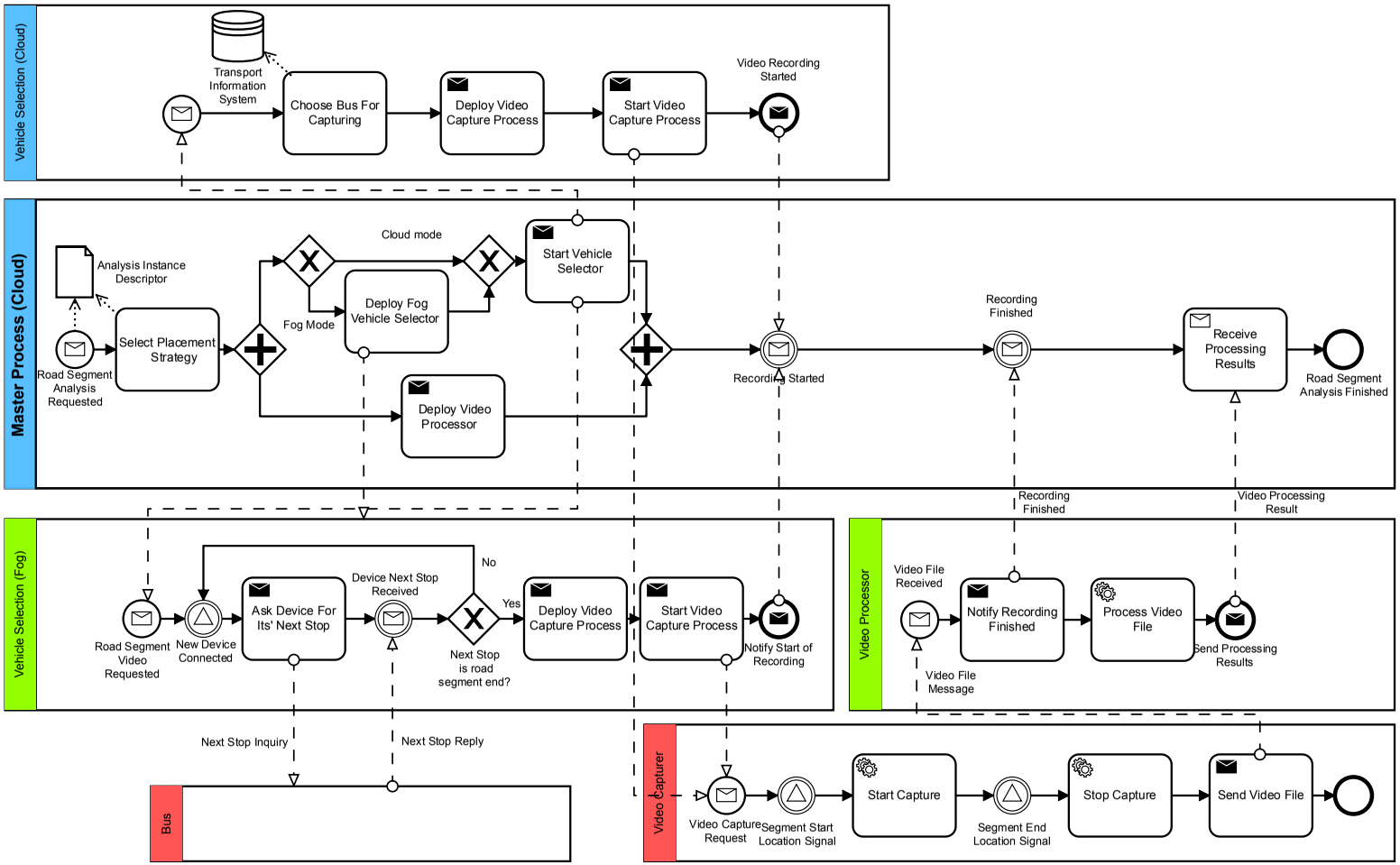


Figure 33: Smart City Road Monitoring Process Collaboration Diagram

Video Capturer Processes. After receiving a start message with the AID, the capturer process awaits for a coordinate-based signal event (*Segment Start Location Signal*) matching the segment start coordinate. Once the signal is received, the Start Capture task initiates video capture on the device. Next, another location-based signal triggers the subsequent "Stop Video Capture" task, after which the recorded file is sent to the Video Processor as indicated by the address from the message which started the process. The Video Processor will be either the Cloud (4G) or the Fog server (WiFi).

Video Processor. This process starts when receiving a message including a road segment video file and AID. Before processing, the Master process is notified that the video has been captured. Then, the resource-intensive video processing Simulated Work task is executed, the results of which are sent to the Master process through the end message event.

5.3.4. Process Implementation in STEP-ONE

The Master Process, two variants of Vehicle Selection Processes, Video Capturer and Video Processor workflows were created as separate BPMN 2.0 processes (Fig. 33) designed with the STEP-ONE Flowable designer.

The Message tasks were implemented as STEP-ONE Process Message tasks, to which the relevant variables of AID were attached. For Deploy message tasks, e.g. "Deploy Video Capture Process", the STEP-ONE Deploy Message task was used, with the attached resource specified by a file path in the XML.

For the Start Capture, Stop Capture, and Process Video File tasks we used the Simulated Work task. The capture-related tasks had a worksize of 500 millions instructions (MI), while the Video Processing Task had a 7000 billion instructions task size.

With the exception of "Video File Message", the message sizes are relatively small, below 1024 kB. In our setup, the Video File Message has been fixed at 100 MB.

For the "New connection" signal catching events in the Fog Vehicle Selection we used the STEP-ONE connection signals, while the Video Capturer Location signals use the Coordinate-based STEP-ONE signals that are based on a custom execution listener *ee.mass.epm.sim.LocationSignalExecutionListener* using the segment end coordinate variable from the AID. Other process parts, such as the gateways and their evaluation expressions or start events were defined following typical process modelling approaches for Flowable.

Custom Code for Tartu Scenario. To realize this scenario, two custom ONE applications supplement the above processes. First, the scheduling of Road Segment Analyses is performed by an application which maintains the set of all road segments based on the world map, and then periodically sends the Segment Analysis Request messages to the Cloud node, initiating the Master process.

The second custom application is the Bus response application, which re-

sponds to "Next Stop Inquiry" messages with a message containing the bus's next stop coordinates.

5.3.5. Configuring Networking and Reporting

The configuration of ONE takes places with the `settings.txt` file, STEP-ONE-specific settings are also to be defined there.

For Fog and Cloud nodes, we used the *SharedBandwidthInterface* network interface for 54Mbps wired network links in the simulation. Additionally, Fog and Bus nodes are equipped with a WLAN interface with a max transmit speed of 54Mbps, while Bus and Cloud nodes are interconnected with an interface representing 4G LTE connection with a 12Mbps speed.

We captured the full details of processes using the DetailedBpmReporter, and some messaging data with ONE-s DeliveredMessagesReport.

5.4. Evaluation

We analysed the behaviour of the presented scenario when varying 2 parameters: 1) how often new analysis requests are scheduled; and 2) how many analyses are scheduled concurrently. Further, we investigated how the processes perform with different process placement strategies: when run on Cloud, Fog, or both interchanged.

5.4.1. Experiment Setup

First, we describe how Road Segment Analysis Requests were scheduled and the parameters which affected the process placement strategy.

Rate of starting processes . A custom ONE application, *RoadmonitoringApp*, maintains the set of all road segments - S, and the set of currently running road segment analyses - $S_{running} \subseteq S$.

At an interval of every i seconds, it randomly chooses a set of new analysis requests to schedule $S_{new} \subseteq S \setminus S_{running}$, $|S_{new}| \leq n$, where n is the no. of requests to start each interval.

For each $s \in S_{new}$, a Master Process instance in the cloud is started, and the set of running analyses is updated, $S_{running} := S_{running} \cup S_{new}$

When a segment request process finishes, it is removed from $S_{running}$.

Placement strategy - fog or cloud . The output of the "Select Placement Strategy" task is a pre-configured as a simulation parameter: $ratio_{cloud}$. The ratio determines what fraction of processes use cloud in the placement. For instance, if the value is 0.0, all processes use Fog placement strategy, if it is 0.5, then every other process uses Fog placement.

This acts as a placeholder for a dynamic resource scheduling algorithm, that could be easily implemented in STEP-ONE. For this case study, such an algorithm implementation is out of scope, instead, we wish to focus on how the testbed

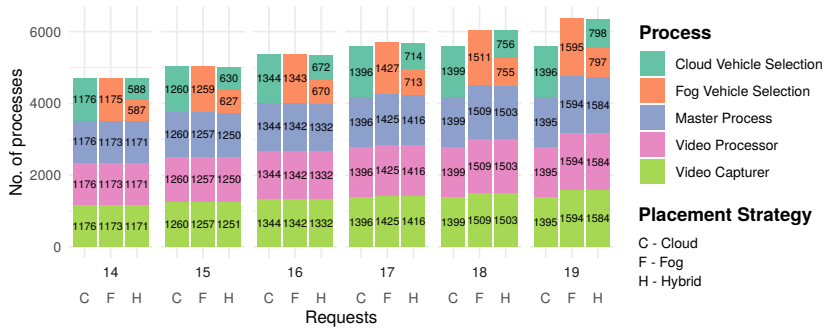


Figure 34: No. of different Processes run across experiments

supports evaluating such algorithms through its general features and refer to the 2nd contribution described in section 4.1.4 as an example of an algorithm, which was also implemented using an early version of STEP-ONE.

In our evaluation, we considered the values shown in Table 7 for $|S|$, i, n, $ratio_{cloud}$, resulting in a total of 15 simulations. In each simulation the no. of hosts is 71: 1 cloud, 60 fog servers, and 10 buses.

Parameter	**Value(s)**
$\|S\|$	87
i	400
n	14..19
$ratio_{cloud}$	0.0, 0.5, 1.0
Fog Computation Conf.	CPUs: 4 (31500 DMIPS)
Cloud Computation Conf.	CPUs: 8 (35500 DMIPS)

Table 7: Evaluation parameters and their values

5.4.2. Case Study

Fig. 34 shows a breakdown of how many processes were run by type, across all hosts. As can be expected based on the process design from Fig. 33, for each Road Analysis Master Process, there is 1 Video Processor, 1 Video Capturer and 1 of either Cloud- or Fog Vehicle Selection process - depending on the process placement decision. For instance, scheduling 15 requests at 400 second intervals started a total of 1250 road analysis processes.

At up to 16 parallel requests, all process placement strategies run the same no. of processes (with a small amount of jitter). At above 16 requests, the no. of Solo Cloud processes becomes smaller than the other cases and peaks at near 1400. We discuss the reason for this below in subsection 5.4.4

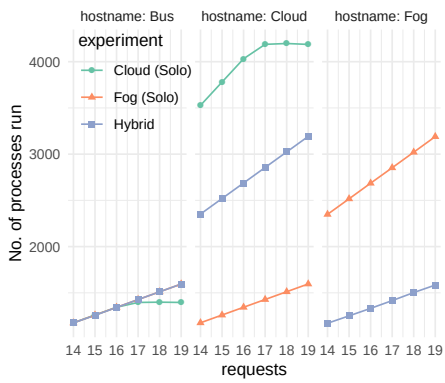


Figure 35: Processes run in different layers

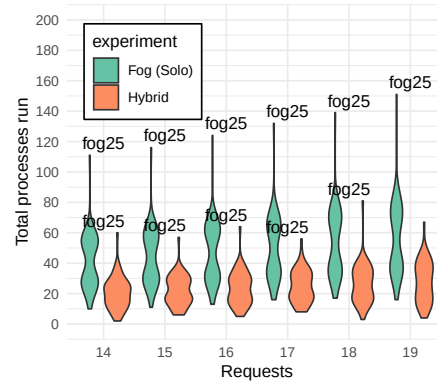


Figure 36: Process distribution among fog hosts

5.4.3. Process allocation

Using the Fog placement strategy reduces the load for Cloud in terms of total number of processes run, as visible in Fig. 35, which shows for each host type the number of processes executed in each experiment. The effect is significant in the "Solo Fog" case, where the load for cloud is reduced by 2/3 compared to Solo Cloud. The hybrid approach does reduce load for cloud, but by 1/3.

Looking at the distribution of processes among fog hosts, Fig. 36, we can note that fog server 25 is handling more processes than others. Server 25 is actually located at the crossing of the two bus-lines so higher utilization of this server is expected (see Fig. 32).

5.4.4. Process Performance - Duration

The issues with the cloud stem from the Video Analysis process. At lower loads, Video Analysis in cloud takes 200s to complete, in all placement configurations. However, for Cloud Solo, already from 17 parallel requests at 400s interval, the cloud cannot complete Video Analysis processes in time before new requests are scheduled, hugely increasing the median duration of the Video Analysis process and by extension the Road Analysis Master Process as well (see Fig. 37).

On Fig. 37, the different behaviours of the cloud and Fog placement strategies can be seen as well - in the Fog case, the Vehicle Selector takes more time to delegate the capturing to a bus compared to the clouds Vehicle Selector. On the other hand, since the bus is already at the segment start once it starts the capture process, the Video Capture process duration is lower for Fog, since in the Cloud case, the Capturer process includes the time spent travelling to the road segment.

When comparing the Solo Fog and Solo Cloud approaches, the Road Analysis Process finishes faster in the fog case, which in our simulation is caused by the network interface differences between Cloud and Fog.

Fig. 38 shows the Empirical Cumulative Distribution Functions (ECDF) of

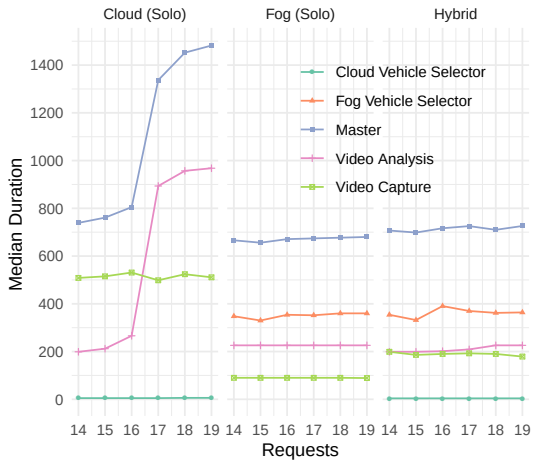


Figure 37: Median Durations of Different Processes

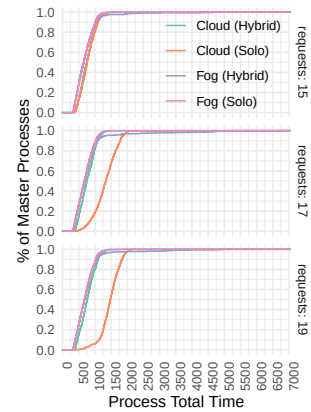


Figure 38: ECDFs of Master Road Analysis Process

Master Process durations in cases where Cloud was not lagging due to a high request rate. Here, the Hybrid case has been further separated depending on whether the analysis was placed in Fog or Cloud.

We see that all processes take more than 200s to complete, and that in both Hybrid or Solo Fog configurations at 15 requests, 90% of processes run on Fog finish under 1000s, while for cloud, this is around 80%. This suggests that the bus-selection algorithm has room for improvement - the slower behaviour of Cloud is explained by the Cloud choosing a bus which is not optimal. At request rates higher than 15, the slow-down of cloud is again visible.

Bandwidth allocation. The impact of moving some of the communication towards the edge networks can be seen from Fig. 39, which shows the message count on the y-axis, and the bandwidth per message type with text. While the total no. of messages handled by the cloud does not reduce drastically when considering the hybrid or solo fog case, the impactful difference comes from the Video File Messages, which make almost the entirety of the bandwidth consumption. The sum of bandwidth taken by Video File messages is 117600MB, and we can see how either half or the entirety of that is shifted to the Fog nodes depending on the process placement. Shifting large file transfers to the local networks has an impact on the overall process speed as well, since we consider Fog servers to have speedier network interfaces than the 4G LTE interface which connects the buses with the cloud.

5.4.5. Discussion

STEP-ONE is a set of tools to define and study EPM scenarios using BPMN 2.0 processes. The core of STEP-ONE is a Flowable process engine embedded into the ONE simulator that has been interfaced with the simulators' environment.

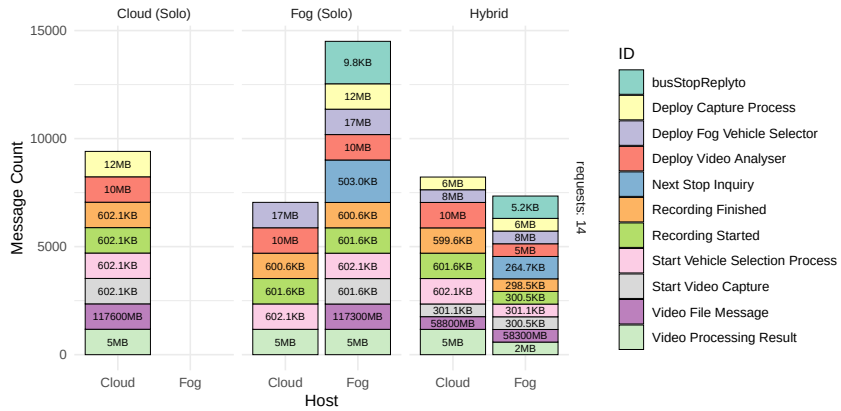


Figure 39: No. of messages and bandwidth transferred by message type

The conducted case study confirmed how shifting both the processing load and bandwidth from the cloud to Fog is feasible. In this study, we saw shortcomings in scalability of a cloud-based solution - notably that at a 400s interval, the Cloud alone cannot support more than 15 parallel requests. Meanwhile, Fog-oriented placement could support 19 parallel requests without slow-downs.

While in these results Cloud showed worse scalability, it's important to note that in our simulation, the network interface of the cloud was slower than the fog server's and that the processing capability of Cloud, while higher than the fog nodes', was still in the same order of magnitude.

The results gave insight into the (over)-utilization of nodes (such as Fog server 25 or the Cloud node), which is useful for planning the geographical placement and hardware needs of nodes.

With these experiments, we wished to show that STEP-ONE can be used for in-depth study on the effect of load-balancing, scheduling etc. algorithms within realistic scenarios. The algorithms could be implemented as process sub-tasks. While Fog yielded promising results, an aspect which we did not consider in our study are the capital- and operational expenses of deploying Fog infrastructure. With some additions, other researchers could explore these directions with STEP-ONE as well, e.g. by using the *CostHelper* described in section 5.2.3.

Another research theme are scenarios where data transfer is subject to temporal quotas - e.g. the cloud or individual fog nodes have a bandwidth budget per time unit, which can be used in process-level decision-making. This would require creating an application or service for ONE nodes that can report their data usage during runtime, then this data can be forwarded to the processes as variables and be used in the decision gateways, deployment and service tasks.

6. CONCLUSION

The Business Process Management and Internet of Things domains have a lot to offer each other. Implementing BPM for IoT in the scope of new computing paradigms such as Edge & Fog computing can be challenging, especially for mobility-oriented IoT applications, which were the focus of this thesis. We distinguish 2 styles of Fog Computing - a centralised, top-down approach and a more autonomous, bottom-up approach. The former has received more attention in the state of the art, in the context of process management, it marks the approach where a global vision on the system promises optimal decisions (e.g. which resources to invoke), but privacy is reduced, and constant stable connectivity is relied upon. We call the notion of orchestrating and execution of processes in the edge - *Edge Process Management* (EPM).

In the bottom-up approach, the edge device, such as a Mobile Process Host, is the initiator and manager of the process and it may involve additional resources (Cloud, Fog, other MPHs) in the process opportunistically, as they are available, thereby representing a far more autonomous approach. Regardless, for Mobile IoT, both top-down and bottom-up approaches need to take into account locations and trajectories of hosts, bandwidth and processing size of tasks/data being offloaded.

In this thesis, we showed in the 1st contribution how MPHs can be realized technically, by implementing a prototype of a process-engine-embedded smartphone. Importantly, our MPH implementation includes the capability of migrating process instances device-to-device during runtime, increasing the resilience of process execution in the edge, as execution is not bound to a single device, without relying on a central manager for this. We showed how D2D edge process migration between MPHs can be useful for a mobility-heavy domain - logistics & transport. Practical experiments confirmed that the time-performance of the migration procedure make this a feasible approach.

While for logistics & transportation use cases scheduling the interactions between a MPH and other devices can be left as a manual human task, other scenarios need the scheduling decisions to be automated. For this, we proposed a heuristic scheduling algorithm that optimizes the decisions about which cloudlet a MPH should offload to, given the trajectory of the MPH, locations of nearby cloudlets, offloaded task size, MPH energy level and current cloudlet load and constraints for quality-of-service in terms of temporal performance. We then observed the effectiveness of the algorithm in 5 specific mobility-related task offloading scenarios in a simulated setting, varying the positioning, load, and number of nodes for each scenario. The results showed that the heuristic algorithm outperformed a naive opportunistic baseline approach in terms of successfully finished tasks and energy consumed.

Preparing and conducting the above experiments convinced us of the need for a simulation tool which would support the design and study of various business

processes, their execution, algorithms for their management and task scheduling in realistic mobility-oriented scenarios. We proposed the STEP-ONE simulator as such a tool, which extends the ONE simulator by embedding Flowable BPMS into the simulated hosts, and integrating the ONE simulation world with the process engines. STEP-ONE leverages ONE simulator's existing networking and mobility features, including map-based mobility models, while adding to it modules for modelling necessary Fog computing features such as task modelling, task execution cost estimation and adapting the graphical process designers and ONE graphical interface to be directly usable for designing and monitoring simulated business processes. We showed how STEP-ONE can be used to simulate a complex smart-city road monitoring process choreography, based on a top-down Fog application design. The process choreography involves Cloud resources, Fog nodes and MPHs, the invocation of which is managed at the process-level. A detailed analysis of this scenario showed the capability of STEP-ONE to be used for rich evaluation of edge process management. For instance, how processes perform, which nodes become bottlenecks, and how STEP-ONE can be used to compare how including or excluding Fog resources in the choreography affects results.

6.1. Future Research Directions

In the 1st contribution, we focused on the Process Executors in the edge, while the presented system design included more parties, such as the central Process Owner. As possible future research, the Process Owner-related modules can be studied, to identify technical challenges of implementing the full end-to-end system and applying it in practice. In the case study of STEP-ONE's smart city, the city initiating the road monitoring processes can be considered as the Process Owner, so an initial idea of how the process-level integration may work was attained, but a practical implementation outside of simulated scenarios would surely reveal more gaps and challenges of this approach. Additionally, STEP-ONE currently does not implement the D2D migration feature, as we previously implemented on Activiti, not Flowable. Adapting this feature to STEP-ONE should be straight-forward, as Flowable is a fork of Activiti. Once this is done, the migration capability can be studied under more scenarios besides the presented logistics case.

As part of future work, it is important to pursue more detailed modelling of wireless radio, taking into account physical factors such as radio and obstacle interference. To transform the simulation-based approach into a real-life deployment, introducing a trajectory prediction component may also be necessary if the user is not willing to specify their destination and trajectory explicitly (e.g. when using a navigation software). The presented heuristic scheduling algorithm can be fine-tuned with per-parameter weights, however a means to easily infer the weight values for a given application's requirements is still needed.

STEP-ONE can be used to model various vehicular applications [MCK19]. A direction to analyse, for example, is the effect of other edge nodes such as

pedestrians and cars, on the discussed scenario. On one hand, these nodes may also be using Fog services or invoking processes simultaneously with the road analysis processes, affecting the compute load of the fog servers or bandwidth usage of the networks, as we saw in chapter 4. On the other hand, even if the other nodes are not directly invoking functions of the fog resources, the radio interference of their presence may still affect the performance.

As part of future work for the proposed testbed, modelling of networking and computing for cloud-layer nodes can be improved, by introducing virtualization and cluster-based concepts. STEP-ONE currently mainly focuses on catching events from the simulation and mapping them to the process execution. To complement messages, more tasks which influence the simulation world could be added, such as a "Move to" task, which would allow to direct host movement from processes and would be interesting to model self-driving cars, drones, robots.

The BPMN 2.0 processes used in this thesis have a fairly fixed structure. As we noted in the State of the Art (cf. 2.5.1), CMMN based modelling of applications could also be suitable for modelling IoT applications, due to having more relaxed control flow structure constraints. Because CMMN is compatible with BPMN (it can include BPMN-based sub-processes), it would be interesting to explore introducing the proposed concepts of this thesis to a CMMN-based approach.

BIBLIOGRAPHY

[AC21] Saif Aljanabi and Abdolah Chalechale. "Improving IoT Services Using a Hybrid Fog-Cloud Offloading". In: *IEEE Access* 9 (2021), pp. 13775–13788. ISSN: 2169-3536. DOI: 10.1109/ACCESS.2021.3052458.

[AG19] Flowable AG. *The Flowable Engine as a Serverless Function*. https://www.flowable.com/blog/flowable-engine-as-a-serverless-function. 2019.

[Al-+15] Ala Al-Fuqaha et al. "Internet of Things: A Survey on Enabling Technologies, Protocols, and Applications". In: *IEEE Communications Surveys Tutorials* 17.4 (2015), pp. 2347–2376. ISSN: 1553-877X. DOI: 10.1109/COMST.2015.2444095.

[AMS19] Mainak Adhikari, Mithun Mukherjee, and Satish Narayana Srirama. "DPTO: A Deadline and Priority-aware Task Offloading in Fog Computing Framework Leveraging Multi-level Feedback Queueing". In: *IEEE Internet of Things Journal* (2019), pp. 1–1. ISSN: 2372-2541. DOI: 10.1109/JIOT.2019.2946426.

[App+14] Stefan Appel et al. "Modeling and Execution of Event Stream Processing in Business Processes". In: *Information Systems* 46 (Dec. 2014), pp. 140–156. ISSN: 0306-4379. DOI: 10.1016/j.is.2014.04.002.

[APZ18] Yuan Ai, Mugen Peng, and Kecheng Zhang. "Edge Computing Technologies for Internet of Things: A Primer". In: *Digital Communications and Networks* 4.2 (Apr. 2018), pp. 77–86. ISSN: 23528648. DOI: 10.1016/j.dcan.2017.07.001.

[Ash09] Kevin Ashton. "That 'Internet of Things' Thing". In: *RFID Journal* 222.7 (2009), pp. 97–114.

[Ass18] IEEE Standards Association. "IEEE Standard for Adoption of OpenFog Reference Architecture for Fog Computing". In: *IEEE Std 1934-2018* (Aug. 2018), pp. 1–176. DOI: 10.1109/IEEESTD.2018.8423800.

[ATH16] M. G. R. Alam, Y. K. Tun, and C. S. Hong. "Multi-Agent and Reinforcement Learning Based Code Offloading in Mobile Fog". In: *2016 International Conference on Information Networking (ICOIN)*. Jan. 2016, pp. 285–290. DOI: 10.1109/ICOIN.2016.7427078.

[Bar+12] Angineh Barkhordarian et al. "Migratability of BPMN 2.0 Process Instances". In: *Service-Oriented Computing - ICSOC 2011 Workshops*. Ed. by George Pallis et al. Lecture Notes in Computer Science. Berlin, Heidelberg: Springer, 2012, pp. 66–75. ISBN: 978-3-642-31875-7. DOI: 10.1007/978-3-642-31875-7_8.

[Beh+11] Michael Behrisch et al. "SUMO – Simulation of Urban MObility: An Overview". In: *Proceedings of SIMUL 2011, The Third Interna-*

tional Conference on Advances in System Simulation. Ed. by SINTEF & University of Oslo Aida Omerovic, RTI International-Research Triangle Park Diglio A. Simoni, and RTI International-Research Triangle Park Georgiy Bobashev. Barcelona: ThinkMind, Oct. 2011. ISBN: 978-1-61208-169-4.

[Ber+16] Maria Bermudez-Edo et al. "IoT-Lite: A Lightweight Semantic Model for the Internet of Things". In: *2016 Intl IEEE Conferences on Ubiquitous Intelligence & Computing, Advanced and Trusted Computing, Scalable Computing and Communications, Cloud and Big Data Computing, Internet of People, and Smart World Congress (UIC/ATC/ScalCom/CBDCom/IoP/SmartWorld)*. Toulouse: IEEE, July 2016, pp. 90–97. ISBN: 978-1-5090-2771-2. DOI: 10.1109/UIC-ATC-ScalCom-CBDCom-IoP-SmartWorld.2016.0035.

[Bit+15] Luiz Fernando Bittencourt et al. "Towards Virtual Machine Migration in Fog Computing". In: *2015 10th International Conference on P2P, Parallel, Grid, Cloud and Internet Computing (3PGCIC)*. Krakow: IEEE, Nov. 2015, pp. 1–8. ISBN: 978-1-4673-9473-4. DOI: 10.1109/3PGCIC.2015.85.

[Bit+17] L. F. Bittencourt et al. "Mobility-Aware Application Scheduling in Fog Computing". In: *IEEE Cloud Computing* 4.2 (Mar. 2017), pp. 26–35. ISSN: 2325-6095. DOI: 10.1109/MCC.2017.27.

[Bon+12] Flavio Bonomi et al. "Fog Computing and Its Role in the Internet of Things". In: *Proceedings of the First Edition of the MCC Workshop on Mobile Cloud Computing*. MCC '12. New York, NY, USA: ACM, 2012, pp. 13–16. ISBN: 978-1-4503-1519-7. DOI: 10.1145/2342509.2342513.

[Bot+16] Alessio Botta et al. "Integration of Cloud Computing and Internet of Things: A Survey". In: *Future Generation Computer Systems* 56 (Mar. 2016), pp. 684–700. ISSN: 0167-739X. DOI: 10.1016/j.future.2015.09.021.

[Bou+17] Khouloud Boukadi et al. "From VM to Container: A Linear Program for Outsourcing a Business Process to Cloud Containers". In: *On the Move to Meaningful Internet Systems. OTM 2017 Conferences*. Ed. by Hervé Panetto et al. Lecture Notes in Computer Science. Cham: Springer International Publishing, 2017, pp. 488–504. ISBN: 978-3-319-69462-7. DOI: 10.1007/978-3-319-69462-7_31.

[BR07] Mike Botts and Alexandre Robin. "OpenGIS Sensor Model Language (SensorML) Implementation Specification. Version 1.0. 0." In: Open Geospatial Consortium (2007).

[Bro+18] N Brouns et al. "Modeling IoT-aware Business Processes". In: *IBM Research Report* (2018), p. 42.

[Bro+21] Antonio Brogi et al. “Declarative Application Management in the Fog: A Bacteria-Inspired Decentralised Approach”. In: *Journal of Grid Computing* 19.4 (Dec. 2021), p. 45. ISSN: 1570-7873, 1572-9184. DOI: 10.1007/s10723-021-09582-y.

[Buc+12] Antonio Bucchiarone et al. “Dynamic Adaptation of Fragment-Based and Context-Aware Business Processes”. In: *2012 IEEE 19th International Conference on Web Services*. June 2012, pp. 33–41. DOI: 10.1109/ICWS.2012.56.

[BVS13] Rajkumar Buyya, Christian Vecchiola, and S. Thamarai Selvi. *Mastering Cloud Computing: Foundations and Applications Programming*. Amsterdam ; Boston: Morgan Kaufmann, 2013. ISBN: 978-0-12-411454-8. DOI: 10.1016/C2012-0-06719-1.

[Cat+11] Tiziana Catarci et al. “WORKPAD: Process Management and Geo-Collaboration Help Disaster Response”. In: *International Journal of Information Systems for Crisis Response and Management (IJISCRAM)* 3.1 (Jan. 2011), pp. 32–49. ISSN: 1937-9390 DOI: 10.4018/JISCRM.2011010103. DOI: 10.4018/jiscrm.2011010103.

[CG03] Di Caro and A Gianni. “Analysis of Simulation Environments for Mobile Ad Hoc Networks”. In: *Dalle Molle Institute for Artificial Intelligence, Tech. Rep* (2003), p. 28.

[Che+18a] X. Chen et al. “ThriftyEdge: Resource-Efficient Edge Computing for Intelligent IoT Applications”. In: *IEEE Network* 32.1 (Jan. 2018), pp. 61–65. ISSN: 0890-8044. DOI: 10.1109/MNET.2018.1700145.

[Che+18b] Y. Cheng et al. “A Service-Based Fog Execution Environment for the IoT-Aware Business Process Applications”. In: *2018 IEEE International Conference on Web Services (ICWS)*. July 2018, pp. 323–326. DOI: 10.1109/ICWS.2018.00052.

[Cir+19] Flavio Cirillo et al. “A Standard-Based Open Source IoT Platform: FIWARE”. In: *IEEE Internet of Things Magazine* 2.3 (Sept. 2019), pp. 12–18. ISSN: 2576-3199. DOI: 10.1109/IOTM.0001.1800022.

[Cis14] Cisco. “The Internet of Things Reference Model”. In: *Internet of Things World Forum*. 2014, pp. 1–12.

[Com+20] Ivan Compagnucci et al. “Modelling Notations for IoT-Aware Business Processes: A Systematic Literature Review”. In: *Business Process Management Workshops*. Ed. by Adela Del Río Ortega, Henrik Leopold, and Flávia Maria Santoro. Lecture Notes in Business Information Processing. Cham: Springer International Publishing, 2020, pp. 108–121. ISBN: 978-3-030-66498-5. DOI: 10.1007/978-3-030-66498-5_9.

[Con+13] Marco Conti et al. “Service Selection and Composition in Opportunistic Networks”. In: *2013 9th International Wireless Communi-*

cations and Mobile Computing Conference (IWCMC). July 2013, pp. 1565–1572. DOI: 10.1109/IWCMC.2013.6583789.

[CSB16] Chii Chang, Satish Narayana Srirama, and Rajkumar Buyya. "Mobile Cloud Business Process Management System for the Internet of Things: A Survey". In: *ACM Comput. Surv.* 49.4 (Dec. 2016), 70:1–70:42. ISSN: 0360-0300. DOI: 10.1145/3012000.

[CTC17] V. Chamola, C. K. Tham, and G. S. S. Chalapathi. "Latency Aware Mobile Task Assignment and Load Balancing for Edge Cloudlets". In: *2017 IEEE International Conference on Pervasive Computing and Communications Workshops (PerCom Workshops)*. Mar. 2017, pp. 587–592. DOI: 10.1109/PERCOMW.2017.7917628.

[CWB17] Xianfu Chen, Celimuge Wu, and Mehdi Bennis. "An Oblivious Game-Theoretic Approach for Wireless Scheduling in V2V Communications". In: *GLOBECOM 2017 - 2017 IEEE Global Communications Conference*. Dec. 2017, pp. 1–6. DOI: 10.1109/GLOCOM.2017.8254668.

[Dar+15] Kashif Dar et al. "A Resource Oriented Integration Architecture for the Internet of Things: A Business Process Perspective". In: *Pervasive and Mobile Computing* 20 (July 2015), pp. 145–159. ISSN: 1574-1192. DOI: 10.1016/j.pmcj.2014.11.005.

[Ded+18] J. Dede et al. "Simulating Opportunistic Networks: Survey and Future Directions". In: *IEEE Communications Surveys Tutorials* 20.2 (2018), pp. 1547–1573. ISSN: 1553-877X. DOI: 10.1109/COMST.2017.2782182.

[DFB14] Evert Ferdinand Duipmans, Luís Ferreira Pires, and Luiz Olavo Bonino da Silva Santos. "A Transformation-Based Approach to Business Process Management in the Cloud". In: *Journal of Grid Computing* 12.2 (June 2014), pp. 191–219. ISSN: 1572-9184. DOI: 10.1007/s10723-013-9278-z.

[Din+13] Hoang T. Dinh et al. "A Survey of Mobile Cloud Computing: Architecture, Applications, and Approaches: A Survey of Mobile Cloud Computing". In: *Wireless Communications and Mobile Computing* 13.18 (Dec. 2013), pp. 1587–1611. ISSN: 15308669. DOI: 10.1002/wcm.1203.

[DMC15] Dulce Domingos, Francisco Martins, and Lara Caiola. "Decentralising Internet of Things Aware BPMN Business Processes". In: *Sensor Systems and Software*. Ed. by Eiman Kanjo and Dirk Trossen. Lecture Notes of the Institute for Computer Sciences, Social Informatics and Telecommunications Engineering. Cham: Springer International Publishing, 2015, pp. 110–119. ISBN: 978-3-319-17136-4. DOI: 10.1007/978-3-319-17136-4_12.

[DPW04] A. Davis, J. Parikh, and W. E. Weihl. “Edgecomputing: Extending Enterprise Applications to the Edge of the Internet”. In: *Proceedings of the 13th International World Wide Web Conference on Alternate Track Papers & Posters*. WWW Alt. ’04. New York, NY, USA: Association for Computing Machinery, May 2004, pp. 180–187. ISBN: 978-1-58113-912-9. DOI: 10.1145/1013367.1013397.

[Du+18] Jianbo Du et al. “Computation Offloading and Resource Allocation in Mixed Fog/Cloud Computing Systems With Min-Max Fairness Guarantee”. In: *IEEE Transactions on Communications* 66.4 (Apr. 2018), pp. 1594–1608. ISSN: 1558-0857. DOI: 10.1109/TCOMM.2017.2787700.

[Dum+18] M. Dumas et al. *Fundamentals of Business Process Management: Second Edition*. Fundamentals of Business Process Management: Second Edition. 2018. DOI: 10.1007/978-3-662-56509-4.

[FGB18] J Fonseca, P Guillemin, and M Bauer. “Context Information Management (CIM); NGSI-LD API”. In: 1 (2018), pp. 1–159.

[Gin+10] Pau Giner et al. “Developing Mobile Workflow Support in the Internet of Things”. In: *IEEE Pervasive Computing* 9.2 (Apr. 2010), pp. 18–26. ISSN: 1536-1268. DOI: 10.1109/MPRV.2010.14.

[Gre+18] Paul Grefen et al. “Complex Collaborative Physical Process Management: A Position on the Trinity of BPM, IoT and DA”. In: *Collaborative Networks of Cognitive Systems*. Ed. by Luis M. Camarinha-Matos, Hamideh Afsarmanesh, and Yacine Rezgui. Vol. 534. Cham: Springer International Publishing, 2018, pp. 244–253. ISBN: 978-3-319-99127-6. DOI: 10.1007/978-3-319-99127-6_21.

[Gua+18] Yunguo Guan et al. “Data Security and Privacy in Fog Computing”. In: *IEEE Network* 32.5 (Sept. 2018), pp. 106–111. ISSN: 0890-8044, 1558-156X. DOI: 10.1109/MNET.2018.1700250.

[Guo+18] Y. Guo et al. “Mobile Cyber Physical Systems: Current Challenges and Future Networking Applications”. In: *IEEE Access* 6 (2018), pp. 12360–12368. ISSN: 2169-3536. DOI: 10.1109/ACCESS.2017.2782881.

[Gup+17] Harshit Gupta et al. “iFogSim: A Toolkit for Modeling and Simulation of Resource Management Techniques in the Internet of Things, Edge and Fog Computing Environments”. In: *Software: Practice and Experience* 47.9 (2017), pp. 1275–1296. ISSN: 1097-024X. DOI: 10.1002/spe.2509. arXiv: 1606.02007.

[HA19] Faruk Hasić and Estefanía Serral Asensio. “Executing IoT Processes in BPMN 2.0: Current Support and Remaining Challenges”. In: *2019 13th International Conference on Research Challenges in Information Science (RCIS)*. May 2019, pp. 1–6. DOI: 10.1109/RCIS.2019.8876998.

[Hac+06] Gregory Hackmann et al. “Sliver: A BPEL Workflow Process Execution Engine for Mobile Devices”. In: *Service-Oriented Computing – ICSOC 2006*. Ed. by Asit Dan and Winfried Lamersdorf. Lecture Notes in Computer Science. Springer Berlin Heidelberg, 2006, pp. 503–508. ISBN: 978-3-540-68148-9.

[HKS09] Stephan Haller, Stamatis Karnouskos, and Christoph Schroth. “The Internet of Things in an Enterprise Context”. In: *Future Internet – FIS 2008*. Ed. by John Domingue, Dieter Fensel, and Paolo Traverso. Lecture Notes in Computer Science. Berlin, Heidelberg: Springer, 2009, pp. 14–28. ISBN: 978-3-642-00985-3. DOI: 10.1007/978-3-642-00985-3_2.

[HM11] Stephan Haller and Carsten Magerkurth. “The Real-time Enterprise: IoT-enabled Business Processes”. In: *Interconnecting Smart Objects with the Internet Workshop 2011*. 2011, p. 3.

[Hol95] David Hollingsworth. “The Workflow Reference Model”. In: *workflow management coalition TC00-1003* (1995).

[Hon+13] Kirak Hong et al. “Opportunistic Spatio-Temporal Event Processing for Mobile Situation Awareness”. In: *Proceedings of the 7th ACM International Conference on Distributed Event-Based Systems*. DEBS ’13. New York, NY, USA: Association for Computing Machinery, June 2013, pp. 195–206. ISBN: 978-1-4503-1758-0. DOI: 10.1145/2488222.2488266.

[Hu+15] Yun Chao Hu et al. “Mobile Edge Computing—A Key Technology towards 5G”. In: *ETSI white paper* 11.11 (2015), pp. 1–16.

[Hua+16] Hong-Cheng Huang et al. “Interference-Limited Device-to-Device Multi-User Cooperation Scheme for Optimization of Edge Networking”. In: *Journal of Computer Science and Technology* 31.6 (Nov. 2016), pp. 1096–1109. ISSN: 1860-4749. DOI: 10.1007/s11390-016-1685-8.

[Hub+16] Steffen Huber et al. “Using Semantic Queries to Enable Dynamic Service Invocation for Processes in the Internet of Things”. In: *2016 IEEE Tenth International Conference on Semantic Computing (ICSC)*. Feb. 2016, pp. 214–221. DOI: 10.1109/ICSC.2016.75.

[HZL12] Kristof Hamann, Sonja Zaplata, and Winfried Lamersdorf. “Process Instance Migration: Flexible Execution of Distributed Business Processes”. In: *2012 First International Workshop on European Software Services and Systems Research - Results and Challenges (S-Cube)*. June 2012, pp. 21–22. DOI: 10.1109/S-Cube.2012.6225502.

[ITU12] ITU-T. “2060: Overview of the Internet of Things”. In: *ITU-T–International Telecommunication Union* (2012).

[Jan+20] Christian Janiesch et al. “The Internet of Things Meets Business Process Management: A Manifesto”. In: *IEEE Systems, Man, and Cybernetics Magazine* 6.4 (Oct. 2020), pp. 34–44. ISSN: 2333-942X. DOI: 10.1109/MSMC.2020.3003135.

[Jar+14] Antonio J. Jara et al. “Mobile Digcovery: Discovering and Interacting with the World through the Internet of Things”. In: *Personal and Ubiquitous Computing* 18.2 (Feb. 2014), pp. 323–338. ISSN: 1617-4917. DOI: 10.1007/s00779-013-0648-0.

[Kép+16] Kálmán Képes et al. “Situation-Aware Execution and Dynamic Adaptation of Traditional Workflow Models”. In: *Service-Oriented and Cloud Computing*. Ed. by Marco Aiello et al. Lecture Notes in Computer Science. Cham: Springer International Publishing, 2016, pp. 69–83. ISBN: 978-3-319-44482-6. DOI: 10.1007/978-3-319-44482-6_5.

[KOK09] Ari Keränen, Jörg Ott, and Teemu Kärkkäinen. “The ONE Simulator for DTN Protocol Evaluation”. In: ICST, 2009. ISBN: 978-963-9799-45-5. DOI: 10.4108/ICST.SIMUTOOLS2009.5674.

[Koz10] Tomas Kozel. “BPMN Mobilisation”. In: *Proceedings of the European Conference of Systems, and European Conference of Circuits Technology and Devices, and European Conference of Communications, and European Conference on Computer Science*. ECS’10 / ECCTD’10 / ECCOM’10 / ECCS’10. Tenerife, Spain: World Scientific and Engineering Academy and Society (WSEAS), Nov. 2010, pp. 307–310. ISBN: 978-960-474-250-9.

[KS14] Anil Kalbag and Gerald Silverman. “Enriching Business Processes through Internet of Everything”. In: *Cisco IT Article. Business Processes and {IoE}* (2014).

[Kum+13] Karthik Kumar et al. “A Survey of Computation Offloading for Mobile Systems”. In: *Mobile Networks and Applications* 18.1 (Feb. 2013), pp. 129–140. ISSN: 1572-8153. DOI: 10.1007/s11036-012-0368-0.

[Li+13] Jiwei Li et al. “ENDA: Embracing Network Inconsistency for Dynamic Application Offloading in Mobile Cloud Computing”. In: *Proceedings of the Second ACM SIGCOMM Workshop on Mobile Cloud Computing*. MCC ’13. New York, NY, USA: ACM, 2013, pp. 39–44. ISBN: 978-1-4503-2180-8. DOI: 10.1145/2491266.2491274.

[Liu+16] Juan Liu et al. “Delay-Optimal Computation Task Scheduling for Mobile-Edge Computing Systems”. In: *2016 IEEE International Symposium on Information Theory (ISIT)*. July 2016, pp. 1451–1455. DOI: 10.1109/ISIT.2016.7541539.

[LKC17] Jongwon Lee, Jaejun Ko, and Young-June Choi. “Dhrystone Million Instructions per Second–Based Task Offloading from Smartwatch to

Smartphone". In: *International Journal of Distributed Sensor Networks* 13.11 (Nov. 2017), p. 1550147717740073. ISSN: 1550-1477. DOI: 10.1177/1550147717740073.

[LMM19] Francesco Leotta, Andrea Marrella, and Massimo Mecella. "IoT for BPMers. Challenges, Case Studies and Successful Applications". In: *Business Process Management*. Ed. by Thomas Hildebrandt et al. Lecture Notes in Computer Science. Cham: Springer International Publishing, 2019, pp. 16–22. ISBN: 978-3-030-26619-6. DOI: 10.1007/978-3-030-26619-6_3.

[LS13] K. Lee and I. Shin. "User Mobility-Aware Decision Making for Mobile Computation Offloading". In: *2013 IEEE 1st International Conference on Cyber-Physical Systems, Networks, and Applications (CPSNA)*. Aug. 2013, pp. 116–119. DOI: 10.1109/CPSNA.2013.6614257.

[Mah+19] Redowan Mahmud et al. "Quality of Experience (QoE)-Aware Placement of Applications in Fog Computing Environments". In: *Journal of Parallel and Distributed Computing* 132 (Oct. 2019), pp. 190–203. ISSN: 0743-7315. DOI: 10.1016/j.jpdc.2018.03.004.

[Mah+20] Redowan Mahmud et al. "Profit-Aware Application Placement for Integrated Fog–Cloud Computing Environments". In: *Journal of Parallel and Distributed Computing* 135 (Jan. 2020), pp. 177–190. ISSN: 0743-7315. DOI: 10.1016/j.jpdc.2019.10.001.

[Man+19] Sankalita Mandal et al. "A Classification Framework for IoT Scenarios". In: *Business Process Management Workshops*. Ed. by Florian Daniel, Quan Z. Sheng, and Hamid Motahari. Vol. 342. Cham: Springer International Publishing, 2019, pp. 458–469. ISBN: 978-3-030-11640-8 978-3-030-11641-5. DOI: 10.1007/978-3-030-11641-5_36.

[Mas+18] D. Mascitti et al. "Service Provisioning in Mobile Environments through Opportunistic Computing". In: *IEEE Transactions on Mobile Computing* 17.12 (Dec. 2018), pp. 2898–2911. ISSN: 1536-1233. DOI: 10.1109/TMC.2018.2824325.

[MBR15] Roberto Minerva, Abyi Biru, and Domenico Rotondi. "Towards a Definition of the Internet of Things (IoT)". In: *IEEE Internet Initiative* 1.1 (2015), pp. 1–86.

[MCK19] Leo Mendiboure, Mohamed-Aymen Chalouf, and Francine Krief. "Edge Computing Based Applications in Vehicular Environments: Comparative Study and Main Issues". In: *Journal of Computer Science and Technology* 34.4 (July 2019), pp. 869–886. ISSN: 1000-9000, 1860-4749. DOI: 10.1007/s11390-019-1947-3.

[MCS16a] J. Mass, C. Chang, and S. N. Srirama. "WiseWare: A Device-to-Device-Based Business Process Management System for Industrial

Internet of Things". In: *2016 IEEE International Conference on Internet of Things (iThings) and IEEE Green Computing and Communications (GreenCom) and IEEE Cyber, Physical and Social Computing (CPSCom) and IEEE Smart Data (SmartData)*. Dec. 2016, pp. 269–275. DOI: 10.1109/iThings-GreenCom-CPSCom-SmartData.2016.69.

[MCS16b] Jakob Mass, Chii Chang, and Satish N. Srirama. "Workflow Model Distribution or Code Distribution? Ideal Approach for Service Composition of the Internet of Things". In: *2016 IEEE International Conference on Services Computing (SCC)*. San Francisco, CA, USA: IEEE, June 2016, pp. 649–656. ISBN: 978-1-5090-2628-9. DOI: 10.1109/SCC.2016.90.

[MCS18] J. Mass, C. Chang, and S.N. Srirama. "Context-Aware Edge Process Management for Mobile Thing-to-Fog Environment". In: *ACM Int. Conf. Proc. Ser.* Association for Computing Machinery, 2018. ISBN: 9781450364836 (ISBN). DOI: 10.1145/3241403.3241449.

[MCS19] Jakob Mass, Chii Chang, and Satish Narayana Srirama. "Edge Process Management: A Case Study on Adaptive Task Scheduling in Mobile IoT". In: *Internet of Things* 6 (June 2019), p. 100051. ISSN: 2542-6605. DOI: 10.1016/j.iot.2019.100051.

[MD17] Francisco Martins and Dulce Domingos. "Modelling IoT Behaviour within BPMN Business Processes". In: *Procedia computer science* 121 (2017), pp. 1014–1022.

[Mey+11] Sonja Meyer et al. "Towards Modeling Real-World Aware Business Processes". In: *Proceedings of the Second International Workshop on Web of Things*. WoT '11. San Francisco, California, USA: Association for Computing Machinery, June 2011, pp. 1–6. ISBN: 978-1-4503-0624-9. DOI: 10.1145/1993966.1993978.

[MM17] Andrea Marrella and Massimo Mecella. "Adaptive Process Management in Cyber-Physical Domains". In: *Advances in Intelligent Process-Aware Information Systems: Concepts, Methods, and Technologies*. Ed. by Gregor Grambow, Roy Oberhauser, and Manfred Reichert. Intelligent Systems Reference Library. Cham: Springer International Publishing, 2017, pp. 15–48. ISBN: 978-3-319-52181-7. DOI: 10.1007/978-3-319-52181-7_2.

[MMS16] Andrea Marrella, Massimo Mecella, and Sebastian Sardina. "Intelligent Process Adaptation in the SmartPM System". In: *ACM Trans. Intell. Syst. Technol.* 8.2 (Nov. 2016), 25:1–25:43. ISSN: 2157-6904. DOI: 10.1145/2948071.

[Moh+20] Nitinder Mohan et al. "Pruning Edge Research with Latency Shears". In: *Proceedings of the 19th ACM Workshop on Hot Topics in Networks*. HotNets '20. New York, NY, USA: Association for Comput-

ing Machinery, Nov. 2020, pp. 182–189. ISBN: 978-1-4503-8145-1. DOI: 10.1145/3422604.3425943.

[Mou+18] C. Mouradian et al. “A Comprehensive Survey on Fog Computing: State-of-the-Art and Research Challenges”. In: *IEEE Communications Surveys Tutorials* 20.1 (2018), pp. 416–464. ISSN: 1553-877X. DOI: 10.1109/COMST.2017.2771153.

[MRB21] Redowan Mahmud, Kotagiri Ramamohanarao, and Rajkumar Buyya. “Application Management in Fog Computing Environments: A Taxonomy, Review and Future Directions”. In: *ACM Computing Surveys* 53.4 (July 2021), pp. 1–43. ISSN: 0360-0300, 1557-7341. DOI: 10.1145/3403955.

[MRH15] Sonja Meyer, Andreas Ruppen, and Lorenz Hilty. “The Things of the Internet of Things in BPMN”. In: *Advanced Information Systems Engineering Workshops*. Ed. by Anne Persson and Janis Stirna. Vol. 215. Cham: Springer International Publishing, 2015, pp. 285–297. ISBN: 978-3-319-19242-0 978-3-319-19243-7. DOI: 10.1007/978-3-319-19243-7_27.

[MSC20] Jakob Mass, Satish Narayana Srirama, and Chii Chang. “STEP-ONE: Simulated Testbed for Edge-Fog Processes Based on the Opportunistic Network Environment Simulator”. In: *Journal of Systems and Software* 166 (Aug. 2020), p. 110587. ISSN: 01641212. DOI: 10.1016/j.jss.2020.110587.

[MSM11] Sonja Meyer, Klaus Sperner, and Carsten Magerkurth. “Towards Real World Aware Enterprise Systems - Reflecting the Quality Information of Physical Resources in Services and Processes”. In: *2011 IEEE Eighth International Conference on Mobile Ad-Hoc and Sensor Systems*. Oct. 2011, pp. 843–848. DOI: 10.1109/MASS.2011.98.

[MSW18] M. Mukherjee, L. Shu, and D. Wang. “Survey of Fog Computing: Fundamental, Network Applications, and Research Challenges”. In: *IEEE Communications Surveys Tutorials* 20.3 (2018), pp. 1826–1857. ISSN: 1553-877X. DOI: 10.1109/COMST.2018.2814571.

[MVJ18] Majid Makki, Dimitri Van Landuyt, and Wouter Joosen. “Towards PaaS Offering of BPMN 2.0 Engines: A Proposal for Service-Level Tenant Isolation”. In: *Advances in Service-Oriented and Cloud Computing*. Ed. by Zoltán Ádám Mann and Volker Stolz. Vol. 824. Cham: Springer International Publishing, 2018, pp. 5–19. ISBN: 978-3-319-79089-3 978-3-319-79090-9. DOI: 10.1007/978-3-319-79090-9_1.

[NAG19] Mahda Noura, Mohammed Atiquzzaman, and Martin Gaedke. “Interoperability in Internet of Things: Taxonomies and Open Challenges”. In: *Mobile Networks and Applications* 24.3 (June 2019),

pp. 796–809. ISSN: 1383-469X, 1572-8153. DOI: 10.1007/s11036-018-1089-9.

[Ni+17] L. Ni et al. “Resource Allocation Strategy in Fog Computing Based on Priced Timed Petri Nets”. In: *IEEE Internet of Things Journal* 4.5 (Oct. 2017), pp. 1216–1228. DOI: 10.1109/JIOT.2017.2709814.

[Ope17] OpenFog Consortium Architecture Working Group and others. “OpenFog Reference Architecture for Fog Computing”. In: *OPFRA001* 20817 (2017), p. 162.

[Ott+14] Beate Ottenwälder et al. “MCEP: A Mobility-Aware Complex Event Processing System”. In: *ACM Transactions on Internet Technology* 14.1 (Aug. 2014), 6:1–6:24. ISSN: 1533-5399. DOI: 10.1145/2633688.

[Pen+14] Tao Peng et al. “Business Process Assignment and Execution in Mobile Environments”. In: *2014 International Conference on Collaboration Technologies and Systems (CTS)*. May 2014, pp. 267–274. DOI: 10.1109/CTS.2014.6867574.

[Pha14] Thomas Phan. “Intelligent Energy-Efficient Triggering of Geolocation Fix Acquisitions Based on Transitions between Activity Recognition States”. In: *Mobile Computing, Applications, and Services*. Ed. by Gérard Memmi and Ulf Blanke. Lecture Notes of the Institute for Computer Sciences, Social Informatics and Telecommunications Engineering. Cham: Springer International Publishing, 2014, pp. 104–121. ISBN: 978-3-319-05452-0. DOI: 10.1007/978-3-319-05452-0_9.

[Pou+17] Shaya Pourmirza et al. “A Systematic Literature Review on the Architecture of Business Process Management Systems”. In: *Information Systems* 66 (June 2017), pp. 43–58. ISSN: 0306-4379. DOI: 10.1016/j.is.2017.01.007.

[Pry+11] Rüdiger Pryss et al. “Towards Flexible Process Support on Mobile Devices”. In: *Information Systems Evolution*. Ed. by Pnina Soffer and Erik Proper. Lecture Notes in Business Information Processing. Springer Berlin Heidelberg, 2011, pp. 150–165. ISBN: 978-3-642-17722-4.

[Pry+15] Rüdiger Pryss et al. “Supporting Medical Ward Rounds through Mobile Task and Process Management”. In: *Information Systems and e-Business Management* 13.1 (Feb. 2015), pp. 107–146. ISSN: 1617-9854. DOI: 10.1007/s10257-014-0244-5.

[Qi+16] Qi Qi et al. “Software Defined Resource Orchestration System for Multitask Application in Heterogeneous Mobile Cloud Computing”. In: *Mobile Information Systems* (2016). DOI: 10.1155/2016/2784548.

[Rah+16] A. Rahman et al. “A Cloud Robotics Framework of Optimal Task Offloading for Smart City Applications”. In: *2016 IEEE Global Com-*

munications Conference (GLOBECOM). Dec. 2016, pp. 1–7. DOI: 10.1109/GLOCOM.2016.7841487.

[Reh+17] Muhammad Habib ur Rehman et al. "Towards Next-Generation Heterogeneous Mobile Data Stream Mining Applications: Opportunities, Challenges, and Future Research Directions". In: *Journal of Network and Computer Applications* 79 (Feb. 2017), pp. 1–24. ISSN: 1084-8045. DOI: 10.1016/j.jnca.2016.11.031.

[RML12] Alessandro Russo, Massimo Mecella, and Massimiliano Leoni. "ROME4EU - A Service-Oriented Process-Aware Information System for Mobile Devices: ROME4EU - A SERVICE-ORIENTED PAIS FOR MOBILE DEVICES". In: *Software: Practice and Experience* 42.10 (Oct. 2012), pp. 1275–1314. ISSN: 00380644. DOI: 10.1002/spe.1138.

[Sat+09] Mahadev Satyanarayanan et al. "The Case for VM-Based Cloudlets in Mobile Computing". In: *IEEE Pervasive Computing* 8.4 (Oct. 2009), pp. 14–23. ISSN: 1558-2590. DOI: 10.1109/MPRV.2009.82.

[Sat+13] M. Satyanarayanan et al. "The Role of Cloudlets in Hostile Environments". In: *IEEE Pervasive Computing* 12.4 (Oct. 2013), pp. 40–49. ISSN: 1536-1268. DOI: 10.1109/MPRV.2013.77.

[Sat+15] M. Satyanarayanan et al. "Edge Analytics in the Internet of Things". In: *IEEE Pervasive Computing* 14.2 (2015), pp. 24–31. ISSN: 1536-1268. DOI: 10.1109/MPRV.2015.32.

[Sat17] M. Satyanarayanan. "The Emergence of Edge Computing". In: *Computer* 50.1 (Jan. 2017), pp. 30–39. ISSN: 0018-9162. DOI: 10.1109/MC.2017.9.

[Sch+13] Stephan Scheuren et al. "Spatio-Temporally Constrained Planning for Cooperative Vehicles in a Harvesting Scenario". In: *KI - Künstliche Intelligenz* 27.4 (Nov. 2013), pp. 341–346. ISSN: 1610-1987. DOI: 10.1007/s13218-013-0267-y.

[Sch+16] Johannes Schobel et al. "A Lightweight Process Engine for Enabling Advanced Mobile Applications". In: *On the Move to Meaningful Internet Systems: OTM 2016 Conferences*. Ed. by Christophe Debruyne et al. Lecture Notes in Computer Science. Springer International Publishing, 2016, pp. 552–569. ISBN: 978-3-319-48472-3.

[Sei+15] Ronny Seiger et al. "Modelling Complex and Flexible Processes for Smart Cyber-Physical Environments". In: *Journal of Computational Science* 10 (Sept. 2015), pp. 137–148. ISSN: 1877-7503. DOI: 10.1016/j.jocs.2014.07.001.

[Sei+19] Ronny Seiger et al. "Toward a Framework for Self-Adaptive Workflows in Cyber-Physical Systems". In: *Software & Systems Modeling* 18.2 (Apr. 2019), pp. 1117–1134. ISSN: 1619-1366, 1619-1374. DOI: 10.1007/s10270-017-0639-0.

[Sha+19] Muhammad Shafiq et al. “Ranked Sense Multiple Access Control Protocol for Multichannel Cognitive Radio-Based IoT Networks”. In: *Sensors* 19.7 (Apr. 2019), p. 1703. ISSN: 1424-8220. DOI: 10.3390/s19071703.

[SHA17] R. Seiger, S. Herrmann, and U. Aßmann. “Self-Healing for Distributed Workflows in the Internet of Things”. In: *2017 IEEE International Conference on Software Architecture Workshops (ICSAW)*. Apr. 2017, pp. 72–79. DOI: 10.1109/ICSAW.2017.36.

[SHA19] Ronny Seiger, Peter Heisig, and Uwe Aßmann. “Retrofitting of Workflow Management Systems with Self-X Capabilities for Internet of Things”. In: *Business Process Management Workshops*. Ed. by Florian Daniel, Quan Z. Sheng, and Hamid Motahari. Lecture Notes in Business Information Processing. Cham: Springer International Publishing, 2019, pp. 433–444. ISBN: 978-3-030-11641-5. DOI: 10.1007/978-3-030-11641-5_34.

[Shi+16] Weisong Shi et al. “Edge Computing: Vision and Challenges”. In: *IEEE Internet of Things Journal* 3.5 (Oct. 2016), pp. 637–646. ISSN: 2327-4662. DOI: 10.1109/JIOT.2016.2579198.

[SHS18] Ronny Seiger, Steffen Huber, and Thomas Schlegel. “Toward an Execution System for Self-Healing Workflows in Cyber-Physical Systems”. In: *Software & Systems Modeling* 17.2 (May 2018), pp. 551–572. ISSN: 1619-1366, 1619-1374. DOI: 10.1007/s10270-016-0551-z.

[SOE17] Cagatay Sonmez, Atay Ozgovde, and Cem Ersoy. “EdgeCloudSim: An Environment for Performance Evaluation of Edge Computing Systems”. In: *2017 Second International Conference on Fog and Mobile Edge Computing (FMEC)*. May 2017, pp. 39–44. DOI: 10.1109/FMEC.2017.7946405.

[Spi+09] Patrik Spiess et al. “SOA-based Integration of the Internet of Things in Enterprise Services”. In: *Proceedings of the IEEE 7th International Conference on Web Services (ICWS)*. Los Angeles, US: IEEE Computer Society Press, July 2009, pp. 968–975. ISBN: 978-0-7695-3709-2.

[SRG08] Rohan Sen, Gruia-Catalin Roman, and Christopher Gill. “CiAN: A Workflow Engine for MANETs”. In: *Coordination Models and Languages*. Ed. by Doug Lea and Gianluigi Zavattaro. Lecture Notes in Computer Science. Springer Berlin Heidelberg, 2008, pp. 280–295. ISBN: 978-3-540-68265-3.

[SRT18] David Richard Schafer, Kurt Rothermel, and Muhammad Adnan Tariq. “Replication Schemes for Highly Available Workflow Engines”. In: *IEEE Transactions on Services Computing* (2018), pp. 1–1. ISSN: 2372-0204. DOI: 10.1109/TSC.2018.2813368.

[SS17] Pallavi Sethi and Smruti R. Sarangi. “Internet of Things: Architectures, Protocols, and Applications”. In: *Journal of Electrical and Computer Engineering* 2017 (2017), pp. 1–25. ISSN: 2090-0147, 2090-0155. DOI: 10.1155/2017/9324035.

[Sta+18] Michael Stach et al. “Towards a Beacon-based Situational Prioritization Framework for Process-Aware Information Systems”. In: *Procedia Computer Science*. The 15th International Conference on Mobile Systems and Pervasive Computing (MobiSPC 2018) / The 13th International Conference on Future Networks and Communications (FNC-2018) / Affiliated Workshops 134 (Jan. 2018), pp. 153–160. ISSN: 1877-0509. DOI: 10.1016/j.procs.2018.07.156.

[STB17] L Shakkeera, Latha Tamilselvan, and B.S. Abdur Rahman University. “Towards Maximum Resource Utilization and Optimal Task Execution for Gaming IoT Workflow in Mobile Cloud”. In: *International Journal of Intelligent Engineering and Systems* 10.1 (Feb. 2017), pp. 134–143. ISSN: 21853118. DOI: 10.22266/ijies2017.0228.15.

[Sun20] Ali Sunyaev. *Internet Computing: Principles of Distributed Systems and Emerging Internet-Based Technologies*. Cham: Springer, 2020. ISBN: 978-3-030-34957-8 978-3-030-34956-1.

[SZ15] Koh Song Sang and Bo Zhou. “BPMN Security Extensions for Healthcare Process”. In: *2015 IEEE International Conference on Computer and Information Technology; Ubiquitous Computing and Communications; Dependable, Autonomic and Secure Computing; Pervasive Intelligence and Computing*. Oct. 2015, pp. 2340–2345. DOI: 10.1109/CIT/IUCC/DASC/PICOM.2015.346.

[Tra+12] Stefano Tranquillini et al. “Process-Based Design and Integration of Wireless Sensor Network Applications”. In: *Business Process Management*. Ed. by Alistair Barros, Avigdor Gal, and Ekkart Kindler. Lecture Notes in Computer Science. Berlin, Heidelberg: Springer, 2012, pp. 134–149. ISBN: 978-3-642-32885-5. DOI: 10.1007/978-3-642-32885-5_10.

[Tri+17] Sacha Trifunovic et al. “A Decade of Research in Opportunistic Networks: Challenges, Relevance, and Future Directions”. In: *IEEE Communications Magazine* 55.1 (Jan. 2017), pp. 168–173. ISSN: 1558-1896. DOI: 10.1109/MCOM.2017.1500527CM.

[TVM18] Giacomo Tanganelli, Carlo Vallati, and Enzo Mingozzi. “Edge-Centric Distributed Discovery and Access in the Internet of Things”. In: *IEEE Internet of Things Journal* 5.1 (Feb. 2018), pp. 425–438. ISSN: 2327-4662. DOI: 10.1109/JIOT.2017.2767381.

[Van+09] Wil MP Van der Aalst et al. “ProM: The Process Mining Toolkit.” In: *Proceedings of the BPM 2009 Demonstration Track*. CEUR Workshop Proceedings. Ulm, Germany, 2009, pp. 1–4.

[Var10] Andras Varga. “OMNeT++”. In: *Modeling and Tools for Network Simulation*. Ed. by Klaus Wehrle, Mesut Güneş, and James Gross. Berlin, Heidelberg: Springer Berlin Heidelberg, 2010, pp. 35–59. ISBN: 978-3-642-12330-6 978-3-642-12331-3. DOI: 10.1007/978-3-642-12331-3_3.

[Ver+11] Ovidiu Vermesan et al. “Internet of Things Strategic Research Roadmap”. In: *Internet of things-global technological and societal trends* 1.2011 (2011), pp. 9–52.

[W3C20] W3C. *Web of Things (WoT) Architecture*. https://www.w3.org/TR/2020/REC-wot-architecture-20200409/. 2020.

[Wai+21] Philipp Waibel et al. “ViePEP-C: A Container-Based Elastic Process Platform”. In: *IEEE Transactions on Cloud Computing* 9.4 (Oct. 2021), pp. 1657–1674. ISSN: 2168-7161. DOI: 10.1109/TCC.2019.2912613.

[WE16] Michael Weyrich and Christof Ebert. “Reference Architectures for the Internet of Things”. In: *IEEE Software* 33.1 (Jan. 2016), pp. 112–116. ISSN: 1937-4194. DOI: 10.1109/MS.2016.20.

[Wie+15] Matthias Wieland et al. “Towards Situation-Aware Adaptive Workflows: SitOPT — A General Purpose Situation-Aware Workflow Management System”. In: *2015 IEEE International Conference on Pervasive Computing and Communication Workshops (PerCom Workshops)*. Mar. 2015, pp. 32–37. DOI: 10.1109/PERCOMW.2015.7133989.

[Xu+19] Xiaolong Xu et al. “Multiobjective Computation Offloading for Workflow Management in Cloudlet-Based Mobile Cloud Using NSGA-II”. In: *Computational Intelligence* 35.3 (2019), pp. 476–495. ISSN: 1467-8640. DOI: 10.1111/coin.12197.

[Yan+18] Yang Yang et al. “MEETS: Maximal Energy Efficient Task Scheduling in Homogeneous Fog Networks”. In: *IEEE Internet of Things Journal* 5.5 (Oct. 2018), pp. 4076–4087. ISSN: 2372-2541. DOI: 10.1109/JIOT.2018.2846644.

[You+16] Alaaeddine Yousfi et al. “uBPMN: A BPMN Extension for Modeling Ubiquitous Business Processes”. In: *Information and Software Technology* 74 (June 2016), pp. 55–68. ISSN: 0950-5849. DOI: 10.1016/j.infsof.2016.02.002.

[You+18] Alaaeddine Yousfi et al. “Toward uBPMN-Based Patterns for Modeling Ubiquitous Business Processes”. In: *IEEE Transactions on Industrial Informatics* 14.8 (Aug. 2018), pp. 3358–3367. ISSN: 1941-0050. DOI: 10.1109/TII.2017.2777847.

[You+19] Ashkan Yousefpour et al. “All One Needs to Know about Fog Computing and Related Edge Computing Paradigms: A Complete Survey”. In: *Journal of Systems Architecture* 98 (Sept. 2019), pp. 289–330. ISSN: 1383-7621. DOI: 10.1016/j.sysarc.2019.02.009.

[Yu+18] Wei Yu et al. “A Survey on the Edge Computing for the Internet of Things”. In: *IEEE Access* 6 (2018), pp. 6900–6919. ISSN: 2169-3536. DOI: 10.1109/ACCESS.2017.2778504.

[Zap+10a] Sonja Zaplata et al. “Flexible Execution of Distributed Business Processes Based on Process Instance Migration”. In: *Journal of Systems Integration* 1.3 (July 2010), pp. 3-16–16. ISSN: 1804-2724. DOI: 10.20470/jsi.v1i3.52.

[Zap+10b] Sonja Zaplata et al. “Towards Runtime Migration of WS-BPEL Processes”. In: *Service-Oriented Computing – ICSOC 2007*. Ed. by David Hutchison et al. Vol. 4749. Berlin, Heidelberg: Springer Berlin Heidelberg, 2010, pp. 477–487. ISBN: 978-3-540-74973-8 978-3-540-74974-5. DOI: 10.1007/978-3-642-16132-2_45.

[Zen+16] Deze Zeng et al. “Joint Optimization of Task Scheduling and Image Placement in Fog Computing Supported Software-Defined Embedded System”. In: *IEEE Transactions on Computers* 65.12 (Dec. 2016), pp. 3702–3712. ISSN: 0018-9340. DOI: 10.1109/TC.2016.2536019.

[Zha+14] Y. Zhang et al. “Dynamic Offloading Algorithm in Intermittently Connected Mobile Cloudlet Systems”. In: *2014 IEEE International Conference on Communications (ICC)*. June 2014, pp. 4190–4195. DOI: 10.1109/ICC.2014.6883978.

[Zha+15] Ben Zhang et al. “The Cloud Is Not Enough: Saving Iot from the Cloud”. In: *Proceedings of the 7th USENIX Conference on Hot Topics in Cloud Computing*. HotCloud’15. USA: USENIX Association, July 2015, p. 21.

[Zha+18] T. Zhang et al. “Lightweight SOA-Based Multi-Engine Architecture for Workflow Systems in Mobile Ad Hoc Networks”. In: *IEEE Access* 6 (2018), pp. 14212–14222. DOI: 10.1109/ACCESS.2018.2815617.

[Zhu+16] Xinwei Zhu et al. “Enabling Flexible Location-Aware Business Process Modeling and Execution”. In: *Decision Support Systems* 83 (Mar. 2016), pp. 1–9. ISSN: 0167-9236. DOI: 10.1016/j.dss.2015.12.003.

[ZNF17] H. Zakeri, Fereidoon Moghadas Nejad, and Ahmad Fahimifar. “Image Based Techniques for Crack Detection, Classification and Quantification in Asphalt Pavement: A Review”. In: *Archives of Computational Methods in Engineering* 24.4 (Nov. 2017), pp. 935–977. ISSN: 1886-1784. DOI: 10.1007/s11831-016-9194-z.

[Zou+16] M. Zou et al. “D3W: Towards Self-Management of Distributed Data-Driven Workflows with QoS Guarantees”. In: *2016 IEEE 9th International Conference on Cloud Computing (CLOUD)*. June 2016, pp. 148–155. DOI: 10.1109/CLOUD.2016.0029.

ACKNOWLEDGEMENTS

My gratitude goes to all who supported me on this journey of learning - to my parents and grandparents, to Brita, to my friends and co-workers. Thank you for the companionship and guidance during this journey - my supervisors Satish and Chii, colleagues at the Mobile & Cloud Lab with whom countless discussions and coffee have been had and all the other inspiring people at the Institute of Computer Science. Thank you - Amnir and Ulrich - for the help in improving this thesis, and Urmo - for introducing me to exciting projects and problems outside the University.

This thesis was supported by Telia Eesti AS and the Archimedes Foundation Smart specialisation scholarship. The work was also supported by the Estonian Centre of Excellence in IT (EXCITE) and by European Social Fund via “ICT programme” measure.

SISUKOKKUVÕTE

Mobiilse värkvõrgu protsessihaldus

Sensorite, aktuaatorite ja mikrokontrolleritega varustatud seadmeid toodetakse aina väiksemates mõõtmetes ja üha soodsamate hindade juures. Tänu sellele sageneb taoliste seadmete integreerimine meid igapäevaelus ümbritsevaisse esemeisse - alates kodumasinatest ja tööstuslikest tootmisliinidest kuni tänavavalgustite ja loomakarjadeni. Nutikatele seadmetele ka sidevõimekuse lisamine (nt WiFi mooduli näol) võimaldab nende igapäevaelu objektide arvutivõrkudega liitmist, kaugjälgimist ja -juhtimist.

Värkvõrgu ehk Asjade Interneti (ingl Internet of Things, lüh IoT) visioon tõotab selle trendi jätkumist ning keskendub sellele, kuidas seadmeid üksteise ja väliste süsteemidega suhtlema panna; ning kuidas hallata süsteeme ja rakendusi, mis nõnda suuri seadmehulki hõlmavad. Varajaste IoT tehnoloogiate kasutajate seas paistavad silma valdkonnad nagu tööstuslik tootmine, logistika ja tervishoid. Need on valdkonnad, kus äriprotsesside haldus (ingl Business Process Management, lüh BPM) on olnud olulisel kohal. BPM teadusharu uurib äriprotsesside seiret, kaardistamist, disaini, analüüsi, automatiseerimist ning täiustamist. Sestap on värkvõrgu populaarsuse tõus äratanud huvi probleemi vastu, kuidas ühildada IoT-d tehnoloogiatega nagu äriprotsesside haldussüsteemid.

BPM on seni olnud edukas selliste äriprotsesside parendamisel, mis kasutavad ära konventsionaalseid ettevõttesüsteeme, viimased järgivad tihti teenusekeskse arhitektuuri (ingl Service Oriented Architecture, lüh SOA) printsiipe. Teenusekeskne lähenemine on oluline ka IoT jaoks, lahendades üht IoT tuumikprobleemidest - ühilduvuse saavutamine heterogeensete seadmete ja nende teenuste vahel. Juhul, kui IoT seadmed pakuvad oma funktsionaalsusi (nt sensorandmete lugemine, mootori lülitamine) standardsete veebiteenustena, on IoT kasutamine BPM protsessides võrdlemisi lihtne.

Samas pole selge, kuidas rakendada BPM-i IoT lahendustele, mis hõlmavad uusi arvutusparadigmasid nagu serva- ja uduarvutus (ingl Edge Computing, Fog Computing). Need paradigmad nihutavad arvutuste, võrgu ja andmetega seonduvaid ülesanded (mis tüüpiliselt paiknevad andmekeskustes) võrgu servas asuvaile seadmetele - nagu näiteks võrku ühendatud autod, nutitelefonid, ruuterid ja lüüsseadmed. Kommunikatsioon servavõrgus toetub lokaalse kohtvõrgu kasutamisele või seadmelt-seadmele (ingl device-to-device) suhtlusele. See aga muudab seadmete mobiilsusega arvestamise suureks väljakutseks, sest pidev levialade vahel liikumine vahel tähendab katkendlikku ühenduvust. Käesolevas töös nimetatakse IoT stsenaariume, mis taolist liikuvust hõlmavad, mobiilseks asjade internetiks (ingl Internet of Mobile Things, lüh IoMT).

Teiseks, uduarvutuse kontseptsioon hõlmab uduserverite kasutuselevõttu, mis peaksid paiknema võrgu servas, lõppkasutajate ja IoT seadmete vahetus läheduses. Tegemist on üldotstarbeliste serveritega, nn mini-andmekeskustega - see

avab võimaluse delegeerida osad arvutuslikud ülesanded IoMT seadmetelt uduserveritele. Ent optimaalne delegeerimisotsus peab arvestama kompromissidega jõudluse, energiatarbimise ja ajakulu vahel. Oletame näiteks, et seadme liikumistrajektoor kulgeb läbi serveri kohtvõrgu leviala vaid põgusalt ning ajal, kui server on koormatud juba ka teiste IoMT seadmete poolt delegeeritud tööde protsessimisega. Sellisel juhul on tõenäosus, et järjekordse töö delegeerimine ebaõnnestub suur ning eelistatud oleks valida mõni teine piirkonnas asuv server.

Käesolev doktoritöö uurib, kuidas võimaldada äriprotsesside käitamist servas, paigaldades protsessikäitusmootorid mobiilseadmetele, muutes nad seeläbi nn mobiilseteks protsessihostideks (MPH). MPH-d suudavad servavõrgus autonoomselt protsesse käitada, kesksest süsteemist (ja ühendusest) sõltumata. Lähenemine kasutab olemasolevaid BPM standardeid ja tehnoloogiaid, nagu nt BPMN 2.0 modelleerimiskeel, mis suurendab tulemuste otsest rakendatavust päris-süsteemidel. Me demonstreerime, kuidas on võimalik erinevaid IoMT näidisrakendusi BPM standarditega modelleerida ja käitada.

Esitleme kahte funktsionaalsust, mis toetavad mobiilseadmetel autonoomset, pidevat ja adaptiivset protsessikäitust: seadmelt-seadmele protsessi-isendite käitusaegne migreerimine; ning plaanimisalgoritm, mis otsustab, milliseid uduservereid servaprotsessides delegeerimisel kaasata.

Seadmelt-seadmele äriprotsesside migreerimine võimaldab protsessikäitusel pidevalt jätkuda ka siis, kui antud äriloogikale olulises asukohas olevate seadmete hulk muutub ja pole garantiid, et leiduks MPH-d, mis oleks alati lähedal kättesaadav. Näiteks kauba transpordil, kui protsess hõlmab kauba sensorite pidevat jälgimist, ent kaup liigub eri organisatsioonide ja isikute (ja nende seadmete levialade) vahel. Migratsioonifunktsioon võimaldab protsessi jätkata eri servaseadmetel, kesksest süsteemist sõltumata. Katsed Android nutitelefonidega näitasid, et migratsioonifunktsionaalsus toimib piisavalt kiiresti, et olla taolistes rakendustes kasutatav.

Plaanimisalgoritm arvestab uduserverite asukohti, MPH liikumistrajektoori ja käitatava arvutuse karakteristikuid, et leida antud hetkel sobivaim uduserver. Analüüsisime algoritmi diskreetse sündmuspõhise simulatsioonitööriistaga ONE erinevates olukordades, varieerides seadmete asukohti, trajektoore ja serverite koormust. Tulemused näitasid, kuidas algoritm parandab energiasäästlikkust ning arvutusi delegeerivate protsesside eduka käitamise tõenäosust.

Läbiviidud uurimustest saadud õppetundide põhjal lõime STEP-ONE nimelise testkeskkonna, et soodustada edasisi serva- ja uduprotsesside haldamisega seonduvaid uuringuid. STEP-ONE on tööriistade komplekt simuleerimaks IoMT süsteeme, kus rakenduste haldus ja kompositsioon põhineb BPMN 2.0 modelleerimiskeelel. STEP-ONE võimaldab protsesse simuleerida realistliku mobiilsusega, näiteks on võimalik protsessimudelites arvestada tänavakaartidel põhinevat liikumistrajektooride infot. Demonstreerime STEP-ONE-i targa linna stsenaariumiga, kus näidisrakendus on modelleeritud protsesside koreograafiana, mille käitamine leiab aset erinevatel ressurssidel - pilves, uduserveritel ja MPH-del. Näitame,

kuidas STEP-ONE toetab selliste stsenaariumite ehitamist, simulatsioonipõhist käitamist ja uuringuid ning kuidas see hõlbustab selliste protsesside adaptiivset juhtimist algoritmide ja otsustusmehhanismidega.

CURRICULUM VITAE

Personal data

Name:	Jakob Mass
Date and Place of Birth:	22. January, 1992, Estonia
Contact:	firstname.lastname@eesti.ee

Education

2016–2022	University of Tartu, PhD in Computer Science
2014–2016	University of Tartu, MSc in Software Engineering
2011–2014	University of Tartu, BSc in Computer Science
1999– 2011	Kadrioru Saksa Gümnaasium

Employment

2020–	University of Tartu, junior lecturer of distributed systems
2017–	Telia Eesti AS, IoT systems developer / intern

Scientific work

Main fields of interest:

- Mobile Application Development
- Fog- and Edge Computing for Internet of Things
- Data integration and Mobility for Smart Cities
- Business Process Management

ELULOOKIRJELDUS

Isikuandmed

Nimi:	Jakob Mass
Sünniaeg ja koht:	22. jaanuar, 1992, Eesti
Kontakt:	eesnimi.perenimi@eesti.ee

Haridus

2016–2022	Tartu Ülikool, PhD Informaatika
2014–2016	Tartu Ülikool, MSc Tarkvaratehnika
2011–2014	Tartu Ülikool, BSc Informaatika
1999– 2011	Kadrioru Saksa Gümnaasium

Teenistuskäik

2020–	Tartu Ülikool, hajussüsteemide nooremlektor
2017–	Telia Eesti AS, IoT süsteemide arendaja-praktikant

Teadustegevus

Peamised uurimisvaldkonnad:

- Mobiilirakenduste arendamine
- Udu- ja servarvutused Asjade Interneti jaoks
- Andmete integreerimine ning mobiilsus targas linnas
- Äriprotsesside haldus

LIST OF ORIGINAL PUBLICATIONS

1. J. Mass, C. Chang, and S. N. Srirama. "WiseWare: A Device-to-Device-Based Business Process Management System for Industrial Internet of Things". In: *2016 IEEE International Conference on Internet of Things (iThings) and IEEE Green Computing and Communications (GreenCom) and IEEE Cyber, Physical and Social Computing (CPSCom) and IEEE Smart Data (SmartData)*. Dec. 2016, pp. 269–275. DOI: 10.1109/iThings-GreenCom-CPSCom-SmartData.2016.69
2. Jakob Mass, Chii Chang, and Satish Narayana Srirama. "Edge Process Management: A Case Study on Adaptive Task Scheduling in Mobile IoT". in: *Internet of Things* 6 (June 2019), p. 100051. ISSN: 2542-6605. DOI: 10.1016/j.iot.2019.100051
3. Jakob Mass, Satish Narayana Srirama, and Chii Chang. "STEP-ONE: Simulated Testbed for Edge-Fog Processes Based on the Opportunistic Network Environment Simulator". In: *Journal of Systems and Software* 166 (Aug. 2020), p. 110587. ISSN: 01641212. DOI: 10.1016/j.jss.2020.110587

DISSERTATIONES INFORMATICAE PREVIOUSLY PUBLISHED IN DISSERTATIONES MATHEMATICAE UNIVERSITATIS TARTUENSIS

19. **Helger Lipmaa.** Secure and efficient time-stamping systems. Tartu, 1999, 56 p.
22. **Kaili Müürisep.** Eesti keele arvutigrammatika: süntaks. Tartu, 2000, 107 lk.
23. **Varmo Vene.** Categorical programming with inductive and coinductive types. Tartu, 2000, 116 p.
24. **Olga Sokratova.** Ω-rings, their flat and projective acts with some applications. Tartu, 2000, 120 p.
27. **Tiina Puolakainen.** Eesti keele arvutigrammatika: morfoloogiline ühestamine. Tartu, 2001, 138 lk.
29. **Jan Villemson.** Size-efficient interval time stamps. Tartu, 2002, 82 p.
45. **Kristo Heero.** Path planning and learning strategies for mobile robots in dynamic partially unknown environments. Tartu 2006, 123 p.
49. **Härmel Nestra.** Iteratively defined transfinite trace semantics and program slicing with respect to them. Tartu 2006, 116 p.
53. **Marina Issakova.** Solving of linear equations, linear inequalities and systems of linear equations in interactive learning environment. Tartu 2007, 170 p.
55. **Kaarel Kaljurand.** Attempto controlled English as a Semantic Web language. Tartu 2007, 162 p.
56. **Mart Anton.** Mechanical modeling of IPMC actuators at large deformations. Tartu 2008, 123 p.
59. **Reimo Palm.** Numerical Comparison of Regularization Algorithms for Solving Ill-Posed Problems. Tartu 2010, 105 p.
61. **Jüri Reimand.** Functional analysis of gene lists, networks and regulatory systems. Tartu 2010, 153 p.
62. **Ahti Peder.** Superpositional Graphs and Finding the Description of Structure by Counting Method. Tartu 2010, 87 p.
64. **Vesal Vojdani.** Static Data Race Analysis of Heap-Manipulating C Programs. Tartu 2010, 137 p.
66. **Mark Fišel.** Optimizing Statistical Machine Translation via Input Modification. Tartu 2011, 104 p.
67. **Margus Niitsoo**. Black-box Oracle Separation Techniques with Applications in Time-stamping. Tartu 2011, 174 p.
71. **Siim Karus.** Maintainability of XML Transformations. Tartu 2011, 142 p.
72. **Margus Treumuth.** A Framework for Asynchronous Dialogue Systems: Concepts, Issues and Design Aspects. Tartu 2011, 95 p.
73. **Dmitri Lepp.** Solving simplification problems in the domain of exponents, monomials and polynomials in interactive learning environment T-algebra. Tartu 2011, 202 p.

74. **Meelis Kull.** Statistical enrichment analysis in algorithms for studying gene regulation. Tartu 2011, 151 p.
77. **Bingsheng Zhang.** Efficient cryptographic protocols for secure and private remote databases. Tartu 2011, 206 p.
78. **Reina Uba.** Merging business process models. Tartu 2011, 166 p.
79. **Uuno Puus.** Structural performance as a success factor in software development projects – Estonian experience. Tartu 2012, 106 p.
81. **Georg Singer.** Web search engines and complex information needs. Tartu 2012, 218 p.
83. **Dan Bogdanov.** Sharemind: programmable secure computations with practical applications. Tartu 2013, 191 p.
84. **Jevgeni Kabanov.** Towards a more productive Java EE ecosystem. Tartu 2013, 151 p.
87. **Margus Freudenthal.** Simpl: A toolkit for Domain-Specific Language development in enterprise information systems. Tartu, 2013, 151 p.
90. **Raivo Kolde.** Methods for re-using public gene expression data. Tartu, 2014, 121 p.
91. **Vladimir Šor.** Statistical Approach for Memory Leak Detection in Java Applications. Tartu, 2014, 155 p.
92. **Naved Ahmed.** Deriving Security Requirements from Business Process Models. Tartu, 2014, 171 p.
94. **Liina Kamm.** Privacy-preserving statistical analysis using secure multi-party computation. Tartu, 2015, 201 p.
100. **Abel Armas Cervantes.** Diagnosing Behavioral Differences between Business Process Models. Tartu, 2015, 193 p.
101. **Fredrik Milani.** On Sub-Processes, Process Variation and their Interplay: An Integrated Divide-and-Conquer Method for Modeling Business Processes with Variation. Tartu, 2015, 164 p.
102. **Huber Raul Flores Macario.** Service-Oriented and Evidence-aware Mobile Cloud Computing. Tartu, 2015, 163 p.
103. **Tauno Metsalu.** Statistical analysis of multivariate data in bioinformatics. Tartu, 2016, 197 p.
104. **Riivo Talviste.** Applying Secure Multi-party Computation in Practice. Tartu, 2016, 144 p.
108. **Siim Orasmaa.** Explorations of the Problem of Broad-coverage and General Domain Event Analysis: The Estonian Experience. Tartu, 2016, 186 p.
109. **Prastudy Mungkas Fauzi.** Efficient Non-interactive Zero-knowledge Protocols in the CRS Model. Tartu, 2017, 193 p.
110. **Pelle Jakovits.** Adapting Scientific Computing Algorithms to Distributed Computing Frameworks. Tartu, 2017, 168 p.
111. **Anna Leontjeva.** Using Generative Models to Combine Static and Sequential Features for Classification. Tartu, 2017, 167 p.
112. **Mozhgan Pourmoradnasseri.** Some Problems Related to Extensions of Polytopes. Tartu, 2017, 168 p.

113. **Jaak Randmets.** Programming Languages for Secure Multi-party Computation Application Development. Tartu, 2017, 172 p.
114. **Alisa Pankova.** Efficient Multiparty Computation Secure against Covert and Active Adversaries. Tartu, 2017, 316 p.
116. **Toomas Saarsen.** On the Structure and Use of Process Models and Their Interplay. Tartu, 2017, 123 p.
121. **Kristjan Korjus.** Analyzing EEG Data and Improving Data Partitioning for Machine Learning Algorithms. Tartu, 2017, 106 p.
122. **Eno Tõnisson.** Differences between Expected Answers and the Answers Offered by Computer Algebra Systems to School Mathematics Equations. Tartu, 2017, 195 p.

DISSERTATIONES INFORMATICAE UNIVERSITATIS TARTUENSIS

1. **Abdullah Makkeh**. Applications of Optimization in Some Complex Systems. Tartu 2018, 179 p.
2. **Riivo Kikas**. Analysis of Issue and Dependency Management in Open-Source Software Projects. Tartu 2018, 115 p.
3. **Ehsan Ebrahimi**. Post-Quantum Security in the Presence of Superposition Queries. Tartu 2018, 200 p.
4. **Ilya Verenich**. Explainable Predictive Monitoring of Temporal Measures of Business Processes. Tartu 2019, 151 p.
5. **Yauhen Yakimenka**. Failure Structures of Message-Passing Algorithms in Erasure Decoding and Compressed Sensing. Tartu 2019, 134 p.
6. **Irene Teinemaa**. Predictive and Prescriptive Monitoring of Business Process Outcomes. Tartu 2019, 196 p.
7. **Mohan Liyanage.** A Framework for Mobile Web of Things. Tartu 2019, 131 p.
8. **Toomas Krips.** Improving performance of secure real-number operations. Tartu 2019, 146 p.
9. **Vijayachitra Modhukur.** Profiling of DNA methylation patterns as biomarkers of human disease. Tartu 2019, 134 p.
10. **Elena Sügis.** Integration Methods for Heterogeneous Biological Data. Tartu 2019, 250 p.
11. **Tõnis Tasa.** Bioinformatics Approaches in Personalised Pharmacotherapy. Tartu 2019, 150 p.
12. **Sulev Reisberg.** Developing Computational Solutions for Personalized Medicine. Tartu 2019, 126 p.
13. **Huishi Yin.** Using a Kano-like Model to Facilitate Open Innovation in Requirements Engineering. Tartu 2019, 129 p.
14. **Faiz Ali Shah.** Extracting Information from App Reviews to Facilitate Software Development Activities. Tartu 2020, 149 p.
15. **Adriano Augusto**. Accurate and Efficient Discovery of Process Models from Event Logs. Tartu 2020, 194 p.
16. **Karim Baghery.** Reducing Trust and Improving Security in zk-SNARKs and Commitments. Tartu 2020, 245 p.
17. **Behzad Abdolmaleki.** On Succinct Non-Interactive Zero-Knowledge Protocols Under Weaker Trust Assumptions. Tartu 2020, 209 p.
18. **Janno Siim.** Non-Interactive Shuffle Arguments. Tartu 2020, 154 p.
19. **Ilya Kuzovkin.** Understanding Information Processing in Human Brain by Interpreting Machine Learning Models. Tartu 2020, 149 p.
20. **Orlenys López Pintado.** Collaborative Business Process Execution on the Blockchain: The Caterpillar System. Tartu 2020, 170 p.
21. **Ardi Tampuu.** Neural Networks for Analyzing Biological Data. Tartu 2020, 152 p.

22. **Madis Vasser.** Testing a Computational Theory of Brain Functioning with Virtual Reality. Tartu 2020, 106 p.
23. **Ljubov Jaanuska.** Haar Wavelet Method for Vibration Analysis of Beams and Parameter Quantification. Tartu 2021, 192 p.
24. **Arnis Parsovs.** Estonian Electronic Identity Card and its Security Challenges. Tartu 2021, 214 p.
25. **Kaido Lepik.** Inferring causality between transcriptome and complex traits. Tartu 2021, 224 p.
26. **Tauno Palts.** A Model for Assessing Computational Thinking Skills. Tartu 2021, 134 p.
27. **Liis Kolberg.** Developing and applying bioinformatics tools for gene expression data interpretation. Tartu 2021, 195 p.
28. **Dmytro Fishman.** Developing a data analysis pipeline for automated protein profiling in immunology. Tartu 2021, 155 p.
29. **Ivo Kubjas.** Algebraic Approaches to Problems Arising in Decentralized Systems. Tartu 2021, 120 p.
30. **Hina Anwar.** Towards Greener Software Engineering Using Software Analytics. Tartu 2021, 186 p.
31. **Veronika Plotnikova.** FIN-DM: A Data Mining Process for the Financial Services. Tartu 2021, 197 p.
32. **Manuel Camargo.** Automated Discovery of Business Process Simulation Models From Event Logs: A Hybrid Process Mining and Deep Learning Approach. Tartu 2021, 130 p.
33. **Volodymyr Leno.** Robotic Process Mining: Accelerating the Adoption of Robotic Process Automation. Tartu 2021, 119 p.
34. **Kristjan Krips.** Privacy and Coercion-Resistance in Voting. Tartu 2022, 173 p.
35. **Elizaveta Yankovskaya.** Quality Estimation through Attention. Tartu 2022, 115 p.
36. **Mubashar Iqbal.** Reference Framework for Managing Security Risks Using Blockchain. Tartu 2022, 203 p.