Department of Informatics
Technical University of Munich

TECHNICAL UNIVERSITY OF MUNICH

DEPARTMENT OF INFORMATICS

MASTER'S THESIS IN INFORMATICS

Optimization of IoT service placement

Paulius Šukys

Technical University of Munich

Department of Informatics

Master's Thesis in Informatics

Optimization of IoT service placement

Optimierung von Dienstplatzierung in IoT Umgebungen

Author:	Paulius Šukys
Supervisor:	Prof. Dr.-Ing. Georg Carle
Advisor:	Dr.-Ing Marc-Oliver Pahl
	M. Sc. Stefan Liebald
Date:	March 27, 2019

I confirm that this Master's Thesis is my own work and I have documented all sources and material used.

Garching, March 27, 2019

Location, Date	Signature

Abstract

Internet of Things (*IoT*) heterogeneous computing environments introduce a challenge for executing multiple services and maintaining quality of service. A way for maintaining and improving is by managing the location of deployed services. By analysing the domain and related areas for service placement, this thesis overviews the impact of placement strategies. Distributed Smart Space Orchestration System (*DS2OS*) is focused as a base system to verify the design and implementation. Evaluation relied on a simulation framework, that imitates relevant *DS2OS* components, and has shown that placement strategies are not significant where execution relies on a single service, and has impact in lowering runtime for executions that rely on composed services. Finally, integration and evaluation limitations are discussed, as well as feasible feature work directions.

Zusammenfassung

IoT verschiedenartige Rechenumgebungen stellten eine Herausforderung für mehrere Dienstdurchfuhrüngen und erhaltung der Dienstleistungsqualität dar. Ein Verfahren zur Wartung und Verbesserung ergibt sich aus Verwaltung der Dienststandortes. Die Masterarbeit gibt einen Überblick über die Auswirkungen mit Domain-Analyse für verwandate Dienstplatzierung Forschungsbereiche dar. *DS2OS* ist fokusiert wie ein Grundlage für Design und Bewertung. Die Bewertung wurde im Simulationsrahmen, die relevanten *DS2OS* Komponenten hat, durchgeführt. Es hat sich gezeigt, dass die Dienstplatzierungsstrategie keine Bedeutung für einzelne Dienstausführung hat, jedoch eine niedrigere Laufzeit für zusammengesetzte Dienstleistungen aufweist. Zum Schluss werden die Integrations,- und Bewertungsgrenzen, sowie machbare zukünftige Arbeitsrichtungen diskutiert.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

Pervasive computing is a concept where computing is ubiquitous. A common part of applications that follow this concept fall under the domain of *IoT*. Here devices come in all kinds of sizes and capabilities. Due to their variety and resource limitations, it is in the interest of the operator to obtain better utilization.

In a similar fashion, microservice based application comprises of multiple services, and it is only natural that a set of services need to closely cooperate. By lining resource usage against specific goals to improve system's performance or efficiency on price, average load, energy usage, microservice based applications can perform better.

By combining the approaches of microservices for service composition, it is worthy to investigate its effects for *IoT* environments, to reap the benefits that microservices attempt to gain.

The goal of this thesis is exploring the possibilities and effects of service location adjustment application in *IoT*, by attempting to combine the domains of *IoT* and *microservices.* With evaluation of what the constraints and possibilities of both domains are through analysis, and relevant related work, thesis seeks to provide an evaluation whether introduction of service placement methods provide any improvements.

1.1 Methodology

As this thesis covers two distinct domains, namely general service placement and *IoT*. By firstly analyzing the domains of interest and saturating out research question, a set of requirements would be set for reviewing related work.

Related work initially relies on areas of relevance drawn from analyzed domains, as well as unstructured search that comes as advised literature from advisers. By evaluating literature overview based papers and further investigating their references, requirement fulfillment can be assessed.

Thesis structure continues with chapter 2 for overviewing domains of interest and building research questions based on aforementioned analysis. Then, in chapter 3, related work analysis is conveyed on top of requirements for research questions. By evaluating common and fulfilled requirements in related works, design can be formalized in chapter 4. The details of design implementation get documented in chapter 5. Finally, evaluation of implemented design is given in chapter 6 to indicate whether service placement has significance in IoT environments. Conclusion in chapter 7 recaps thesis contents and future work prospects are discussed.

Chapter 2

Analysis

This chapter overviews the landscape for service placement in *IoT* applications. The problem connects two domains of requirements - what are the problems and respective approaches that occur with services; and what constraints does IoT introduce for service placement. To answer these question, this chapter is layed by sequentially analyzing constraints in both domains, application framework for IoT service management (Section 2.4) and general approaches for service placement. Firstly, *IoT* domain is overviewed in section 2.1. As a step back, services themselves need to be analyzed and this is done in section 2.2. Following comes an *IoT* middleware framework *DS2OS* (Section 2.4). Finally, to tackle the service placement, resource allocation problem is stated and domains that attempt to solve it for general service placement are discussed in sections 2.5 and 2.6. Section 2.7 summarizes and granulates research questions that arise from conducted analysis.

The methodology of conveying analysis is based on saturating challenges that come of relevance of service placement in each analyzed domain. These challenges are aggregated and converted to research questions of this thesis. Looking ahead into related work, research questions are too general, therefore a set of requirement objectives are drawn from research questions. Specifics of how challenges convert to research question and they - to requirement objectives are discussed in section 2.7.

2.1 Internet of Things

“Internet of Things is a combination of a technological *push* and a human pull for more and ever-increasing connectivity with anything happening in the immediate and wider environment – a logical extension of the computing power in a single machine to the environment: *the environment as an interface*"[18]. The *Environment as an Interface* builds an understanding that surrounding objects are capable of transmitting and receiving data. As these interfaces can easily be built, it is but inevitable that management and development research emerges for IoT systems and architectures.

One way of developing an *IoT* system is *IoT reference model* published by [1]. Generally, *IoT reference model* [1] is used to instruct how to compose an *IoT* based system. The proposed model is based on 7 layers:

1. **Collaboration & Processes** - Businesses and clients
2. **Application** - Reports, analytics & control.
3. **Data Abstraction** - Aggregation and access
4. **Data Accumulation** - Storage
5. **Edge (Fog) Computing** - Data analysis & Transformation
6. **Connectivity** - communication
7. **Physical devices & Controllers**

The layers are designed to decouple logic from data (operational from informational) technologies. List in descending order aggregates and/or transforms data from one layer to another. Here control and policies can only be set from ascending order layers, i.e. *Connectivity* layer controls and introduces policies to *Physical devices & Controllers* layer.

As systems vary in environment and device characteristics, loose coupling allows the system to be operational by separating intents into separate components.
Challenge 1 (Decoupled IoT) *IoT devices are loosely coupled.*

"The term Internet of Things (IoT) refers to the interconnection of small devices able to interact with each other and cooperate in order to accomplish common tasks"[32]. While physical size and resource ability of a device may vary, the task might have estimated requirements for devices. This means that not every device can run a specific task due to not being able to provide amount or type of resources needed. On the other hand, devices are heterogeneous software-wise as well: using specific architectures, system packages, and libraries.
Challenge 2 (Heterogeneous IoT) *IoT devices have heterogeneous characteristics.*

With the diversity of abilities each *IoT* device contains, it is assumed that these characteristics are managed, and this is out of scope of this thesis.

2.2 Services

Ideally, a service is a computational unit that performs a specific task. In cloud computing context, services are supposed to be flexible by adapting their ability to function regardless of the environment. Due to the sheer amount of applications being used, it was inevitable motivation for packaging, deployment, configuration, and most importantly architectural pattern development. Further sections overview common architectural approaches in general cloud computing.

2.2.1 Service Oriented Architecture

Service Oriented Architecture (*SOA*) is a software development design, where the core element is a service. *SOA* specifically discusses variants of services, their usage, and how they are managed independently and in clusters.

SOA has six core values[1]:

1. Business value over technical strategy
2. Strategic goals over project-specific benefits
3. Intrinsic interoperability over custom integration
4. Shared services over specific-purpose implementations
5. Flexibility over optimization
6. Evolutionary refinement over pursuit of initial perfection

1 and 2 are oriented towards ability to develop and maintain applications in an agile manner. 3 and 4 state looseness of services and being independent of service local implementation in order to provide same functionality. Finally, 5 and 6 relate to first two by prioritizing rapid development and good maintainability.

While some of the mentioned values correspond to the development process itself, the relevance of *SOA* to this thesis is the architectural concept that allows to develop and maintain services in a scalable and sustainable way.

[1] As published in http://www.soa-manifesto.org

2.2.2 Microservices

While *SOA* is the base design principle, similar conceptual designs emerged, one of them being Microservices. Microservices propose a loosely coupled architecture, where separated services communicate through standardized protocols. This modularity is beneficial for developers, when only a single component is needed to be modified. Main difference from *SOA* is service independence from others, thus providing better fault-tolerance.

Microservices attempt to have services loosely coupled, just like how devices are in *IoT* domain.
Challenge 3 (Loosely coupled Microservices) *Microservice based services need to be loosely coupled and independent.*

This akin feature gives insight that likely both domains tackle similar problems. Services can be deployed onto different devices, and thus be interpreted the same way as services are in microservices.

2.3 Virtual State Layer

Virtual State Layer (*VSL*) is a programming abstraction introduced by [25], "its purpose is providing all kinds of services in the service domain with the context they need to reach their goals". *VSL* simplifies communication by providing semantic search to resolve targets. Context Model (*CM*) is used for assisting in semantic search within available services.

In [25] context models are "templates for representing properties of the physical world in the virtual world". *CM*s are shared over a global Context Model Repository (*CMR*). For accessing context models, hierarchical addressing is used in the following format:

```
/grandparent/parent/child
```

The context is represented as context nodes, they provide the means for services to hook into specific context and expose their functionality there. As the importance of exposed functionality might be of different severity, it is possible to store the values for context nodes inside the Knowledge Tree, and to supply them on-demand. *VSL* has two types of nodes:

1. regular - coupling through subscription mechanism, values persisted in Knowledge Tree,

2. virtual - direct coupling to service, values are queried on-demand from a service itself.

By providing these means, *VSL* helps to simplify complex application communication.

2.4 The Distributed Smart Space Orchestration System

[25]'s *DS2OS* is a framework that is developed with the help *VSL* (Section 2.3) middleware.

It is an extensive and feasible application framework to enable simplified and secure *IoT* application communication and management.

DS2OS adds functionality for: service management; global application store; and further support for crowdsourced software development. Three main components of *DS2OS* are:

1. *VSL* (Section 2.3) as communication middleware.
2. Smart Space Service management (*S2S*) for managing VSL services.
3. Smart Space Store (*S2Store*) (Section 2.4.8) crowdsourced software development based global service management.

2.4.1 DS2OS Hierarchy

DS2OS implements hierarchical management with 5 core components, as shown in Figure 2.1. Abbreviations used in figures:

- Knowledge Agent (*KA*) - *VSL* communication endpoint, described in section 2.4.3.
- Service Hosting Environment (*SHE*) - low-level service management functionality: service control and monitoring in Real-Time Environment (*RTE*). Described in section 2.4.4.
- Node Local Service Manager (*NLSM*) - node-local service management: start, stop, pause, and monitor VSL services in the node, described in section 2.4.5.
- Service Local Service Manager (*SLSM*) - manages NLSMs within a site. It provides service management and resource usage optimization, described in section 2.4.6.
- *S2Store* - *CMR*, collects, publishes statistics from sites. Described in section 2.4.8.

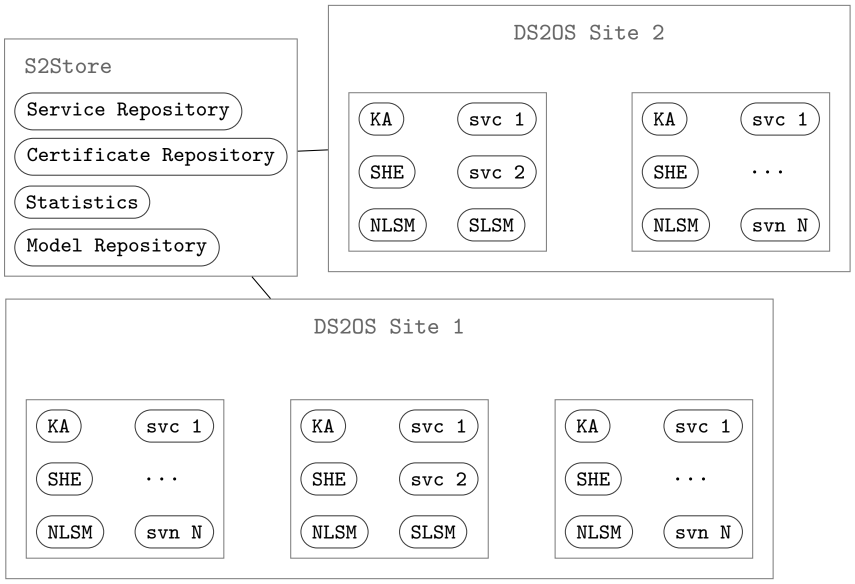

Figure 2.1: DS2OS hierarchy

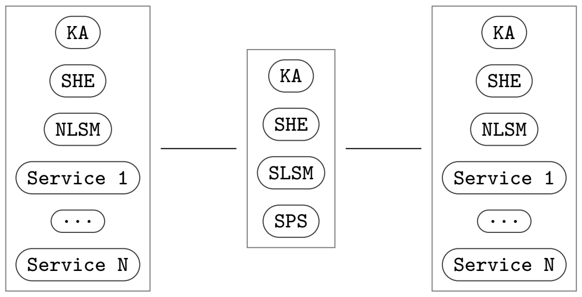

Figure 2.2: DS2OS site structure

2.4.2 DS2OS site

Figure 2.1 refers to overall structure, whereas this thesis orients towards service deployment within a *DS2OS* site, as shown in Figure 2.2:

In Figure 2.2 each box represents a single host. Each box's *NLSM* uses its own *KA* to communicate to *SLSM* through *SLSM*'s *KA*. The following sections overview each of *DS2OS* site's components.

2.4.3 Knowledge Agent

KA is a separate instance that enables communication, and manages its connectivity to other *KA*s and thus contributes in maintaining a knowledge tree. A knowledge tree is a specific data structure and can be generally regarded as a database. As services connect to *KA*s, they are able to query data from other *KA*s and publish themselves in their own *KA*'s domain.

2.4.4 Service Hosting Environment

SHE interacts and manages the execution environment. When *NLSM* receives a service to be deployed, *SHE* prepares and starts the service. In general, *SHE* is responsible for service executable execution management. SHE utilizes Open System Gateway initiative (*OSGi*), that specifies dynamic component usage for Java applications. Further analysis and implementation were carried on in [24].

2.4.5 Node Local Service Management

NLSM, in contrast to *SHE*, manages services as *VSL* services. Maintains communication with SLSM and receives services to be deployed. By *VSL*'s separation of service logic from service state, which is stored in *KA*, thus *NLSM* can persist its state through restarts. This allows seamless restarts on failure.
Challenge 4 (Dynamic DS2OS) *NLSMs join and leave DS2OS site anytime.*

2.4.6 Site Local Service Management

SLSM manages *DS2OS* site and is responsible for various functionalities:

1. Maintain communication with *NLSM*s.
2. Maintain local service repository, which is updated from *S2Store*.
3. Deploy and manage running services on *NLSM*s:
   (a) deploy by specific constraints.
   (b) migrate services in-between *NLSM*s.
   (c) stop, and remove services from NLS*NLSM*s.
4. Monitor NLSMs and services running in them

2.4.7 Site Local Certificate Authority

Site Local Certificate Authority (*SLCA*) is responsible for managing *DS2OS* site's certificates. It autonomously provisions certificates within the site, by renewing old certificates, and issuing new ones to running services in the site.

Further analysis and implementation was done in [6]

2.4.8 Smart Space Store

S2Store is a centralized instance that is considered as upstream service repository. The repository houses three types of resources relevant to a *DS2OS* site (See Figure 2.1):

1. Context Models, described in 2.3.
2. Services, which are deployed into *NLSM*s.
3. Certificate repository

As discussed in Section 2.4.6, *SLSM*s update their local repositories from *S2Store*.

2.4.9 DS2OS Site Implementation

The combined implementation by [3][24][6] for *DS2OS* site components of *SLSM*, *NLSM*, *S2Store*, *SHE*, *SLCA* had limited documentation and compilable source code resulting in the system being non-functioning. While contents of their Theses described mechanisms, the intended design and execution details were non trivial to map to actual implementation.

With these problems and failure to reproduce even minimally functioning *DS2OS* site environment, I and *M.Sc.. Christian Luebben* attempted to rectify it.

Main issues were accumulated within source code repository issue system and a summary of them is as follows:

1. Hard-coded absolute file paths that refer to author local file systems for dependencies - configurations, certificates, working directories,
2. Communication misuse leading to overloaded systems,
3. Uncontrolled life-cycles leading to over-utilization where the service should be on idle,
4. Failing service package transmission.
5. No development no documentation and usage system usage instructions
6. No functionality verifying tests or workflows

Most of the issues were minor changes, without implementation structure deviation.

Main problem was [24] implementation as *SHE* lacked *OSGi* service management implementation for running and managing *VSL* based services. Namely, there was no service bootstrapping, as typically a service needs information about *KA* and valid certificates to be able to communicate. As well, *SHE* implementation used generic *OSGi* templates, which did not have any *VSL* dependencies supplied.

To resolve this, I created a template *OSGi VSL* service that was the final missing piece to execute whole implementation stack. Due to lack of time, no further architectural fixes and implementations were conveyed, and hard-coded values, to workaround missing SHE resolution mechanisms (bootstrapping new services), were applied.

Most of the system services did not have sane application life-cycles and their execution yielded full utilization of host's processing power. To fix these problems, artificial stops and architectural changes needed to be applied, namely forcing the responsible thread to sleep for a short amount of time, and creating event-based main-loops.

Most and yet not sufficient work was applied onto service packages. They:

1. Contained incompatible artifacts,
2. Were not able to be deployed multiple times,
3. Had flawed communication between services.

A lot of the problems, namely artifact compatibility and fixes in communication between services, were fixed, yet due to time limitations the rest are documented as issues. Due to the problems with target framework contained, evaluation uses simulation framework which implements relevant *DS2OS* parts in section 6.1.

2.4.10 Service placement

Current implementation of *DS2OS* implementation of *service placement* relies on load balancing: services are deployed to the least utilized *NLSM*.
Challenge 5 (DS2OS resource usage interpretation) *DS2OS service placement treats resource usage of equally.*

While the utilization formula compares sums of memory bytes and Central Processing Unit (*CPU*) relative (percentage) usage, the underlying design ideas remain clear.

2.5 Resource Allocation Problem

Resource allocation is a problem common not only in computer science, but also in economics [9][33].

As resource allocation is generally solved as a single operation for resources, without frequent recurrence, a common definite optimal solution is done by applying linear programming ($($LP)).

Linear Programming (LP) uses a mathematical model that contains linear relationships to produce an optimal solution. In essence, linear programming: contains a cost function, which needs to be maximized or minimized, depending on function's goal; a set of linear function constraints; and variable constraints (most commonly that some variables cannot be negative). It can be formalized as follows:

Cost function f:

$$f(x_1, x_2, ..., x_n) = c_1x_1 + c_2x_2 + ... + c_n + x_n$$

Given constraints:

$$a_11x_1 + a_12x_2 + ... + a_1nx_n < b_1$$

$$a_21x_1 + a_22x_2 + ... + a_2nx_n < b_2$$

$$...$$

$$a_n1x_1 + a_n2x_2 + ... + a_nnx_n < b_n$$

and given variable constraints:

$$x_1 >= 0$$

$$...$$

$$x_n >= 0$$

Notable that in above descriptions, given the conditions, amount of constraints and variable constraints can vary.

2.6 Service placement

In the heart of this thesis, is the problem of *service placement*. The following subsections overview various domains that attempt to solve service placement, and later attempt compare problems that arise with these domains and *IoT* domain. In this case, service placement also includes migrations thus including continuous service placement planning.

2.6.1 Service Composition

Service Composition describes problem domain for bundling services together in order to fulfill a common task. [16] conveyed a systematic literature review, and outlined the following issues that service composition attempts to solve:

1. **User requirement satisfaction**
2. **Quantitative QoS**
3. **Algorithm improvements**
4. **Data storage and indexing structures**
5. **Self-adaptability, automatism, reliability and accuracy, and quality assurance**
6. **QoS mathematical models**
7. **Revenue maximization**
8. **Service discovery optimization**
9. **New frameworks and structures**

As all of the listed entries can be considered as relevant for service placement, it quickly becomes clear that a combination that solving all of the listed out domains is out of scope for this thesis.

Issue 1 tackles allowing users to describe their requirements and means to abide to them. Qualitative Quality of Service (*QoS*) is simpler to define, yet more difficult to grasp, thus issue 2 attempts to transform from qualitative to quantitative *QoS* parameters.
Challenge 6 (Service composition QoS optimization) *Service composition relies on QoS parameter optimization.*

As Cloud Computing Service Composition (*CCSC*) is defined as NP-hard problem, algorithmic advances are significant either by using heuristic or non-heuristic methods in issue 3. To support scalability and recurrent composition, issue 4 attempts to solve *CCSC* efficiently with data storage and indexing structures.
Challenge 7 (Service composition algorithms) *Service composition relies on algorithmic and data structure solutions.*

Issue 5 tackles reliability in providing service and tooling automatization, which is mostly user oriented. By using mathematical *QoS* to investigate all relevant aspects, issue 6 tries to model service composition better. Again, by orienting towards user goals, issue 7 attempts models the composition goals as revenue based and tries to maximize them. In cases where service composer does not automatically register services on predefined requirements, issue 8 rectifies this by proposing discovery optimizations.
Challenge 8 (Service composition service discovery) *Service composition needs service characterization for service discovery.*

Finally, issue 9 groups proposals that provide full-blown solutions that might combine partial solutions to aforementioned issues.

2.6.2 Service Selection

Service selection is a problem domain, where services are ranked and filtered based on preset set of constraints.

[34] distilled five major service selection topics:

1. **Decision-making techniques**,
2. **Data representation models**,
3. **Service parameters and characteristics**,
4. **Contexts**,
5. **Purposes**.

Topic 1 outlines algorithmic solutions that given specific input provides that provides refined output for service selection.
Challenge 9 (Service selection decision making) *Service selection relies on algorithmic decisions.*

Alternatively, topic 2 overviews service characterization options, such as semantics and providing models to represent data. Topic 3 generally overviews what service evaluation metrics are useful for service selection.

Challenge 10 (Service selection characterization) *Service selection characterizes services, their contexts and goals.*

These metrics are not concrete values, but derivatives that represent some non-functional requirement, i.e. security, performance, accessibility, usability. Topic 4 groups actuality of service selection for a given context: Infrastructure as a Service (*IaaS*), Platform as a Service (*PaaS*), Software as a Service (*SaaS*), and general. Grouping by service selection goals is done in topic 5, which contains single tenant, composite, and multi-tenant service selection.

2.6.3 Cloud Brokerage

Cloud broker is a service that connects cloud service providers with consumers through a unified interface. This allows to integrate several cloud service providers seamlessly to consumer. As cloud service providers offer heterogeneous resources differently, broker maps them to internal model.

[8] overviewed cloud brokerage approaches. The paper provided an extended overview on how cloud brokerage solution have been developed. An important issue, bidding style for brokers were identified as the following:

1. **Resource-driven** - user defines resource requirements,
2. **Deadline-driven** - user defines execution goals,
3. **QoS-oriented** - user states certain *QoS* requirements,
4. **Budget-based** - where usage is priced and monetary budget is stated,
5. **Resale** - broker buying from vendors and reselling to users,
6. **Marketplace** - both providers and users place bids and broker is responsible for matching.

Challenge 11 (Cloud brokerage goal optimization) *Cloud brokerage uses various methods to provide optimal service.*

As some of the bidding styles rely on resource requirements and Service Level Agreement (*SLA*) bind brokers to provide a agreed quality of services, [8] identified that monitoring is conducted in hardware, Virtual Machine (*VM*), and service levels.

Challenge 12 (Cloud brokerage metrics as descriptors) *Cloud brokerage uses various level metrics for service description.*

2.6.4 Network Functions Virtualization (NFV)

Network Function Virtualization (*NFV*) was introduced by [26]. It is a network architecture for modularizing communication services as Virtual Network Function (*VNF*)s. This is akin to the concept of Microservices (Section 2.2.2). [26] technical challenges *NFV* faces are as follows:

1. **Portability/Interoperability** - a unified interface to decouple software from hardware,
2. **Performance Trade-off** - as vendor software is most likely optimized for their hardware, performance degradation is inevitable,
3. **Migration and co-existence of legacy & compatibility with existing platforms** - ability to work with hybrid networks composed of virtual and physical appliances,
4. **Management and Orchestration**,
5. **Automation** - for being able to seamlessly scale,
6. **Security & Resilience** - the introduction of *NFV* should not degrade security, availability, and resilience,
7. **Network Stability** - *NFV* and scaling of virtual appliances should not impact network stability,
8. **Simplicity** - virtualized network platforms being simpler to operate,
9. **Integration** - ability to "mix&match" appliances from different vendors/hypervisors

A set of ordered *VNF*s build a Network Service (*NS*), which represents a complex, specific service.

[10] conducted a comprehensive survey on Resource Allocation in NFV (*NFV-RA*). Authors stated that *NFV* is still forming and further developments can yield new challenges, and their paper overviews current ones and attempts to solve them. They identified, that *NFV-RA* is composed of three stages:

1. *Chain composition (VNFs-CC)* - finding an ordered chain *VNF*s,
2. *Forwarding Graph Embedding (VNFs-FGE)* - finding where *VNF*s should be placed,
3. *Scheduling (VNFs-SCH)* - finding parallel execution orders for *VNF*s to reduce execution time.

It is stated that while these stages have rather distinct results, it is possible to optimize their functionality by *coordinating* them. This would mean running two stages at the same and gradually integrating one's results into other. *Coordination* is stated as one of the challenges that *NFV* should tackle.

As *NS*s represent an ordered set of *VNF*s, *VNFs-CC* attempts to compose them into a chain based on *VNF* characteristics. *VNFs-FGE* being the next step, attempts to map *VNF*s in network infrastructure based on node or in-between the nodes characteristics. Finally, *VNFs-SCH* attempts to optimize runtime of several *NS*s that run on the same infrastructure and possibly share same nodes.

[10] outlined that *VNFs-FGE* are goal based (e.g. reliability, availability, and performance). Given that *NFV* is a pretty saturated domain, each *VNF*s can be of computing, storage, or networking type. Each has an implicit hardware and QoS requirements attached to it, thus the types can be interpreted as generalized requirement groups.

2.6.5 Virtual Machine Migration

Providing a homogeneous environment for running the applications - *VM* - it became possible to virtualize resources and provide them in a single machine seamlessly. Regardless, applications inside VMs do not utilize their resources to their full extent, therefore making machines under-loaded. It is in the interest of infrastructure providers to benefit from cost efficiency by running more infrastructure and thus having higher utilization.

VM migration is a solution that helps in achieving better utilization. In [46] an effective cloud resource management system is of the following characteristics:

1. Estimates the risk of overload;
2. Identifies the best *VM*s to migrate;
3. Identifies the most appropriate target physical machine to migrate to;
4. Minimizes the impact on the underlying infrastructure.

Here 1 is an analytical result as a metric for determining for sensitive a specific resource is to overload. 2 is for mainly identifying either with the help of 1 or other means that a specific set of *VM*s should be migrated. As well by the help of 1 and external tooling 3 searches for an appropriate and compatible machine. Finally, since migration is a infrastructure management procedure not accounted to the user, the importance of *VM*s migration overhead is tackled in 4.

2.6.6 IoT

Autonomic computing refers systems being able of self-management. In the context of *IoT*, edge devices are difficult to manage manually due to heterogeneity and decentralization. *Autonomic computing* also means that a set of edge devices are capable achieving a solution without having a centralized manager.

From self-management systems to systems that are orchestrated in big scale *Fog computing* tackles this. It is an architecture that relies on edge devices to convey full or partial computations that are passed into next computational layer. Layered computations process several, thus making edge devices as virtual resources that fulfill some computational stage - parsing, data transformation, aggregation, analysis, or run computations that require specific equipment. [23] overviewed state-of-the-art and research challenges in *Fog computing.* They indicated that two major groups of research conducted was in Fog computing algorithms, and architectures. Not only did they review other surveys and papers, but also composed a sophisticated criterion system for evaluating research papers. The criteria had 2 aforementioned groups (dimensions) - architectural and algorithmic:

1. Heterogeneity - ability to cope with very heterogeneous devices,
2. *QoS* management - due to varying environment, *QoS* has to be managed,
3. Scalability - ability to elastic resource scaling,
4. Mobility - ability to provide service for mobile nodes (join or leave network),
5. Federation - provisioning several providers that run local fog domains,
6. Interoperability - provider interoperability.

Differently than *Fog computing*, Mobile Edge Computing (*MEC*) pushes the whole services into *edge devices.* [20] identified that *Edge Computing* was suffering unconstrained battery consumption, execution and communication delays, and stated that *MEC* emerged to cope with delay problems. Three main groups of research were distinguished:

1. Decision on computational-offloading,
2. Allocation of computing resources,
3. Mobility management.

1 helps to identify whether offloading is profitable in terms of energy consumption, and execution delay. When computation is offloaded, 2 helps to efficiently allocate

resources, and balance load. 3 refers to reaction to edge device parameters changing: getting out/into *MEC*, having worse/better communication characteristics.

2.7 Requirements

To reiterate challenges from analysis, the list with challenge descriptions and their locations is as follows:

Challenge 1	Decoupled IoT	4
Challenge 2	Heterogeneous IoT	4
Challenge 3	Loosely coupled Microservices	6
Challenge 4	Dynamic DS2OS	9
Challenge 5	DS2OS resource usage interpretation	11
Challenge 6	Service composition QoS optimization	13
Challenge 7	Service composition algorithms	14
Challenge 8	Service composition service discovery	14
Challenge 9	Service selection decision making	14
Challenge 10	Service selection characterization	15
Challenge 11	Cloud brokerage goal optimization	15
Challenge 12	Cloud brokerage metrics as descriptors	15

In order to better reflect these challenges when reviewing related work, the research questions were formed.

RQ1. What characteristics are needed for optimal *DS2OS service placement*?

- **a)** What heterogeneous node characteristics are important?
- **b)** What service specific characteristics are important?
- **c)** How can varying characteristics be taken into account in the general optimization strategy?

RQ2. How can service deployment be improved to provide better execution performance?

- **a)** How can **RQ1.** characteristics be used to elevate decision making for *service placement*?

b) What methodologies can be used to run an online *service placement and migration* strategy?

c) How can the methodologies be designed to be feasible to run in *IoT* edge devices?

d) How significant is placement strategy's ability to scale with the increase of *IoT* edge devices?

RQ3. What effect does the proposed design and implementation have regarding optimization strategies such as *least-nodes*, and *resource balancing* heuristics?

a) *least nodes* - attempt to place all services in as little nodes as possible

b) *resource balancing* - attempt to achieve a balanced resource usage between all nodes.

Challenges 1, and 3 correspond well with the thesis target domain, which is described by challenge 4 - allowing *NLSMs* to dynamically join and leave the site, thus it being loosely coupled.

Challenges 2, 6, 10, and 12 form **RQ1.** to describe characteristics.

Challenges 10, and 11 form **RQ2.** for elevating goals. Challenges 7, and 9 refer to **b**) by tackling specifics of implementation. As current implementation, challenge 5 has initial answer for **a**), whereas the scope of this thesis is to evaluate feasibility of alternatives.

Finally, **RQ3.** mainly presents evaluation by validating proposed solutions and benchmarking performance.

For evaluating related work with raised research questions, requirements were formed as follows:

RO1 - unified description for heterogeneous resources,

RO2 - ability to run recurrently and online,

RO3 - *IoT* resource-sensitivity,

RO4 - low scaling overhead,

RO5 - placement methodology proposal,

RO6 - evaluation methodology.

RO1 covers **a**), **b**), and **c**) by evaluating ability to describe varying characteristics. **RO2** maps to **b**), **RO3** to **c**), and **RO4** to **d**). For deducing whether related work paper proposes a relevant solution **RO5** is formed and it covers **a**). Finally, for evaluation and comparison research question **RQ3.**, requirement object **RO6** was formed.

Chapter 3

Related work

Related work research has been conducted by keyword search in respective analysis areas. In the case multiple related work pieces distinctly reference a specific topic, it was included in related work research. The following sections indicate analysis on related work papers that potentially provide sophisticated answers for requirement objectives. Each subsection separates a specific piece of related work and in the end of each is a description of what requirement objectives are fulfilled.

3.1 Service composition

Service composition is the domain for implementing and be able to modularly scale complex applications. In heterogeneous cloud environments cloud service composition distributes multiple services and attempts to adhere to their requirements.

"Service composition execution optimization based on state transition matrix for cloud computing"

[19] proposes a *QoS* aware optimal service composition method. It "regards the cost averaged for one time of successful execution" as actual and minimizes that cost. Paper indicates four major *QoS* components:

1. **Cost** - execution cost, free for service invocation,
2. **Availability** - immediate service availability,
3. **Reliability** - probability of receiving right results within a time frame,

4. **Time** - duration from service invocation to result retrieval.

Service composition is described by Business Process Execution Language for Web Services (*BPEL4WS*). This method relies on Status Transition Matrix (*STM*) "to analyze the dynamic execution process of service composition". The contents of *STM* are based on Markov system - set of nodes that have fixed probabilities for passing from one node to another. A single service status transition matrix looks like:

$$P_{ws} = \begin{bmatrix} 0 & r_{ws} & 1 - r_{ws} \\ 1 & 0 & 0 \\ 1 & 0 & 0 \end{bmatrix}$$

Here P_{ws} - status transition probability matrix; r_{ws} - service reliability probability. The paper includes the possibility that a service can possibly not execute or fail during execution, thus decreasing the value of service composition. By using Markov system and evaluation via *STM*s, failure domain is dealt too.

By evaluating the cost of every successful execution in the composition, *STM* allows to choose an optimal composition.

This work covers the following requirements: **RO5**.

"A brokerage-based approach for cloud service selection"

[35] modeled a "brokerage-based architecture in the Cloud, where the Cloud brokers is responsible for the service selection". The fundamentals are: building index of service provider properties; querying user's requirements on the indexed properties.

Indexing relies on B^+-tree for *Cloud Service Provider* (CSP) index. Each index entry is of the following format: $< Key_{sp}, SID, p_1, p_2, \ldots, p_{10} >$. Here Key_{sp} is the indexing key; SID - service provider's identity, p_x the following:

1. **Service** type - varying type between on-demand, reserved, or specialized services
2. Security - level of security: high if three or more compliances with [22], one-two compliances - medium, else low.
3. *QoS* - determined through broker's and other vendor evaluation.
4. Measurement units - terms that the service can be charged for (i.e. memory, transactions, time).
5. Pricing units - how long a service is reserved for (i.e. hourly, monthly, yearly).

6. Instance sizes - amount of resources used.
7. Operating system - in cases where licensing is necessary for operating system usage.
8. Pricing - actual price for service usage.
9. Pricing sensitivity - price variability by region.
10. Subcontractors - indicates presence of subcontractors, and their services provided.

Every entry properties and relationships are encoded into binary strings, and concatenated. To generate the key, Hamming distance between every service provider and their closest cluster center is calculated. Then, *scaling value* is used to saturate points closer to each's cluster center.

Querying the index is based on:

1. Query encoding - as indexes were generated through Hamming distance, and clustering, having same encoding allows direct querying.
2. k-NN search - based on paper's trial-and-error, $k = 0.1 * N$: k - neighbor amount; N - service provider amount.
3. Result refinement - specific fit to query requirements.
4. Special criteria consideration - case of collision (requirement fulfillment varies), or collusion (correlating usage with users).

This work covers requirements: **RO2**, **RO4**, and **RO5**.

Agent-based cloud service composition

[11] proposes agent-based approach to compose services for different types of Cloud services. The architecture comprises of 6 elements:

1. **Web service** - interfaces to cloud resources,
2. **Service ontology** - functionality, input, and output describing specification,
3. **Resource agents (RAs)** - web service control and access orchestration,
4. **Service provider agents (SPAs)** - manage resources by organizing *RAs*,
5. **Broker agents (BAs)** - single interface for connecting *SPAs* into one virtualized resource,

6. **Consumer agents (CAs)** - parse consumer requirements and select best fitting *BA* and submit service composition request to it.

The architecture essentially composes distinct functionality components in order to provide virtualization in service level.

[11] covers requirements: **RO1**, **RO2**, **RO4**, **RO6**.

“A parallel branch and bound algorithm for workflow QoS optimization”

[17] devised a mathematical model for mapping abstract workflows to concrete ones. “A workflow is composed of web services selected in accordance with user requirements”.

Multidimensional Multi-choice Knapsack Problem (MMKP) based model helps to define a happiness measurement, which considers requirements and their importance by weighing against user preferences. To maximize the happiness measurement, a branch-and-bound algorithm was used. This paper covers requirements: **RO1**, **RO4**, **RO5**.

“Cloud Computing Service Composition and Search Based on Semantic”

[44] proposes *service matching algorithm* (SMA), “which considers the semantic similarity of concepts in parameters based on WordNet”. The proposed solution consists of several components and steps for matching services:

Here *Service Main Table*, *Service Parameter Table*, and *Service Operator Table* store main service elements. *Concept Similarity Table* is populated via pre-computed WordNet similarity relationships. Highly matching results of the tables are calculated and stored in *One-way Matching Table. Automatic Service Composition Module* processes data from *One-way Matching Table*, builds all possible service compositions, and stores accordingly to *Service Composition Table* and *Composition Path Table.*

Service Matching Algorithm relies on matching two services and their input parameters to others’ output parameters. [44] transforms service matching problem into semantic similarity.

Data from *One-way Matching Table* can be represented as weighted directed graph, where nodes are services and edges - semantic matching. For this, authors introduce an improved *EP-JOIN* algorithm called *Fast-EP* with time complexity of $O(N * log(N))$. This time complexity is achieved by exploiting used table architecture by storing composite service as a single one.

This work covers requirements: **RO1**, **RO3**.

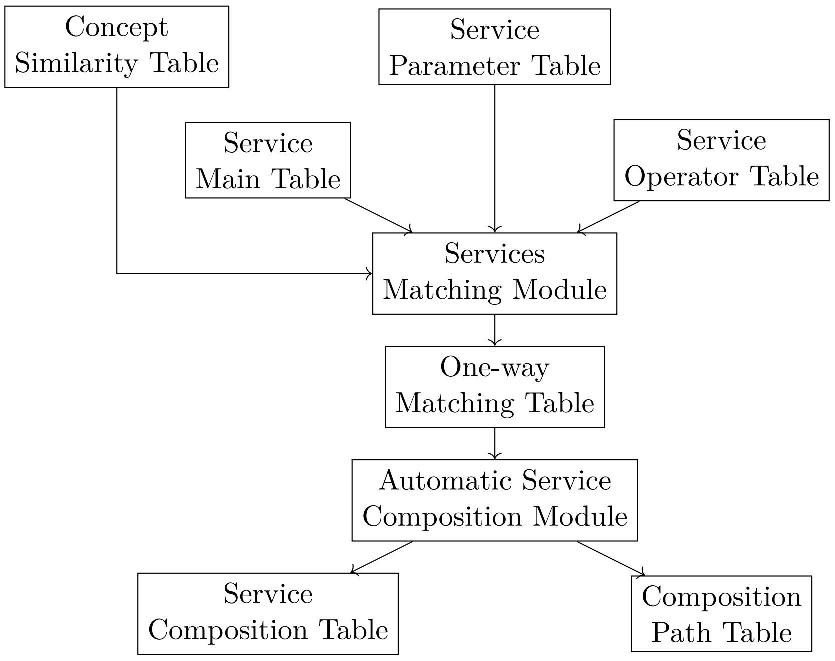

Figure 3.1: [44] *SMA* module diagram

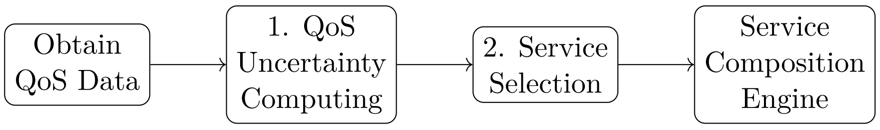

Figure 3.2: SSCM procedures [36]

"Cloud model for service selection"

[36] propose *QoS*-aware service selection based on mixed integer programming. *Service selection via Cloud model (SSCM)* two phased approach is used (See Figure 3.2):

1. Transform qualitative QoS to quantitative QoS,
2. Service selection

QoS-awareness is modeled as calculation on *QoS* fulfillment (uncertainty), as it can be seen in Figure 3.2.

This work covers requirements: **RO2**, **RO5**, and **RO6**.

"Resource allocation for service composition in cloud-based video surveillance platform"

[12] developed a platform for resource allocation optimization. They posed resource allocation as a linear programming problem. The objective function aims to minimize physical server usage. The constraints are as follows:

1. One virtual machine per physical server
2. Virtual machine demands don't overload physical server's capacity
3. Specific latency threshold
4. Specific resource utilization threshold
5. Specific *CPU* utilization

The paper recognizes that *LP* solution does not scale well. To tackle this, a *best-fit decreasing* heuristic was proposed. In this heuristic, *VM* is placed to a physical server that leaves the least left over resources. This heuristic can be considered as a measure for balancing resource usage between all machines.

Requirements covered by this work: **RO2**, **RO3**, **RO5**, and **RO6**.

"Revenue maximization with quality assurance for composite web services"

[41] developed a method to determine policy under three decision criteria:

1. Service availability at the decision instant,
2. Execution cost,
3. Rest time to deadline.

Arrival decision rule would select services with the lowest cost. One of the constraints are that workflows are expected to be sequential. For optimizing expected revenue, dynamic programming is used for dynamic service selection.

Authors did not find correlation between cost and higher quality assurance. To tackle this, a dynamic-programming algorithm to increase end-to-end quality assurance in place of revenue, was developed.

This work covers requirements: **RO2**, **RO4**, **RO5**, and **RO6**.

“Service-Oriented Computing”

[40] devised an approach to capture Cloud service capabilities and requirements using variability modeling. To use variability modeling, authors adapted Feature Model (*FM*) to *Cloud Feature Model (CFM)* by introducing extensive changes to attributing and modeling. Different types of models are used for CFM:

1. Domain model - representation of all relevant abstract decision aspects for selection.
2. Service model - single concrete Cloud service offer.
3. Requirements model - representation of requirements.
4. Alternative model - single valid configuration derived from *service model*.

Requirements covered by this work: **RO1**, **RO2**,

“HireSome-II: Towards privacy-aware cross-cloud service composition for big data applications”

[7] proposed a service composition method HireSome-II. Services are evaluated by some of their *QoS* history records with k-nearest-neighbor method as representative data selector. Paper’s authors pointed out that different type of services have different functionality and thus *QoS*. To tackle the variability problem, MCDM and SAW techniques developed by [13] were used to derive five typical *QoS* criteria:

1. Price
2. Duration
3. Reputation
4. Success rate
5. Availability

This work covers **RO2**, **RO3**, **RO4**, **RO5** requirements.

“A QoS-satisfied prediction model for cloud-service composition based on a hidden Markov model”

[42] present a *QoS*-satisfied prediction model based on a hidden Markov model. It uses the following general flowchart for service composition:

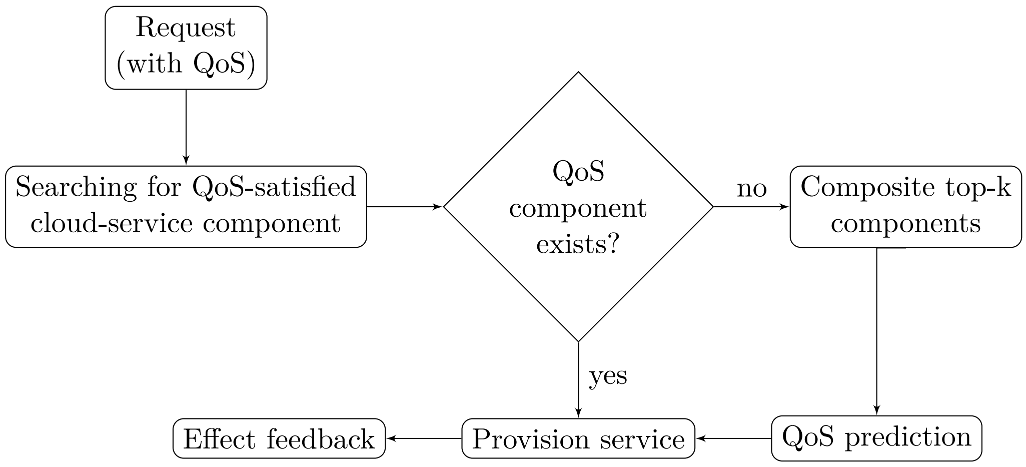

Figure 3.3: [42] service composition flowchart

Here the component of interest is *QoS* prediction. It is based on Hidden Markov Model (*HMM*) to predict whether a composition of cloud service components can satisfy user's *QoS*.

Requirements covered by this work: **RO2**, **RO5**.

"QoS ranking prediction for cloud services"

[48] proposes a *QoS* ranking prediction framework. It focuses on client side *QoS* properties, such as response time; throughput; and failure probability. The framework follows the following scheme:

1. Based on user-provided *QoS* similarities between user and training users are calculated.
2. Based on calculated similarities, a set of similar users is deducted.
3. Several algorithms are proposed for personalized service ranking by correlating usage experiences from similar trained users.
4. Results are presented to user.

Service ranking is presented by two proposed ranking algorithms - CloudRank1 and CloudRank2. *CloudRank1* is a greedy algorithm, that has the following steps:

1. Rank employed services by *QoS* values
2. Calculate service's sum of preference values with other services
3. Rank services by their sum of preference values

4. Pick highest ranking service and update ranks by removing picked service
5. Update ranks for all picked and non-picked services.

CloudRank2 algorithm is similar to *CloudRank1*, yet it does not sum preference values and treats them individually, thus give more accurate results.

Requirements covered by this work: **RO2**, **RO3**, **RO4**, **RO5**, **RO6**.

"A hybrid imperialist competitive-gravitational attraction search algorithm to optimize cloud service composition"

[15] proposed a hybridization of modified Gravitational Attraction Search with an Imperialist Competetive Algorithm to optimize response time and execution times simultaneously. *Imperialist Competetive Search* algorithm is based on sociopolitical evolution of humans. It uses countries as individuals, which are ranked by some characteristic. A set of "better" countries are selected as imperialist and others constitute as imperialist colonies. The allocation procedure of assigning colony to imperial country is uniformly distributed. As the total power of empire also consists of its colonies, during algorithm execution colonies are absorbed by the imperialist country. This increases the power absorbed from the colony.

Further on, empire competition power is decreased for the weakest empire until the weakest empire is destroyed.

Another part of the hybrid solution proposed by [15] is *Gravitational Attraction Search*. Main idea behind it is that problem search space contains many particles in n dimensions that have masses. Mass is considered as a quality of solution and is managed through *Gravitation law*, and *Laws of Motion*.

The hybridized solution obtains normalized weights for response-time and execution fee (sum equal to 1). As response-time and execution fee are of different unit types, they are normalized by deviation from average:

$$SV(x_i) = \frac{x_i - \bar{x}}{SD}$$

Here $\bar{x}$ - average set value; x_i service specific value; SD - standard deviation of the set.

The whole process of hybrid modified Imperialist Competetive (IC) and Gravitational Attraction Search (GAS) is as follows:

1. Generating countries

  - Country eligibility calculation on applicant's normalized response-time and execution fee.

2. Sort by country power
   - Select imperialist countries and divide colonies among them
3. Move colonies towards their imperial country
4. Country eligibility re-calculation on applicant's normalized response-time and execution fee with respect to applicant's weight.
   - Replace each imperialist with most powerful country in empire.
5. Every 10th iteration - imperialistic competition
   - Attract weakest country of the weakest empire
   - Using roulette wheel selection, generate number of countries selected in each empire for local search
6. GAS execution on previous phase-selected countries and imperialist country replacement with best found solution
   - Sort imperialists based on their power
   - Save best found imperialist
   - Go to step 3

This work covers requirements: **RO2**, **RO3**, **RO4**, **RO5**.

"A service composition framework for market-oriented high performance computing cloud"

[28] introduces a framework for on-demand composing and deploying available application on clouds. It builds up a knowledge base, which is compared to dependencies extracted from user's requirements. The architecture is as follows

Here, composition agent parses composition request and uses knowledge base to resolve all of the dependencies. Service Discovery is used to retrieve and update relevant service information. If all dependencies are successfully resolved, the configuration is passed to Packaging Engine and then to Service Delivery.

Knowledge base uses ontology developed by Web Ontology Language (OWL). It describes the components that are directly required in composition process. Ontology's relationships between components are the following:

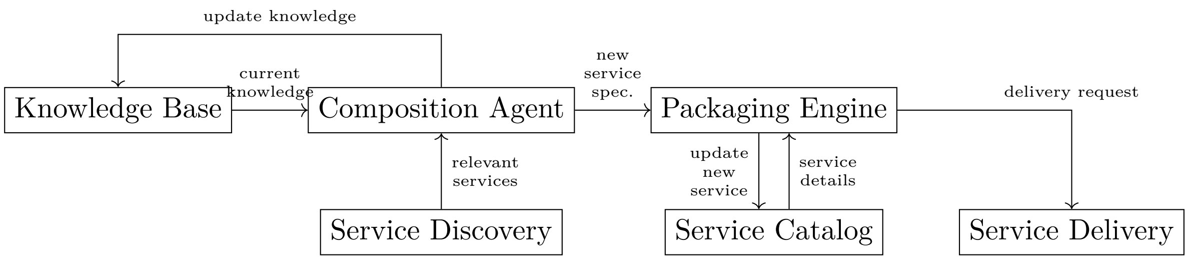

Figure 3.4: [28] framework's architecture

- Dependency - one component must exist for the other function
- Compatibility - two components are functional in a single environment
- Conflict - two components do not function in a single environment
- Whole-part - one component is part of the other
- Type - grouping instances that have the same set of functionality

This work covers the following requirements: **RO1**, **RO2**, **RO5**.

3.2 Service selection

Selecting cloud services under complex factors is one of related research areas. Mostly, it involves characterizing service providers to fulfill requirements. This topic is relevant for this thesis, as services, their requirements and functionalities are treated as heterogeneous (every service provides a different functionality). Main considerations when looking into related works for service selection are to answer **RQ1.** and **a**).

"Cloud Service Selection Based on the Aggregation of User Feedback and Quantitative Performance Assessment"

Inclusion and evaluation of user feedback was proposed as a framework by [30] to deduct service satisfaction. The framework consists of four components:

1. Cloud selection service - accepting and processing consumer requests.
2. Benchmark testing service - third-party to test scenarios for performance.
3. User feedback management service - collection and management of consumer feedback.

4. Assessment aggregation service - collects subjective data from *User feedback management service*, objective data from *Benchmark testing service*, and computing final score for service.

Some of the subjective and objective attributes correspond to same metric and thus this union is called *associated attributes.* These attributes allow direct comparison in their proposed approach:

1. Convert subjective attributes into scalar ratings
2. Convert objective attributes into scalar ratings
3. Filter unreasonable subjective assessments - comparing *associated attributes*, where feedback evaluation is significantly (in paper 80% of the maximum Euclidean distance) different than benchmarked.
4. Weight importance of every attribute - predetermined significance of every attribute.
5. Aggregate all attributes - filtered and weighted attributes summed up to give a final score.

This work covers **RO2**, **RO3**, **RO4**, and **RO6** requirements.

"A Declarative Recommender System for Cloud Infrastructure Services Selection 2 A System for Cloud Service Selection"

[45] proposed a system with an ontology model for characterizing providers in service configuration selection. *Cloud Configuration Management Layer* maintains essential domain model for compute, storage, and network services. For service discovery, based on their functionality and QoS parameters, a *Web Ontology Language (OWL)* defined ontology is proposed. To describe services, and configurations and metrics, two ontologies are used:

- **Cloud Service Ontology** - to describe the service type itself,
- **Cloud QoS Ontology** - configuration and metrics.

Cloud Service Ontology mainly allows to describe the service itself, for example, what functionalities/services it provides. On the other hand, *Cloud QoS Ontology* is essential from a management point of view for each service.

This work mainly covers **RO1** requirement.

3.3 Service management

“Large-scale cluster management at Google with Borg”

[38] describes *Google*’s approach on their own cluster management framework - Borg. Borg cell is a set of machines that are considered as unit. Given different service nature, Borg categorizes into long-running and short-term services. Short-term services have lower priority and are killed (placed in backlog) in favor of long-term ones. Borg scheduler is responsible for deducing placement and running services from backlog. Scheduler first filters feasible machines and then applies scoring to evaluate machine’s efficiency and availability.

In case of lack of resources, lower-priority tasks are preempted and put into scheduler’s pending queue. Scheduler is responsible for reassigning new location for the tasks from it’s pending queue.

This work covers requirements: **RO2**, **RO3**, **RO4**, and **RO5**.

“Ontological Map of Service Oriented Architecture for Shared Services Management”

[39] proposes an ontological map of *Service Oriented Architecture* for *Shared Services Management (SSM).* Shared Services contain common functions for organization in order to “reduce information process duplication and increase information and knowledge sharing”. Accordingly Shared Services Management is a set of activities that attempt to optimize Shared Service efficiency and effect. The authors propose usage of Zachman framework to map to SSM. Zachman framework is an information architecture for relating information system taxonomies.

In a nutshell, SOA to SSM mapping allows better ontological organization by defining objects, subjects and relationships for entities, see Figure 3.5.

Mapping is as follows:

1. What - service unit:
    - Refers to identifiable software application
    - Described by hierarchical structure and represented by tree of services
2. How - service procedure
    - Set of actions between client and host system

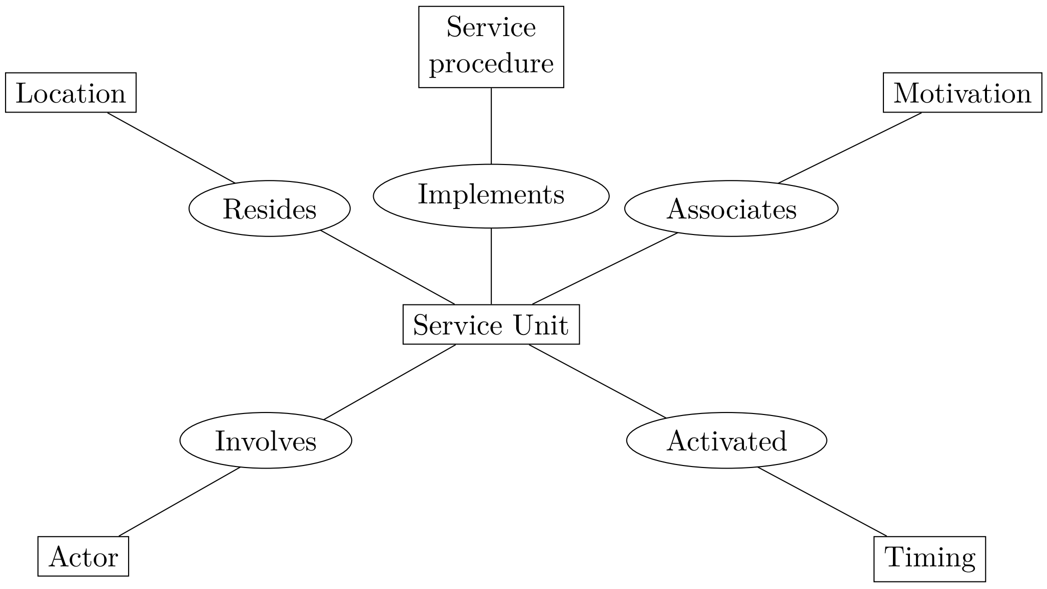

Figure 3.5: [39] primary ontological view of SOA for SSM

- Contains inputs, output, logic and persistence method.

3. Where - service provider and client location (URL)
4. Who - actors, characterized by dimensions:
    - Organizational - control and responsibility
    - Technical - agent's skills and expertise
    - Human cognition - cognitive characteristics
5. When - time
6. Why - motivations for service sharing, generally cost saving.

This work covers requirement **RO1**.

3.4 Service placement

In order to find a set of nodes that comply with resource requirements is yet another research field. Service placement tackles this by determining which nodes are qualified for service deployment.

"Hierarchical network-aware placement of service oriented applications in clouds"

[21] propose *Integer Linear Programming* formulation for *Cloud Application Placement Problem (CAPP).* The authors as well pose another solution as heuristic hierarchical algorithm based on *Particle Swarm Optimization* and *Genetic Algorithms.* Paper notes specific resource constraints: CPU and memory capacities; usage of specific hardware; and network capacity. Problem optimization is split into multiple objectives that are sequentially executed and planned in a way, that previous optimization objective results are used by next one:

1. Request maximization by utilizing CPU usage
2. Network bandwidth maximization for inter-service communication
3. Computation node minimization to improve energy efficiency and placement cost
4. Minimization of service migrations
5. Minimization of hops between nodes for latency reduction

The optimization objectives are posed as variations of equations that formally describe the problem. *Integer Linear Programming* based algorithm was used to solve the optimization objectives.

One of the heuristics - Particle Swarm Optimization (PSO) - is based on swarm behavior simulation. Every particle determines its best position and swarm is aware of best position in it. Particle movement depends on local (their) and global best positions meaning that while they search locally, direction is towards best known position. In this paper's perspective, every particle is a solution to CAPP. Solution is defined as an array in 3-dimensions: application; application request number; and services. PSO objective function is:

$$obj = (\frac{PlacedCPU}{RequestedCPU})^2 * (\frac{PlacedRequests}{TotalRequests}) * (1 - \frac{UsedNodes}{\mid N \mid +1}) * (1 - \frac{MigrationCount}{\mid N \mid * \mid S \mid +1})$$

Second heuristic is based on *Genetic Algorithm (GA).* In essence, GA relies on natural selection based on solution fitness. New solutions are created by combining two existing ones. For solution and combination, equations used in PSO approach are used, thus making this only a different approach mechanism.

This work covers: **RO2**, **RO3**, **RO5**, **RO6** requirements.

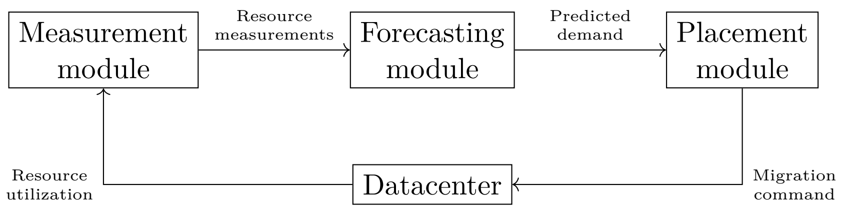

FIGURE 3.6: [47] VM placement framework

“COOL CLOUD: A PRACTICAL DYNAMIC VIRTUAL MACHINE PLACEMENT FRAMEWORK FOR ENERGY AWARE DATA CENTERS”

[47] proposes a framework for least resource wastage and power consumption. The framework (Figure 3.6) has three distinct modules:

1. **Measurement module** - for CPU, memory, network, and storage usage metrics,
2. **Forecasting module** - for decision logic that produces VM migration plan,
3. **Placement module** - for deployment and migration of VMs

Authors modeled forecasting as *Integer Linear Programming (ILP)* problem with CPU, memory, network, and storage resources. While proposed *ILP* solution provides optimal placement, authors admit that it is “unpractical for large size data centers”. To tackle this, they developed a low computational complexity heuristic described in Algorithm 1. The variables are described in table 3.1.

Variable	Description
$Period$	predefined execution time period
L'	current placement matrix
$\mathbb{N}_{\mathbb{VM}}$	set of *Virtual Machines (VMs)*
$\mathbb{N}_{\mathbb{PM}}$	set of *Physical Machines (PMs)*
P	power consumption
P_{active}	power level of physical machines in active mode
P_{sleep}	power level of physical machines in sleep mode
U^{CPU}	*VM* CPU utilization
U^{MEM}	*VM* memory utilization
U^{HD}	*VM* storage utilization
U^{BW}	*VM* bandwidth utilization
$T^{migrate}$	time for *VM* migration
H^{CPU}	*VM* CPU limit
H^{MEM}	*VM* memory limit
H^{HD}	*VM* storage limit

H^{BW}	*VM* bandwidth limit
$P^{migrate}$	power level for *VM* migration
L	placement matrix (decision variable)
G	migration matrix (decision variable)
O	operation mode vector (decision variable)

Table 3.1: [47] ILP based algorithm's variable descriptions

```
Input  : $Period, L', \mathbb{N}_{\mathbb{VM}}, \mathbb{N}_{\mathbb{PM}}, P, P_{active}, P_{sleep},$
         $U^{CPU}, U^{MEM}, U^{HD}, U^{BW}, T^{migrate},$
         $H^{CPU}, H^{MEM}, H^{HD}, H^{BW}, P^{migrate}$
Output: $L, G, O$
while There exists a resource constraint violation do
    Perform VM migration to find a feasbile solution;
    if A feasible solution cannot be found then
        Adopt the alternative for operation;
        break;
    end
end
repeat
    Seek a better solution to consume energy at a lower level;
until The solution cannot be improved;
return $L, G, O$
```

Algorithm 1: [47] energy saving heuristic

Requirements covered by this work: **RO2**, **RO3**, **RO5**.

3.5 Cloud configuration

"Towards Multi-Cloud Configurations Using Feature Models and Ontologies"

[31] presents a model driven approach to cloud configuration. The paper listed out cloud variability, configuration dimensions, and multi-cloud configurations as main challenges. Their proposed solution includes two parts:

1. describing cloud provider variability characteristics,
2. connecting application requirements to those characteristics.

Part 1 uses *FM* for each cloud provider to describe. An ontology maps cloud concepts to cloud providers *FM*'s features. Another ontology maps resources and their capabilities to cloud providers *FM*'s attributes.

FM was based on the concepts from [29] and extended by [2] and [5]. As cloud providers have different *FM*s and thus present heterogeneity, the paper proposes abstraction by ontologies. For this, two ontologies are proposed: $Onto_{Cloud}$ and $Onto_{Dim}$. $Onto_{Cloud}$ models technical requirements (software) supported by the cloud provider, and $Onto_{Dim}$ describes dimension properties (hardware) to be used for $Onto_{Cloud}$ technical requirements. $Onto_{Dim}$ allows part 2 to be fulfilled.

By having an extensive cloud provider description methodology, the paper uses round-robin-based algorithm for picking viable machines to deploy applications.

This work covers requirements: **RO1**, **RO2**, **RO4**, **RO6**

3.6 Cloud brokerage

Cloud brokers "intermediate between cloud customers and providers to assist the customer in selecting the most suitable cloud service". As cloud services expose varied functionality and have a varying amount of *SLA* satisfiability, it is non-trivial to deduct services that best meet defined set of requirements.

"Smart Cloud Marketplace - Agent-Based Platform for Trading Cloud Services"

[4] proposes *Smart Cloud Marketplace* - marketplace-like platform for cloud services. Marketplace's agents are both service providers and consumers that employ various trading policies for local efficiency. Authors solve *Cloud Services Allocation Problem* with *Greedy-RP* and *Adaptive-Greedy* mechanisms.

Greedy-RP mechanism is based upon on a greedy heuristic and allocation, and pricing schemes. Output of this mechanism is user that obtains the resources needed. Greedy-RP sorts all users by their *price-per-item* and checks against *Available-Resource-Constraint* and *Reserve-Price-Constraint.* "Pricing scheme aims to establish the prices that the users will have to pay for the granted resources" and helps deduce minimum possible item price to win the auction.

This work covers requirements: **RO2**, **RO4**, and **RO5**.

"Preference-based cloud service recommendation as a brokerage service"

[27] proposes service ranking. Their work allows fuzzy numbers and intervals to define *Key Performance Indicator* and user requirements. The ranking is done by deriving "fuzzy comparison matrices and subsequently using a fuzzy Analytical Hierarchical Process". Ranking procedure consists of 4 phases:

1. Expressing ranking problem into a hierarchical structure
2. Computation of relative *QoS* attribute weights
3. Computation of relative service performances
4. Aggregation of relative service weights

This work covers requirements: **RO1**, and **RO5**.

"A Broker-based Framework for Multi-Cloud Workflows Steinbuch Centre for Computing"

By evaluating services as tasks and clustering, [14] proposes a workflow based framework. It consists of Workflow Engine, which interacts with Service Broker. Workflow Engine is responsible for parsing and clustering proposed workflow, and Service Broker matches providers, schedules tasks, and starts execution. Full interaction can be seen in Figure 3.7.

From the figure it is clear that tasks preparation and deployment is done in sequential order. This imposes a workflow which limits parallelization. One of the limitations is that with parallel deployment and execution some tasks cannot be optimally grouped. Another would be architectural of having a full-stop whenever there is a need to get results, full-stop milestones: deployment of the tasks; tasks grouping; task execution.

The paper specified only the sequence and architecture without specifics about methods or algorithms applied for clustering and scheduling.

This work covers requirements: **RO2**, **RO4**, **RO5**, and **RO6**.

3.7 Resource allocation

"A Game of Things: Strategic Allocation of Security Resources for IoT"

[32] proposes security tool resource allocation methodology through game theory. In paper a threat model is described with defendant strategies that consider resource allo-

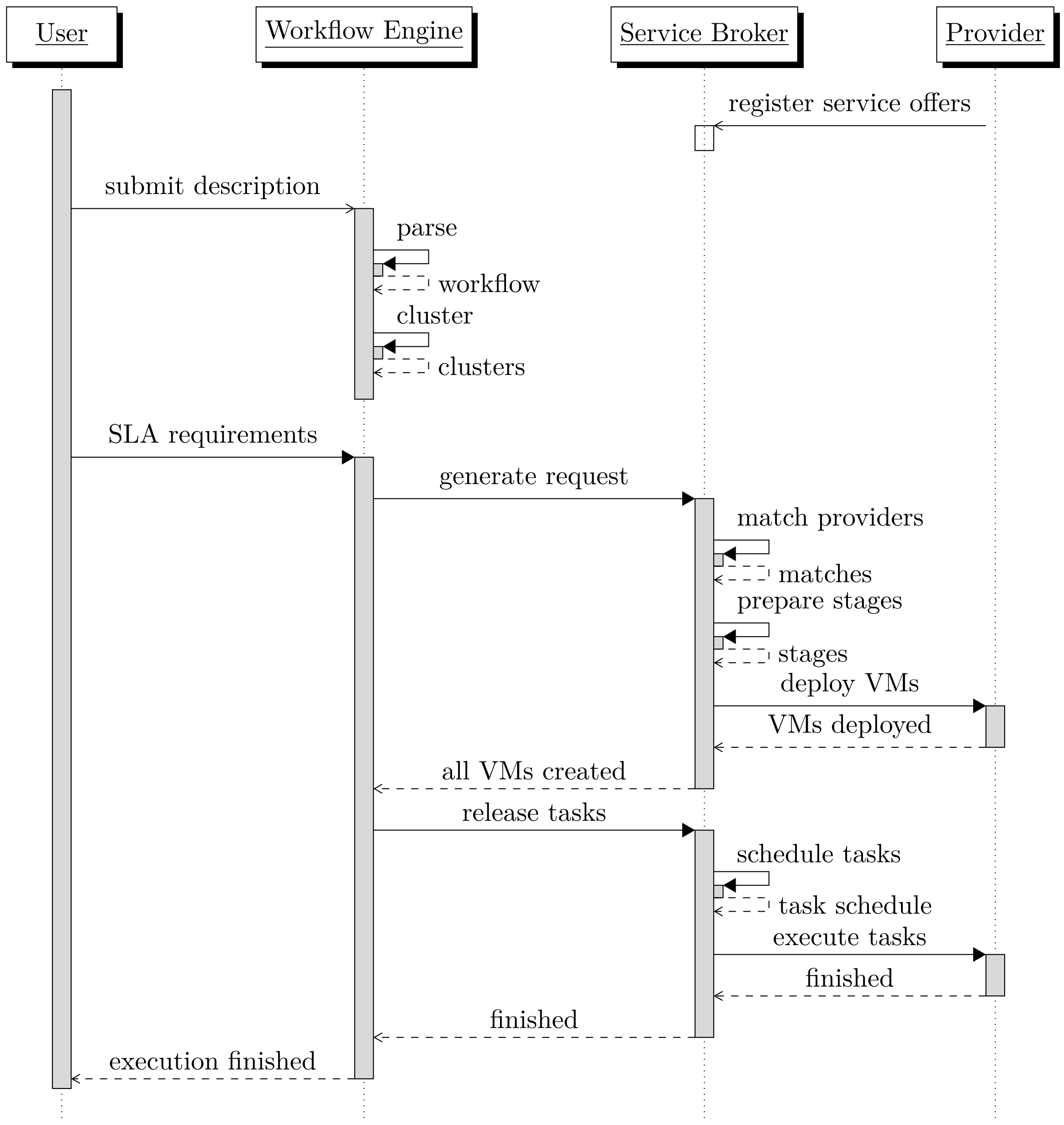

Figure 3.7: [14] workflow deployment sequence diagram

cation cost, energy consumption, and resource's criticality. For evaluating characteristics, Pareto analysis is conducted for computing Pareto points (allocation plans). After computing Pareto curve, a set of Pareto points, a risk minimization function is solved to obtain Pareto point that best corresponds to requirements.

This work covers requirements: **RO2**, **RO3**, **RO5**, and **RO6**.

"Cloud resource allocation schemes: review, taxonomy, and opportunities"

[43] overviews cloud resource allocation schemes. The paper derived resource allocation taxonomy in 8 categories:

1. Allocation with optimization objective
2. Design approach for allocation
3. By target resource type - instances, bandwidth, storage and work items
4. By optimization method
5. By utility function - user-center, system-center, or hybrid
6. By processing mode - centralized or distributed
7. By target instance - one or multiple datacenters
8. By evaluation setup - workload or experimental platform

Resource allocation taxonomy is relevant as it provides information of what [43] overview found as means and goals.

One of the *Optimization Objective* is *resource pricing* for either monetary or operational expenditures. Another objective is *resource utilization* - attempt to have as few unused resources as possible. Third objective is *availability*, to increase operational performance. Lastly, *QoS* objective relies upon metrics visible by client.

Design approaches analyzed by [43] were:

1. Market-oriented - using economic principles
2. Queue theoretic models - for forecasting needed resources and performance metrics
3. General Machine Learning - statistical prediction-demand functions
4. Graph theory - graph model abstraction
5. Clustering (bin packing) - grouping items and trying to minimize the groups

6. Heuristics

These design approaches give several insights for the actuality of the thesis. First of all, the approach describes the data model that the solution relies on. Secondly, a methodology on utilizing the aforementioned data model is set.

Optimization methodologies are mostly well known approaches:

1. Integer programming
2. Linear programming
3. Dynamic programming
4. Convex optimization
5. Inspired by biological problems
6. Statistical

These optimization methodologies directly answer **RO5** by supplying the underlying methodologies.

3.8 Resource management

"Exploiting Service Usage Information for Optimizing Server Resource Management"

[37] show that exposing and using detailed service usage information allows to provide improved *QoS* and optimize resource utilization. The paper describes four service access attributes for service usage information.

First service access attribute is *request flow.* It gives high-level information, such as client identity, request type, and request time. In [37] there attributes are interpreted as: Overall request rate, specific type request rate, specific type request rate in session, percentage of specific type of requests, average session length, average session inter-request time, rate of new sessions, structure of web sessions.

Second attribute is *coarse-grained resource utilization and reward.* It gives information about request type execution cost. Costs can be inflated by varying system load - longer request processing under heavy system load -, and different amount of data being processed. Having these problems in mind, this paper analyses average typed request processing time.

Third attribute is *fine-grained server resource utilization.* This request access attribute describes detailed information about request type processing inside the server. Paper programming language's *VM* profiling to obtain metrics and relative information for request processing.

Last attribute is *data access patterns.* Application's data access by request type is provided for this requests access attribute. This includes specific data segments accessed and access consequences.

This work covers requirements: **RO2**, **RO3 RO4**.

Reference	**RO1**, heterogeneous resources	**RO2**, run recurrently and online	**RO3**, IoT resource-sensitivity	**RO4**, low scaling overhead	**RO5**, placement methodology	**RO6**, evaluation methodology
"Smart Cloud Marketplace - Agent-Based Platform for Trading Cloud Services"[4]	x	✓	x	✓	✓	x
"Preference-based cloud service recommendation as a brokerage service"[27]	✓	x	x	x	✓	x
"A Broker-based Framework for Multi-Cloud Workflows Steinbuch Centre for Computing"[14]	x	✓	x	✓	✓	✓
"Towards Multi-Cloud Configurations Using Feature Models and Ontologies"[31]	✓	✓	x	✓	x	✓
"A Game of Things: Strategic Allocation of Security Resources for IoT"[32]	x	✓	✓	x	✓	✓
"Exploiting Service Usage Information for Optimizing Server Resource Management"[37]	x	✓	✓	✓	x	x
"Service composition execution optimization based on state transition matrix for cloud computing"[19]	x	x	x	x	✓	x
"A brokerage-based approach for cloud service selection"[35]	x	✓	x	✓	✓	x
Agent-based cloud service composition[11]	✓	✓	x	✓	x	✓
"A parallel branch and bound algorithm for workflow QoS optimization"[17]	✓	x	x	✓	✓	x
"Cloud Computing Service Composition and Search Based on Semantic"[44]	✓	x	✓	x	x	x

“Cloud model for service selection”[36]	x	✓	x	x	✓	✓
“Resource allocation for service composition in cloud-based video surveillance platform”[12]	x	✓	✓	x	✓	✓
“Revenue maximization with quality assurance for composite web services”[41]	x	✓	x	✓	✓	✓
“Service-Oriented Computing”[40]	✓	✓	x	x	x	x
“HireSome-II: Towards privacy-aware cross-cloud service composition for big data applications”[7]	x	✓	✓	✓	✓	x
“A QoS-satisfied prediction model for cloud-service composition based on a hidden markov model”[42]	x	✓	x	x	✓	x
“QoS ranking prediction for cloud services”[48]	x	✓	✓	✓	✓	✓
“A hybrid imperialist competitive-gravitational attraction search algorithm to optimize cloud service composition”[15]	x	✓	✓	✓	✓	x
“A service composition framework for market-oriented high performance computing cloud”[28]	✓	✓	x	x	✓	x
“Large-scale cluster management at Google with Borg”[38]	x	✓	✓	✓	✓	x
“Ontological Map of Service Oriented Architecture for Shared Services Management”[39]	✓	x	x	x	x	x
“Hierarchical network-aware placement of service oriented applications in clouds”[21]	x	✓	✓	x	✓	✓
“Cool Cloud: A Practical Dynamic Virtual Machine Placement Framework for Energy Aware Data Centers”[47]	x	✓	✓	x	✓	x
“Cloud Service Selection Based on the Aggregation of User Feedback and Quantitative Performance Assessment”[30]	x	✓	✓	✓	x	✓
“A Declarative Recommender System for Cloud Infrastructure Services Selection 2 A System for Cloud Service Selection”[45]	✓	x	x	x	x	x

Table 3.2: Related work comparison overview

3.9 Conclusions

This section summarizes the take-away from previous related work.

Not a lot of related work has tackled the heterogeneity of resources (**RO1**), yet those who did mostly chose to model an ontology. Majority of algorithms were oriented towards recurrent execution (**RO2**). Common activation approaches were: time based - i.e. every minute; event based - i.e. on new node in cluster; and *SLA* violation based.

While not a lot of related work directly referenced IoT domain (**RO3**), some of them relied on close monitoring of resource usage and reaction. Take away message would exactly mean thorough resource monitoring with appropriate measures.

RO4 requirement fulfillment in related work was rather saturated. On one side - deterministic optimal solutions by using *LP* or variations of it. LP problems do not fulfill this requirement as they do not scale.

Not all of the related-work papers proposed placement algorithm (**RO5**). Even more, not all of the works that proposed a placement strategy contained an algorithm that could model future resource usage.

Most **RO6** works did not disclose their evaluation methodology. The minority, however, described what resources, how and with what data they evaluated their solution with.

In design, alternative solution to related work will repeatedly discussed.

Chapter 4

Design

This chapter describes design and decisions for enabling *DS2OS* service placement strategy usage. A discussion of alternative design approaches is provided at the end of each section.

4.1 Solution requirements

In section 2.7 requirements were formed:

RO1 unified description for heterogeneous resources,

RO2 ability to run recurrently and online,

RO3 *IoT* resource-sensitivity,

RO4 low scaling overhead,

RO5 placement methodology proposal,

RO6 evaluation methodology.

This chapter overviews how this design fulfills these requirements. Section 4.3, with guidance from related work, proposes a solution to **RO1**.

Design for requirements **RO2**, **RO3**, and **RO4** is described in section 4.4. Multiple placement strategies, that utilize resource descriptions from section 4.3, are described in section 4.5.

Evaluation methodology, fulfilling **RO6** is designed and conveyed in chapter 6.

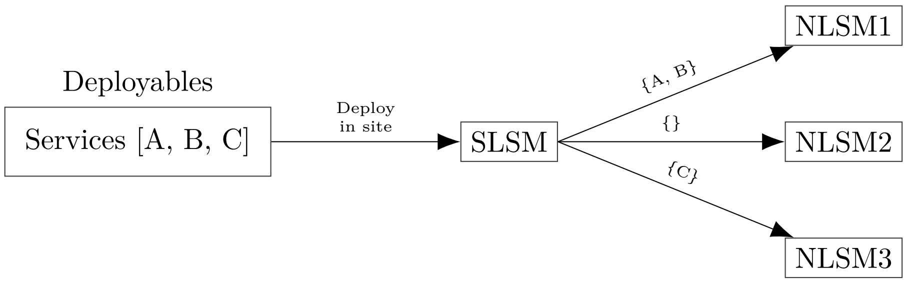

Figure 4.1: DS2OS deployment in [3]

4.2 DS2OS integration

This section overviews current service placement design by [3] and proposes alterations for carrying out thesis's requirements.

In [3] section 4.3 described service deployment and migration.

In summary, in perspective of service placement, the procedure is defined as optimization and is carried out by *SLSM* itself.

Figure 4.1 portrays current implementation as designed in [3]. Here services are deployed through *SLSM*, it internally decides which service goes to which *NLSM*.

In figure 4.1 services are marked as A, B, C and $NLSM1$ receives services A and B, $NLSM2$ does not receive any services, and finally $NLSM3$ receives C.

The proposed integration in figure 4.2 removes internal decision component from *SLSM* while leaving the operational deployment responsibility. Here two types of services are introduced: Placement Strategy Picker (*PSP*) and Placement Strategy (*PS*)s.

PSP is an intermediary which analyses placement goals from *SLSM* to provide the optimal *PS*. *PS* is an abstract strategy which given current placement layout of the *DS2OS* site, proposes alternative service layout. It completely depends on the implementation of *PS* on how the proposed services are placed.

In figure 4.2 *SLSM* communicates with a *PSP* goals and services. *PSP* does an intermediate decision by analyzing goals and *picking* a specific *PS*. *PS* does the final decision internally on service placement.

Architecture and communication between *SLSM*, *PSP*, and *PS* is described in section 4.4.

As an alternative for integration with *DS2OS*, all of the components could be incorporated inside of *SLSM*. This would likely provide better performance in the aspect of

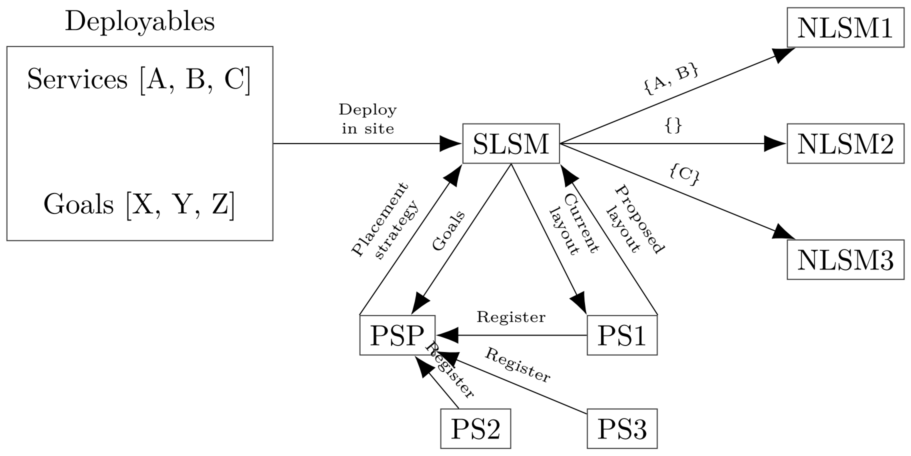

Figure 4.2: Proposed DS2OS deployment with separated placement strategies

SLSM, *PSP* and *PS* communication. On the other hand, the performance benefits of this communication can be negligible as the communication is not intensive, and most of *PS* communication is related towards *NLSM* resource and capability monitoring. Additionally, by coupling *PSP* and *PS* with *SLSM* it becomes non-trivial to extend with additional *PS*s. Another negative aspect of incorporating *PS* into an *SLSM* is process resource utilization. *PS* execution requirements are not restricted, which in the sense of *PS* being in *SLSM* would introduce the risk of deficiency in resource time. In other words, *PS* direct integration would induce degradation of *NLSM* performance.

Another integration alternative would be combination of *PSP* and *PS*s into a single service. While this does not introduce all of the *SLSM* related limitations, in aforementioned alternative approach, introduction of new *PS*s would still be coupled and limited with *PSP*.

In general, *PSP* exposes distinctively different functionality than that of *PS*, therefore applying architectural methodology of microservices (section 2.2.2), would provide loose coupling and individuality.

4.3 Resource description

In order to model a heterogeneous environment, whose devices not only have different amounts of resources, but also different capabilities, a resource description methodology needs to be devised.

The parties that in current *DS2OS* integration (section 4.2) operate with resource exposure are *PS*s and *NLSM*s (and *SHE*s, as they are responsible for managing *NLSM*'s environment).

A design decision of grouping by resource types was taken for resource description. This decision is akin to proposed solutions in related works that propose ontologies or specialized labeling methodologies.

While the principle of grouping by resource types does not provide equal evaluation of resource qualities, it provides an abstract way of monitoring a definitive subset of resources. For example, *NLSM* hosts can have multiple types of storage (such as local and networked), different type of operating memory (dedicated and swap). In this case *PS* does not indicate various types of grouped resources and attempt to micromanage monitoring results for each of them, but instead would get information about *NLSM*s capability and usage of specific type of resource.

Another consideration is distribution of resource description. Plainly the problem can be described as follows: given that both *NLSM* and *PS* have the same resource type group definitions, how would an update be distributed?

One way of doing this would be development of a central service, as it was done in [15], by having a knowledge base. This would increase the complexity as both *PS* and *NLSM* would have to adjust their internal models to what the resource description service would offer. It is unclear how would a new unknown type be universally utilized inside *PS*, whereas for *NLSM* the central service could provide resource to type group mappings.

To avoid the aforementioned complexity introduction with additional service, the type groups can be explicitly set and synchronized between *NLSM* and *PS* through software project versioning. This would also allow *PS* developers to understand and exploit available type groups better, than writing custom software that attempts to utilize updates from a service.

It is very well expected that *PS*s will not utilize all of the available resources as this is the decision of the developer itself, yet posing this problem as full-utilization one, allows to understand the implications of how should it be implemented. In this case, second approach, that does not use central service, will be used.

4.4 Architecture and communications

As mentioned in section 4.2, proposed design is by removing placement optimization from *SLSM* into *PSP* and *PS*. Figure 4.2 shows a general model of how the service deployment would work with the proposal.

Communication is conveyed through *VSL* (Section 2.3) as it is used in current implementation (Section 2.4). Alternatively, it is possible to use other communication protocols, such as Hyptertext Transfer Protocol (*HTTP*), but that in the long run would introduce higher technical debt. Technical debt is an additional maintenance cost in software development. In using a different communication protocol it would mean that communication clients and their handlers have to be maintained in *PSP*, *PS*, *SLSM* for service placement deduction, and *NLSM* for resource monitoring. Additionally, in the case of different *transport layer*, which *VSL* might support, *HTTP* clients would need to support as well, as the environment might not offer any other fall-backs. On the other hand, since the communication relies on *KA*s, they will receive higher process load due to introduction of monitoring. This additional load could interfere with intended use of the system - service communication. A solution for this could be load balancing or using different *KA*s, but that is outside the scope of this thesis.

As *PSP* only supplies matching to specific goals, it is redundant to maintain any persistant storage or state in it. On the other hand, *PS* might maintain model of the *DS2OS* site, depending on strategy itself. These conditions require *PS* persistence. Additionally, given that *SLSM* is responsible for site's management, *PS* only needs to maintain an internal model.

SLSM initiates placement layout retrieval in figure 4.3. Figure can be split into three parts: registration, initialization, and querying.

In initialization *SLSM* uses *PSP* to get optimal *PS* for given goals. After *PSP* returns optimal *PS* for given goals, *SLSM* queries for *PS*'s monitoring requirements.

Once initialization is done, *SLSM* can query *PS* whenever to retrieve placement layout for services inside NLSMs. This functionality fulfills the requirement **RO2** for continuous, online running.

4.4.1 Placement Strategy Picker

PSP is responsible for interpreting submitted goals in order to deduce an optimal *PS*. *PS*s register to *PSP* themselves and their capabilities in registration phase, as it can

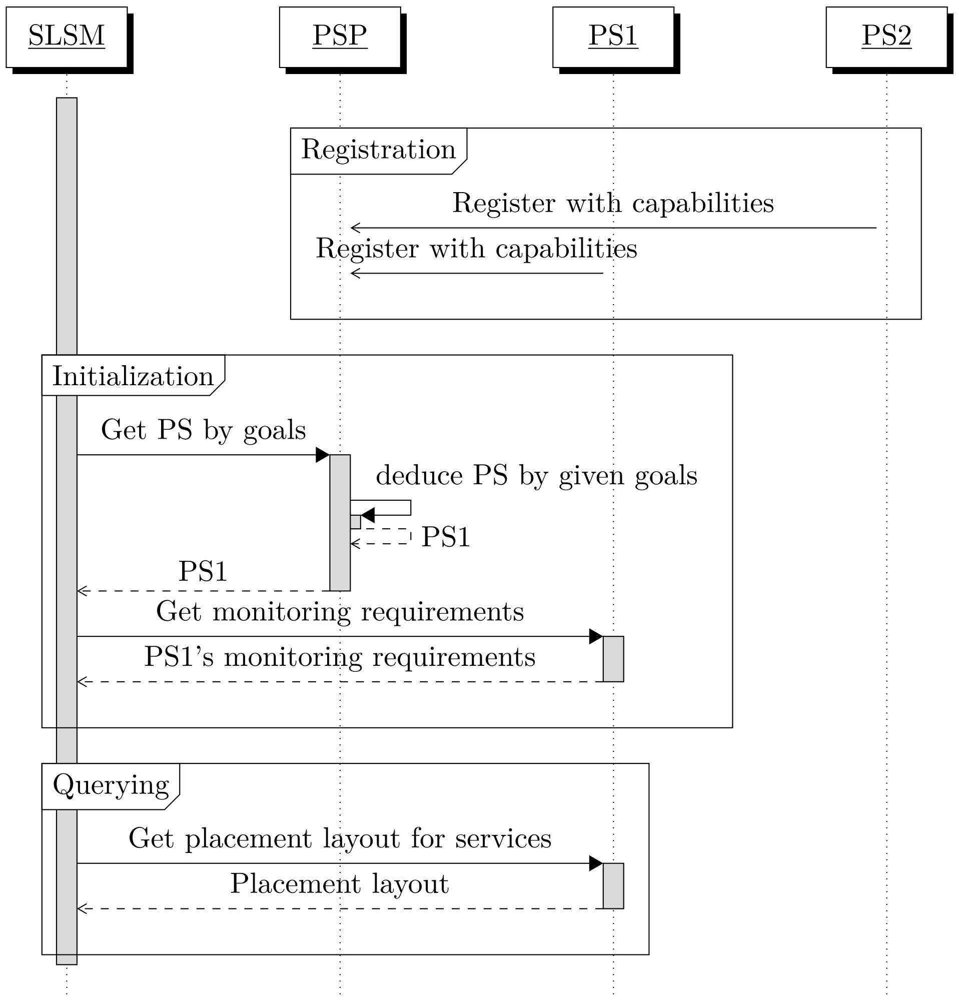
SLSM
PSP
PS1
PS2
Registration
Register with capabilities
Register with capabilities
Initialization
Get PS by goals
deduce PS by given goals
PS1
PS1
Get monitoring requirements
PS1's monitoring requirements
Querying
Get placement layout for services
Placement layout

Figure 4.3: SLSM workflow to receive placement layout

be seen in 4.3 *Registration* block. Registration block is independent only for the case of better visibility of functionality. By design the *PSP* should be able to dynamically accept new registrations at any time.

From design perspective, there are multiple ways of modeling the registration procedure, although it is a trivial where a different design would not indicate significant changes.

Actions between *SLSM* and *PSP* are in the following order:

1. *SLSM* send a list of goals to the *PSP*,
2. *PSP* parses goals,
3. *PSP* evaluates from a list of registered *PS*s and picks best match,
4. *PSP* returns the *PS* to *SLSM*.

From communication perspective, *PSP* exposes two endpoints:

1. **placementStrategies** - *VSL* list node that is a repository of registered *PS*s,
2. **optimalStrategy** - *VSL* node for getting optimal *PS* for given goals.

Using *VSL* list node for **placementStrategies** provides transparency by giving means for *PS* to verify success of registration with *PSP*. Listing 4.1 is for **placementStrategies** item's *CM*. Here an entry contains the address to the *PS*, and strategy's optimization targets.

Listing 4.1: placementStrategy context model

```
<placementStrategy>
    <address type="/basic/text" />
    <optimizationTargets type="/basic/list" allowedTypes="/basic/text" />
</placementStrategy>
```

Listing 4.3 shows *VSL* context model for *PSP*. Here aforementioned **placementStrategies** list is one of the fields. The other field - **optimalStrategy** - is for retrieving an optimal *PS*. For this call, parameters are additionally required. The format is as follows:

Listing 4.2: optimalStrategy call example

```
/optimalStrategy/processing=10&networking=20&...
```

In 4.2 parameters rely on *Query string*[1]. Name of the parameter is the goal that the strategy has to follow. Value - importance as a weight. Weighing just represents impor-

[1] Part of Uniform Resource Locator (*URL*) structured key-value representation in a string.

tance in comparison to other goals. This means that whatever the scores are assigned, importance is weighted against the sum of the scores, rather than some absolute value.

Listing 4.3: placementStrategyPicker context model

```
<PlacementStrategyPicker>
    <placementStrategies type="/basic/list" allowedTypes=".../placementStrategy">
</PlacementStrategyPicker>
```

optimalStrategy does not provide such transparency as every *PS* has its own way of determining the placement layout.

An alternative could be usage of ontologies or approaches similar to how *PS* and *NLSM* resource type groups are designed, but this introduces another level of complexity and generally is outside the scope of this thesis.

4.4.2 Placement Strategy

For *PS* to be selected, it has to be registered with *PSP*, as seen in figure 4.3 registration block. By registering, *PS* also provides optimization targets that let *PSP* to later match to goals received by *SLSM*.

PS utilizes resource type groups (section 4.3) by providing monitoring requirements to *SLSM*. *SLSM* is responsible for distributing the monitoring requirements to *NLSM*s in *DS2OS* site. Figure 4.4 displays the sequence of how *PS*'s monitoring requirements are distributed through *SLSM*.

In listing 4.4, *PS* has two fields - **monitoringRequirements** for exposing its monitoring requirements; and **placementLayout** for providing latest optimal service placement layout in *NLSMs*.

Listing 4.4: placementStrategy context model

```
<placementStrategy>
    <monitoringRequirements type="/basic/list" allowedTypes="/basic/text">
    <placementLayout type="/basic/list" allowedTypes=".../placement">
</placementStrategy>
```

Placement layout is a list whose entries contain *NLSM* and the service that should be running in it as it can be seen in listing 4.5.

Listing 4.5: single placement layout entry

```
<placement>
    <nlsm type="/basic/text">
    <service type="/basic/text">
</placement>
```

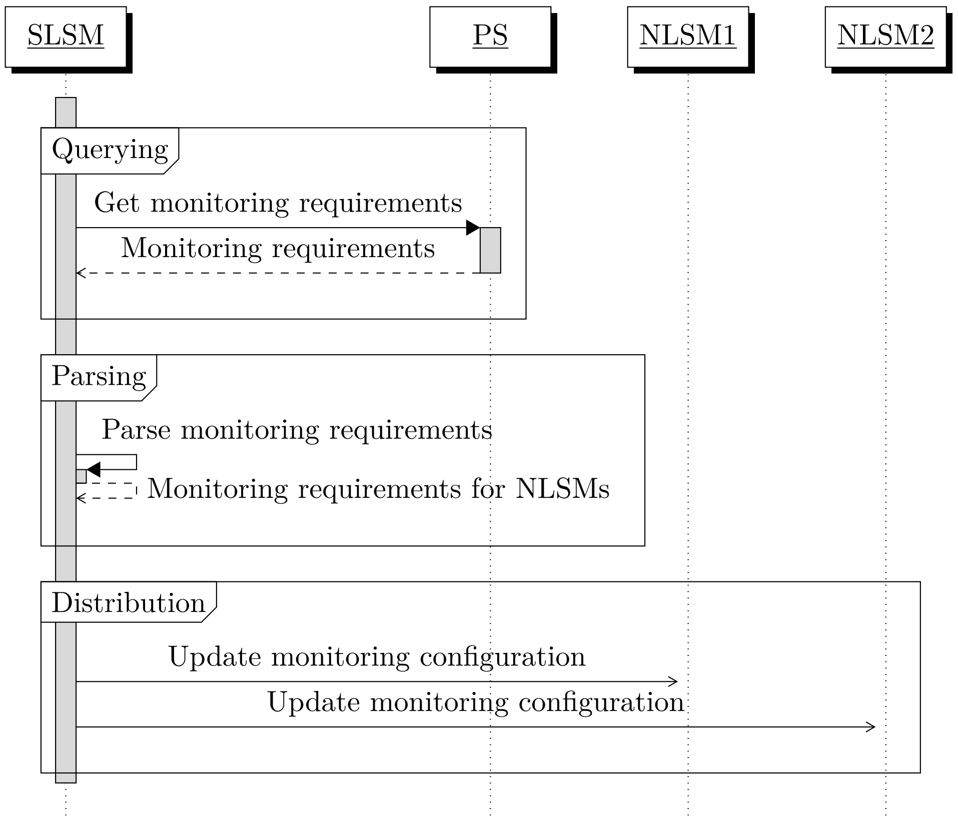
SLSM
PS
NLSM1
NLSM2
Querying
Get monitoring requirements
Monitoring requirements
Parsing
Parse monitoring requirements
Monitoring requirements for NLSMs
Distribution
Update monitoring configuration
Update monitoring configuration

Figure 4.4: Monitoring requirements sequence diagram

Final and essential *PS* functionality is providing placement layout for *SLSM*. By providing current service layout of the cluster, *SLSM* receives updated placement layout in a timely manner, as well adhering to **RO2**.

For resource monitoring *VSL* will be used, as it has been well established between all components already. This way, *PS* can directly query *NLSM*s and their appropriate configurations. Using *VSL*'s type search, allows to avoid a race condition of when monitoring requirements are not fully distributed to all nodes.

Returned placement layout is parsed by *SLSM* that operates on deploying new services or migrating already running ones.

4.5 Placement strategies

This section overviews proposed designs for placement strategies that would integrate with current implementation of *DS2OS*. In general, as discussed in section 4.4, all placement strategies are separate microservices that comply with a proposed *VSL* communication interface. This means that the possibilities of placement strategies are not limited by the proposed design in this section, but rather by the algorithms that provide an updated placement layout.

4.5.1 Resource balance

This placement strategy attempts to model the site so that every *NLSM* uses similar amount of resources. It is noticeable that the balancing relies on absolute (specific usage of resource) metrics and not relative ones (percentage of utilization), due to possibility *IoT*'s heterogeneous environment.

It was the initial approach by [3] (section 2.4.9) and was analyzed in [12] related work as a heuristic.

The approach relies on a sequential service to *NLSM* assignment by finding the least loaded *NLSM*. Algorithm 2 displays the general workflow on how the services are assigned. Due to high dimensionality of resources, as this strategy considers all of the available resource type groups, to evaluate which *NLSM*s have highest resource capability, a sum of all their resources is generated. This resonates with the initial design approach by [3], which was as well listed out as problem 5.

```
input  : $layout_{slsm}$ ;                         // Current SLSM service layout
input  : $usages_{nlsm}$ ;                          // Resource usage per NLSM
input  : $caps_{nlsm}$ ;                       // Resource capabilities per NLSM
output: $layout_{ps}$ ;                // Strategy's proposed new service layout
$layout_{ps} \leftarrow \emptyset$
$usages_{service} \leftarrow ResourceusageperNLSM$
foreach $nlsm \leftarrow usages_{nlsm}$ do
    $usages_{nlsm.services)} \leftarrow$ equal usage split from $nlsm$
end
foreach $service \leftarrow layout_{slsm}$ do
    foreach $nlsm \leftarrow sorted(caps_{nlsm})$ ;          // Sorted by sum of resources
     do
        if $service$ $usage$ $fits$ $in$ $NLSM$ $free$ $resources$ then
            $layout_{ps} \leftarrow service, nlsm$ $caps_{nlsm} \leftarrow cap_{nlsm} - usages_{service}$
        end
    end
end
```

Algorithm 2: Resource balance service to NLSM assignment

As discussed in previous sections in design and related work, resource balancing provides optimization in having a buffer of resources for service usage peaks. This strategy supposedly would allow to better control the whole site's balancing approach.

Alternatives, or most likely directions where this placement strategy could be put in the future would be:

1. Closer resource monitoring
2. Service differentiation
3. Profiling resources

Current resource monitoring is limited by implementation which provides usages per *NLSM*. The assumption is then made that all services use the same amount of resources within the same *NLSM*. While this might sound as obviously wrong solution to be used, an important feature of placement strategies is that they provide continuous updates for placement layout. That way, if the usages differ greatly, further placement layouts would likely granulates higher resource usage services by continuous reordering. The downside means that to archive an optimal state from a resource-balance perspective, it will take multiple migrations, therefore - more time. To avoid this, closer resource monitoring, as in monitoring resource usage by each service itself would allow resource-balance strategy to propose the optimal solution in a single placement layout.

Service differentiation idea was used in [38] to identify long-running services. This identification is significant as long-running services have higher priority over short-running ones, which can be removed in favor of long-running services. In resource-balance perspective, the layout could identify service priority and thus provide greater resource buffers. This would surely introduce complexity by evaluating priority and adjusting the placement procedure. Introduction of priorities would likely require a calculation and scaling of available buffers, and this might be a challenge in a heterogeneous environment.

Finally, resource balancing in current design does not track service usage. Differently from the first future issue of closer monitoring, tracking service usage would enable strategy to profile the service usage between migrations and have make a better assumption of what the service usage is.

4.5.2 Networking and processing performance ranking

This strategy partly relies on service ranking proposed in [48]. Here services are ranked mainly by their monitored network usage, which is considered as highest order resource. In algorithm 3 service placement firstly relies on service network usage sorting. Additionally, next important step is hosting *NLSM*'s processing capability (denoted as *CPU*), which sorts the NLSMs in that manner. The assumption is made that the most network hungry services communicate with each other, as the *VSL* communication is bi-directional (request-response). As in section 4.5.1 services are placed only in NLSMs that have all the requirements fulfilled for the service.

The rationality behind using higher processing capability *NLSM*s for network intensive services is based on the assumption that the network requests would be fulfilled quicker as the transportation over network cost would be diminished.

By attempting to put services that have high network usage into single *NLSM*s this placement strategy expresses its optimization goals as network load minimization.

A straight forward alternative would be introduction of service-service network monitoring to be able deduce which pairs of services conduct high and frequent loads of network communication. This would be an alternative solution for deeper monitoring, discussed in resource balancing strategy (section 4.5.1).

Other than that, service communication profiling would provide a better understanding of how services communicate. This proposal is different of the one in resource balancing strategy, as it saturates on profiling service relations. This could be designed both

```
input  : $layout_{slsm}$ ;                          // Current SLSM service layout
input  : $usages_{services}$ ;                       // Resource usage per service
input  : $caps_{nlsm}$ ;                         // Resource capabilities per NLSM
output: $layout_{ps}$ ;                  // Strategy's proposed new service layout
$net\_sorted\_usage_{services} \leftarrow sortservicesbynetworkusage$
$cpu\_sorted\_caps_{nlsms} \leftarrow sortnlsmsbytheirCPUcapability$
foreach $service$ $in$ $sorted(usages_{services}$ ;            // services sorted by their
 network usage
 do
    foreach $nlsm$ $in$ $sorted(caps_{nlsm})$ ;            // NLSMs sorted by their CPU
     capability
     do
        if $service$ $usage$ $fits$ $in$ $NLSM$ $resources$ then
            $layout_{ps} \leftarrow service, nlsm$ $caps_{nlsm} \leftarrow cap_{nlsm} - usages_{service}$
        end
    end
end
```
Algorithm 3: Networking and processing performance service to NLSM assignment

apriori, by analyzing service packages of their VSL communication calls, and prior deployment, where service communication calls can be tracked in real time.

Chapter 5

Implementation

This chapter overviews decision when implementing design (chapter 4). Given that both *VSL* core libraries and *DS2OS* site implementation are written in Java programming language, all of the design implementation is done in Java as well. Obviously, by using microservices architecture (Section 2.2.2), it is feasible to use any technologies, the main constraint was that *VSL* communication client (connector) is written in Java programming language. Alternatives would be using *Jython* - *Python* programming language running inside Java Virtual Machine (*JVM*), but there is *Jython* minimal project maintenance and it relies on an old version 2.7 of Python, which has end-of-life scheduled as the start of 2020.

5.1 Integration

Additional integration into *DS2OS* site was needed. Apart from essential fixes introduced (section 2.4.9) in attempt to make *DS2OS* site implementation feasible for evaluation, multiple additional features were introduced:

1. Resource monitoring implementation in *SHE* and *NLSM*
2. Relevant communication endpoints in *SLSM*

Previous implementation of resource monitoring relied on custom crafted CPU relative usage calculation which was unmaintainable (see section 2.4.9) and usage information is useless as it provides relative usage (0-100%) on a heterogeneous *DS2OS* site's node. The proposed changes introduce an additional library - OSHi[1].

For exposing resources that *SHE* exposes for monitoring, a VSL node of type */basic/list* was used (listing 5.1).

Listing 5.1: Exposed resource context model

```
<sheResource type="/basic/composed">
    <value type="/basic/text" writer="" />
    <max type="/basic/text" writer="" />
    <min type="/basic/text" writer="" />
</sheResource>
```

Here each resource provides 3 additional nodes which correspond to capabilities of the resource:

1. **max** - maximum possible value,
2. **min** - minimum possible value,
3. **value** - current resource usage value.

SHE exposes *resources* node through VSL as it was the [24] design decision in implementing *NLSM* and *SHE* inter-communication. In general, *SHE* is considered as an isolated node, which only allows *NLSM* to communicate with it. Thus resource exposure only in *SHE* is not sufficient, since services (as minimal example, placement strategy) inside *DS2OS* site need to have access to the resources.

NLSM implements a proxying methodology for replicating the exposed list of resources and routing the monitoring requests to relevant *SHE* resource node. By subscribing to *SHE*'s resource list, NLSM is able to synchronize the changes between its list and *SHE*'s (listing 5.2). Here a list of new resources (*addedResources*) and expired (*deletedResources*) is deduced thus providing specific resources to add and delete from *NLSM*'s replicated resources list.

Listing 5.2: NLSM's SHE resource synchronization method

```
private void synchronizeSheResources() throws VslException {
    String myAddress = connector.getRegisteredAddress();
    String sheAddress = getSheAddress();
    ArrayList<String> sheResources = new ArrayList<String>(
        Arrays.asList(
            connector.get(
                sheAddress + "/resources/elements").getValue().split(";")));
    ArrayList<String> nlsmResources = new ArrayList<String>(
        Arrays.asList(
```

[1] Native Operating System and Hardware Information library: `https://github.com/oshi/oshi`

```
            connector.get(
                myAddress + "/resources/elements").getValue().split(";")));
    //deletedResources = nlsm - she
    //addedResources = she - nlsm
    ArrayList<String> deletedResources = new ArrayList<String>(nlsmResources);
    deletedResources.removeAll(sheResources);
    ArrayList<String> addedResources = new ArrayList<String>(sheResources);
    addedResources.removeAll(nlsmResources);

    LOGGER.info("SHE:␣" + sheResources.toString() +
                ";␣NLSM:␣" + nlsmResources.toString());
    LOGGER.info("Added:␣" + addedResources.toString() +
                ";␣deleted:␣" + deletedResources.toString());

    for (String deletedResource: deletedResources) {
        connector.get(myAddress + "/resources/del/" + deletedResource);
    }

    for (String addedResource: addedResources) {
        connector.get(
            myAddress + "/resources/add/ds2os/sheResource//" + addedResource);
        addResource(addedResource);
    }
}
```

In similar fashion, *SLSM* method responsible for deducing which services have to be migrated is described in algorithm 4. Here $services_{new}$ and $services_{removed}$ are mapping of services added and deleted from an *NLSM*.

```
input  : $layout_{curr}, layout_{ps}$
output: $services_{new}, services_{removed} foreachNLSM$
foreach $NLSM$ do
    $services_{new}(NLSM) \leftarrow layout_{ps}(NLSM) - layout_{curr}(NLSM)$
    $services_{removed}(NLSM) \leftarrow layout_{curr}(NLSM) - layout_{ps}(NLSM)$
end
```

Algorithm 4: SLSM's migration deduction method

5.2 Placement Strategy Picker

Due to time limitations, current *PSP* implementation directly compares deployment goals with *PS* proposed optimization targets. Similarly to how *NLSM* replicates *SHE*'s exposed resources section 5.1, *PSP* subscribes to *placementStrategies* VSL list for changes and updates internal model of every *PS*.

On request for optimal placement strategy, *PSP* compares internal model with posed requirements and deduces best match.

Goal matching is implemented has three distinct branches:

1. If all goals are matched by multiple *PS*s, first one (non-deterministic) *PS* is picked,
2. If some goals are matched, they are weighted and best match *PS* is picked.
3. If none goals are matched, first one (non-deterministic) *PS* is picked.

Namely, 2nd branch can be taken as non-trivial task is described in algorithm 5. Here $goals.getRelativeWeight$ retrieves the relative weight of a specific goal. Goals are defined as a list of items with relative weight, which by default is equal between all goals (if not set).

```
input  : $PSs, goals$
output: $PS_{optimal}$
$scores \leftarrow listofscoresforeveryPS, initializedas0$ foreach $PSs$ do
    foreach $PS.optimizations$ do
        $scores_{PS} \leftarrow scores_{PS} + goals.getRelativeWeight(optimization_{PS})$
    end
end
return $PSwithmax(scores)$
```
Algorithm 5: PSP goal matching when some goals are matched by PSs

5.3 Placement strategies

This section overview implemented placement strategies and most importantly the base platform that they're implemented upon. Base reduces the overhead of custom implementation for ontology based resource monitoring, *VSL* communication, and monitored data management. As most of the implementation relies on template that covers peripherals, the implementation of each placement strategy just relies on the template and appropriate algorithms proposed in design chapter (chapter 4).

5.3.1 Ontology

Ontology based monitoring relies on predefined resource groups. A snippet example is shown in listing 5.3.

Listing 5.3: Ontology resource group distinction

```
switch(resourceName) {
    case "CPU":
        return ResourceGroup.CPU;
    case "Memory":
    case "RAM":
    case "SWAP":
        return ResourceGroup.Memory;
    case "Network":
    case "Bandwidth":
        return ResourceGroup.Network;
    ...
}
```

Important issue when defining such resource groups is that this has to be initially coordinated between *PS* base platform and *SHE* itself. While this issue couples, there's weak coupling between the base platform and implementation of *PS* itself. This is because of dependency on the resource groups that base presents. It is assigned to project development maturity by exposing wide range of resource group mappings, thus making the coupling stable, and is out of scope for this thesis.

5.3.2 VSL communication

Since VSL context model for *PS* is set, the base platform handles and parses the requests, thus removing the need for *PS* developer to re-implement communication layer.

Listing 5.4: Placement Strategy's interface for custom PS implementation

```
public interface PlacementStrategy {
    public void start();

    public void stop();

    public String getCurrentState();

    public ResourceGroup[] getMonitoringRequirements();

    public Map<String, String> getPlacementLayout(Map<String, String> oldLayout);

    public void pushResourceUsage(Map<String, Map<String, String>> resourceUsage);
}
```

In general, the developer needs to implement the *PlacementStrategy* interface, shown as listing 5.4. Here *start*, *stop*, and *getCurrentState* provide state management for *PS* implementation. Expected states are:

1. **STOPPED** - when *PS* is not performing any operations,
2. **LOADING** - when preparation is in process. Eventually should change to **STOPPED** or **RUNNING**,
3. **RUNNING** - indicator that placement strategy is operational.

$getMonitoringRequirements$ provides a list of resources for monitoring, and $pushResourceUsage$ receives a snapshot of all resources monitored in whole *DS2OS* site and handles internally. Finally, *getPlacementLayout* provides a mapping of new layout $< NLSM, service >$ from an old layout.

5.3.3 Monitored data management

Monitoring is a data intensive activity providing a lot of information. Without any management, this information would overload the process's operating memory even though the process does not utilize it. To tackle this, two data management solution have been implemented:

1. Archiver - archiving data into storage after a specific threshold,
2. Archive rotation - purging archived data from storage after a specific threshold.

Every data point is a timestamp and the resource usage value. Current implementation's threshold relies on data size (i.e. data entries), but for future work it is possible to extend it to threshold by time range. In a similar fashion, archives are rotated by a trivial size threshold.

Chapter 6

Evaluation

This chapter evaluates service placement implementation for *DS2OS* site. To evaluate in a quantitative fashion, a simulation framework has been developed and is described in section 6.1. Only feasible evaluation is described in section 6.2.

6.1 Simulation framework

Due to *DS2OS* implementation with applied fixes (Section 2.4.9) being inappropriate for performing stable and reproducible evaluations, a simulation framework has been developed. It replicates the relevant functionalities for placement strategy microservice:

1. *SLSM* running services (current layout) interface,
2. *NLSM* resource monitoring interface.

As all of the communication is done through VSL (Section 2.3),

real *SLSM* and *NLSM* context models were used and VSL nodes, that are relevant for *PS* evaluation, were implemented.

Figure 6.1 displays the simulation framework components. Here $SLSM_{sim}$ and $NLSM_{sim}$ are parts of simulation, whereas PS is real.

Here services and their resource usages are as well simulated. The rationality behind resource usage simulation is that artificially modelled service resource usage behavior model would not deviate significantly. While it does not represent realistic resource usage, an artificially modeled service would not represent actual resource usage and requirements as well.

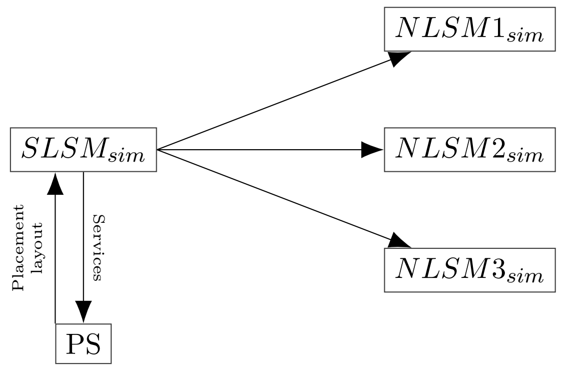

Figure 6.1: Evaluation simulation framework's components

Additionally, since the only interface that resource usage is capable to be monitored through is *NLSM*'s *VSL* node, it already provides an abstraction over resource usage.

Discussing corner cases, where service resource usage disrupts *NLSM* functionality is out of scope for this thesis, as it is a domain of service management within *SHE*.

Simulation framework works on the premise that an evaluation is a test that has multiple stages:

1. Initial state - site setup with already running services,
2. Migration transfer and installation - whenever *PS* proposes a different service layout, migration is triggered,
3. Post-migration state - site with newly setup service layout.

Service execution induces a time constraint based suspension. For example, a service can *work* for around $50ms$. The same time constraint is put during migration and communication domains: an operation has a preset time that it will take as a minimum. To not rely on service always finishing in constant time, the sleep time is randomized with gaussian distribution.

Comparison of result between initial and post-migration states allows to validate *PS*'s positive impact. Additionally it is possible to estimate the benefit of the migration. Evaluating change during migration state allows to track and estimate the cost of the migration. Comparison of costs and benefits of migrations enables modeling when proposed layout pays off.

In retrospective, using simulation framework raises questions regarding the meaningfulness of results. As it attempts to simulate the real environment as close as possible, it is relevant from architectural perspective. This means that the evaluation test cases could be seamlessly plugged into working *DS2OS* implementation (which is not due to the reasons mentioned in section 2.4.9). Of course, the measures of fixing and stabilizing the *DS2OS* are currently applied and it could likely modify the *VSL* interface. Then this simulation framework would lose relevance.

As this simulation framework provides mocked environment with resource usages, another question can be raised on meaningfulness of results. The evaluation of placement strategy is to evaluate whether it provides improvements into the *DS2OS* site. As the state of the site is discrete (discrete services running in discrete *NLSM*s controlled by *SLSM*), it provides equal initial state for any placement strategy. Additionally, placement strategy does not provide anything else just a proposition of how services should be placed (layout). Generally, if not for dependence on resource monitoring simulation, evaluation could be conveyed by analyzing proposed layout in how well it corresponds to *DS2OS* site.

6.2 Execution time

Measuring task (call to service inside *DS2OS* site and response) execution time is an evaluation for placement strategies. Due to the limitations of current *DS2OS* site implementation and simplifications applied in evaluation simulation framework (section 6.1), this section evaluates overall capability and possible workload of a system.

The clearest benefit the *PS*s present is faster execution. By imposing different service placement, migrations are needed. Figure 6.2 presents expected evaluation execution times flow on strategies. This expectation figure can be split into two parts: migration, and execution change after migration. It is expected that migration induces load on the nodes and thus the execution quality suffers. On ther other hand, evaluation method attempts to find whether placement strategy provides improval and degradation. In figure 6.2 it is expected that placement strategy introduced placement layout will help to decrease the execution time. Here vertical axis t_e represents overall time of execution, whereas horizontal axis t - general timeline.

This evaluation can be split into two domains:

1. Single service placement evaluation,
2. Composite service placement evaluation.

Even though service usage is simulated, single services are more prone to migration as migration benefit is only a single request, whereas composite services communicate

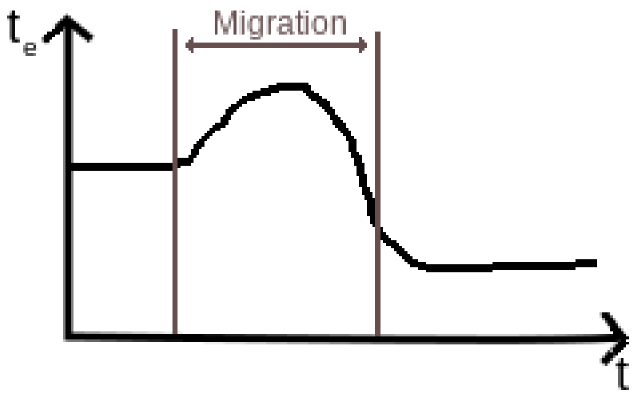

Figure 6.2: Mock figure to display expectations in task execution times

in-between thus saturating migration benefits. Additionally, due to full simulation, relevance of resource type is negligible.

The evaluation scenario can be split into the following stages:

1. Initial state with deployed services is provided and recorded,
2. Placement strategy layout is incrementally applied (migration) and recorded,
3. After-migration site's state is recorded.

In constant time intervals task execution time evaluation requests are submitted, spanning throughout all of the stages.

By recording relevant states, in this case - task execution time, it is then feasible to evaluate impact of proposed placement layout.

Before describing setups it is important to note that placement strategies do not have access to usage per-service rather than general *NLSM* resource usage, as described in section 4.2. Additionally, migration costs are also unknown apriori.

6.2.1 Single service evaluation

Table 6.1 describes the resource requirement for services and their mappings in *DS2OS* site. Here CPU, Memory, and network usages are just mocked values as the simulation framework (section 6.1) does not provide actual measured values.

Table 6.2 describes the *NLSM* resource capabilities. In the same manner, the values are mocked, but these values with table's 6.1 values are exposed to placement strategies.

Service ID	NLSM	CPU usage	Memory usage	Network usage	Runtime (ms)
1	NLSM1	100	100	100	60
2	NLSM1	150	50	120	60
3	NLSM2	200	150	150	60
4	NLSM3	50	200	100	60

Table 6.1: Single service setup usages

NLSM	CPU	Memory
1	300	300
2	300	200
3	500	300
Migration	20	20

Table 6.2: NLSM single service setup capabilities

The runtime of a service is a mocked value that simulation framework's service idles to imitate load, before responding to a request.

It is assumed that all *NLSM*s are connected with 10ms latency. This latency is mainly used and should be visible in the case of composed services.

In this setup, for single-service evaluations the service of interest is 1. By design it is independent and does not rely on other services.

Resource balancer

Figure 6.3 displays the results of resource balancer placement strategy. The horizontal axis represents the general timeline, as in time that took the evaluation method to execute. The vertical axis is the evaluation target value - service execution time. It describes how long did it overall take for the client to request the service, the service to process the request, and return the value back to the client. A spike of execution time can be seen as migration has been triggered.

The pre-migration average execution time is $74.5ms$ and post-migration $80.2ms$, the difference 5.7. Red enclosing lines mark the formal migration start and finish in the simulation framework, yet it can be seen that execution time did not immediatelly get reduced post-migration. This can be labeled as partly problem of the simulation framework, as the simulation is not efficiently parallelized. Another noticeable feature in figure 6.3 is that pre-migration and post-migration execution times do not significantly differ.

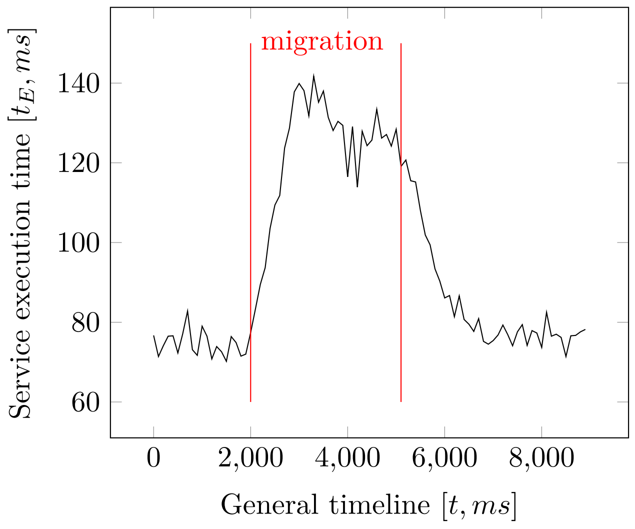

Figure 6.3: Single service execution evaluation for resource balancing strategy

Network and processing ranking

Similarly to results of resource balance single strategy's service execution time in section 6.2.1, pre-migration and post-migration execution times do not differ significantly in figure 6.4. The average execution time pre-migration was $74.5ms$, and post-migration was $76.2ms$ spanning $1.7ms$ difference. The differences in migration length and load pattern between resource balance strategy and network and processing ranking strategy can be simply stated as difference of proposed layouts and thus differently executed migrations. As migration management is not part of this thesis, evaluation relevance in negligible.

6.2.2 Composite service evaluation

While using the same setup as in single-service execution evaluation, the difference here is that the service of interest is 2. In simulation framework it's set-up that it relies on services 1 and 3 for communication.

Resource balancer

In 6.5 it can be seen that pre-migration and post-migration execution times do not have a significant difference. The average execution time pre-migration was $239.0ms$ and post-migration $241.8ms$, making the difference $2.8ms$. This could be explained as resource balancer strategy executed it's placement layout calculation algorithm without any composite service communication in mind.

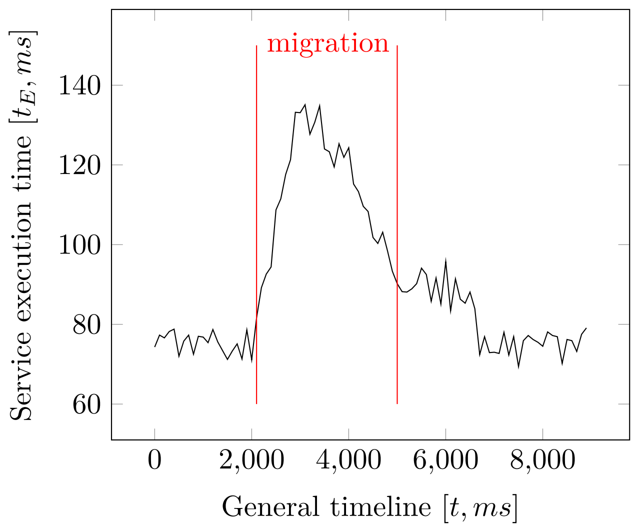

Figure 6.4: Single service execution evaluation for network and processing ranking strategy

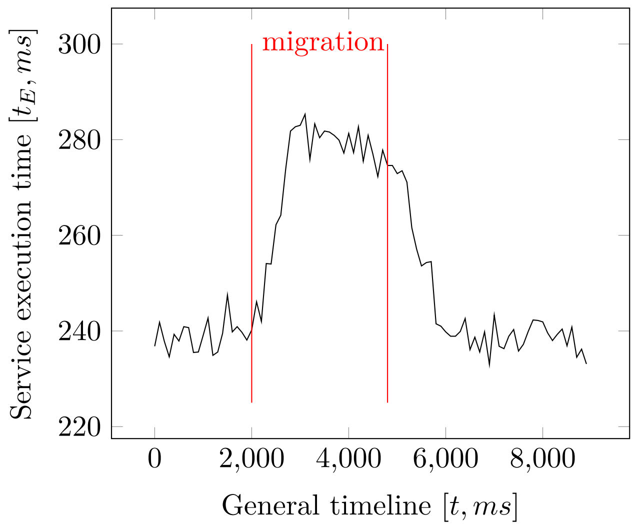

Figure 6.5: Composite service execution evaluation for resource balancing strategy

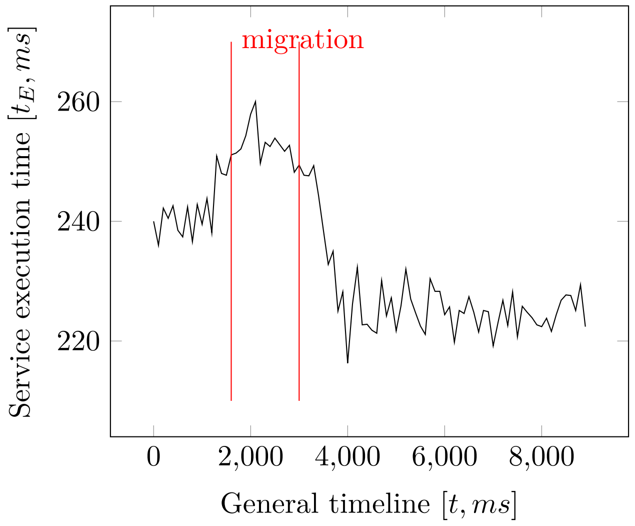

Figure 6.6: Composite service execution evaluation for network and processing ranking strategy

Network and processing ranking

Figure 6.6 has a significant change of execution time. Average execution time before migration was $242.2ms$ and post-migration $226.9ms$, resulting in $-15.3ms$ difference. It can be assumed that the proposed placement layout grouped relevant services together thus building difference in execution time.

6.2.3 Conclusions

Resource balance strategy did not provide clear benefit for execution time optimization both in single service and composite service scenario. Network and processing ranking strategy had no clear benefit for single service execution, whereas composite service execution evaluation shows a significant improvement in execution time.

While evaluations are highly limited by the flexibility of simulation framework to give a thorough inspection, the results, namely from figure 6.6, give incentive to see that placement strategy would improve (reduce) the execution time.

Chapter 7

Conclusion

This thesis overviewed service placement approaches to apply them for *IoT* environments via *VSL* based *DS2OS* framework. Two distinct problem domains were identified in analysis for service placement: need for description, and methodology on placements. Analyzed related work gave insights in partial solutions, as it was observed in table 3.2, for designing a placement strategy integration into *DS2OS*. While reparations for implementations by [3] and [24] were not sufficient to use for evaluation of this thesis, a mock simulation framework that reuses relevant interfaces from aforementioned theses. For future perspective, when the *DS2OS* service management implementation is stable enough, the evaluation methodologies, due to reuse of interface, will remain relevant. As simulation framework mocks elements of the system, the results can be interpreted with the bias of simulation framework's implementation (section 6.1). The evaluation results show that placement strategies applications have significant impact for *IoT* environment solutions, of which one was *DS2OS*.

Finally, by reiterating the research questions and attempting to answer them, concludes this thesis. This thesis found that answering **RQ1.** "What characteristics are needed for optimal *DS2OS service placement*" (and respective subquestions) is non trivial. First of all, optimality is subjective to goals and thus characteristics depend on the service placement strategy itself. The fulfillment from thesis was to design and implement a flexible enough characteristic description system where *NLSM*s update to placement strategy's required resources for monitoring. **a**) "How can **RQ1.** characteristics be used to elevate decision making for *service placement*" has partial connection to previous research question as generally placement strategies rely on a set of monitored *DS2OS* site characteristics. **b**) "What methodologies can be used to run an online *service placement and migration* strategy?" specfically identified that any *LP* solutions are unsuitable for scalable and continuous running, especially in *IoT* domain. Most of the methodologies that comply are heuristics that attempt to replicate the solutions of *LP* service placement algorithms. Thesis did not extentively tackle **c**) "How can the methodologies be designed to be feasible to run in *IoT* edge devices?". Part of the assumptions were placed on fulfilling **b**), another part by using *DS2OS* framework with *VSL* to simplify complex communication. **d**) "How significant is placement strategy's ability to scale with the increase of *IoT* edge devices?" was not evaluated due to usage of simulated framework for evaluations. Answer **RQ3.** "What effect does the proposed design and implementation have regarding optimization strategies such as *least-nodes*, and *resource balancing* heuristics?" relied on resource balancing placement strategy only. A placement strategy that took networking and processing resources into account was designed. The evaluation results have shown that composite service execution time can be reduced, whereas single service execution time can not.

7.1 Future work

During thesis, a lot of distinct directions were found to work in the future.

In short they are:

FW1 Placement strategy picker

FW2 Deeper resource monitoring

FW3 Placement strategy switching

FW4 Combining placement strategies

FW5 Migration management

FW1's *PSP*, described in section 4.4, can have extended functionality in controlling which *PS* is chosen. Currently the choosing phase is a one-off, yet it could be adapted to end-users changing requirements during execution, as *PS* integration is designed to be seamless.

FW2 would provide thorough monitoring data. While this would need *PS* adjustment, it would allow better modeling of resource requirements for services, by providing data and parameters for *Machine Learning* algorithms. As communication relies on *VSL* (Section 2.3), analyzing requests between services would additionally provide another dimension in which services impose their requirements.

Given that a single *PS* is oriented towards a specific set of goals, with extension of **FW1**, *PS* switching would be inevitable in **FW3**. While technically it is a trivial task, the actual point of interest is rationale behind switching. In **FW1** one example was given as end-users changing their requirements. Another case would be identification of *PS* effect and *deciding* to switch to another *PS* as provided benefits are better.

FW4 was a frequent discussion in thesis meetings whether running multiple *PS*s is feasible. Rationale behind that was that a composition of services from a single deployment would prefer a specific goal, whereas another deployment would have different goals defined. Without much doubt, it is clear that there's a possibility to have two clashing placement strategies which would produce completely different layouts. Is it feasible to combine *PS*s and if yes - how? Does *PS* combination provide better performance for *DS2OS* site than running separately.

Lastly, **FW5** is about migration management. As it can be seen in evaluation, migration overhead is noticeable. It is important to note, that evaluation was done on the base of a single migration. This thesis did not cover when and how often should migrations be performed.

Another migration related future work could be regarding inclusion of migration costs. This would give incentive to model and attempt to minimize amount of service migrations performed.

It was not included, but a clear future work can be design of yet another placement strategy for the *DS2OS* site. Given current evaluation methodology and independent simulation framework from section 6.1, it is possible to apply a custom *PS* and evaluate its performance.

Due to the *DS2OS* site implementation problems described in section 2.4.9, relevance of simulation framework is short-lived. With further fixing and development on the *DS2OS* site, it is likely better to apply described evaluation method on the actual implementation itself, which therefore would provide a more realistic performance evaluation of the *PS* and big-scale integration test for the *DS2OS* implementation too.

Chapter A

List of acronyms

BPEL4WS	Business Process Execution Language for Web Services
CCSC	Cloud Computing Service Composition
CM	Context Model
CMR	Context Model Repository
CPU	Central Processing Unit
DS2OS	Distributed Smart Space Orchestration System
FM	Feature Model
HMM	Hidden Markov Model
HTTP	Hyptertext Transfer Protocol
IaaS	Infrastructure as a Service
IoT	Internet of Things
JVM	Java Virtual Machine
KA	Knowledge Agent
LP	Linear Programming
MEC	Mobile Edge Computing
NFV	Network Function Virtualization
NFV-RA	Resource Allocation in NFV
NLSM	Node Local Service Manager
NS	Network Service
OSGi	Open System Gateway initiative
PaaS	Platform as a Service

PS	Placement Strategy
PSP	Placement Strategy Picker
QoS	Quality of Service
RTE	Real-Time Environment
S2S	Smart Space Service management
S2Store	Smart Space Store
SaaS	Software as a Service
SHE	Service Hosting Environment
SLA	Service Level Agreement
SLCA	Site Local Certificate Authority
SLSM	Service Local Service Manager
SOA	Service Oriented Architecture
STM	Status Transition Matrix
URL	Uniform Resource Locator
VM	Virtual Machine
VNF	Virtual Network Function
VSL	Virtual State Layer

Bibliography

[1] Martin Bauer et al. "IoT Reference Model". In: *Enabling Things to Talk: Designing IoT solutions with the IoT Architectural Reference Model.* Ed. by Alessandro Bassi et al. Berlin, Heidelberg: Springer Berlin Heidelberg, 2013, pp. 113–162. ISBN: 978-3-642-40403-0. DOI: 10.1007/978-3-642-40403-0_7. URL: https://doi.org/10.1007/978-3-642-40403-0_7.

[2] David Benavides, Pablo Trinidad, and Antonio Ruiz-Cortés. "Automated Reasoning on Feature Models". In: *LNCS, ADVANCED INFORMATION SYSTEMS ENGINEERING: 17TH INTERNATIONAL CONFERENCE, CAISE 2005.* Springer, 2005, p. 2005.

[3] Deniz Celik. "Semi-Autonomous IoT Service Management on Unattended Nodes". PhD thesis. 2018.

[4] Sergei Chichin et al. "Smart Cloud Marketplace - Agent-Based Platform for Trading Cloud Services". In: *2014 IEEE/WIC/ACM Int. Jt. Conf. Web Intell. Intell. Agent Technol.* (2014), pp. 388–395. DOI: 10.1109/WI-IAT.2014.193. URL: http://ieeexplore.ieee.org/document/6928211/.

[5] Krzysztof Czarnecki, Simon Helsen, and Ulrich Eisenecker. "Formalizing cardinality-based feature models and their specialization". In: *Software Process: Improvement and Practice.* 2005, p. 2005.

[6] Lorenzo Donini. "Autonomous Certificate Management for Microservices in Smart Spaces". PhD thesis. 2018. ISBN: 6122738700.

[7] Wanchun Dou et al. "HireSome-II: Towards privacy-aware cross-cloud service composition for big data applications". In: *IEEE Trans. Parallel Distrib. Syst.* 26.2 (2015), pp. 455–466. ISSN: 10459219. DOI: 10.1109/TPDS.2013.246.

[8] Abdessalam Elhabbash et al. "Cloud Brokerage: A Systematic Survey". In: (2018). arXiv: 1805.09018. URL: http://arxiv.org/abs/1805.09018.

[9] Rodica Gherghina and Ioana Duca. "Using Linear Programming in order to Optimize the Allocation of Resources for Investment". In: *Journal of Knowledge*

Management, Economics and Information Technology 3.1 (2013), pp. 1–12. URL: https://ideas.repec.org/a/spp/jkmeit/1354.html.

[10] Juliver Gil Herrera and Juan Felipe Botero. "Resource Allocation in NFV: A Comprehensive Survey". In: *IEEE Trans. Netw. Serv. Manag.* 13.3 (2016), pp. 518–532. ISSN: 19324537. DOI: 10.1109/TNSM.2016.2598420.

[11] J. Octavio Gutierrez-Garcia and Kwang Mong Sim. *Agent-based cloud service composition.* Vol. 38. 2013. DOI: 10.1007/s10489-012-0380-x.

[12] M. Shamim Hossain et al. "Resource allocation for service composition in cloud-based video surveillance platform". In: *Proc. 2012 IEEE Int. Conf. Multimed. Expo Work. ICMEW 2012* (2012), pp. 408–412. DOI: 10.1109/ICMEW.2012.77.

[13] C.I. Huang and K. Yoon. "Multiple Criteria Decision Making: Methods and Applications". In: 1981.

[14] Foued Jrad, Jie Tao, and Achim Streit. "A Broker-based Framework for Multi-Cloud Workflows Steinbuch Centre for Computing". In: *Proc. 2013 Int. Work. Multi-cloud Appl. Fed. clouds* (2013), pp. 61–68. DOI: 10.1145/2462326.2462339.

[15] Amin Jula, Zalinda Othman, and Elankovan Sundararajan. "A hybrid imperialist competitive-gravitational attraction search algorithm to optimize cloud service composition". In: *Proc. 2013 IEEE Work. Memetic Comput. MC 2013 - 2013 IEEE Symp. Ser. Comput. Intell. SSCI 2013* 3 (2013), pp. 37–43. DOI: 10.1109/MC.2013.6608205.

[16] Amin Jula, Elankovan Sundararajan, and Zalinda Othman. "Cloud computing service composition: A systematic literature review". In: *Expert Syst. Appl.* 41 (2014), pp. 3809–3824. DOI: 10.1016/j.eswa.2013.12.017. URL: http://romisatriawahono.net/lecture/rm/survey/networksecurity/Jula-CloudComputing-2014.pdf.

[17] Kevin Kofler, Irfan Ul Haq, and Erich Schikuta. "A parallel branch and bound algorithm for workflow QoS optimization". In: *Proc. Int. Conf. Parallel Process.* (2009), pp. 478–485. ISSN: 01903918. DOI: 10.1109/ICPP.2009.34.

[18] Thorsten Kramp, Rob van Kranenburg, and Sebastian Lange. "Introduction to the Internet of Things". In: *Enabling Things to Talk.* Berlin, Heidelberg: Springer Berlin Heidelberg, 2013, pp. 1–10. ISBN: 9783642404030. DOI: 10.1007/978-3-642-40403-0_1. arXiv: arXiv:1011.1669v3. URL: https://link.springer.com/book/10.1007/978-3-642-40403-0http://link.springer.com/10.1007/978-3-642-40403-0{_}1.

[19] Sheng Liu et al. "Service composition execution optimization based on state transition matrix for cloud computing". In: *Proc. 10th World Congr. Intell. Control Autom.* (2012), pp. 4126–4131. DOI: 10.1109/WCICA.2012.6359167. URL: http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=6359167.

[20] Pavel Mach and Zdenek Becvar. "Mobile Edge Computing: A Survey on Architecture and Computation Offloading". In: *IEEE Commun. Surv. Tutorials* 19.3 (2017), pp. 1628–1656. ISSN: 1553877X. DOI: 10.1109/COMST.2017.2682318. arXiv: 1702.05309.

[21] Hendrik Moens et al. "Hierarchical network-aware placement of service oriented applications in clouds". In: *IEEE/IFIP NOMS 2014 - IEEE/IFIP Netw. Oper. Manag. Symp. Manag. a Softw. Defin. World* (2014). DOI: 10.1109/NOMS.2014.6838230.

[22] Arlen J. Mogull R. "Security Guidance for critical areas of focus in Cloud Computing V4.0". In: *CSA (Cloud Security Alliance), USA. Online: https://cloudsecurityalliance.org/guidance v4.0* (2017).

[23] 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.

[24] Fabian Benedikt Ohlenforst. "Providing a Remotely Manageable Runtime Environment for IoT Services". PhD thesis. 2018.

[25] Marc Oliver Pahl, Georg Carle, and Gudrun Klinker. "Distributed smart space orchestration". In: *Proc. NOMS 2016 - 2016 IEEE/IFIP Netw. Oper. Manag. Symp.* 2016, pp. 979–984. ISBN: 9781509002238. DOI: 10.1109/NOMS.2016.7502936.

[26] NFV White Paper. *Network Functions Virtualisation: An Introduction, Benefits, Enablers, Challenges & Call for Action. Issue 1.* Tech. rep. ETSI, Oct. 2012.

[27] Ioannis Patiniotakis, Yiannis Verginadis, and Gregoris Mentzas. "Preference-based cloud service recommendation as a brokerage service". In: *Proc. 2nd Int. Work. CrossCloud Syst. - CCB '14*. New York, New York, USA: ACM Press, 2014, pp. 1–6. ISBN: 9781450332330. DOI: 10.1145/2676662.2676677. URL: http://dl.acm.org/citation.cfm?doid=2676662.2676677.

[28] Tran Vu Pham et al. "A service composition framework for market-oriented high performance computing cloud". In: *Proc. 19th ACM Int. Symp. High Perform. Distrib. Comput.* (2010), pp. 284–287. DOI: 10.1145/1851476.1851511.

[29] Klaus Pohl, Günter Böckle, and Frank Linden. *Software Product Line Engineering: Foundations, Principles, and Techniques.* Jan. 2005. ISBN: 978-3-540-24372-4. DOI: 10.1007/3-540-28901-1.

[30] Lie Qu, Yan Wang, and Mehmet A Orgun. "Cloud Service Selection Based on the Aggregation of User Feedback and Quantitative Performance Assessment". In: *2013 IEEE Int. Conf. Serv. Comput.* (2013), pp. 152–159. DOI: 10.1109/SCC.2013.92. URL: http://ieeexplore.ieee.org/document/6649690/.

[31] Clément Quinton et al. "Towards Multi-Cloud Configurations Using Feature Models and Ontologies". In: (). URL: http://delivery.acm.org.eaccess.ub.tum.

de/10.1145/2470000/2462332/p21-quinton.pdf?ip=129.187.254.46{\&}id=2462332{\&}acc=ACTIVESERVICE{\&}key=2BA2C432AB83DA15.B4538F6A74FA55F8.4D4702B0C3E38B35.4D4702B0C3E38B35{\&}{_}{_}acm{_}{_}=1523363827{_}e128d2234a1b4d879db9802b9784c8ab.

[32] Antonino Rullo et al. "A Game of Things: Strategic Allocation of Security Resources for IoT". In: *Proc. Second Int. Conf. Internet-of-Things Des. Implement.* (2017), pp. 185–190. DOI: 10.1145/3054977.3055001. URL: http://doi.acm.org/10.1145/3054977.3055001.

[33] Thomas L. Saaty, Luis G. Vargas, and Klaus Dellmann. "The allocation of intangible resources: the analytic hierarchy process and linear programming". In: *Socio-Economic Planning Sciences* 37.3 (2003), pp. 169 –184. ISSN: 0038-0121. DOI: https://doi.org/10.1016/S0038-0121(02)00039-3. URL: http://www.sciencedirect.com/science/article/pii/S0038012102000393.

[34] Le Sun et al. "Cloud service selection: State-of-the-art and future research directions". In: *Journal of Network and Computer Applications* 45 (2014), pp. 134 –150. ISSN: 1084-8045. DOI: https://doi.org/10.1016/j.jnca.2014.07.019. URL: http://www.sciencedirect.com/science/article/pii/S108480451400160X.

[35] Smitha Sundareswaran, Anna Squicciarini, and Dan Lin. "A brokerage-based approach for cloud service selection". In: *Proc. - 2012 IEEE 5th Int. Conf. Cloud Comput. CLOUD 2012* (2012), pp. 558–565. ISSN: 2159-6182. DOI: 10.1109/CLOUD.2012.119.

[36] Switching Technology et al. "Cloud model for service selection". In: *2011 IEEE Conf. Comput. Commun. Work. (INFOCOM WKSHPS)* 60821001 (2011), pp. 666–671. DOI: 10.1109/INFCOMW.2011.5928896. URL: http://ieeexplore.ieee.org/xpls/abs{_}all.jsp?arnumber=5928896{\%}5Cnhttp://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=5928896.

[37] Alexander Totok. "Exploiting Service Usage Information for Optimizing Server Resource Management". In: *ACM Trans. Internet Technol* 11.1 (2011). DOI: 10.1145/1993083.1993084. URL: https://static.googleusercontent.com/media/research.google.com/en//pubs/archive/37194.pdf.

[38] Abhishek Verma et al. "Large-scale cluster management at Google with Borg". In: (). DOI: 10.1145/2741948.2741964. URL: https://static.googleusercontent.com/media/research.google.com/en//pubs/archive/43438.pdf.

[39] Hai Wang and Shouhong Wang. "Ontological Map of Service Oriented Architecture for Shared Services Management". In: *Expert Syst. Appl.* 41.5 (Apr. 2014), pp. 2362–2371. ISSN: 0957-4174. DOI: 10.1016/j.eswa.2013.09.034. URL: http://dx.doi.org/10.1016/j.eswa.2013.09.034.

[40] Erik Wittern, Jörn Kuhlenkamp, and Michael Menzel. "Service-Oriented Computing". In: 7636 (2012), p. 34321. DOI: 10.1007/978-3-642-34321-6. URL: http://link.springer.com/10.1007/978-3-642-34321-6.

[41] Daniel Worm et al. "Revenue maximization with quality assurance for composite web services". In: *Proc. - 2012 5th IEEE Int. Conf. Serv. Comput. Appl. SOCA 2012* (2012). DOI: 10.1109/SOCA.2012.6449452.

[42] Qingtao Wu et al. "A QoS-satisfied prediction model for cloud-service composition based on a hidden markov model". In: *Math. Probl. Eng.* 2013 (2013). ISSN: 1024123X. DOI: 10.1155/2013/387083.

[43] Abdullah Yousafzai et al. "Cloud resource allocation schemes: review, taxonomy, and opportunities". In: *Knowledge and Information Systems* 50.2 (2017), pp. 347–381. ISSN: 0219-3116. DOI: 10.1007/s10115-016-0951-y. URL: https://doi.org/10.1007/s10115-016-0951-y.

[44] Cheng Zeng et al. "Cloud Computing Service Composition and Search Based on Semantic". In: 2007 (2009), pp. 290–300. DOI: 10.1007/978-3-642-10665-1_26. URL: http://link.springer.com/10.1007/978-3-642-10665-1{_}26.

[45] Miranda Zhang et al. "A Declarative Recommender System for Cloud Infrastructure Services Selection 2 A System for Cloud Service Selection". In: *9th Int. Conf. Econ. Grids, Clouds, Syst. Serv. GECON 2012* (2012), pp. 102–113. DOI: 10.1007/978-3-642-35194-5{_}8.

[46] Xiangliang Zhang et al. "Virtual Machine Migration in an Over-committed Cloud". In: Vm ().

[47] Zhiming Zhang, Chan Ching Hsu, and Morris Chang. "Cool Cloud: A Practical Dynamic Virtual Machine Placement Framework for Energy Aware Data Centers". In: *Proc. - 2015 IEEE 8th Int. Conf. Cloud Comput. CLOUD 2015* (2015), pp. 758–765. DOI: 10.1109/CLOUD.2015.105.

[48] Zibin Zheng et al. "QoS ranking prediction for cloud services". In: *IEEE Trans. Parallel Distrib. Syst.* 24.6 (2013), pp. 1213–1222. ISSN: 10459219. DOI: 10.1109/TPDS.2012.285.