BRNO UNIVERSITY OF TECHNOLOGY
VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ

FACULTY OF INFORMATION TECHNOLOGY
FAKULTA INFORMAČNÍCH TECHNOLOGIÍ

DEPARTMENT OF INFORMATION SYSTEMS
ÚSTAV INFORMAČNÍCH SYSTÉMŮ

GENERIC DECENTRALIZED SELF-ADAPTIVE CONTEXT-AWARE ARCHITECTURE MODEL

GENERIC DECENTRALIZED SELF-ADAPTIVE CONTEXT-AWARE ARCHITECTURE MODEL

PHD THESIS
DISERTAČNÍ PRÁCE

AUTHOR **Ing. M. MOHANNED KAZZAZ**
AUTOR PRÁCE

SUPERVISOR **Doc. Ing. JAROSLAV ZENDULKA, CSc.**
ŠKOLITEL

BRNO 2019

Abstract

The evolution in information system continuously raises demands for more efficient, effective and adaptive cooperation between system's components to cope with changes in the system and to guarantee its best performance. Two main approaches have been introduced to achieve these requirements. First, the self-adaptation approach which enables information system to adapt to the changes in context information of the system and its surrounding environment based on an adaptation strategy. Second, context-awareness approach which enables to monitor the context information and recognize those changes that can trigger the adaptation process.

In this work we introduce a generic context-aware self-adaptive architecture model to support software system with adaptation functionalities that guarantee system's availability, operation conditions and performance. Moreover, we provide two real-life case studies as a proof-of-concept of the applicability and re-usability of our proposed adaptation approach.

Abstrakt

Vývoj v informačním systému neustále zvyšuje nároky na účinnou, efektivní a adaptivní spolupráci mezi komponenty systému, aby se vyrovnal se změnami v systému a zaručil tak nejlepší výkon. K dosažení těchto požadavků byly zavedeny dva hlavní přístupy. Přístup k adaptaci umožňuje informačnímu systému přizpůsobit se změnám v kontextu informací systému a jeho okolního prostředí na základě adaptační strategie. Přístup ke zvyšování informovanosti zase napomáhá sledovat informace o kontextu a rozpoznat změny, které mohou proces adaptace vyvolat.

V této práci představujeme obecný kontextově orientovaný model vlastní adaptivní architektury pro podporu softwarového systému s adaptačními funkcemi, které zaručují dostupnost systému, provozní podmínky a výkon. Navíc poskytujeme dvě případové studie v reálném životě jako důkaz konceptu použitelnosti našeho navrhovaného adaptačního přístupu.

Keywords

Self-adaptation, software architecture, context model, decentralized control, context awareness.

Klíčová slova

Adaptabilita, softwarová architektura, model kontextu, decentralizované řízení, sledování kontextu.

Reference

KAZZAZ, M. MOHANNED. *Generic decentralized self-adaptive context-aware architecture model.* Brno, 2019. PhD thesis. Brno University of Technology, Faculty of Information Technology. Supervisor Doc. Ing. JAROSLAV ZENDULKA, CSc.

Generic decentralized self-adaptive context-aware architecture model

Declaration

I hereby declare that the thesis is my own work that has been created under the supervision of doc. Ing. Jaroslava Zendulky CSc. It is based on the seven papers [42, 43, 44, 41, 45, 46, 47] that I have written jointly with my supervisor specialist RNDr. Marek Rychlý, Ph.D. Where other sources of information have been used, they have been duly acknowledged.

.....
M. MOHANNED KAZZAZ
July 4, 2019

Acknowledgements

First, I would like to thank my parents and family for supporting me with their love and prayers all my life. Thank you so much!

I would like to thank doc. Ing. Jaroslava Zendulka, CSs., for his invaluable advice, support and guidance during his supervision of this work. Also, I would like to thank RNDr. Marek Rychlý, Ph.D., who has been a great source of guidance, inspiration and assistance during my doctoral studies.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

The ever-developing nature of the distributed system arises demanding requirements of the design process for an automatic and robust management means. To satisfy these requirements, Self-adaptation has been proposed to support software systems with the mechanism to modify their behaviour and maintain their goals flexibly and robustly through an automatic reaction to information context changes of 1) actor's requirements, 2) surrounding environment, 3) and, the system itself [65, 17]. The reaction is a result of specific monitoring strategy the system should follow. Four functionalities have been defined in [73] as required functionalities in Self-adaptive system, the system must: 1) monitor context information of system and environment, 2) detect changes, 3) decide the adaptation plan to perform and 4) act by executing the chosen adaptation plan. On the other hand, Context Awareness approach has been proposed to solve the problem of information misunderstanding between system distributed components over different domains. It addresses the need to provide a unified model of information context which helps the realization of system adaptation by supporting system information context understanding, monitoring and discovery of context changes in the operational environment.

The self-adaptive context-aware framework for stationary and mobile devices presented in this thesis is a framework for enabling self-adaptation and context-awareness in information system. It utilizes ontology-based model to describe system components including their properties and preferences. Using an ontology-based model enables the adoption of context awareness approach concepts of context modelling, monitoring and context reasoning. Moreover, the framework supports the utilization of a decision-making process to choose the best adaptation scenario based on defined set of criteria. To present the application of the decentralized context-aware adaptive architecture model proposed in this research, we provided two case-studies with a detailed description of system configuration and framework's analysis and evaluation.

The introduced adaptive architecture model provides an answer to the question of how to support information systems with a dynamic response to changes in their surrounding environment. In other words, how to design a formal architectural model that supports information system reconfiguration during runtime based on the changes in system components and the current state of system environment. Moreover, the contribution of this thesis leverages the adoption of system adaptation in service-oriented mobile architectures. Also, it provides researchers with an adaptive architecture model supported with a multi-criteria decision-making process, which facilitates and easies the design and implementation of new adaptation scenarios through the utilization of the provided adaptive architecture model and implemented framework.

In the next section we present the motivation scenario behind this work and our proposed approach to support adaptation in software system and to solve the limitations of current approaches.

1.1 Motivation

Inspired by [93, 74], let us consider a traffic jam detection system as a real-life scenario of utilizing adaptation in car navigation system. The motivation behind utilizing the adaptation is to solve service's loss situation and to help drivers to avoid traffic bottlenecks on roadways. To design such a system, it is required to enable traffic information exchanging between cars. For that, traffic information should be formally modelled and correctly understood by the navigation system application on each car. On the other hand, the navigation system requires to be context-aware by continuously monitoring and analysing traffic status to detect traffic jam situations that trigger a process of re-planning the route. Moreover, a process of deriving of alternative routes and recommending the best one should be provided. These requirements can be satisfied by defining a unified taxonomy of context terms to describe the system and its components. Context information like car speed, position, traffic status, route details, number of surrounding cars, etc., are pieces of information that should be realized and exchanged between cars during run-time. This information should be noted formally as car context model and published to be discovered by other cars. The usage of car context model facilitates the integration of context-awareness approach by enabling context changes monitoring and discovery. Moreover, it supports the utilization of self-adaptation to traffic information changes (i.e., traffic jam status), by defining adaptation conditions and the utilization of a proper decision-making process used to choose the best adaptation decision.

The importance of our research resides in the following points:

- The introduced context model of system components allows to provide a formal and common understanding of system context information between different application domains.
- The abstract core self-adaptive architecture model can be extended and customized to adopt new adaptation scenarios.
- The proposed framework facilitates system adaptation and service provision in mobile architectures.
- The proposed framework supports the integration of self-adaptation on existing information systems regardless their technical implementations which minimizes the upgrade effort to enable self-adaptation in those systems.

1.2 Important Definitions

In this section we introduce definitions of the important basic concepts we use throughout the thesis. These definitions provide the meaning of few topic-related terms and concepts that will be explained in more details later in the thesis.

- **System** : the set of related hardware and software units implemented together to perform specific business application.

- **Controller** : the software unit responsible for performing the considered adaptation strategies in the system.
- **Orchestration** : the set of protocols, coordinations, and activities between software and hardware systems designed to achieve an automated process [25].
- **Centralized Controlled System** : a system that is controlled using a central processing unit that is responsible for performing the whole business processes.
- **Decentralized Controlled System** : system that is controlled by two or more computers physically distributed to different places (for example, at the location of system database or data source) to improve system's performance (for example, by avoiding transferring of large amount of data) and security.
- **Environment** : the operational surrounding and conditions where the System and its users operate.
- **Context** : the information representing the status of the System and Environment parts. This information can be acquired and processed for specific applications.
- **Service** : a piece of code designed to perform a specific computation in the system [26].
- **Device** : a machine operating in the system. (i.e., mobile device, printer, camera, etc.).
- **Service Provider** : a device set up with a running HTTP server to host services.
- **Sensor/Context Provider** : a device that provides specific type of data (i.e., light sensor).
- **Service Migration (Mobility)** : The ability of a Controller to move a service from one Service Provider to another [60, 15].
- **Efficiency** : The ratio of needed computation work dedicated to avoid operational violations in the system and improve system performance to the total impact of performed computation work on system performance.

1.3 Aim of the Thesis

Software architecture can be described as the "blueprints" of a software system at the highest level of abstraction [79]. It describes the software system as a set of components and their interactions through notation and documentation to provide better understanding and analysing of the system [79, 88]. The importance of software architecture is realized in its evolution and customization factors which enable the architecture to adapt itself and to evolve to match new usages [67].

The evolution of software architecture is categorized into two types [16]: the static evolution, which refers to the need to stop and restart the system if architectural modifications is required and the dynamic evolution, that allows the application of these modifications without causing interruptions or failures in the system. However, many approaches to model software architectures were contributed, using different notations of components and

connectors which raised a problem of misunderstanding or interpretation of these documentations as there is no formal description available [51].

From an architectural point of view, Service-Oriented Architecture (SOA) [26], Component-based development (CBD) [12], and Microservice Architecture [63] are the state-of-the-art approaches introduced to provide a formal architecture design style for modern information systems and to cope with their distributed nature by supporting software system components reusability, communications and interoperability.

Self-adaptive approach has been proposed to guarantee software system goals by dynamically reacting and adapting to changes in the environment and user requirements to continue providing a reliable service. Researchers concentrate on self-adaptive system requirements by modelling the properties of these requirements from goal-oriented [1] perspective, aspect-oriented perspective [62] and agent-oriented [8]. On the other hand, other researchers [66, 76] addressed the context-aware adaptation and context management as challenges to design self-adaptive system regarding its uncertainty in understanding the meaning of context information and its changes over different development environments.

Contributions on self-adaptation [1, 62, 85] have used the goal-directed approach [19] to model and define the objectives of the system as goals and sub-goals. Researches like [95, 22], tried to achieve system goals using the component-based approach by supporting components composition and defining the process plan through different implementation techniques.

In [92], the authors provided a new way to improve self-adaptive architecture by addressing the need to separate between the application and adaptation concerns which is considered a method towards coping the requirements of adaptive systems.

In [93], the authors addressed the decentralized self-adaptation by defining the attributes of decentralized self-adaptive systems and demonstrating the defined attributes through two case studies. The adaptation requires a reconfiguration of the system that costs more adaptation time. The context providers cooperate in different system nodes to avoid the lack of information through a centralized control (Master/Slave) organization.

The thesis focuses on investigating the design requirements needed to achieve self-adaptation in information system by introducing a self-adaptation context awareness architectural model. The proposed architecture model shall support dynamic discovery of system components, an automatic identification of critical situations requiring the adaptation and provide a mechanism to survive these critical situations by implementing a defined adaptation plan. This thesis studies existing approaches of context-information modelling to provide a formal model that can be used to describe system components statuses and operating conditions so that changes in these statues can be recognized and understood to be used in planning and performing the adaptation processes.

1.4 Thesis Objective

The general goal of this research is to design a decentralized self-adaptive architecture. In this architecture, we want to provide the possibility of implementing self-adaptation not only on the user side (i.e. end user interface) but also on the system itself in the way that allows the system to adapt regarding context's changes (i.e. light, temperature, communication bandwidth, battery status) through a decentralized adaptation which can minimize the adaptation costs, guarantee the quality of provided services and improve system performance. The specific objectives supporting the general objective can be summarized as follows:

O1 To provide a formal system context model which enables the understanding of context and context's changes meanings and promotes context awareness in the system.

O2 To provide a context-aware self-adaptive architecture model that adapts to the context's changes of system components and its surrounding environment.

O3 To provide a framework that supports adaptation in the system. The framework allows different types of devices to cooperate in centralized controlled orchestrations to solve a problem of context's loss or uncertainty by including new services that do not affect with the changes causing the problem.

O4 To analyse the efficiency of implementing the decentralized adaptation on system side on appropriate case studies.

1.5 Thesis Contribution

According to the proposed objectives, the following contributions are provided:

C1 A system ontology to support the usage of common understanding of context information between different domains. A detailed description of the proposed ontology is provided in Section 3.4 and Section 5.4.1.

C2 A formal ontology-based context model that allows to describe system component context model and provides a method to ease the discovery process of new context providers in the system. See Section 3.5 for more details.

C3 A framework for distributed context-aware self-adaptive system (presented in Section 6.2), is provided to support an automatic system adaptation to context's changes of the system and environment. The adaptation guarantees the operation conditions to keep a desired Quality of Service (QoS) performance level. Moreover, the distributed mechanism will improve system performance by distributing system tasks of adaptation and context processing over several controllers which helps to avoid system overloads that could happen when utilizing the centralized approach.

C4 An extensible adaptation architecture model (see Section 5.5) that can be easily customized by researchers over new case studies. The extensibility of the model eases and stimulates conducting research work on both self-adaptation and context-awareness.

C5 A decision-making process to support choosing the best adaptation scenario from a set of alternative adaptations based on set of prioritized criteria. See Sections 4.1 and 4.2 For more details.

1.6 Structure of the Thesis

This thesis is divided into four parts as follows: Part I "Theoretical Background", Part II "Proposed Approach", Part III "Implementation and Experimental Results", and Part IV "Future Work and Conclusion", which contains the following chapters: Chapter 2, Chapters 3–5, Chapters 3–6, and Chapters 7–8-9, respectively.

Theoretical Background: In Chapter 2, we provide the state-of-the-art study of this work starting with an introduction of the Service-Oriented Architecture approach and the

motivation behind adopting it in this work. Later, we review the existing literature implementing the self-adaptive and context awareness approaches. In addition to the provided state-of-the-art study, a supplementary information related to the state-of-the-art and related work dealing with more specific areas is covered in parts II and III. Finally, we demonstrate a real-life problem statement and a summary of the-state-of-the-art study.

Proposed Approach: In Chapter 3, we introduce our proposed context-aware self-adaptive SOA meta-model. We present the ontology-based component context model proposed to describe system components. In Chapter 4, we demonstrate the decision making process and algorithms developed to support decision making in our approach. In Chapter 5, we provide an adaptive SOA based model by applying the meta-model proposed in Chapter 3. The demonstrated architecture model adopts service migration between service providers as the adaptation strategy to survive violations in service and providers preferences.

Implementation and Experimental Results: In Chapter 6. we provide the analysis of requirements and implementation description of the proposed service migration system. In Chapter 7, we provide two case studies to present the application and efficiency of our service migration approach to solve this thesis motivation example provided in Chapter 2. In Chapter 8, we discuss the future work and in Chapter 9, we summaries the thesis approach and highlight its contributions.

Part I

Theoretical Background

Chapter 2

State of the Art

In this Chapter, we analyse some of the most important state-of-the-art approaches of self-adaptive software systems. In Section 2.1 we provide a review of Service-Oriented Architecture which provides the needed abilities for consumers to invest system services between the distributed information systems and reusing their available resources. Context and its categorization and models are described briefly in Section 2.1.6. In the later sections, we present the existing approaches used in software system. In Section 2.2, we deal with adaptive system approaches, presenting the existing contributions based on system adaptation and reconfiguration of systems based on context's system changes. Later, we review the existing Context-aware systems and the context-aware adaptation approaches. Finally, we provide summary and conclusion of the state-of-the-art in Section 2.1.

2.1 Service Oriented Architecture

Service Oriented Architecture (SOA) is a paradigm for organizing and utilizing distributed capabilities that can be under the control of different ownership domains. It provides a uniform means to offer, discover services and use their capabilities to produce desired effects consistent with measurable preconditions and expectations [56].

A Service provides documentation about its capabilities and the information necessary to interact with it. SOA provides the ability for service consumers to discover and use the functionalities and capabilities of the participated services. The main role of SOA is to match these consumers requirements and capabilities of each party through a matching process of the provided documentations.

In SOA, interaction between a service consumer and a service is achieved by a series of information exchanges which is controlled through certain policies.

2.1.1 The Motivation of Service Oriented Architecture

The central objective of a service-oriented approach is to reduce dependencies between „software islands“ which basically comprise the services and clients accessing those services [83].

The main drivers for SOA-based architectures are to facilitate the manageable growth of large-scale enterprise systems, to facilitate Internet-scale provisioning and use of services and to reduce costs in organization to organization cooperation [56].

SOA based architectures enable the cooperation between software entities from different environments to achieve new needed business processes. Reusing these software and avail-

able resources from different networks reduces time and cost of developing new software systems.

2.1.2 Service

In the context of SOA, the term service can be defined as „an implementation of a well-defined piece of business functionality, with a published interface that is discoverable and can be used by service consumers when building different applications and business processes“ [26].

Another definition of a service is provided by [56], „A service is a mechanism to enable access to one or more capabilities, where the access is provided using a prescribed interface and is exercised consistent with constraints and policies as specified by the service description“.

2.1.3 Service Attributes

As service represents the corner stone in SOA, it must have common attributes to be able to satisfy SOA standards. These attributes are described by [26] as a common set of service-level design principles mostly associated with service orientation:

- Services are reusable. Regardless of whether immediate reuse opportunities exist, services are designed to support potential reuse.
- Services share a formal contract. For services to interact, they need not share anything but a formal contract that describes each service and defines the terms of information exchange.
- Services are loosely coupled. Services must be designed to interact without the need for tight, cross-service dependencies.
- Services abstract underlying logic. The only part of a service that is visible to the outside world is what is exposed via the service contract. Underlying logic, beyond what is expressed in the descriptions that comprise the contract, is invisible and irrelevant to service consumers.
- Services are composable. Services may compose other services. This allows logic to be represented at different levels of granularity and promotes reusability and the creation of abstraction layers.
- Services are autonomous. The logic governed by a service resides within an explicit boundary. The service has control within this boundary and is not dependent on other services for it to execute its governance.
- Services are stateless. Services should not be required to manage state information, as that can impede their ability to remain loosely coupled. Services should be designed to maximize statelessness even if that means deferring state management elsewhere.
- Services are discoverable. Services should allow their descriptions to be discovered and understood by humans and service requestors that may be able to make use of their logic.

With these principles of service, SOA will be easily configurable by developers and reusable in different implementations by modifying one service or more to achieve the business goal of the system.

2.1.4 Modelling of Web Service

„A Web Service (WS) is a software system designed to support interoperable machine-to-machine interaction over a network. It has an interface described in a machine-processable format (specifically WSDL). Other systems interact with the Web service in a manner prescribed by its description using SOAP-messages, typically conveyed using HTTP with an XML serialization in conjunction with other Web-related standards“ [18].

From this definition it can be realized the ease of WS usage in software system as it uses standardized technologies through the unique Uniform Resource Identifier (URI) in the system and the Extensible Markup Language (XML). XML is used to describe Web service methods in the Web Service Description Language (WSDL) document and also the Simple Object Access Protocol (SOAP) messages used to achieve information communications between Web services [26, 21].

The Web service model presented by [21] consists of three entities, the *Service Provider*, the *Service Registry* and the *Service Consumer*. The service provider creates and offers the web service with its standardized XML description on the service registry. The service registry has the required information about the service provider and technical documentation of the service. The service consumer uses, locates and retrieves the information from the registry then uses the obtained service description to invoke the web service.

The Representational State Transfer (REST) architecture style has been introduced in [28] to standardize the description and interaction with Web services. REST defines a Web service as set of resources that can be identified and reached by users through a URI. Each resource can be managed through a set of operations (retrieve, create, update and delete) through the Hypertext Transfer Protocol (HTTP) methods (GET, PUT, POST and DELETE) representing these operations respectively.

2.1.5 Web service Composition

The interoperability of the Web services enables it to invoke other Web services in the environment to achieve the business logic. This invocation of different Web services is called *web service composition*. A new *composite web service* can be created by composing *basic services* and composite services. A Web service composition provides a method to achieve business logic of the system by web services of more complex tasks.

Service composition strategies have been categorized in [21] into five different categories. We briefly note two that match with the interest of our work.

The first strategy is static and dynamic composition. It is related to time of composition. Static composition takes place during design-time of planning the architecture. In contrast, dynamic composition is planned and takes place during run-time of the application.

The second strategy is manual and automatic composition. The main aspect of this strategy deals with whether the composition is made by a human intervention or by the system itself.

2.1.6 Context Information

The need to get better and specific understanding the meaning of information about the system and its surrounding environment has become highly important. This information is referred by term „context“. Different definitions have been provided according to the usage of this information as one of the first contributions [75] defines context as location, identity of users, and nearby people.

In [2] the authors define context as follows: *„Context is any information that can be used to characterize the situation of any entity. An entity is a person, place, or object that is considered relevant to the interaction between a user and an application, including the user and applications themselves.“*.

A refined definition of context information is provided in [76]: *„Context is a combination of any information that can be sensed or received by an entity which is useful to catch events and situations.“*.

2.1.7 Categorization of the Context

Regarding to the nature of context and its different recourses, the need for classifying the context is considered an important factor to understand and model it [35].

In [35], the authors provided four main characteristics of context information:

- *Context is temporal:* Context relates to time factor, so it can be *static* or *dynamic*.
- *Context is imperfect:* Context ranges between *correct, incorrect, inconsistent, and incomplete* according to its representation of the true state of the environment.
- *Context has many alternative representations:* to fit with different types of application.
- *Context information is highly interrelated:* by presenting the relationships between the entities which describe the context.

2.1.8 Context Models

A good context information modelling formalism of reduces the complexity of context-aware applications and improves their maintainability and evolvability [10].

Bettini et al. [10] proposed the following requirements which should be met by the context information models:

- *Heterogeneity and mobility:* Regarding the vast of context sources, sensors, databases and user data, the model of context should be able to express about the captured meaningful data from these resources with a consideration of its different capturing rates and changes (i.e., dynamic from sensors or static from user data). The mobility expresses the ability to continuously model context obtained from mobile sources which can be captured from different environments.
- *Relationships and dependencies*: The context model should also contain relational information about the captured context information of the entities, objects or location that may interact together in the environment or have dependencies on other entities.

- *Timeline:* As it is not possible to save all the captured information during all the life time of the system, the ability to summarize this information should be applied for some future usage.
- *Imperfection:* Regarding the imperfect nature of context as it is described in Section 2.1.7, there is need for good modelling to evaluate the usage of context information.
- *Reasoning:* To determine if there is a change in the observed context of environment. A reasoning technique is applied to enable the system to capture context changes and evaluate their importance to ensure the consistency verification of context model.
- *Usability of modelling formalisms:* Models should provide an easy way to obtain a readable understandable context for both developers and applications.
- *Efficient context provisioning:* Context models should be supported by good techniques *(i. e. indexes)* to enable efficient access and usage of the modelled context.

In the same work [10], three context models were proposed to fit with these requirements:

1. *Object-role based models of context information*

   This model is based on the database modelling techniques to support query processing and reasoning. The main strength of the object-role based model is its support for various stages of the software process. On the other hand, its main weakness is the „flat“ information model, through it all types of context are uniformly represented as atomic facts.

2. *Spatial models of context information*

   It organizes its context information by the physical location which could be predefined as static locations or obtained as sensed locations by mobile sensors. Spatial context models are well suited for applications that are mainly location-based, like many mobile information systems. However, the drawback of spatial context models is the effort of gathering up to date location data of the context information.

3. *Ontology-based models of context information.* Ontological models of context provide clear advantages both in terms of heterogeneity and interoperability. They support the usability by user-friendly graphical tools that make the design of ontological context models viable to developers who are not particularly familiar with description logics. The drawback of ontology-based models is the very little support of modelling temporal aspects in ontologies and the serious performance issues [10] caused by the ontological reasoning with OWL-DL.

2.2 Existing Approaches and Frameworks

In this section we provide an overview on the current approaches of self-adaptive systems and context-aware systems. Later, we address the limitations in these works and introduce our proposed contribution to solve these limitations.

2.2.1 Adaptive Systems

There have been many researches to introduce a formal or semi-formal architecture model of adaptive systems that can adapt its behaviour or architecture in response to changes in its environmental context [85, 82, 24].

Self-adaptation was first introduced by IBM through the „Autonomic Computing“ approach [48] describing self-managing system using a central controller. The following functionalities have been introduced to define self-managing software system.

- Self-configuration which presents a system's ability to configure itself automatically according to high level policies of its objectives.
- Self-optimization which is achieved by a system through continuously seeking to improve and upgrade itself and its functionality by applying the latest versions of its components.
- Self-healing of adaptive system which is the ability to detect, diagnose and repair its components automatically.
- Self-protection which is the ability of continually predicting and defending system failures or attacks.

AgentScape [94], is a multi-agent middleware provided to support the development of adaptive and reconfigurable applications. An AgentScape application is described as a set of agents and objects that can be invoked by agents to perform some processing. AgentScape model uses naming and location services to enable agents to communicate using message-passing communications and to migrate to other locations in the system.

Mobile-C [15] is a mobile agent platform that facilitates mobile agent communication and migration. It has been provided to improve agent cooperation by reducing data transmission between agents which leads to improve response time in real-time applications. The communication is achieved through IEEE Foundation for Intelligent Physical Agents (FIPA) [64] agent communication language (ACL) messages encoded in XML. Mobile-C supports system adaptation by providing mobile agents with mechanism to discover changes and perform unanticipated actions by dynamically deploying new algorithms and code.

Rainbow [30], an adaptation framework developed to support software systems with the self-adaptation functionalities mentioned above through a reusable infrastructure. It proposes the usage of an architectural style to define and encode system-specific knowledge during design time. This knowledge describes system as sets of components, properties, rules, analysis, operations, and strategies. The Rainbow framework supports a software system with the infrastructure to perform monitoring of the defined properties and constraints of system components, evaluating the constraints, discovering violations in the constraints and triggering adaptation process to react to any violation through the adaptation strategies defined to guarantee specific system concerns (i.e., system performance).

Jade [8], an agent development framework that facilitates the development and management of agent-based self-adaptive applications. It provides the tool to define agent platform, containers and agents and their tasks. Agent tasks can be extended by defining new behaviour class together with a behaviour ontology describing the term of this behaviour and then assigning the new behaviour class to the agent object. Jade framework supports the utilization of different ontologies to support different application domains. Using the ontology guarantees the correct understanding of messages between agents. Moreover, the

framework facilitates the integration of Web service through supporting a bidirectional interconnectivity between agents and Web services. Web services can be registered/deregistered in the Universal Description, Discovery, and Integration (UDDI) registry to be discovered and invoked by Jade agents. Jade supports the mobility of its mobile agents between different containers of the same platform. However, moving a mobile agent between Jade containers is only supported by a manual process requiring the definition of platform hops for an agent to visit till it reaches the destination.

Da Silva et al. [82], proposed a generic framework for the automatic generation of processes for self-adaptive software systems so that it can be applied to different application domains. The framework uses workflows, model-based and artificial intelligence (AI) planning techniques to design adaptation plans. They used a standardized AI planning language, the Planning Domain Definition Language (PDDL) [29], to define a system model that is composite of 1) a domain representation stating system's actions (or available tasks that can be used to formulate the adaptation plans), and 2) a problem representation that defines the system initial state and the desired goal. However, the proposed representation does not define relationships between system components and/or system properties. Moreover, the defined system context representation is limited to a static set of terms defined during design time. Which does not answer the question of how to support the automatic usage of newly available resources and tasks in the system during runtime.

Tang et al. [85], have proposed a goal-directed model of self-adaptive software architecture. They presented a goal requirement specification model to formulate a self-adaptive software architecture model. The presented goal requirements model is defined using the Keep All Objectives Satisfied (KAOS) modelling framework [87]. The defined self-adaptive architectural model consists of two sub models: the structural model and the behavioural model. The structural model is derived by mapping system goal and sub goals into components. Each component has a controller and adaptation manager. The behavioural model is provided through a set of processes mapping the interactions between system components into Finite-State Machine (FSM) based control flows. Each process is demonstrated as a subset of Communicating Sequential Processes (CSP) notations [37] representing the goal's and sub goals' processes. The CSP annotations supports the description of goal as hieratical decomposition tree of other sub-goals based on Parallel composition (i.e., AND based composition) and Sequential Compositions (i.e., OR based composition). The adaptation is performed in two steps. First, building a goal decomposition tree. Second, reconfiguring the bindings between software components based on the corresponding goal decomposition tree. To do that, each component's controller collects data about its component, analyses the data in comparison to a defined component's knowledge base of goal plans. Then each adaptation manger adapts its component's structural configuration and dynamic behaviour in accordance to its planned goals. At the end of the adaptation processes of adaptation managers, the goal decomposition trees of all components will be generated assembling the goal decomposition tree of the software. Then, the goal decomposition tree is translated into an FSM presenting the software control flows.

FUSION framework for self-adaptive systems is based on self-tuning approach [24]. It uses a technique of analysing system features to define a system model that copes dynamically with the unanticipated system conditions. Feature relationships are used to improve the adaptation planning during runtime. The feature model of the system consists of one core and other features. These features are related by two kinds of relationships: 1) dependency which defines the prerequisites of this feature, and 2) mutual exclusion that helps to enable only one feature from the group of similar features. System context is presented as

a *metric* which is a measured value of system properties. Moreover, *utility* refers to a user's context or his/her preference about a specific metric. They presented an analytical method to derive the behavioural model of the system by enabling or disabling system features depending on metrics and utilities.

We see from these studies that system adaptation can be provided through modelling both system's behaviour and architecture explicitly. A formal model representing context information must be defined to facilitate a shared understanding of system context information and their possible changes to be discovered and uniformly understood between all system components.

2.2.2 Context-aware Systems

Context-aware computing was introduced for the first time by [75] as the ability of a mobile user's application, which is constantly monitoring information about the surrounding environment, to discover and react to changes in this environment.

Dey et al. [2] considered a system is context-aware if it uses context to provide relevant information and/or services to the user, where relevancy depends on the user's task.

ContextServ [81], is a platform for development of context-aware Web services based on ContextUML [80], which is a modelling language for the model-driven development of context-aware web services based on the Unified Modelling Language (UML). ContextUML copes with both context modelling and context-awareness modelling. In context model, the low-level context is defined as *AtomicContext* which represents simple information acquired from context provider meanwhile the high-level context is referred as *CompositeContext* that comprises either atomic or composite contexts. Context is provided by two source types ContextService and ContextServiceCommunity. ContextService encapsulates sensors details and derives context information by interpreting and transforming the sensed information. ContextServiceCommunity plays the role in selection of the most appropriate context service which will provide the requested context information.

CONON [91] is an OWL-based ontology which provides a formal context model and implements a Description Logic (DL) reasoning. A reasoning rules were used to reason over a low-level (explicit) context to derive a high-level (implicit) context based on the proposed ontology and by means of DL and Resource Description Framework (RDF) reasoners.

In [54], the authors use the Web Ontology Language (OWL) ontology and the Semantic Web Rule Language (SWRL) rules to model context in a context-aware system using Rule-Based Inference engine.

Ejigu et al. [23] similarly proposed Context Management Model (GCOMM) to provide reasoning and decision making in context-aware system. They used an ontology-based context model and defined rules on given data instances.

An OWL-based device ontology was provided by Bandara et al. [6] to describe devices and their hardware and software components. However, the proposed ontology lacks for full service descriptions as it only provides an initial representation for device's services using a relationship called *hasService* without providing a description of the service's attributes. On the other hand, in [11], an ontology has been used to provide a service's description and preferences and to allow match-making techniques on these descriptions.

Context-awareness modelling is presented by two mechanisms, context binding and context triggering. The first one concerns with mapping between the input of service operator and context sources automatically. The second one represents the contextual

adaptation of the services according to defined context constraints and a set of actions. The context binding mechanisms enable more possibilities for automatic execution of service.

SOCAM is a service-oriented context-aware middleware architecture for building context-aware services [32]. It provides a formal context model based on ontology using OWL for representing context semantics and context reasoning. The context-awareness lies in services ability to adapt regarding to context changes depending on predefined rules which specify the adaptation plan. Low level context is captured directly from sensors or from user's data which the particular person defines using the user interface of the application. High-level context can be derived using context reasoner indirectly by interpreting new context from low level contexts. The authors addressed the dependency relationship between context using a sematic description of datatypes and objects properties. Moreover, they propose the ability to track and adapt the dynamic changes in context providers (i.e. adding or removing physical sensors) and their context information through the locating service. This service enables context providers to advertise their new context forms by deploying a multiple matching mechanism between application's query and context providers.

2.2.3 Context-aware Adaptive Software Systems

As the context-awareness contribution is proposed to model, process, and manage context information, self-adaptation approach focuses on the ability of a system to adapt its structure, goals, mechanisms regarding changes in the operating environments. A novel approach was proposed by [39] to apply self-adaptive and context awareness together in software systems. In this survey, the requirements of integrating of these two approaches together have been identified as follow:

- The context modelling requirements need to be considered from the system-context relationship perspective.
- Self-adaptivity needs to have a system that can cope with the context/requirements changes (both anticipated and unanticipated), and then the system needs to be designed with adaptation in mind.
- The requirements for the mechanism that integrates the context-awareness and self-adaptivity needs to be considered.

An abstract architecture has been proposed in a later work [40] and it consists of three layers:

- *The functional system and its context layer* comprise 1) the *functional system element* which presents system functionality through the running components and inactive ones. 2) the *context element* that manipulates the system operation and/or adaptation 3)the*interfaces* with the management layerwhich role is divided to find out the changes in the system or in its context and to apply the decided adaptation plans of the management layers on system functionalities.
- *The system and its context representation layer* provide up-to-date *context model of the* environmental context and a *system model* of its running state. Moreover, it provides the implementation of the operation for applying the actions of the change management layer.

- *The change management layer* checks any possible system consistency violation, derives the high-level context information and decides the suitable adaptation plans regarding to the context and/or requirements changes.

In this approach, system operation will adapt for both changes of *system model* and*context model*, which could cause extra processing cost especially for modelling and deriving context information and applying adaptation actions on the system.

2.2.4 Decentralized Self-Adaptive System

Self-adaptation approaches mainly presented adaptation as centralized or hierarchically controlled systems. A new contribution [93] addressed self-adaptation in decentralized managed software architecture. They divided the computational requirements for decentralized controlled system into four groups: 1) coordinated monitoring through sharing locally collected data of the partial system and its synchronization globally in the system, as the monitoring process is managed locally and each partial system has only access to his own knowledge; 2) coordinated analysis to provide a full analysis of each subsystems data to provide full knowledge analysis of system data; 3) coordinated planning between different planning units which could have different private goals that need to be reformed in one adaptation plan avoiding any possible confliction; 4) coordinated execution needed to synchronize and to manage execution plans of each partial system.

In [93], authors presented a case study of decentralized self-healing for camera failures in traffic monitoring system. In this system, a camera analyses the partial collected data to monitor and detect any possible traffic jam. Cameras will collaborate to monitor the traffic jam located in their viewing range by organization of different nodes. This organization of the same viewing range cameras can join other neighbour organization if traffic jam grows. Organization dynamics are managed by organization controllers, each controller centrally controls its organization and one of these controllers is selected as a master by the other organization controllers which will be as slaves.

Self-healing subsystem is provided to recover camera failures using a self-healing manager component which analyses the monitored data about the status of the cameras. This manager executes a recovery strategy to ensure the consistency of the main system when a camera failure or loss is detected. This approach presented a framework approach of decentralized self-adaptive subsystems which can avoid the bottleneck in processing of monitored data of participated devices or recovering process in each subsystem. However, the authors did not address the possibility of using other types of devices that can be participated in the proposed organization, which could provide more possible adaptation plans and requires modelling of the acquired data of different devices.

2.2.5 Discussion

In this section we provide our evaluation of current approaches demonstrated in the previous sections. We identify the drawbacks that should be addressed to support self-adaptation in software system. These limitations are listed as follows:

L1 Limitation in system extensibility, as in [82, 93] that provides an adaptive design pattern with a pre-defined set of tasks that can be considered in the design of adaptation plan.

L2 Limitation in context information modelling, as in [85, 82, 81] addressed the adaptation but with a limited context model representation. However, context modelling must be guaranteed to allow common understating of context information between several domains and to support system extensibility. Moreover, context modelling is essential to implement context-awareness and adaptation processes in the system.

L3 Limitation in adaptation strategies, as in [81] that requires a pre-defined adaptation strategy and limits the proposed approach for limited context-aware adaptation scenarios.

L4 Limitation in decision making during adaptation process, as in [85, 82, 93, 24] due to the limitation in the system context model representation or due to the consideration of a goal-oriented approach during system design. The decision-making process should be extensible to adopt new terms of system context model that can be used in making decision to support adaptation process.

Based on our evaluation of current approaches, we see that a context-aware self-adaptive SOA model is required to overcome the existing limitations of current approaches. It is required to provide a generic adaptation model that can promote service reusability and system extensibility in software system through the implementation the SOA principles during the design of system architecture. Moreover, it is required to provide a formal context representation of system components to support context awareness and monitoring of context changes in the system. Finally, we it is required to provide a dynamic decision-making process to support the utilisation of different adaptation strategies and to select the best adaptation plan based on metrics defined in the system context model.

2.3 Problem Statement

In this section, we present the statement of the problem caused by the limitations of current approaches. Consider a mobile navigation application (System) providing location and traffic information to drivers. The mobile application uses a Traffic Information Web service (Service) that provides the required location, map and traffic information. The application uses driver (Context) information in planning the best route. Considering driver plan to stop by a supermarket to do shopping.

The system applies Context-awareness approach to cope with driver's plan by suggesting a route with one stop at the supermarket close to driver house. If the location service is down or unreachable (see Figure 2.1), the system (Controller) adapts to this change in the (Environment) Context (i.e., missing location and traffic information of the Traffic Information Web service) by searching and acquiring the required information from the traffic services provided by neighbouring cars or road's traffic information points. As alternative, the controller can perform a (Migration) of its Traffic Detection Service to another car so that it can use the Traffic Information Service available on that car and perform the computation needed to get the missed traffic information and send it back to the original car.

To achieve this system, it is required to provide a framework that supports system extensibility by enabling the automatic discovery of neighboring cars or road's traffic information points so that they can be used in the adaptation process as new source of traffic information service. The framework should enable system developers to describe the context information of system components in a way that guarantees a correct understanding and

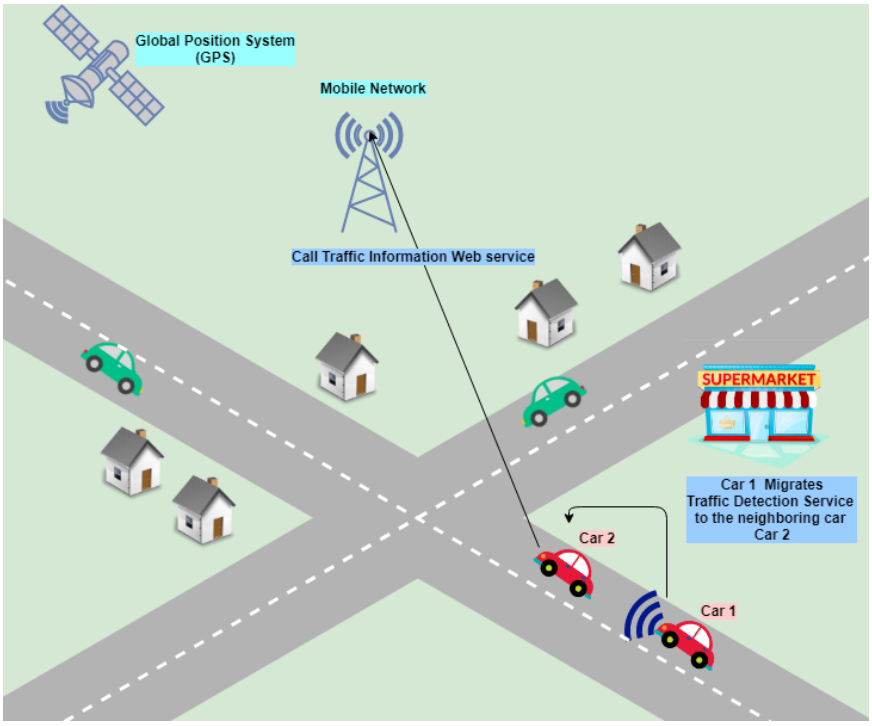

Figure 2.1: Traffic Detection Service as adaptation to traffic information service loss in car mobile navigation application.

utilization of these information between all system components. Moreover, the framework should support a decision-making process that allows to decide which available service to use or to which car to migrate the Traffic Detection Service.

2.4 Summary

In our approach, we use context-awareness in decentralized self-adaptive systems. On one hand, we choose to use the ontology-based approach for context-modelling not only because it describes a system semantically with a proper definition of the relationships between its components, but also regarding to its capability to reason with the Semantic Web. For example, Ontology Based Language (OWL) uses DL reasoner to derive new contextual information about system component which will be used in making the adaptation decision. On the other hand, we think that using decentralized controlled adaptive units can improve system adaptation performance by distributing adaptation efforts over several orchestrations adopting the decentralized control approach overviewed in Section 2.2.4. Also, the utilization of different types of devices should be supported during the self-adaptation process so that new adaptation plans can be performed in the system. For example, an adaptation plan can suggest the utilization of a mobile device's camera during a surveillance camera failure. Moreover, this variation of device types can enable the system to avoid possible device-type-specific failures regarding some special environmental circumstances (see Objective O3). We think that self-adaptation can be applied in each subsystem to avoid such device failure or change in its context information by executing a recovery plan to replace the device with another one that can provide similar context information.

Part II

Proposed Approach

Chapter 3

Context-aware Self-Adaptive SOA Meta-Model

In this chapter, we introduce our meta-model proposed to describe SOA-based system and enable the utilization of context-awareness and self-adaptation. The meta-model is presented in our ontology-based component model schema. The proposed component model allows to describe system components including their properties and preferences. It allows system architects to describe general system operational conditions that should be guaranteed during the run-time. Moreover, it supports the definition of system adaptive behaviour to context changes through a planned self-adaptation strategy.

3.1 Introduction

In cloud computing, web-services are plenty available with their various functionalities. The need for investing their power becomes highly requested to connect these services in some compositions which perform a business process according the user needs instead of spending more time on redeveloping new software. This is addressed by a new approach of service composition with semantic descriptions that provide helpful notation for developers. The developers can reuse these services to create new composite services which reduces the development time and costs.

With this ability, services requesters and providers are communicating to perform user's needs. The user sends his request with the required data to be processed and waits until the provider finishes processing the data and sends the results back to the user. Such process requires a shared definition of the meaning of exchanged data so that it can be understood and processed correctly by the users, service requestors and service providers.

3.2 Semantic Web

The semantic web is a framework of standards provided by the World Wide Web Consortium (W3C) that makes an extension of the standard World Wide Web [9, 57]. It provides a common machine-readable data document over the Web which promotes to read, share, and reuse data in independent way from applications and websites which leads to achieve the Web of data [78].

Semantic Web is an XML-based to describe data to provide the mediator language that serves to understand the interchanged data between different applications and domains.

The Semantic Web provides a way to link between data based on the similarities of its contextual semantic meanings. It also defines relationships between data and increases the understanding and knowledge domain about them.

The Semantic Web Stack shown in Figure 3.1, presents the architecture of Semantic Web, more specifically the components of technologies and formats that Semantic Web is built on [9].

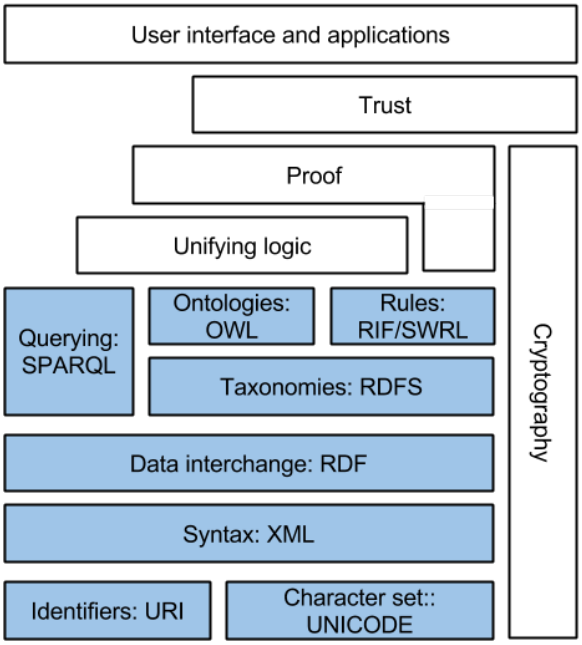

Figure 3.1: The Semantic Web Stack.

Resource Description Framework (RDF)

It is a formal model provided by W3C as a standardized method to describe information used in Web resources. It is a data model that uses a contextual metadata of syntax notations and serialization formats. An RDF-based modelled data is presented in a format of *subject-predicate-object* triples describing Web resources as *subject* or *object* in addition to the relationship between them as a *predicate* [53, 13].

Resource Description Framework Schema (RDFS)

RDFS is a general-purpose language consisting of RDF-defined classes and properties used to provide the vocabularies required to define ontologies of Web resources among different domains [53, 13].

3.3 OWL-S Semantic Web Service Description

OWL-S is an OWL-based Web service ontology that provides a semantic description of web services in the way that facilitates the automation of Web service tasks (i.e., Web service discovery, execution, composition and interoperation) [59]. This ontology describes the service through three main questions: „what the service does“, „how the service works“, and „how to access the service“. Each answer of these questions presents a property of the service, so the properties are „*presents*“, „*describedBy*“, and „supports“.

Finally, the service description will consist of three main components: „*ServiceProfile*“, „*Service Model*“, and „*Service Grounding*“. Service profile provides a helpful information that searching agents needs to find the best service for each request. The service model describes the process of executing the service. Service grounding supports the clients with the needed information about how to access this service regarding of communication protocols and messaging formats [76].

While the OWL-S is a SOAP-based Web Service model, we realize that the need to provide a generic service description profile is demanding to enable the adaptation functionality in SOA-based system, so that it must support the adaptation with different architectural styles of Web services.

3.4 System Core Ontology

Many ontology languages and models, such as OWL, SWRL, and RDF were developed to provide formal semantic models for Web resources and to apply rules on these semantics in order to derive new meaningful data. Ontology terms must be understood and shared between system components, which removes any ambiguity of the used terminology during system design and enables a correct utilization of exchanged messages and information between system components.

In this section, we introduce the OWL-based core ontology that will be used to describe service-oriented architecture. The core ontology is a set of terms of architecture components and relations between them. It basically describes SOA-based system as a set of Services and Service Providers. Presented in Figure 3.2, the core ontology consists of the following classes. The *Service* class, defining *Service* components in SOA, has a relationship with the *ServiceProvider* class called *providedBy*. A *ServiceProvider*, is identified by *hostname* and *protocol* properties. The relationship between *ServiceProvider* and its *Service* is called *provides*.

During the development of the ontology, we consider the possibility to extend other SOA standard ontologies such as the SOA Open Group (OG) ontology [31] by our ontology to support them with our proposed adaptation approach. To facilitate the integration of our ontology in any OG based system, we consider both *Service* and *ServiceProvider* classes as sub classes of *Element* class as defined in the OG ontology. While the *use* property is defined to express an interaction between elements of the OG ontology, we choose to specify the interaction relationship between *Service* and *ServiceProvider* with *provides* to define the type of relationship between a *Service* and *ServiceProvider* and avoid any inconsistency and/or incoherence between the considered ontologies.

An object property *hasProperty* is defined in the ontology to describe properties of system components. A component's property object is defined as an instance of ontology class *Property*.

The *Property* class has the following attributes

- *propertyName:* is the name of the *Property*.
- *propertyType*: is the data type of the *Property*.
- *propertyValue:* is the value of the *Property*.
- *criteria:* it states the *CriteriaProperty* that affects the *Property* instance during the decision-making process.

The *CriteriaProperty* has the following attributes:

- *name*: is the name of criterion that governs the property of system component.
- owner: it has a value of "origin", "destination", and "service", and limits possible owners of the properties which are referring to a particular criterion (for example, a criterion with the owner value set to "service" can be referred only from service context models' properties, i.e., it can be applied only on services, not on service providers).
- *valueWithHighestWeight:* indicates which values of properties referring a particular criterion are considered to be the most important during the decision-making process.
- *valueWithLowestWeight:* indicates which values of properties referring a particular criterion are considered to be the least important during the decision-making process.
- *criteriaPriority:* indicates a general importance of a particular criterion by integer values between 1 and 10. The *criteriaPriority* property of a criterion determines the priority of a preference rule affected by the criterion.

Using the semantics of proposed ontology, we can define context models for *Service*-s and *ServiceProvider*-s stating their operation preferences. A detailed description of component model is provided in the following section.

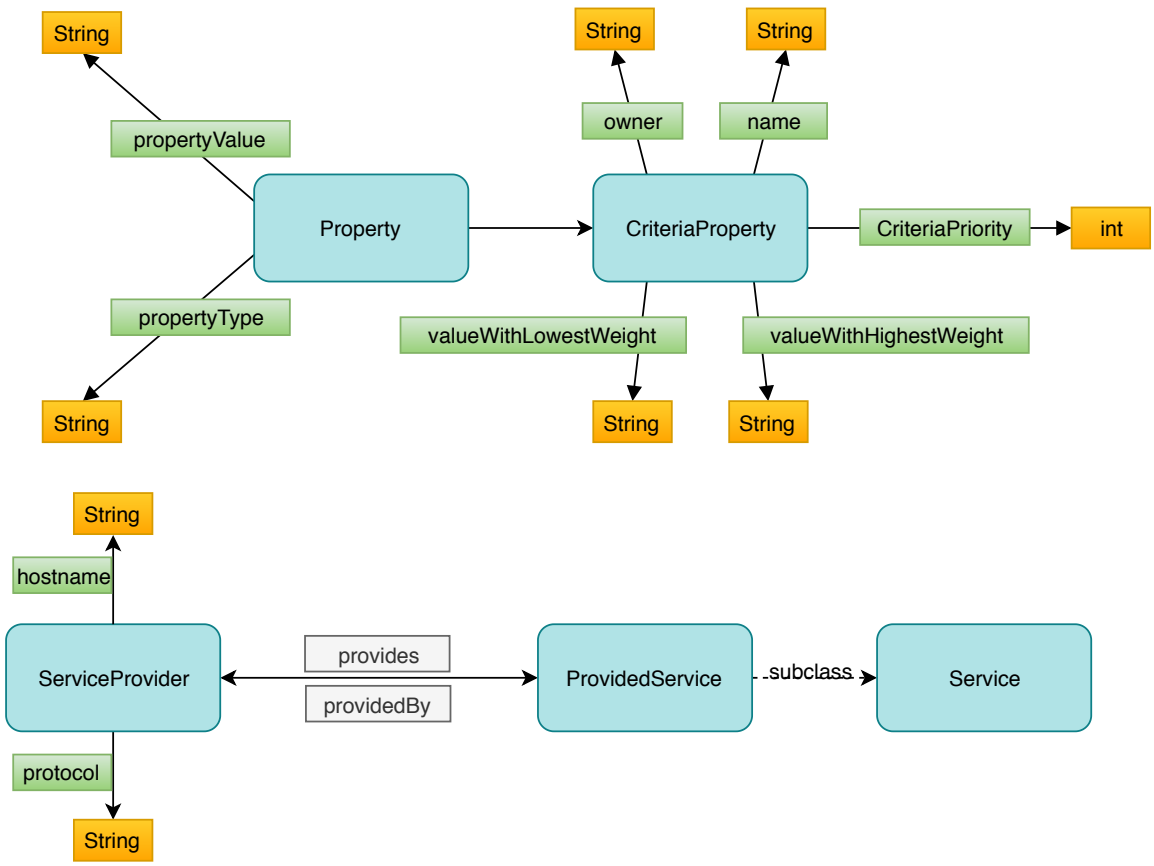

Figure 3.2: System Core Ontology proposed to describe SOA component.

3.5 Ontology-based Context Model

In this section we demonstrate our context model defined based on the core ontology introduced in the previous section. Using the semantic terms proposed in the core ontology, the context model defines SOA components and provides a semantic description of the joined providers and hosted Web services by stating their specifications and properties. The context model describes the preferences and conditions that control system's components through semantic-based rules. The full system context model is created by aggregating system components models in one model. A higher-level context is inferred from the final composed system context model through a reasoning process so that new context information can be derived and used to make adaptation decisions.

The importance of utilizing a dynamically generated model of SOA service providers becomes highly demanding in order to identify and perform the proper reactions to changes in service providers resources. On the other hand, the fixed pre-defined service model can be enriched to contain relevant information about system preferred resources that possibly affect service performance.

```
{ „name“:„defines the name of system component“,
„type“:„contains the type of system component. i.e., “Service„ or “ServiceProvider„„
“noPreferenceRules„:“ „true“ if there is rules element in the model, „false“ if there is not„„
“properties„: {“propertyName„:“states the name of the property„„
“propertyValue„:“states the value of the property„„
“propertyType„:“states the data type of the property, for example, „INT“„„
“criteria„:“ServicePriorityCriterion„},
“rules„:“[ RDFS rule: ...]„
}
```

Figure 3.3: A simplified schema of the system component.

In SOA, SOAP Web service can be accessed through its URI and WSDL file which describes a Service as a set of network endpoints. Further standard description of a service is provided in its WSDL file. This description contains an abstract definition of the exchanged data called Message meanwhile the defined service actions are described through Operation tag. Similarly, a document called the Web Application Description Language (WADL) file is provided to contain description of RESTful service stating its resources and operations.

In our proposed system we utilize RESTful services and consider to integrate service context model in WADL file so that it can be retrieved through its URI.

A *Service* model holds the context information designed to be used in planning of the adaptation process. Similarly, a *ServiceProvider* model contains context information stating its properties, work preferences and conditions that specify the possible hosted services. In Figure 3.3, we present this proposed component context model which consists of the following elements:

- *name*: is a JSON element stating the name of a Service to be identified in the system.
- *type*: is a JSON elements stating the type of related service.
- *properties*: is an array of JSON Objects, describing component's object properties.
- *rules:* is an JSON array of RDFS-based rules. Each rule describes a component's operation preference.

- *noPreferenceRules:* is a data property with a value of *'false'* if the service has preferences and '*true'* if it has not.

Chapter 4

The Decision-Making Process

The process of enabling software system to make an adaptation must contain a sub-process of making a decision. In Section 2.2.5, we discussed the need to support the adaptive systems with a decision-making mechanism that allow to integrate and use the terminology (i.e., the ontology used to describe system components context models) in the decision making process. Such a mechanism is needed to make the decision that satisfies system's and components' rules and requirements. Usually there will be a set of possible decisions to choose from and several factors affecting the decision-making process. These factors must be considered and optimized in order to make more reliable and optimal decisions.

When addressing a decision-making process, the main problems to solve are: how to take that most suitable decision, on what bases to make this judgment, and how to include factors that affect the decision.

Considering specific adaptation scenario, there will be different adaptation possibilities. And to provide an adaptation strategy, it is necessary to know how and what adaptation/s to choose to perform these possibilities. To solve this problem, first, we need to classify each adaptation according its components so that adaptations with similar set of components will be grouped and subjected to a particular decision-making process. Second, in order to select the most suitable adaptation from several alternatives, each adaptation should be prioritized based on the properties of its components and their values.

There are many existing mathematical methods which have been implemented in computer science to enable decision making processes. Some of these methods depend on an analysis process that ranks the decisions using a defined set of quantitative and/or qualitative criteria through a series of pairwise comparisons of these decisions. This type of decision-making process is called multiple-criteria decision-making (MCDM) or multiple-criteria decision analysis (MCDA) method.

To make an adaptation decision from between a set of adaptation scenarios, we choose to implement the Analytic Hierarchy Process (AHP, [71]) as the decision-making method. The AHP method has been very widely used in computer science researches as the best MCDA method because of its simplicity, flexibility [36] and accuracy [3]. It uses a multi-criterion prioritizing mechanism to be used in analyzing, evaluating and prioritizing the alternative decisions and selects the one with the highest priority.

While some other MCDA approaches such as the Analytic Network Process (ANP) [72] considers the criteria to be interoperated so that the relational weights between them will be considered in the decision-making process. We consider the criteria to be independent from each other in the decision problem for the following reasons 1) to reduce the required effort to define a consistent criteria relation matrix during design time of the decision making

problem, 2) to avoid extra processing time when considering the additional ANP weighting factors (i.e., criteria relations) during the decision making process 3) and to avoid complexity while presenting the steps of our adaptation approach the decision making process.

The AHP ranks all alternative decisions by calculating the composite weight for each possible adaptation while considering a specific adaptation strategy in the system. It enables a prioritized consideration of the properties of system components (*Service*s and *Service-Provider*s) involved in the adaptation process. The AHP prioritizes each adaptation and finally chooses the adaptation with the highest weight and priority from other suggested adaptations.

4.1 Decision Making Using the AHP

The AHP starts with creating the comparison criteria matrix A. A is $m \times m$ matrix where m is the number of considered criteria. a_{ij} is the importance of the i^{th} criterion over the j^{th} one. a_{ij} entry is set to 1 in the case of the diagonal entries and also if the i^{th} criterion has the same importance of the j^{th} one. On the other hand, the value of a_{ij} ranges over 3, 5, 7, 9, 1/3, 1/5, 1/7, or 1/9. An entry a_{ij} has the value of 3, 5, 7, or 9 when the importance of the i^{th} criterion has a moderate important, more important, strongly more important, or absolutely more important, than importance of the j^{th} criterion respectively. On the contrary, a_{ij} has the value of 1/3, 1/5, 1/7, or 1/9 when the importance of the j^{th} criterion has a moderate important, more important, strongly more important, or absolutely more important, than the importance of the i^{th} criterion respectively.

Usually, a_{ij} entries are set through user judgment. To automate this process, we specified a *CriteriaPriority* property for each criterion to have a value of the interval [0, 10]. These values are set by system administrator in the core model and used to prioritize the criteria over each other's.

A *CriteriaPriority* instance will be considered in the decision-making process through the AHP method only if it was considered as a Property's criterion in one of components involved in the adaptation scenario.

4.2 Dynamic Decision-Making Algorithms

In order to make an adaptation decision using the AHP, we developed the following algorithms to generate the needed matrices for AHP computations:

InitializeCriteriaMatrix Algorithm it is required to initialize the main criteria priority matrix A. The *InitializeCriteriaMatrix* algorithm, (see Figure 4.1), calculates a_{ij} entry by comparing the *CriteriaPriority* values of the i^{th} criterion and j^{th} and mapping the difference into one value of set $\{1, 3, 5, 7, 9, 1/3, 1/5, 1/7, 1/9\}$.

After intializing A, the AHP computes the weight vector $W_{(m\times 1)}$ of matrix $A_{(m\times m)}$ by normalizing its entries to make the sum of each column entries equals to 1 through Equation 4.1.

$$\bar{a}_{ij} = \frac{a_{ij}}{\sum_{k=1}^{m} a_{kj}} \tag{4.1}$$

Then it computes the priority vector or the normalized principal Eigen vector of criteria by computing the average value of each row of the normalized matrix through Equation 4.2

Require: $\langle p_1, p_2, \ldots, p_m \rangle$ as values of *CriteriaPriority* of criteria $\langle c_1, c_2, \ldots, c_m \rangle$
Ensure: A is a pair-wise criteria comparison matrix for given criteria $\langle c_1, c_2, \ldots, c_m \rangle$

for $i \leftarrow 1$ **to** m **do**
  $a_{ii} \leftarrow 1$
  for $j \leftarrow i{+}1$ **to** m **do**
    $difference \leftarrow |p_i - p_j|$
    if $difference \geq 8$ **then**
      $judgment \leftarrow 9$
    else
      $judgment \leftarrow 2\lfloor \frac{difference}{2} \rfloor + 1$
    end if
    if $p_i > p_j$ **then**
      $a_{ij} \leftarrow judgment$
    else
      $a_{ij} \leftarrow judgment^{-1}$
    end if
    $a_{ji} \leftarrow a_{ij}^{-1}$
  end for
end for
return A

Figure 4.1: The *InitializeCriteriaMatrix* algorithm to compute a pair-wise criteria comparison matrix for AHP based on *CriteriaPriorities* of individual criteria.

$$w_i = \frac{\sum_{k=1}^{m} \overline{a}_{kj}}{m} \tag{4.2}$$

InitializeDecisionMatrices Algorithm The algorithm, (see Figure 4.2), is developed to be automate the creation of the required AHP decisions comparison matrices. The algorithm is executed for each considered criterion c_k, where $k = 1, \ldots, m$, of criteria set $C = \{c_1, c_2, \ldots, c_m\}$ to generate matrix $V_{n \times m} = [V^{(1)}, V^{(2)}, \ldots, V^{(m)}]$, where n is the number of possible adaptation decisions found before.

In the matrix V, each $V^{(k)}$ is a transpose of the weight vector of matrix $S^{(k)}$ obtained by an individual execution of the *InitializeDecisionMatrices* algorithm. Each $s_{ij}^{(k)}$ entry represents a judgment value between the i^{th} adaptation and the j^{th} one based on the criterion k.

The algorithm use *valueWithHighestWeight* and *valueWithLowestWeight* criterion attributes to map the judgment into one of values of set $\{1, 3, 5, 7, 9\}$ or their reciprocals, which are accepted by AHP.

The judgment s_{ij} is considered to have an inverse proportionality relationship with the difference between the related criterion values s_{ki} and s_{kj} of the considered adaptations i and j respectively, when the difference between the valueWithHighestWeight and the valueWithLowestWeight of the criterion k is less than 0.

By applying *InitializeDecisionMatrices* algorithm on all considered criteria we will have m weight vectors of the possible adaptations, each vector of them is related to one criterion of the m considered criteria. Finally, AHP computes the composite weight of the adaptation decision through Equation 4.3.

n	1	2	3	4	5	6	7	8	9	10
RI	0	0	0.58	0.9	1.12	1.24	1.32	1.41	1.45	1.49

Table 4.1: Random Consistency Index (RI).

$$P_{n \times 1} = V_{n \times m} \cdot W_{m \times 1} \tag{4.3}$$

where V is the $n \times m$ matrix obtained by multiple executions of the *InitializeDecisionMatrices* algorithm as described before (one execution for each criterion) and W is the weight vector of criteria from Equation 4.4.

$$P_{n \times 1} = \begin{pmatrix} p^1 \\ p^1 \\ . \\ . \\ p^n \end{pmatrix} \tag{4.4}$$

The entries of vector P represent the composite weights of the adaptations respectively. By result the adaptation with the highest composite weight entry is selected to be executed from the n alternate adaptation decisions based on the calculated adaptation weight vectors and the criteria weight vector.

The AHP provides a Consistency Index (CI) to measure the consistency of judgment in each matrix generated by *InitializeCriteriaMatrix* and *InitializeDecisionMatrices*. The CI is calculated by the application of the following formula

$$CI = \frac{\lambda_{max} - n}{n - 1} \tag{4.5}$$

Where λ_{max} is the highest eigen value of the principal Eigen vector of a given matrix.

To measure the consistency of the judgments of each generated matrix, AHP uses a Consistency Ration (CR) calculated by comparing the CI with a Random Consistency Index, as defined in the following formula

$$CR = \frac{CI}{RI} \tag{4.6}$$

Where the appropriate RI values are provided in Table 4.1. The AHP accepts a matrix to be consistent if its CR is smaller or equal to 10%.

Require: criterion k and its attributes $valueWithHighestWeight^{(k)}$ and $valueWithLowestWeight^{(k)}$; $\langle p_1, p_2, \ldots, p_n \rangle$ as values of services or service providers properties that consider k as their criterion

Ensure: $n \times n$ matrix $S^{(k)}$ as a decision comparison matrix based on criterion k

1: **for** $i \leftarrow 1$ **to** n **do**
2: $s_{ii} \leftarrow 1$
3: **for** $j \leftarrow i + 1$ **to** n **do**
4: $range \leftarrow valueWithHighestWeight^{(k)} - valueWithLowestWeight^{(k)}$
5: $diffValue \leftarrow p_i - p_j$
6: $fifthOfDiff \leftarrow range/5$
7: **if** $diffValue > 4 * fifthOfDiff$ **then**
8: $s_{ij} \leftarrow 9$
9: **else if** $diffValue \leq 4 * fifthOfDiff \wedge diffValue > 3 * fifthOfDiff$ **then**
10: $s_{ij} \leftarrow 7$
11: **else if** $diffValue \leq 3 * fifthOfDiff \wedge diffValue > 2 * fifthOfDiff$ **then**
12: $s_{ij} \leftarrow 5$
13: **else if** $diffValue \leq 2 * fifthOfDiff \wedge diffValue > fifthOfDiff$ **then**
14: $s_{ij} \leftarrow 3$
15: **else if** $diffValue \leq fifthOfDiff$ **then** $s_{ij} \leftarrow 1$
16: **end if**
17: **if** $range * diffValue < 0$ **then**
18: $s_{ij} \leftarrow s_{ij}^{-1}$
19: **else if** $range * diffValue = 0$ **then**
20: $s_{ij} \leftarrow 1$
21: **end if**
22: $s_{ji} \leftarrow s_{ij}^{-1}$
23: **end for**
24: **end for**
25: **return** S

Figure 4.2: The *InitializeDecisionMatrices* algorithm to compute a decision comparison matrix based on a given criterion.

Chapter 5

Web Service Migration-based Adaptive Service Oriented Architecture Model

To demonstrate the power of our ontology-based SOA component context model, we propose the utilization of Web service migration approach as an adaptation strategy in SOA. The migration of a particular service should be considered if its hosting service provider is not able to guarantee the preference or quality of the service and there is no alternative service or service composition that can be used as a substitute for the original service, i.e., it can provide the same functionality and required quality. In the next section we provide a supplementary study demonstrating specific research work on supporting information system with migration adaptation.

5.1 Related Work

Service mobility have been proposed as a very promising approach to leverage the interoperability and reusability characteristics of SOA.

Lange et al. [52] described an implementation of mobile agents in Java by the Aglets framework. The framework allows reusing system components, i.e., aglets, in different contexts, however, without any utilization in making migration decisions.

Hao et al. [33] developed a Web service migration environment and used a genetic algorithm to find the optimal or near-optimal migration decisions. The algorithm calculates the cost of a total round-trip including dependency calls for each service request and it is used to decide migration according to this cost. However, the authors did not take into account user-defined conditions affecting the migration decision, e.g., specific requirements on a migrated service or a destination provider.

Zheng and Wu [98] presented an infrastructure for runtime migration of services in a cloud which consists of five components with different roles and specific criteria to control the migration decision. One of the components collects CPU load data from all known hosts. Then, when the CPU load of a particular host reaches a predefined threshold, a flag is set to indicate that service migration is needed on this host. The approach does not check compatibility of services and providers during migration and does not address the possibility of running several services on a single provider at the same time.

Schmidt et al. [77] implemented a prototype of an adaptive Web service migration with two types of migration possibilities, namely context-based migration and functionality-based migration. In the context-based migration, services are migrated to the providers which meet the services' requirements, while in the functionality-based migration, the services are migrated according to their roles in a workflow (i.e., to optimize their communication in the workflow). Both migration possibilities can be implemented also in our approach by an appropriate migration decision strategy.

Messig et al. [61] proposed a service migration facility in Service Oriented Grid environment which enables taking migration decision based on matching providers' and services' requirements. In this approach, services are hosted by service providers including the resources needed for execution of the services' operations. The authors made several experiments of service migration between two geographically sparse grids where the first grid had high-performance devices and faster network than the second one. While these experiments demonstrated the process of service migration, they are not suitable for the demonstration of migration decisions (e.g., selection of a migration destination) which should be discussed in more detail.

Multi-agent system approaches, such as JADE [8], Mobile-C [15], and AgentScape [94], provide middle-wares to host and migrate mobile agents in a distributed system. A mobile agent is an autonomous composition of code, state, and data that can be transported from one environment to perform agent's tasks. MobiGo [70], which is another middleware system for seamless mobility, provides a mechanism to migrate services according to user's needs using a simple service description. The description contains information about service name, service type, and I/O devices which can be used to run that service. User can select the desired service from a list of available services on that particular device. Service description is considered the main backbone supporting the reusability and interoperability of services.

In [97], a service migration approach was proposed as a solution to maintain continuous service availability with migration decisions based on QoS goals.

In [27], authors introduced a virtual machines migration framework with several migration decision-making strategies for different resource reservations goals and schemes during migration of virtual machines, namely: sequential migration, parallel migration and workload-aware migration strategies.

In [27], authors proposed an algorithm for dynamic placement of services to servers based on available server resources (such as CPU or memory) and network performance given by SLAs.

In [34], authors described a framework for service migration in cloud computing environments using a genetic algorithm to search and select possible migrations. The algorithm utilized a cost model with various service migration costs, including the costs of consistency maintenance and communication during migrations, a service table with information of all migrated and replicated services, and a general computing platform registry with information about the hosted services. Another approach to support the selection of migratable services in the cloud was provided in [84]. The selection process considered pre-defined criteria related to QoS of the migratable services in the cloud, namely: response time, throughput, availability, reliability, and cost. The selection process utilized the AHP method with comparison matrices defined by a consumer's judgments on the QoS criteria. Although the AHP method is utilized also in our approach, the comparison matrices are, in our case, defined by the ontology of dynamic properties and preferences of automatically

discovered providers and their services, which supports a dynamic multi-criteria migration decision making process.

In [68], a decentralized migration approach was introduced based on monitoring of health of Web Services Resource Framework (WSRF) containers [7]. The approach was trying to minimize possible threats of service level agreements (SLA) violations to preserve QoS of provided services by their migration. A service priority was proposed to be used as a weighting factor in migration decisions in addition to a health metric of each WSRF container. However, this approach was limited by the static health metric of WSRF containers based on their available memory and CPU performance factors, while, in our work, we are addressing this limitation by enabling a dynamic definition of the service and provider properties and preferences together with the criteria so that the newly defined weighting criteria and related properties can be automatically considered in the migration decision making process.

Similarly, many frameworks have been proposed for Mobile-hosted provisioning on mobile devices.

In [90], the author presented Android based framework for hosting mobile services using REST web services to enable Web service provisioning. The framework utilizes a fuzzy controller to monitor the context of joining devices, analyse the context and decide which hosted service to be provided.

A mobile-hosted mobile Web service migration framework is proposed in [50]. The framework utilizes SOAP engine to analyse SOAP messages and execute the corresponding service. They utilized a migration policy [49] in the migration decision making process. However, they did not consider the preferences of the connected devices in the migration decision process.

AlShahwan, et al. [4], provided SOAP- and RESTful-based frameworks for distributed execution of mobile Web services. Based on the performed tests on both frameworks, the authors found that the REST framework has better performance than the SOAP one in relation to hosting Web services on mobile devices.

5.2 Service Migration

The *Service Migration* (for definitions see Section 1.2) starts when a particular *Service* is selected to be migrated to a particular destination *Service Provider* by a migration *Controller* of the migration framework running on each *Service Provider*. An *Orchestration Controller* can perform *Service Migration* by packaging the service and deploying it on the destination service provider.

During the migration process, the migrating service is stopped, and its internal state is stored and sent to the destination provider. All further incoming calls of the service are postponed until the migration is completed, i.e., until the migrated service is initiated in the new location, its internal state is restored, and until the service is able to handle incoming messages.

5.3 Migration Decision Modelling

The migration decision can be described by the Linear Time Logic (LTL) [58, 99] as a sequence of states which are related to time. In this section we are providing a formal description of the service migration process demonstrating its stages and their related pre

and post conditions. LTL formulae are combinations of terms using logical operators $\wedge$ and $\rightarrow$ and temporal operators $\Box$, $\diamond$, and $\circ$. Formulae $\Box p$ and $\diamond p$ means that p always or sometimes holds in the future, respectively, and $\circ p$ means that p is true in the next state. Let $P = \{P_1, P_2, \ldots, P_m\}$ is a set of existing providers and $S = \{S_1, S_2, \ldots, S_n\}$ is a set of the migratable services. To demonstrate the migration adaptation model, we provide the following definitions.

Definition 1: ***FindOriginProvider*** is the process to find the most critical provider in the system that needs to migrate its services regarding its current state and its migration conditions.

Definition 2: ***FindMigratedService*** is the process to find the most appropriate migratable service to be migrated from the *FindOriginProvider*.

Definition 3: ***FindDestinationProvider*** is the process to find the best destination provider for the service selected during the *FindMigratedService* process.

Definition 4: **Decision Making Process (D)** is the process of making a decision to select the best service migration to be performed. The migration decision process is described as a sequence of the three defined processes (*FindOriginProvider*, *FindMigrated-Service*, and *FindDestinationProvider*)

$$\begin{aligned} D \equiv\ & (D \uparrow \wedge FindOriginProvider \uparrow \wedge T_a) \\ & \wedge \circ (FindOriginProvider \downarrow \wedge FindMigratedService \uparrow \wedge T_b) \\ & \wedge \circ (FindMigratedService \downarrow \wedge FindDestinationProvider \uparrow \wedge T_c) \\ & \wedge \circ (FindDestinationProvider \downarrow \wedge T_d) \end{aligned} \tag{5.1}$$

In Formula 5.1, $\uparrow$ and $\downarrow$ represent the start event and the end event of each process, respectively, and $T_a \leq T_b \leq T_c \leq T_d$ represent the corresponding events' times.

Definition 5: ***ProviderCurrentLevel(p)*** is the current performance level of a provider $p \in P$.

Definition 6: ***ProviderPreferredLevel(p)*** is the preferred performance level of provider $p \in P$.

Definition 7: ***ServiceCurrentLevel(s, p)*** is the current quality of service $s \in S$ *hosted on provider* $p \in P$.

Definition 8: ***ServicePreferredLevel***(s) is the preferred quality of service $s \in S$.

Definition 9: ***Provider Preference Violation*** is the state when *ProviderCurrentLevel* of a provider p become less than its *ProviderPreferredLevel*. A preference violation can be modelled as in Formula 5.2. Stating that the *FindOriginProvider* process will start sometime in the future when a preference violation will be met.

$$\begin{aligned} \forall S_k, P_i \text{ s.t. } & (ProviderCurrentLevel\,(P_i) < ProviderPreferredLevel\,(P_i)) \rightarrow \\ & \diamond FindOriginProvider \end{aligned} \tag{5.2}$$

Definition 10: ***Service Preference Violation*** presents the state when the *Service-CurrentLevel* of service S_k; $k \in 1 \ldots, n$ running on provider P_i for $i \in 1 \ldots, m$ is no longer at its *ServicePreferredLevel*.

$$\begin{aligned} \forall \mathrm{S_k}, \mathrm{P_i}\ & s.t.\,(ServiceCurrentLevel\ \ (S_k, P_i) < ServicePreferredLevel\,(\mathrm{S_k})) \\ & \wedge\ FindOriginProvider \rightarrow \diamond FindMigratedService \end{aligned} \tag{5.3}$$

Definition 11: Post Migration Condition presents the requirements to be satisfied after the migration of the selected service to a new provider so that the migration to the

new provider will guarantee that both provider preferences and service preferences will be always satisfied after the migration:

$$\begin{aligned} \forall S_k, \exists P_j s.t.(\quad & D \rightarrow \\ & \Box((ProviderLevel(P_j) \geq ProviderPreferredLevel(P_j)) \\ & \wedge(ServiceCurrentLevel(S_k, P_j) \geq ServicePreferredLevel(S_k))) \end{aligned} \tag{5.4}$$

In Figure 5.1, we present the service migration decision-making process as a finite state automaton based on the provided definitions and rules representing pre- and post-conditions of the migration decision making processes. The FindOriginProvider process will start by the discovery of a Provider Preference Violation or at the end of the FindDestinationProvider when the migration fails to satisfy the Post Migration Condition. It means there is still some unsolved Service Preference Violation or Provider Preference Violation in the system. The Decision Making Process will finish when the Post Migration Condition is satisfied meaning that all ProviderPreferredLevel levels and ServicePreferredLevel are guaranteed.

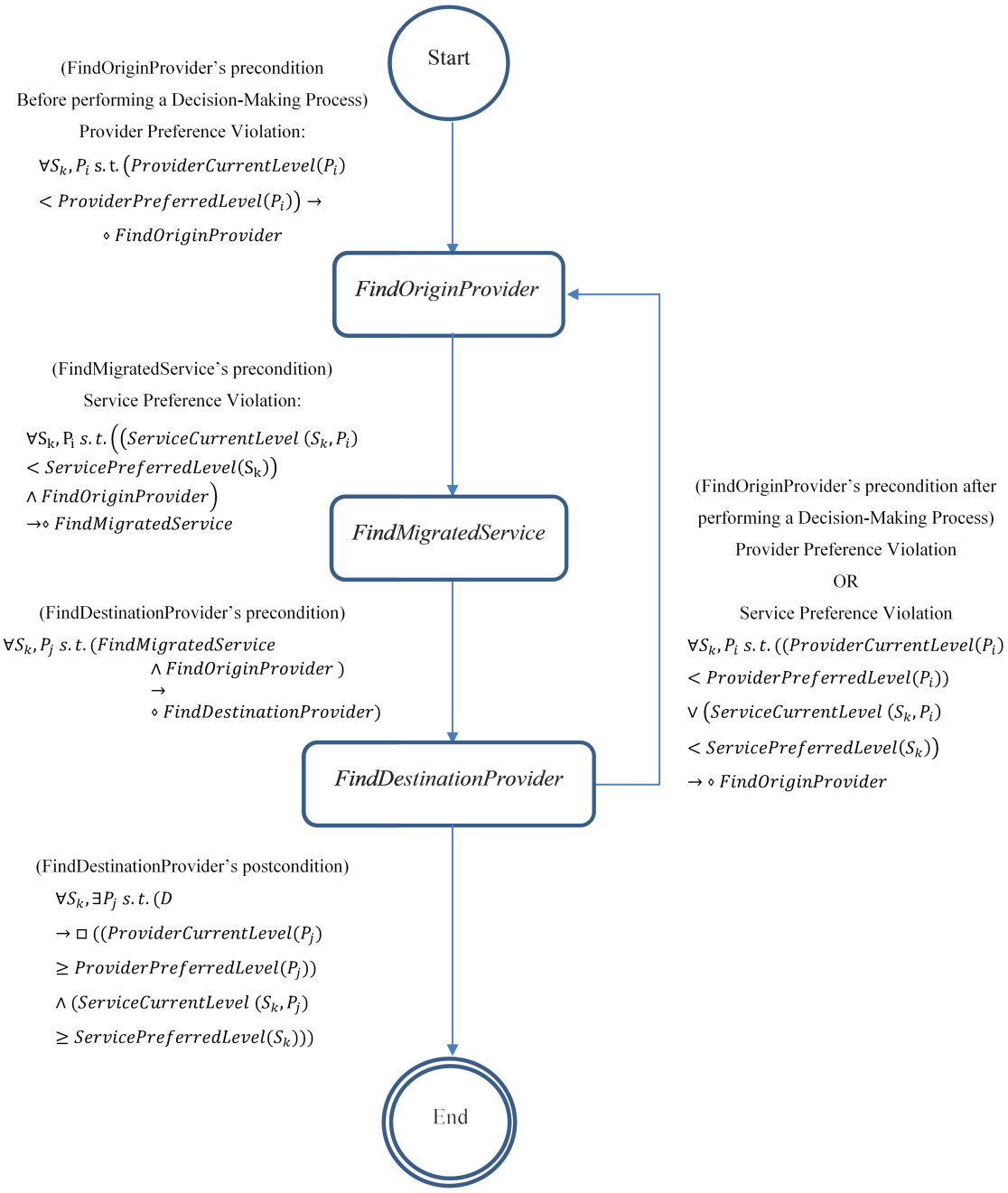

Figure 5.1: The migration-decision process as a finite state automaton.

5.4 Web Service Migration Ontology

To enable service migration through our adaptation approach, we need to define domain specific terms that describe the statuses and properties system components (i.e., *Service* and *ServiceProvider*) during the migration process. To do that, we extend the system core ontology presented in Section 3.4 by defining sub-classes of *Service*, *ServiceProvider* and Property so that it can be used to describe the context model of each system components. The Web service migration ontology is presented in Figure 5.2.

5.4.1 Service Migration Ontology Classes

The Service class is specialised to a *ProvidedService* and *FrameworkService*. The migration ontology adopts the Web Service OWL-S ontology by considering the *presentedBy* object property of a Service so that a *ProvidedService* is considered as a *MigratableService* iff it is *presentedBy* a *MigratableServiceProfile*. A Service is specialized by its *ServiceProfile* to one of the following Service's subtypes:

- ***ProvidedService***: is a subtype Service provided by a *ServiceProvider* of Service.
- ***MigratableService***: a subtype of *ProvidedService* class which possible to be migrated from one *ServiceProvider* to another one.
- ***FrameworkService:*** a subtype of *ProvidedService* class, it is an auxiliary service concerned with managing the migration process.
- ***CandidateForMigrationService***: a subtype of *ProvidedService* class, which represents the service found with violated preference and/or causing violations of its current service provider preferences. Thus, it is recommended to be migrated to another service provider.

On the other hand, properties of *Service*s and *ServiceProvider*s are defined as new classes of Property class, the ontology-based description allows to define new property classes in the models and does not limit the description with a static set of properties. In the Web service migration scenario we are going to define the properties that will be considered in the preferences of services and service providers which control the migration process by evaluating the considered properties values of services and service providers. The following list demonstrates the used sub-classes of *Property* with their descriptions.

- ***ServicePriority***: The priority of a service with single value of the scope [0-100].
- ***FreeMemory***: The current free size of memory on the service provider, measured by Megabytes.
- ***PermanentStorageSize***: The size of permanent memory on the service provider, measured by Megabytes.
- ***BatteryLifeTime***: The current available time of a service provider battery life time, measured by hours.

New sub-classes of *ServiceProvider* class are defined to express the status of the service provider after evaluating its preferences and the preferences of its hosted services. The new classes are as follow

1. ***CandidateOriginServiceProvider***: is a service provider having the required auxiliary *FrameworkService*-s to send its migratable services to another service provider. A CandidateForMigrationService must be hosted on service provider of *CandidateOriginServiceProvider* to be able to send the service successfully.
2. ***CandidateDestinationServiceProvider***: is a service provider that can be a destination for one or more services of the type CandidateForMigrationService.

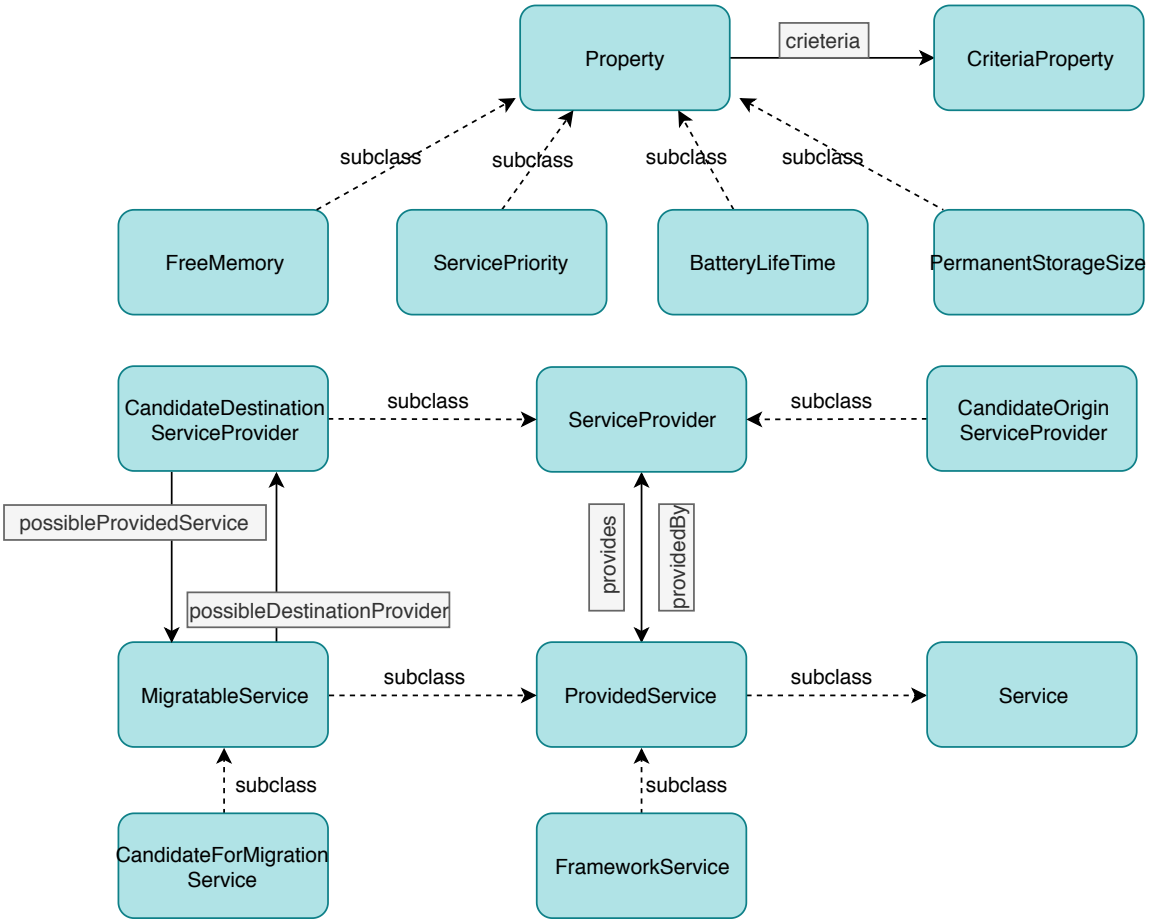

Figure 5.2: The ontology of web service migration system.

By reasoning the full system model, new services will be specialized as CandidateForMigrationServices. Only *ServiceProvider*-s that satisfy the preferences of one or more CandidateForMigrationServices as a new service provider will be classified as a *CandidateDestinationServiceProvider*-s.

A set of auxiliary services were implemented to support the migration process. The services are instances of *FrameworkService* and operate on origin's and destination's providers to provide/collect information about CandidateForMigrationService, CandidateOriginServiceProvider, and *CandidateDestinationServiceProvider*, and to provide the functionalities to perform the migration process.

5.4.2 Service Migration Object Properties

We defined two object properties to express the relationships between the *MigratableService* and *ServiceProvider*.

1. ***possibleDestinationProvider***: is the object property defining the relationship between the MigratableService and CandidateDestinationServiceProvider classes as the domain and range respectively.

2. ***possibleProvidedService:*** is the object property defining the relationship between the CandidateDestinationServiceProvider and *MigratableService* classes as the domain and range respectively.

In Figure 5.3, the *possibleProvidedService* and *possibleDestinationProvider* properties are defined as symmetric properties in the ontology of *CandidateDestinationServiceProvider* and *MigratableService* respectively as follows:

```
<owl:SymmetricProperty rdf:about="WSMF:possibleProvidedService„>
<rdfs:domain rdf:resource="WSMF:CandidateDestinationServiceProvider„/>
<rdfs:range rdf:resource="WSMF:MigratableService„/>
<rdf:type rdf:resource="http://www.w3.org/2002/07/owl#ObjectProperty„/>
</owl:SymmetricProperty>

<owl:SymmetricProperty rdf:about="WSMF:possibleDestinationProvider„>
<rdfs:domain rdf:resource="WSMF:MigratableService„/>
<rdfs:range rdf:resource="WSMF:CandidateDestinationServiceProvider„/>
<rdf:type rdf:resource="http://www.w3.org/2002/07/owl#ObjectProperty„/>
</owl:SymmetricProperty>
```

Figure 5.3: The definition of possibleProvidedService and possibleDestinationProvider Properties in the Service Migration Ontology.

Both of *MigratableService*-s and *ServiceProvider*-s may have operation preferences. Each preference is described as a simplified RDFs rule expressing the operation conditions that satisfy service or provider operation requirements. These rules are evaluated by a RDFS rule reasoner to check whether each service can be a possibleProvidedService by some *ServiceProvider* or not. Based on this reasoning, a *MigratableService* which can not be possibleProvidedService by its current provider will be marked as a CandidateForMigrationService iff its service provider is of type CandidateOriginServiceProvider. The same reasoning is used to find a suitable CandidateDestinationServiceProvider for each *MigratableService* instance of the type CandidateForMigrationService through deriving new instances of the object propertypossibleDestinationProvider. Instance of service and service provider of matched possibleDestinationProvider and possibleProvidedService will be considered as a service migration possibility and will be added to the list of suggested migration decision.

5.4.3 Rules

The power of the using ontology-based models is located in the ability to derive additional information from the service and service provider models and also to express the requirements of each service and provider as preferable working conditions. These conditions can be presented through JENA [14] rules defined based on the provided ontology in the component models (i.e., Service or Service Provider model) and added to the ontology model as a set of component rules. Moreover, JENA rules can describe pre-defined system conditions, not necessarily related to specific component but to the system business process in general. A system rule can be defined directly in the core system model.

Core Rules To enable design and runtime configuration, we provide a set of JENA rules that describe system configuration and preferences. The rules can be used by JENA reasoner

to obtain context information required to identify system components that can participate in the adaptation. For simplicity reason, the defined core rules will be demonstrated using the predicate logic notations. The complete list of the defined rules is provided in Appendix E of the thesis.

The defined rules are listed as follows:

Rule 1: *BecameCandidateOriginServiceProvider*, is the rule defining the required auxiliary *FrameworkService*-s that enable a service provider to become a *Candidate OriginServiceProvider.*

Rule 2: *BecameCandidateDestinationServiceProvider*, is the rule defining the required auxiliary*FrameworkService-s* that enable a service provider to become a *Candidate DestinationServiceProvider.*

By reasoning system ontology model, each *ServiceProvider* instance will possibly be specialized into one of the sub classes CandidateOriginServiceProvider and CandidateDestination ServiceProvider through the implementation of *Rule_1* and *Rule_2* respectively. The specialization process is controlled by each rule that checks about the auxiliary framework services provided by each ServiceProvider. If a *ServiceProvider* provides all the required instances of services responsible for storing the provided *MigratableService* status, creating its deployment package and sending it to another *ServiceProvider*, then that *ServiceProvider* is specialized to CandidateOriginServiceProvider. Meanwhile, a *ServiceProvider* will specialized to a CandidateDestinationServiceProvider when it provides the required auxiliary framework services to receive the deployment package of the *MigratbleService* and to deploy it on its new provider.

The second set of defined system rules is responsible for deriving instances as possibleProvidedService for a CandidateDestinationServiceProvider and possibleDestination-Provider for a *MigratableService.* Two RDFs rules are defined as follows:

Rule 3: *FindServiceForProvidersWithoutPreferences*, is the rule responsible for initiating entries of all discovered *MigratableService*-s as possibleProvidedService of any instance of type CandidateDestinationServiceProvider that has no preferences.

Rule_3 considers any CandidateDestinationServiceProvider which has no preferences as a possibleProvidedService for Service of type *MigratableService.* The instance of a *MigratableService* type is presented by the term 's' element and the instance of *CandidateDestinationServiceProvider.*

Rule 3: *FindServiceForProvidersWithoutPreferences*
Let s = service, p = provider, NP = has no preference rules
MG is a MigratableService, CAND = is a candidate destination provider, PPS = possible provided service.

$$\forall s, p\,(MG\,(s) \wedge NP\,(p) \wedge CAND\,(p) \rightarrow PPS\,(p, s)) \qquad (5.5)$$

Rule 4: *FindProvidersForServicesWithoutPreferences:* is the rule responsible for initiating entries of all discovered instances of the CandidateDestinationServiceProvider as *possibleDestinationProvider* of any *MigratableService* that has no preferences.

Rule_4: *FindProvidersForServicesWithoutPreferences*
Let PDP = possible destination provider

$$\forall s, p\,(MG\,(s) \wedge NP\,(s) \wedge CAND\,(p) \rightarrow PDP\,(s, p)) \qquad (5.6)$$

In brief, by implementing the aforementioned rules in the system, a *MigratableService* without preferences is suggested to be migrated to any CandidateDestinationServiceProvider. On the contrary, a CandidateDestinationServiceProvider that has no preferences will be suggested to host any *MigratableService*.

The last group of core rules helps to generate new instances of type CandidateForMigratonService -s in the model.

Rule 5: *CandidateForMigrationServiceDueToServicesPreferences*, is responsible for noting a *MigratableService* as a CandidateForMigrationService in the model due to the violation of a preference of that *MigratableService*.

Rule_5: *CandidateForMigrationServiceDueToServicesPreferences*
Let o, d = provider, CANO = provider is a candidate origin,
PROVIDES = provider provides a service, CANS = service is a candidate for migration.

$$\begin{aligned} &\forall\ s, o, d\ (MG\,(s)\ \wedge\ \neg NP\,(s)\ \wedge\ \ CANO\,(o)\ \wedge\ CAND\,(d) \\ &\wedge\ PROVIDES\,(o, s) \wedge PDP\,(s, d)\ \wedge\ \neg PDP\,(s, o) \\ &\rightarrow CANS(s)) \end{aligned} \qquad (5.7)$$

Rule 6: *CandidateForMigrationServiceDueToProvidersPreferences* is responsible for noting a *MigratableService* as a CandidateForMigrationService in the model due to a *ServiceProvider* preference violation through hosting the related *MigratableService*.

Rule_6: *CandidateForMigrationServiceDueToProvidersPreferences*

$$\begin{aligned} &\forall\ s, o, d\ (MG\,(s)\ \wedge\ \neg NP\,(d)\ \wedge\ \ CANO\,(o) \\ &\wedge\ CAND\,(o)\ \wedge\ CAND\,(d)\ \wedge\ PROVIDES\,(o, s) \wedge \\ &PPS\,(d, s)\ \wedge\ \neg PPS\,(s, o) \rightarrow CANS(s)) \end{aligned} \qquad (5.8)$$

Component Rules A component rule describes a preference of *Service* or *ServiceProvider*. Each rule is identified in its owner model as a string entry of the rule element as described in Section 3.5. The main purpose of defining a component rule is to state the operation preference of the component. A component rule should identify new possibleProvidedService and *possibleDestinationProvider* properties of *Service*-s and *ServiceProvider*-s in addition to the information acquired by implementing *Rule_3* and *Rule_4* rules.

In Formula 5.9, we demonstrate an example of a preference rule of an instance *ServiceProvider* called sP. The rule called *SamplePreference* is defined in provider model to present the possibility of hosting *MigratableService* 's' only with *ServicePriority* property higher than 50. The derived information by the implementation of this rule will generate new OWL/RDF triples of *MigratableService* entries as possibleProvidedService properties of the *ServiceProvider* "sP„.

Let sP = name of service provider sP, v = service priority property, *hasServicePriority* = has a service priority property.

$$\begin{aligned} &\forall v \ \in \mathbb{Z} \ \forall \ s, o, d \ (MG\,(s) \ \wedge \ CANO\,(o) \ \wedge \ CAND\,(d) \\ &\wedge \ PROVIDES\,(o, s) \wedge (d = sP) \wedge hasServicePriority\,(\text{s},\ \text{v}) \\ &\wedge (v \geq 50) \rightarrow \text{PPS(d, s))} \end{aligned} \tag{5.9}$$

As described in the rule, the 'o' term represents any CandidateOriginServiceProvider originally hosting 's'. The 'd' term represents any CandidateDestinationServiceProvider instance. The property value 'v' of *ServicePriority* property of 's' instance is set to values equal or greater than 50.

5.5 Mobile Web Service Migration Framework Architecture

In this section we demonstrate the framework architecture for service migration presented in Section 5.2. The framework's Controller running on service provider will lead both the context-awareness and adaptation processes. On the first hand, context-awareness process is presented in the system through the following functionalities:

- discovering the connected service providers in the network.
- checking the destination service provider availability.
- monitoring the quality of service after migration and deciding if another migration is required.

On the other hand, system adaptation is presented through the ability of the framework controller on a service provider to migrate a service to a new service provider and assure the availability of the migrated service.

We demonstrate the architecture of proposed framework in Figure 5.4, the architecture consists of:

1) The Presentation Layer: it is the top-level layer of the adaptive system application that allows end-users to access and interact with the system and its services via a Graphic User Interface (GUI) (such as a web page).

2) The Application Layer: it is the layer containing the system Framework Web service, Migratable Web services and controller. The Migratable Web services implement business processes and can be consumed by end-users and external systems. The system controller is responsible for managing the integrity of system business processes by performing service migration according an adaptation strategy. The controller consists of the following modules: 1) Discovery Module: it is the unit responsible for automatic discovery of joining service providers and their services in the system. 2) System Context Manager Module: it is the unit responsible for creating system context model and discovering violations of the rules defined in the context models of discovered services and service providers. 3) Migration Module, which is responsible for making a migration decision, moving a service and deploying it on a suitable service provider.

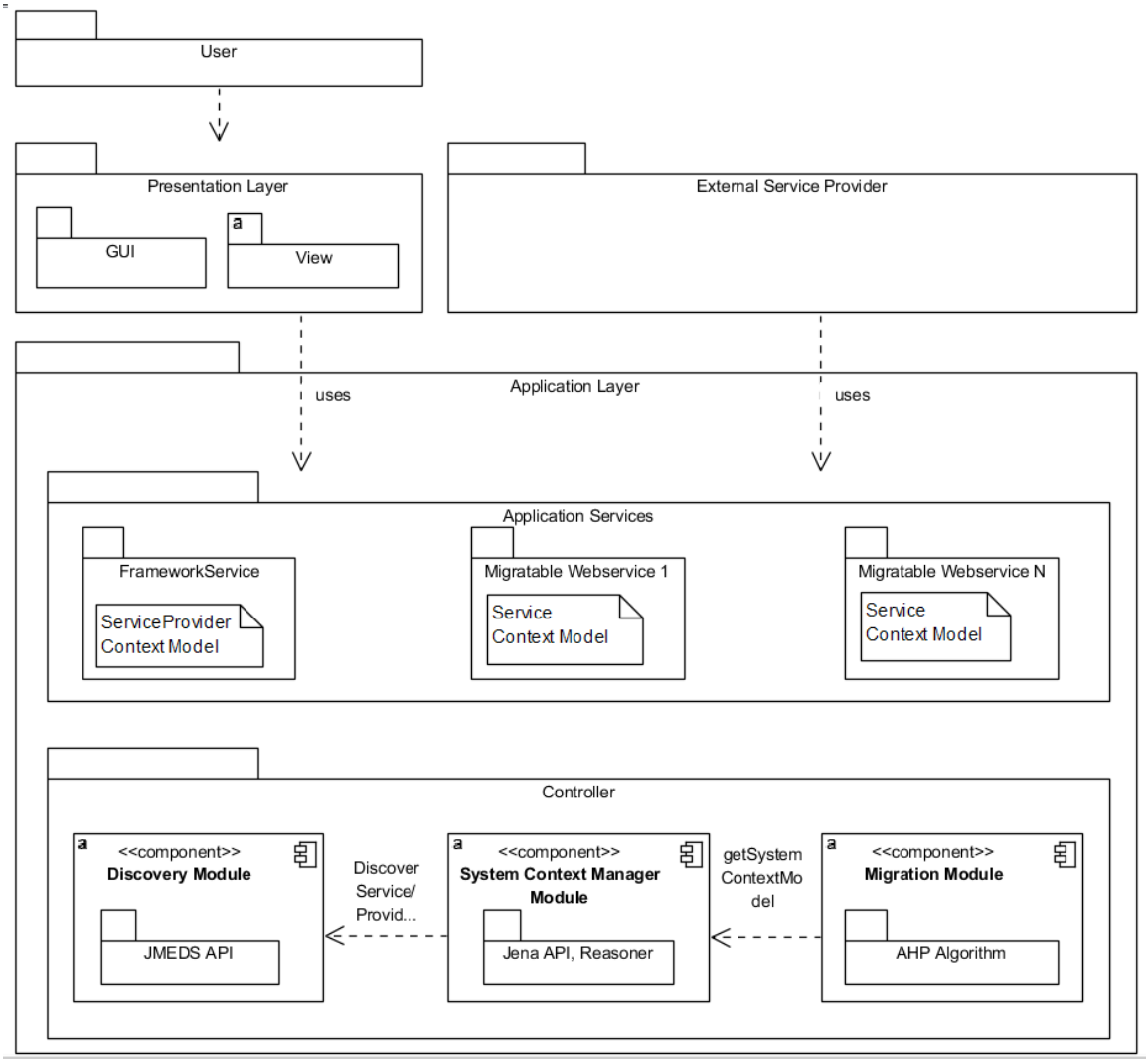

Figure 5.4: Mobile Web service migration architecture.

5.5.1 Discovery Module

The Discovery Module is responsible for a discovery process of the available service providers in the network. Moreover, it is responsible for the discovery of *FrameworkService*-s and *MigratableService*-s (see Section 5.4.1) hosted on each discovered service provider so that their models can be requested in order to be considered in the Context Manager Module's reasoning process.

5.5.2 System Context Manager Module

This module is responsible for generating, monitoring and reasoning system context periodically to enable system context awareness. The module creates system core context model and all partial context models of discovered service providers and *MigratableService* model intended for migration. It is also responsible for generating the partial models of system components that define real-time statuses of properties and preference rules of the subject *MigratableService* and the surrounding service providers.

After creating the system context model of the discovered service providers, the *Controller* looks for a destination service provider that can host *MigratableService* where the pre-defined rules of both *MigratableService* and the destination service provider can be satisfied.

This process is performed through the utilization of an ontology reasoner to derive new context information from system context model and to find the *possibleDestinationProvider* for *MigratableService* that has also *MigratableService* as a *possibleProvidedService*.

The output of this module is a list of triple entries stating the *MigratableService*, the source *ServiceProvider*, and the *possibleDestinationProvider*. This list of entries is the input of the decision-making process performed by the *Migration Module*.

5.5.3 Migration Module

This unit is responsible for selecting the best migration to perform from the input set of possible migrations. It utilizes the algorithms proposed in Section 4.2 to initiate the required matrices used by the AHP decision-making method to choose the migration with the highest priority calculated based on priority values of the defined criteria.

5.5.4 Migration Process

The migration process, demonstrated in Figure 5.5, begins when the *Discovery Module* of a framework controller hosted on some device SP1 starts the Search process to discover the connected devices and services in the network. When a device SP2 is discovered, the SP1 controller's *System Context Manager Module* fetches and adds the partial context models of SP1, SP2 and their hosted services to the core context model including their preference rules found in the partial context model of them. After that, the *System Context Manager Module* utilizes JENA reasoner to reason the generated system model and query it for any derived migration suggestions matching the services and devices preferences. The suggested list of possible migrations is ranked through the *Migration Module* of framework's controller which selects the migration with the highest priority to be performed.

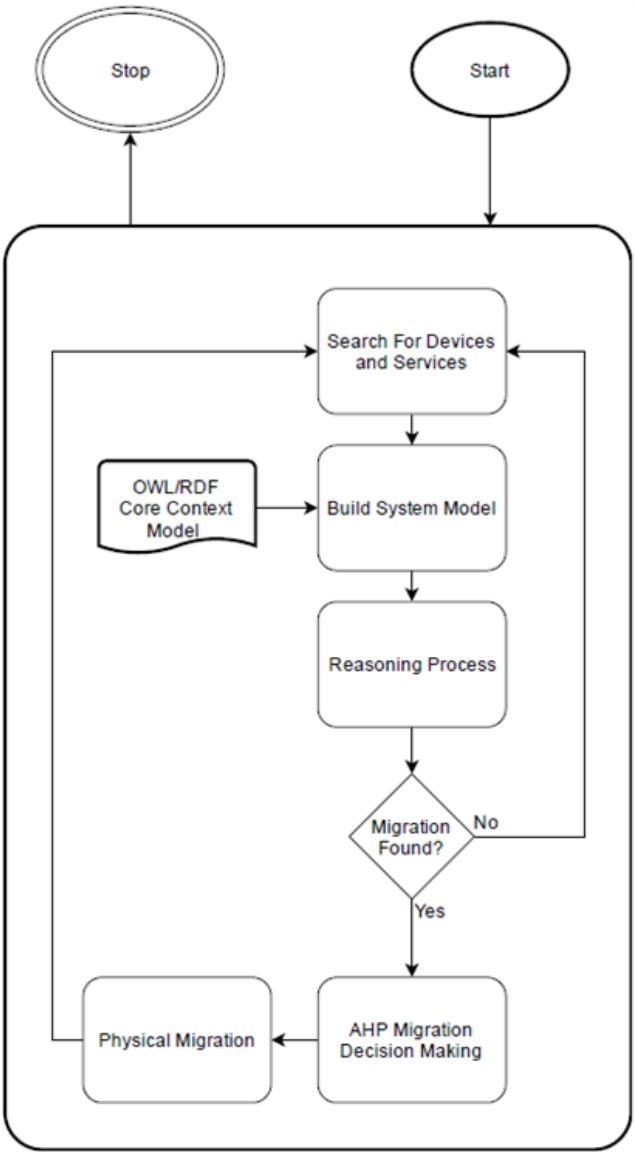

Figure 5.5: Illustration of the Migration Process Steps.

Part III

Implementation and Experimental Results

Chapter 6

Web Service Migration-based Framework Description

In this chapter, we provide the system requirements and implementation description of our Web service migration framework presented in Section 5.5. We adopt Service migration as adaptation strategy to solve violations in preferences of system's services and/or devices (i.e., mobile phones provided with a running http server). First, we demonstrate the requirements analysis of migration system. Later, we provide implementation description including the technical aspects adopted for development of the system. Finally, we provide an example to illustrate our migration approach.

6.1 System Requirements

We provide a context-aware self-adaptive framework to support an automatic Web service migration in SOA. The framework must provide the following functionalities.

- discover services and service providers so that their context models can be retrieved and used in building system context model.
- allow to semantically describe context information which is important for providing the correct understanding/utilization of context information between system components.
- monitor context changes and discover possible context violations in system model.
- react to context violations by triggering the adaptation process.
- suggest possible migrations and automatically choose the most suitable migration to perform based on defined weighting criteria.
- execute the migration decision by physically migrating services to the new destination service provider so that the found violations will be resolved.

6.2 Implementation Description

In this section, we describe the design and implementation of the proposed service migration framework.

6.2.1 Service Migration Framework Architecture

To support the proposed process of Web service migration, we designed a generic context-aware migration-based framework. The framework describes an overall service-oriented architecture supporting the service migration and defines interfaces which can be implemented to adapt the framework to a particular Web service implementation technology. It also provides extension points for user-defined migration decision strategies, i.e., the strategies deciding when the migration of a particular service is needed and how it will be performed. To utilise the framework, demonstrated in Figure 6.1, an implementation of interface *MigrationDecisionStrategy* and auxiliary classes with interfaces *ProviderStatus*, *ServiceStatus*, and *ServiceSemanticDescription*, representing state and semantic information, must be provided. Migration decisions are based on state information extracted from service providers (e.g., available resources, system workload, battery state, etc.) and their services (e.g., utilized resources, number of requests per a unit of time, etc.) and on the services' semantic descriptions (e.g., provided functionality, inputs and outputs, required runtime conditions, etc.). The migration decision strategy has to be able to acquire instances of the mentioned classes (i.e., the objects representing the state and semantic information) from providers supporting service migration and migratable services.

Interface MigrationDecisionStrategy defines methods *getProviderMigrationNecessity*, *getServiceMigrationNecessity*, and *getDestinationSuitability*. The first two methods decide whether some services of a particular service provider or a particular service of this provider, respectively, need to be migrated for some reasons. The third method decides whether a particular provider can be a migration destination for a particular service (i.e., whether a given service can be provided by a given provider after the migration). Returning values of the methods are directly proportional to the necessity of migration of the services or the suitability of the migration destinations. To be able to migrate, services need to implement interface *MigratableService* with the following public methods. Method *getStatus* returns state information that is used in a migration decision strategy to decide whether a particular service needs to be migrated. Method *getSemanticDescription* provides a semantic description of a migrated service which is used in a migration decision strategy to select an appropriate destination service provider. Method *getDeploymentPackage* returns a service deployment package which is used to deploy a new instance of a migrated service at a destination service provider. Finally, *migrateToDestination* transfers a service's internal state from the service's old instance to its previously deployed new instance and finalises the migration.

Service providers with migratable services need to implement interface *MigrationProvider* with the following public methods. Method *getStatus* returns state information that is used in a migration decision strategy to decide whether services hosted by a particular service provider need to be migrated.

Method *getHostedServices* returns all migratable services of a service provider. Method *deployServiceFromPackage* should be able to deploy a service package to create a new instance of the deployed service on a destination service provider. Finally, method *removeService* removes a migrated service from its origin provider.

6.2.2 Device and Service Discovery

We decided to use the Devices Profile for Web Services (DPWS) standard in implementing the framework *Discovery Module* introduced in Section 6.2.2 . DPWS is a middleware with the minimal set of implementation constraints to define the devices and their hosted

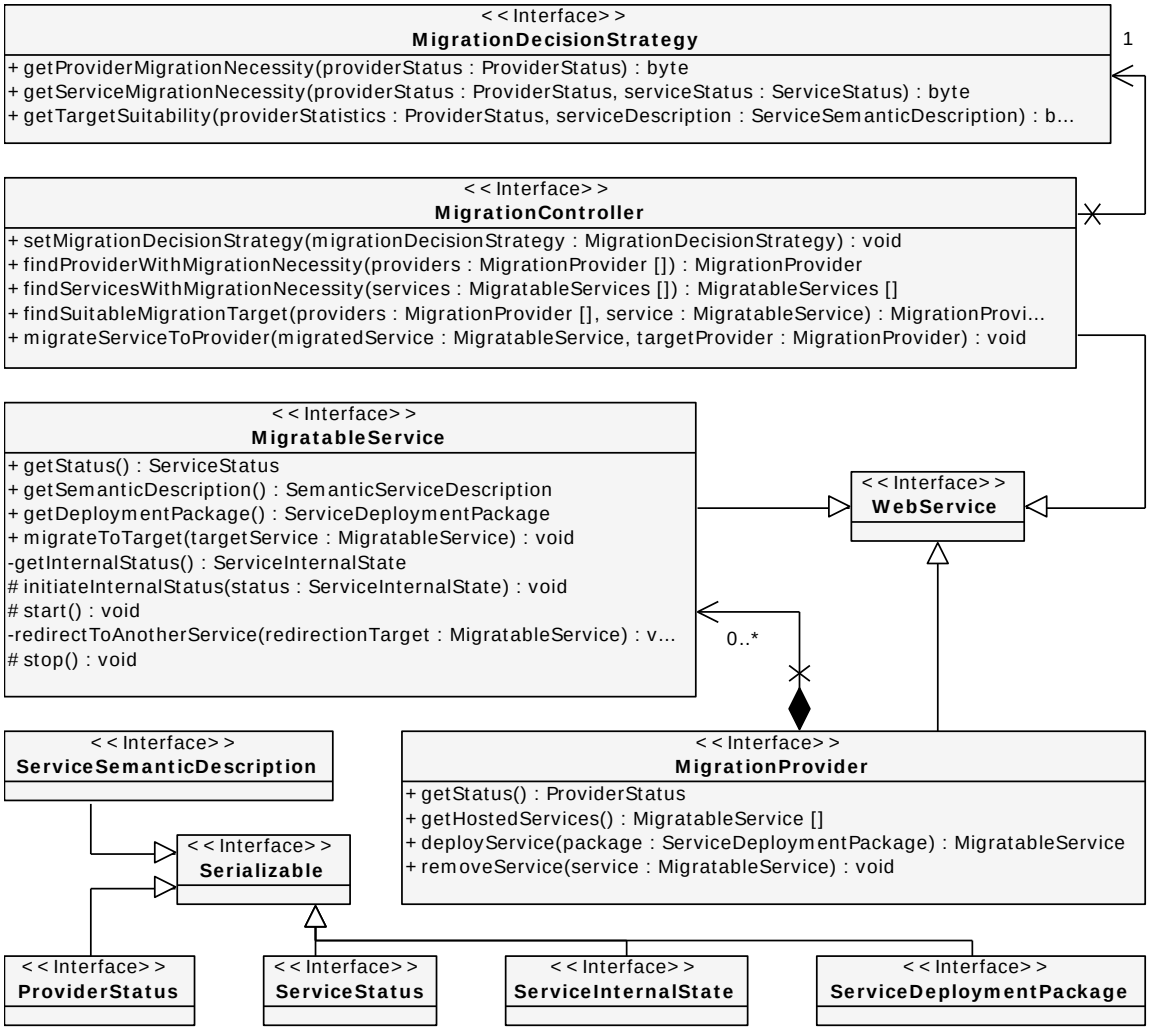

Figure 6.1: The interface of the framework's controller and the interfaces implemented by participating services and service providers to enable the service migration.

services. The hosted services discovery depends on the discovery of their hosting devices, meanwhile the process of device discovery is provided by another type of DPWS built-in services called the discovery services. Each discovery service advertises its Device profile in the network to be discovered by other discovery services. The discovery process of the hosted services is managed by metadata exchange built-in services that can dynamically access to devices hosted services and to their metadata. To enable the automatic discovery of emerging service providers, the DPWS standard has been utilized by integration of WS4D.org Java Multi Edition DPWS Stack (WS4D-JMEDS) [96]. WS4D-JMEDS is a light-weight framework that allows to implement and run DPWS Services, Devices and Clients. Through using JMEDS, the required SOA infrastructure to build the Web service migration system will be provided. Moreover, we will be able to easily discover system's devices and their services and to retrieve the required information to build the system model locally on each device which enables the decentralized adaptation in our framework.

6.2.3 System Services

We adopt the REST architectural style for SOA design due to the lower message payload of REST framework than SOAP framework which makes REST more suitable for mobile devices [89]. For this reason, we utilize the Restlet framework [55] to implement our RESTful Web services for the migration framework as the Restlet APIs supports both JAVA and Android platforms which enables to publish the Web services on stationary and mobile servers (i.e., devices) without the need for modifying its source-code.

For the HTTP server we choose I-Jetty project [5] which is a lightweight HTTP server for Android. Each HTTP server hosts one grounding service *FrameworkService* which has the required functionalities to package, send and receive Web service WAR packages between *FrameworkService* system devices.

We differentiate between three types of resources during the implementation in order to cope with the principals of RESTful architectural style. Each resource is presented as Restlet presentation class file and attached to the device URI.

First, The Service resource class, it provides the necessary methods to deal with the hosted service as a resource

- GET: getServiceWar method is specified to packages the service in a WAR format file path on service provider.
- POST: addService method provides the functionality to initiate new JMEDS service instance on service provider.
- DELETE: deleteService method deletes the service instance from the related service provider.

Second, the Provider resource class:

- GET: getProviderContext, is specified to returns service provider context model through the getContext method.
- POST: addDevice, is defined to start new instance from JEMDS device in the network.
- DELETE: deleteDevice is called to stop and delete JEMDS device instance so that it is not discovered in the system.

And last, the Framework resource class, which provides the required RESTFUL methods to manage required functionalities such as deploying Service package, identifying the operation system on service provider.

On the other hand, each running *MigratableService* has its context model which can be retrieved by its URI. The service OWL/RDF context model is attached to the service's the WADL file by inheriting the *WadlApplication* Restlet class in the main service *Application* class. Then the WADL context model is provided through implementing the getApplicationInfo method and stored as an instance of *DocumentationInfo* class of Restlet framework.

6.2.4 Context Model Reasoning

The process of deriving possible migrations is performed through implementing JENA framework reasoner on the service providers' and services' models with respect to their

Provider	noPreferenceRules	FreeMemory	PermanentStorageSize	BatteryLifeTime
XProvider	True	512 MB	512 MB	1 hour
YProvider	False	2048 MB	2048 MB	2 hours
ZProvider	True	2048 MB	2048 MB	3 hours

Table 6.1: Values of the status properties published in partial models of the service providers.

Service	*noPreferenceRules*	*ServicePriority*	*ServiceType*
Service1	False	50%	major
Service2	True	20%	minor

Table 6.2: Values of the status properties published in partial models of the services.

defined JENA rule preferences. On Android-based framework application, we integrated Androjena APIs [86] for Android to enable JENA based reasoning for the generated system context model. Then the migration with the highest weight from the proposed set of migrations is chosen to be executed. A detailed description of the AHP decision-making process used to select the migration with the highest weight is provided in Section 4.1.

6.3 Migration Example

In this section we provide an illustrative example to evaluate the proposed service migration approach introduced in Section 5.2. The example presents a system of service providers hosting services which cooperate to achieve a particular business logic. Web service migration allows to migrate services from one provider to another in cases of unexpected violations of services or service providers preferences.

The migration guarantees better availability of the services and increases fault-tolerance of the system. Let us suppose the system consists of the following three service providers: *XProvider*, *YProvider*, and *ZProvider*. The status properties of these providers, which are published in their partial context model together with their preference rules are listed in Table 6.1.

For example, the information published by *YProvider* service provider, that is its status properties and preference rules, is listed in Figure 6.2. For simplicity reasons, let us suppose that there are only two migratable services, namely service *Service1* currently provided/hosted by service provider *XProvider* and service *Service2* currently provided/hosted by service provider *YProvider*. A partial context model of each of these two services is available as a part of their WADL file. The status properties published in the context models of these services are listed in Table 6.2. Moreover, *Service1* context model declares the two preferences of *Service1* as *Service1Preference1* and *Service1Preference2,* which limits *Service1* migration to providers with FreeMemory $\geq$ 2048 MB and PermanentStorageSize $\geq$ 2048 MB respectively.

According to the properties published by the individual service providers (see Table 6.1 and Figure 6.2), *OriginBatteryLifeTimeCriterion* and *DestinationBatteryLifeTimeCriterion* are considered as criteria to be included in the decision-making process dealing with *BatteryLifeTime* properties of the providers. Similarly, the services (see Table 6.2 and Figure 6.3) publish their *ServicePriority* properties related to criterion *ServicePriorityCriterion* to make it considered in the decision-making process dealing with these properties.

Migration	Service	Origin	Destination
mig1	Service1	XProvider	YProvider
mig2	Service1	XProvider	ZProvider
mig3	Service2	YProvider	XProvider
mig4	Service2	YProvider	ZProvider

Table 6.3: The migrations found to fix the violated preference rule.

During this example's design-time, the defined preferences rules of *Service1* and *YProvider* (see Figure 6.3 and Figure 6.2) are set to be violated during the run-time. As defined, *Service 1* is hosted by *XProvider* which has *FreeMemory* and *PermanentStorageSize* less than 2048 MB (see Table). Also, YProviderPreference is violated as YProvider is currently hosting *Service2* which does not have the *ServiceType* of *'major'*. Having *Service1ProviderPreference1*, *Service1ProviderPreference2*, and *YProviderPreference* violated, the migration process will be performed to resolve these violations. As described in Section 5.5.4, the framework controller will start the migration process and perform the following steps:

Step 1 - Building System Context Model When the framework controller of service provider discovers other service providers in the system, the controller calls the *FrameworkService* service of each discovered service provider to collect its context model. Next it extracts the context model of each hosted migratable service from the service WADL file. The properties and preference rules extracted from service providers and migratable services context models are combined in one model to be reasoned to generate an inferred context model. Through the reasoning process, the framework's controller will check if there is any violation in the defined preference rules. In this example, the controller detects two preference rule violations:

In the first case, *Service1Preference1* and *Service1Preference2* preference rules of service *Service1* are violated. These preference rules, which permit to host the service only by providers with *FreeMemory* $\geq$ 2048 MB and *PermanentStorageSize* $\geq$ 2048 MB, are violated by service provider *XProvider* which is currently hosting the service and has its both status properties *FreeMemory* and *PermanentStorageSize* set to value 512 MB (see Table 6.1).

In the second case, preference rule *YProviderPreference* of service provider *YProvider* is violated. This preference rule, which permits the provider to host only services with *ServiceType* set to value "major„ (see Figure 6.2), is violated by service *Service2* which is currently hosted by the provider and has status property *ServiceType* set to value "minor„ (see Table 6.2).

Step 2 - System Context Reasoning Process By ontology reasoning with Jena reasoners, four possible migration decisions are found to fix the violations above when performed in the system. These migrations, which are listed in Table 6.3, are *mig1, mig2, mig3* and *mig4*. Migration *mig1* suggests migrating *Service1* from *XProvider* to *YProvider*; migration *mig2* suggests migrating *Service1* from *XProvider* to *ZProvider*, and so on (see Table 6.3). First two migrations are addressing the first violation (of *Service1Preference1* and *Service1Preference2* preference rules of service *Service1*), while the other two migrations are addressing the second violation (of preference rule *YProviderPreference* of service provider *YProvider*).

	Criteria Priority	owner	valueWith HighestWeight	valueWith LowestWeight
ServicePriority Criterion	7	service	100	0
OriginBatteryLife TimeCriterion	9	origin	1	5
DestinationBattery LifeTimeCriterion	3	destination	5	1

Table 6.4: The utilized criteria and values of their attributes.

$\lambda_{max} = 3.0967$; CR $= 0.0833 < 0.1$; matrix A is consistent.				
	ServicePriority Criterion	*OriginBatteryLife TimeCriterion*	*DestinationBattery LifeTimeCriterion*	w_i
ServicePriority Criterion	1.0	0.1428	0.2	0.0737
OriginBatteryLife TimeCriterion	7.0	1.0	3.0	0.6433
DestinationBattery LifeTimeCriterion	5.0	0.3333	1.0	0.2828

Table 6.5: Comparison matrix A, generated by the *InitializeCriteriaMatrix* algorithm and weight vector w computed from the matrix.

Step 3 - AHP based Decision Making Process The list of all possible migrations needs to be processed by AHP to find the best migration to be performed in the system. In order to use AHP, we define the following criteria in the context model: *ServicePriorityCriterion*, *OriginBatteryLifeTimeCriterion*, and *DestinationBatteryLifeTimeCriterion*. Values of individual attributes of the defined criteria are listed in Table 6.4.

By application of the *InitializeCriteriaMatrix* algorithm (see Section 4.2), criteria comparison matrix A is generated as shown in Tabel 6.5. In the last column of Tabel 6.5, there are also values of the weight vector for the criteria which is computed by application of Equation 4.2 on the normalized comparison matrix.

By application of the *InitializeDecisionMatrices* algorithm on the results above, migration comparison matrices $S^{(1)}$, $S^{(2)}$ and $S^{(3)}$ are generated for criteria *ServicePriorityCriterion*, *OriginBatteryLifeTimeCriterion*, and *DestinationBatteryLifeTimeCriterion*, respectively. These matrices are listed in Figure 6.6 together with their weight vectors. For $k \in 1, 2, 3$ weight vector $w^{(k)}$ is computed for matrix $S^{(k)}$ by application of Equation 4.2 on corresponding normalized matrix $\overline{S}^{(k)}$.

Finally, after generating and computing all required matrices and vectors, AHP computes the composite weight vector p through Equation 4.3. The resulting vector is

$$P = \begin{pmatrix} 0.375 & 0.375 & 0.1534 \\ 0.375 & 0.375 & 0.3889 \\ 0.1249 & 0.1249 & 0.0686 \\ 0.1249 & 0.1249 & 0.3889 \end{pmatrix} \cdot \begin{pmatrix} 0.0737 \\ 0.6433 \\ 0.2828 \end{pmatrix}$$

$\lambda_{max} = 4$, $CR = 0$; matrix $S^{(1)}$ is perfectly consistent.					
ServicePriority Criterion	mig1	mig2	mig3	mig4	$v^{(1)}$
mig1	1.0	1.0	3.0	3.0	0.375
mig2	1.0	1.0	3.0	3.0	0.375
mig3	0.3333	0.3333	1.0	1.0	0.1249
mig4	0.3333	0.3333	1.0	1.0	0.1249
$\lambda_{max} = 4$, $CR = 0$; matrix $S^{(2)}$ is perfectly consistent.					
OriginBatteryLife TimeCriterion	mig1	mig2	mig3	mig4	$v^{(2)}$
mig1	1.0	1.0	3.0	3.0	0.375
mig2	1.0	1.0	3.0	3.0	0.375
mig3	0.3333	0.3333	1.0	1.0	0.1249
mig4	0.3333	0.3333	1.0	1.0	0.1249
$\lambda_{max} = 4.0575$, $CR = 0.0213$; matrix $S^{(3)}$ is consistent.					
DestinationBattery LifeTimeCriterion	mig1	mig2	mig3	mig4	$v^{(3)}$
mig1	1.0	0.3333	3.0	0.3333	0.1534
mig2	3.0	1.0	5.0	1.0	0.3889
mig3	0.3333	0.2	1.0	0.2	0.0686
mig4	3.0	1.0	5.0	1.0	0.3889

Table 6.6: Migration comparison matrices $\boldsymbol{S^{(k)}}$ generated by the initializemigrationmatrices algorithm and weight vectors $\boldsymbol{v^{(k)}}$ computed from the matrices.

$$P = \begin{pmatrix} 0.3586 \\ 0.376 \\ 0.1208 \\ 0.1444 \end{pmatrix} \tag{6.1}$$

where p_{11}, p_{21}, p_{31}, and p_{41} entries represent the weights of *mig1*, *mig2*, *mig3*, and *mig4*, respectively. Finally, *mig2* is chosen to be performed as it has the highest priority (p_{21}= 0.376) and *Service1* will be migrated from service provider *XProvider* to service provider *ZProvider* which will fix the violated preference rules of *Service1*, namely *Service1Preference1* and *Service1Preference2*, and satisfy them with the current *FreeMemory* and *PermanentStorageSize* status property values of *ZProvider*. We set the framework controller to perform only the migration with the highest priority for each adaptation round. The chosen migration will only fix the most critical violations of preference rules found before (not all of those violations). For example, preference rule *YProviderPreference* of service provider *YProvider* is still violated by hosting service *Service2* (see Table 6.1) and should be fixed by future migrations. Informally said, migration *mig2* was selected by AHP because it deals with *Service1* which has greater *ServicePriority* than *Service2* in other migrations and because it has destination service provider *ZProvider* which has the best *BatteryLifeTime* in comparison with other possible destination providers (see Tables 6.2 and 6.1, respectively).

```
{
“name„:“YProvider„,
“type„:“ServiceProvider „,
“noPreferenceRules„:“false„,
“properties„:
[
{“propertyName„:“FreeMemory„,
“propertyValue„:“2048„,
“propertyType„:“ INT„},

{“propertyName„:“PermanentStorageSize„,
“propertyValue„:“2048„,
“propertyType„:“ INT„},

{“propertyName„:“BatteryLifeTime„,
“propertyValue„:“2„,
“propertyType„:“ INT„,
“criteria„:[“OriginBatteryLifeTimeCriterion„,
“DestinationBatteryLifeTimeCriterion„]
}
],
“rules„:
“[YProviderPreference:(?service rdf:type core:MigratableService),
(?origin rdf:type core:CandidateOriginServiceProvider),
(?destination rdf:type core:CandidateDestinationServiceProvider),
equal(?destination, core:YProvider),
(?origin core:provides ?service),
(?service core:hasProperty ?property),
(?property rdf:type core:ServiceType),
(?property core:propertyValue ?v1),
eq(?v1, 'major'^^xsd:sting)
->(?destination core:possibleProvidedService ?service)]„
}
```

Figure 6.2: The partial model for service provider YProvider with information of the provider's status properties and preference rule. Preference rule YProviderPreference written in the JENA rules language allows the provider to host only services with ServiceType set to value “major„ (other providers XProvider and ZProvider do not have this restriction).

```
{ “name„:“Service1„,
“type„:“Service„,
“noPreferenceRules„:“false„,
“properties„:
[{“propertyName„:“ServiceType„,
“propertyValue„:“major„,
“propertyType„:“STRING„},

{“propertyName„:“ ServicePriority„,
“propertyValue„:“50„,
“propertyType„:“ INT„,
“criteria„:[“ServicePriorityCriterion„]
}],
“rules„:
[
“[Service1Preference1:
(?service rdf:type core:MigratableService),
(?origin rdf:type core:CandidateOriginServiceProvider),
(?destination rdf:type core:CandidateDestinationServiceProvider),
(?origin core:provides core:service1),
(?destination core:hasProperty ?property),
(?property rdf:type core:FreeMemory),
(?property core:propertyValue ?v1),
ge(?v1, ’2048’^^xsd:int) ->
(core:service1 core:possibleDestinationProvider ?destination)]„
,
“[Service1Preference2: (?service rdf:type core:MigratableService),
(?origin rdf:type core:CandidateOriginServiceProvider),
(?destination rdf:type core:CandidateDestinationServiceProvider),
(?origin core:provides core:service1),
(?destination core:hasProperty ?property),
(?property rdf:type core:PermanentStorageSize),
(?property core:propertyValue ?v1),
ge(?v1, ’2048’^^xsd:int) ->
(core:service1 core:possibleDestinationProvider ?destination)]„
]
}
```

Figure 6.3: The partial model for service *Service1* with information of its preference rules. *Service1ProviderPreference1* and *Service1ProviderPreference2*.

Chapter 7

Case Studies

In this chapter we demonstrate the proposed self-adaptive service-oriented architecture for Web service migration between stationary and mobile service providers through two case studies. The first case study presents our motivation scenario of the thesis proposing to implement self-adaptation in software system to resolve the missing of traffic information service in a cooperative car scenario. It provides a proof-of-concept of the proposed migration adaptation mechanism for Web service in real-life scenarios. In the second case study, we provide a description migration scenario and the migration framework implemented application. The conducted experiments show the applicability and efficiency of the proposed decentralized migration-based service-oriented architecture which is the main goal of the thesis described in Chapter 5.

7.1 Case Study 1 - Traffic Jam Detection Service Migration

In this section we demonstrate a case study of service migration for traffic jam detection using the proposed Web Service migration framework. This case study is inspired by the traffic jam scenario presented in [69] where the migration framework is installed on a group of cooperative cars. In this case study we present the applicability of our service migration-based SOA model proposed in Chapter 5 .

The purpose of the experiment is to show a practical implementation of our context-aware adaptation approach by migrating a traffic jam detection service from an origin car to search for cars inside an area of interest defined by origin or the sender car. In this experiment we will investigate the validity and reliability of the migration framework in rapidly changing environment and evaluate the applicability of the migration approach through a real-life case study.

7.1.1 Related Work

This section presents the related work utilizing context awareness and self-adaptation to solve traffic jam problems. It discusses the differences between these works and the demonstrated work in this case study.

In the work of Hu et al. [38], context-awareness has been proposed to enable the usage of several resources of contextual data such as user's personal activities, social data and environment context (i.e., location, temperature). The authors defined an ontology to describe a mobile smart city system in a crowdsensing scenario. Context-awareness was

implemented through context monitoring and matching of the collected context data and providing system recommendation to the user.

The authors of [20], propose a peer-to-peer architecture for mobile Web service selection and composition. The proposed architecture composer is responsible for discovering the services hosted on nearby mobile devices and composing the required service to respond to a mobile user service request. However, their proposed algorithm uses only service response time factor to select the best services to involve in the composition. In the contrast, this work's framework allows to use a dynamic set of services' and devices' properties and preferences in order to find a set of possible destinations. Moreover, the work authors utilize the Analytic Hierarchy Process (AHP) decision making algorithm with a dynamic set of criteria to determine the best migration to perform. However, this work proposes service migration as an adaptation so that the service can be migrated and hosted on the requesting device, not only to be used while the requester is close to the service origin device.

A cooperative aware vehicle communication system is proposed in the work of Santa et al. [74] to provide information about traffic status and events. A Cooperative Awareness Message (CAM) and Decentralized Environmental Notification Message (DENM) are proposed to describe exchanged messages between cars stating their current status and position. While CAM is used for status notification in one-hop communication, DENM messages are broadcasted over multi-hop communication to cover a specific geographic area. A car hosts services that allow the system to retrieve its position and status to be used in traffic tracking and monitoring applications. Compared to their work, the contribution of this work provides a generic context aware adaptive system that can be customized and utilized in different scenarios including the traffic monitoring system.

In order to use the mobile Web service migration framework introduced in [46] in a traffic jam detection scenario, we customized and extended the ontology provided in [45] with new traffic jam domain-specific classes to describe system components models, properties and related criteria governing the AHP-based decision-making algorithm proposed to select the best migration to perform. On the other hand, we improved the decision-making algorithm with new weighting approach during the decision-making process. For example, when weighting the speed properties based on the speed criterion, a car with speed closer (both higher or lower) to the source car speed should have higher weight and priority to be selected from between all other possible destination cars.

7.1.2 System Description

The car A, can plan its route from point X to point Y and investigate a traffic jam possibility in this route. The traffic jam investigation is performed by migrating *TrafficJamSearch* service of car A and executing it on another car B (i.e., a new service provider), for instance the car that is located in the area of interest defined by car A for planning the route.

In this example, there are two criteria governing the service migration making process:

1. difference between speeds of car A and a possible migration destination car B, and
2. the possible destination car B from the area of interest center of car A.

The first criterion will cause the migration to destination car B with a speed closest to the speed of Car A. While the second criterion will make the selection of service *TrafficJamSearch* migration to the car closest to the center of the area of interest of car A.

When a car B is chosen as a new destination to service *TrafficJamSearch*, the migration controller on Car A starts the physical migration process to Car B. Then, A will execute a search process on B by calling *TrafficJamSearch* to discover the number of cars located in the area of interest of A in order to detect a traffic jam on its planned route.

7.1.3 System Components Context Representation

In order to demonstrate the traffic jam scenario, we added to the migration ontology introduced in Section 5.4 the following semantic component's property classes: 1) *Speed*, 2) *CenterDistance*, and 3) *ServicePriority*. Two criteria are considered in this example, 1) *SpeedCriterion*, and 2) *CenterDistanceCriterion*. The *SpeedCriterion* is the criterion that values the migration to the car that has the closest speed of the service current service provider. While *CenterDistanceCriterion* values the migration to the car that is closest to the center of the area of interest identified by the source car. The traffic jam context ontology-based model is demonstrated in Figure 7.1.

Through the continuous context-awareness monitoring of system context model, when a violation of the rules is sensed, the system launches an adaptation process to migrate a service to a new service provider. In the example scenario, the violation is caused by the traffic information service absence that rises the need to migrate a *TrafficJamSearch* service to a neighboring car in order to perform a car search service and predict traffic jams in an area of interest.

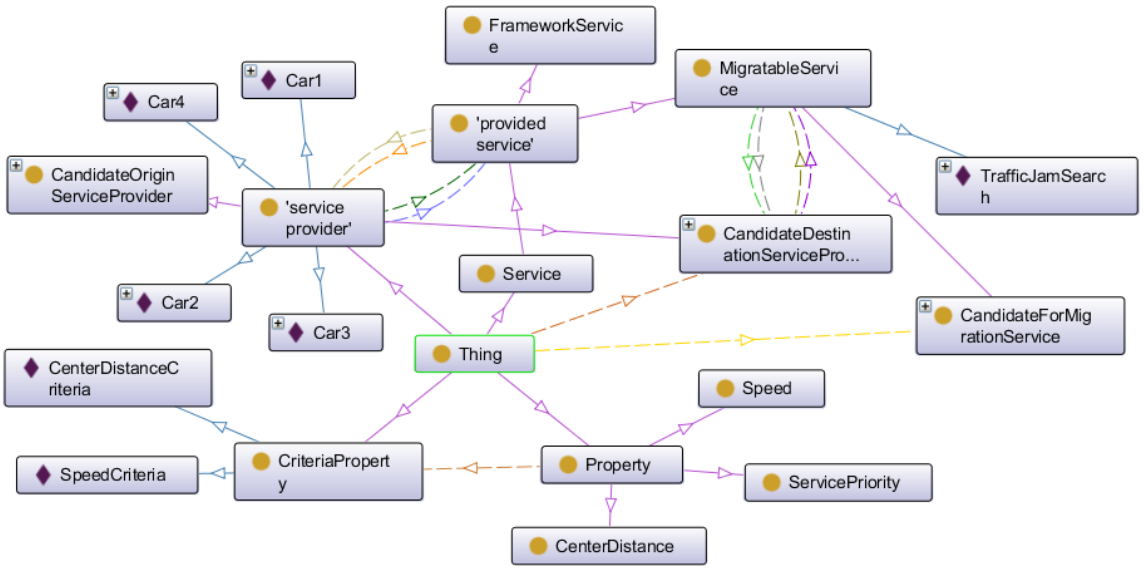

Figure 7.1: Traffic jam detection system ontology-based model representation.

7.1.4 Framework Implementation Description

The framework is divided into two main parts, the framework Web service and service migration Android application. The framework Web service is a Restful-based web service implemented using the Restlet APIs. This service is responsible for publishing the car/service provider instance in the ad hoc network in order to be discovered by other cars. The discovery process is enabled by creating and starting JMEDS device instance on the mobile

service provider by calling the start restful post of the *Frameworkservice*. An example URI request to perform a discovery process on destination car is:

http://{IP}:{Port}/FrameworkService/discover/centerLng/{centerLng}/centerLat/{centerLat}

where the *centerLng* and *centerLat* are the center coordinates of the source car's area of interest. The service provider has the functionalities to retrieve car's context model, GPS location and speed (provided by the google services on Android mobile phone), and the position of its area of interest.

The service migration Android application is responsible for the following tasks:

- discovery process of cars located in a defined area of interest.
- generating system context model by adding the context models of discovered cars.
- system context model reasoning and migration suggestion discovery process.
- migration decision making process.
- publishing service *TrafficJamSearch* migration onto the selected destination car.
- calling *TrafficJamSearch* service to retrieve the information about the traffic jam in the defined area.

The context model of a car is retrieved in JSON format by calling the *getProviderContext* method of its framework service. Based on its IP a car is identified in the network, and its context model can be retrieved as explained in Section 6.2.2. When car A launches the traffic jam detection in a certain area on its route, it starts searching for a car in that area to host its *TrafficJamSearch* service. When a car X is discovered, the source car A's framework application requests the location and speed information to determine whether or not the discovered car exists in its area of interest based on the distance between the discovered car location and the area of interest center. Only cars located in that area will be considered in the migration process so that the framework will request its context model to include their properties and preferences in the model reasoning process. An example of a car preference is defined to limit hosting *TrafficJamSearch* service to only services provided by cars that have a specific manufacturer.

The framework application calculates the distance of each discovered car from the center of the area of interest of car A and adds it to the system model as a CenterDistance property of the related car. The calculation of *CenterDistance* uses the precise location of discovered car and the area of interest center location of the source car. Similarly, the most recent cars speed values are added to the system model as Speed properties. We choose to consider average speed (estimated by the distance traveled during the last 60 s) and precise location values to keep the decision making more reliable and realistic and to avoid service migration failure to a car with an outdated location.

Finally, the decision-making process starts to select the migration with the highest weight, the AHP algorithm weight the migration based on two criteria: (1) *CenterDistanceCriterion*; and (2) *speedCriterion*, so that the migration with the destination car closer to the center of the area of interest will have a higher weight to be chosen by the decision-making process. Similarly, the car with speed equal or around car A's speed will be more highly chosen as destination car. The decision-making process choose the final destination with consideration for the weight of both *CenterDistanceCriterion* and *SpeedCriterion*.

The migration process is provided by *FrameworkService* methods that enable a physical migration of the subject *MigratableService* from a source car to the matching destination car. The *Controller* calls the following URI to perform the migration of *MigratableService*'s implementation from source to destination:

http://{destination.IP}:{destination.Port}/FrameworkService/download/{source.IP}/port/{source.Port}/temp/{source.TempFolder}/service/{MigratableServiceWAR

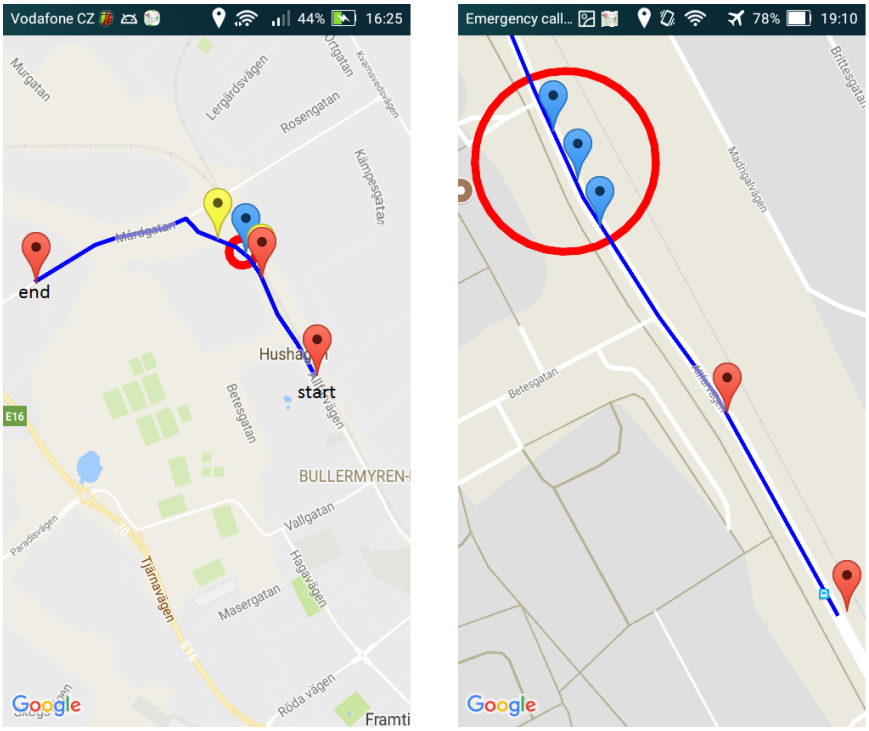

Figure 7.2: Application interface showing the planned route and surrounding cars during the experiment.

7.1.5 Experiment and Results

In this experiment, we use three Genymotion[1] Android emulators and one real mobile phone as service providers. Each mobile phone represents a car on a planned route of a single lane where cars go from point A to point B. Speeds of the emulators are mocked to have random values between 10 to 40 km/h while a fixed speed is set for the source to 20 km/h. To present the possibility of setting rules for migration process, we set a rule for Car3 to not accept services with less priority than 70% while the subject service only has a priority property of 50%.

We initialize the locations of these cars with 100 meters distance consequently. Figure 7.2 demonstrates the interface visualizing the route and the red circle area of interest of the subject car with red color Marker. Each car is presented in blue marker when located inside the area of interest.

On Car1, the framework controller starts searching for other cars located in its area of interest and migrates the search service to be hosted on one of the discovered cars. Later,

[1] https://www.genymotion.com/

Car Name	CenterDistance (m)	Speed (km/h)
Car1	Not applicable	20
Car2	41.64	18
Car3	17.5	30
Car4	12.34	24

Table 7.1: Cars properties during migration example.

Car1 calls the search service to get feedback about the status of traffic in that area. The area of interest, marked as red circle, is set to be 200 meters ahead from Car1 with a diameter of 100 meters.

At first the framework controller on Car1, starts searching for neighboring cars. When a car is found, The Car1 framework requests the *FrameworkService* hosted on the discovered car to get its current location and to check whether the discovered car is located inside Car1's area of interest to consider in the migration process or not. After having a list of discovered cars, Car1 framework starts to create system context model that contains the context model of the discovered cars and its subject *MigratableService*, *TrafficJamService*. In Figure 7.3, the discovered cars are presented with their distances from the center of Car1 area of interest. To demonstrate the decision-making process, we choose the situation when the 3 cars are located inside Car1 area of interest. Table 7.1 contains the *Speed* and *CenterDistance* properties of the cars during the example migration decision.

At this moment, the framework controller suggests migrating the subject service according through two migration plans: 1) *Mig A* to Car2 and 2) *Mig B* to Car4. Even though Car3 is located inside the specified area but it was not to host the service regarding to its preference allowing to host service with a priority higher that 70%. So, this preference rule is not satisfied with Car1's *TrafficJamService*.

The next step for the framework controller is to decide which one of this migration to perform. This is decided based on the AHP process and the defined criteria priorities and the related properties of the migration destinations. For purpose of the experiment, the priority of *SpeedCriterion* and *CenterDistanceCriterion* are set to 9 and 3, respectively.

Based on that, the criteria comparison matrix A is initiated using the AHP-based *InitializeCriteriaMatrix* algorithm, (see Section 4.2). The criteria comparison matrix generated by *InitializeCriteriaMatrix* is listed in Table 7.2. The values show, considering the criteria priorities only, migration to a car with a speed closer to source car speed is 7 times more important than to a car closer to the center of the interest area.

Similarly, the AHP algorithm calculates matrices $n \times n$ of migration weights for each criterion based on the migration properties respecting that criterion. A center distance property is governed by the *centerDistanceCriterion* which has 0 and 50 for the value of the highest and the lowest weight respectively and is statically defined in the system context model. While a speed property is governed by the *speedCriterion* which is dynamically calculated by the algorithm based on the current speeds of cars existing in the area of interest. For each migration the algorithm queries the system context model for the current cars speeds. Later, it evaluates the weights of each car's speed based on the speed of the source car so that the car with the closest speed will have the highest weight values and vice versa. The migration comparison matrices are provided in Table 7.3.

Finally, the AHP algorithm implements Equation 4.2 to computes the composite weight vector $w$so so that the migration with the highest weight will be chosen and executed.

The weights of migrations are noted in Equation 7.1.

$\lambda_{max} = 2, C = 0, CR = 0$			
	centerDistance Criterion	speedCriterion	Priority vector w
centerDistance Criterion	1.0	0.14	0.125
speedCriterion	7.0	1.0	0.875

Table 7.2: Main criteria comparison matrix and its priority vector.

Migration Comparison Matrix with regard to *centerDistanceCriterion*			
$\lambda_{max} = 2, C = 0, CR = 0$			
	Mig A	*Mig B*	Priority vector - $V^{(1)}$
Mig A	1.0	0.2	0.166
Mig B	5.0	1.0	0.833
Migration Comparison Matrix with regard to *speedCriterion*			
$\lambda_{max} = 2, C = 0, CR = 0$			
	Mig A	*Mig B*	Priority vector - $V^{(2)}$
Mig A	1.0	3.0	0.75
Mig B	0.33	1.0	0.25

Table 7.3: Migration comparison matrices and priority vectors.

$$p = \begin{pmatrix} 0.677 \\ 0.323 \end{pmatrix} \tag{7.1}$$

where p_{11} and p_{21} entries represent the weights of *Mig A* and *Mig B* respectively. Based on the highest value of composite weight vector p, *Mig A* will be performed as it has the highest priority (p_{11}= 0.677).

This experiment is repeated 10 times to measure the time spent to perform the migration process and its sub processes (see Figure 7.3). The experiment shows that the average time for the whole migration process is 7.5 s while the average time to create system context is 4.136 s and time to make the decision using the AHP algorithm is 0.251 s. While the authors of [69] used 11 devices in their case study, they set the experiment time-out to 7.5 s, and each device speed is set to a value between 18 to 36 km/h with 5 s intervals. In comparison to that and based on our experiment settings, we see that this time is acceptable as the source car with a speed of 20 km/h will almost cross only 40 meters which leaves it 160 meters away from its area of interest for the routing system to call the *TrafficJamSearch* service and re-plan the route if necessary.

The result shows the validity of the adaptive context-aware service migration approach provided in this case study has been demonstrated through the appropriate adaptation time to perform service migration. The result provides a proof-of-concept of the thesis approach utilized to solve traffic information service absence problem in a real-time application through a seamless decision-making process.

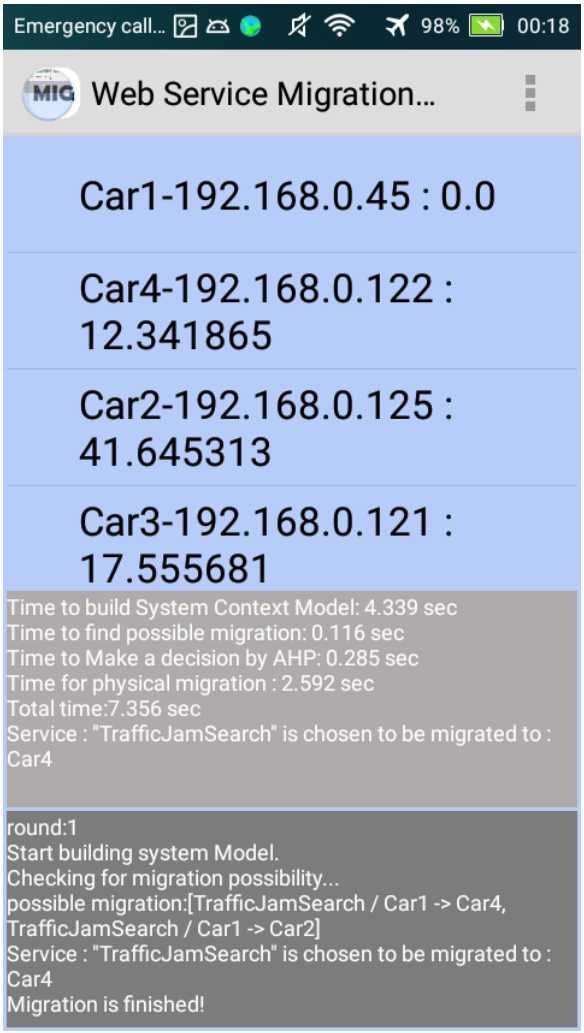

Figure 7.3: Mobile Web service migration framework application.

7.1.6 Conclusion

We presented a case-study to show the applicability of our framework by performing service migration between cooperative cars. Through this case study we present the functionality and usability of the context-aware self-adaptive Web service migration approach proposed in this thesis and the possibility to be customized for different real-life scenarios. The experiment results show the efficiency (see Section 1.2) of the proposed decision-making process demonstrated by the time required to make the decision compared to the total time of the migration process. It is dependent on the static time of service discovery and the physical migration of service deployable package. Moreover, the results show that the framework's performance is seamless and suitable for real-time implementations.

7.2 Case Study 2 - Tourist Video Streaming Mobile Service Migration

In this section, we present a case study of service migration between mobile devices to demonstrate the validity, applicability and efficiency of the self-adaptive migration-based service-oriented architecture approach. We perform experiments to demonstrate the improvement on the QoS gained by the migration based on decentralized adaptation mechanism. Also, we provide a performance analysis to show the light-weight impact of migration adaptation process and framework on the resources of mobile devices as example of resource restricted service providers.

7.2.1 System Description

The case study scenario is based on real life tour programme where tourists are subscribing to a travel company service hosted on a mobile phone of a tourist guide. The service provides information about the scheduled tour, information about the sightseeing located around the user position, and also video editing service that allows passengers to edit their captured videos and publish them on company social webpage. The migration adaptation is performed to migrate company service between different mobile devices based on the resource status of its current hosting device. For example, when the tour guide mobile device has a low memory situation, the framework performs migration adaptation to guarantee specific level of service quality and performance by migrating the service onto service provider with high memory resources. We assume for this scenario the later service provider is located on the travel company bus. Another migration example can happen when passengers leave the bus to a ferry as the service can follow the user by migrating to a company server located on the ferry. Similarly, the service can migrate back to the bus after the ferry journey finished and passengers return on-board the bus.

7.2.2 System Settings

In this section, we describe a test-bed system designed to represent the case study scenario. From an objective perspective, several system's metrics such as (*Service Response Time, CPUUsage and BatteryLevel)* are measured duringa system's resource-stress test. Based on the performed measurements, we provide an analysis of the system performance during the test to show the applicability of the proposed adaptation approach utilized to solve the case study problem.

Two devices are used as service providers in this experiment. Both are Huawei Y560-L01 mobile phones namely: Destination and Source, with 1.1GHz CPU frequency, 1GB RAM and running Android 5.5.1 APIs. The status properties of both providers are listed in Figure 7.4 including their preference rules.

Destination Provider has *CPUUsage* property of 40% and a preference rule defining the least *ServicePriority* of services that can be hosted on *Destination* by 50%. Source provider has 85% of *CPUUsage* and no preference rules.

The subject Web service S is a video transcoding service which converts AVI video files into FLV format. S has *ServicePriority* property of value 50% and has one preference rule that allows it to be migrated only to service provider with $CPUUsage < 45\%$. *ServicePriority* and *CPUUsage* both are sub properties of the Property Class in the context model. *ServicePriority*'s states the priority of the service by a value of [0, 100] while *CPUUsage*

Provider	CPUUsage	Preference Rule
Source	85%	NA
Destination	40%	ServicePriority>=50%

Table 7.4: Values of the Status Properties and Preferences of the Mobile Service Providers.


```
{
“name„:“S„,
“type„:“MigratableService„,
“properties„:
{ “propertyName„:“ServicePriority„,
“propertyValue„:“50„,
“propertyType„:“INT„,
“criteria„:“ServicePriorityCriterion„
},
“rules„:“[SPreference: (?service rdf:type core:MigratableService),
(?origin rdf:type core:CandidateOriginServiceProvider),
(?destination rdf:type core:CandidateDestinationServiceProvider),
(?origin core:provides core:S),
(?destination core:hasProperty ?property),
(?property rdf:type core:CPUUsage),
(?property core:propertyValue ?v1),
le(?v1, „45“^^http://www.w3.org/2001/XMLSchema#int) ->
(core:S core:possibleDestinationProvider ?destination)]„
}
```


Figure 7.4: The Context Model of Service S Presented in JSON Format.

is a service provider property stating the percentage of the device processor in use. The context model of Service S is provided in Figure 7.4.

We install the implemented Web service migration framework on both experimental devices. In this experiment we assume that the migration framework will perform continuous monitoring real-time system context model stating *CPUUsage* of both devices. A reasoning process will be performed on system context model based on the defined preference rules of device *Destination* and Service (*S*). Through this reasoning process, the framework will discover rule violations, suggest and perform service migration decision from its current service provider to another device that satisfies the preference rules of *Destination* and *S*.

7.2.3 Experiment Description

First, we publish Service (*S)* on *Source* mobile service provider and call *S* to perform a transcoding of an AVI sample video file of 17.1 MB size. We measured the response times of service S for 10 times during the test in order to compare the result with the response times of the service after the migration. The time measurement was done using the Advanced REST Client API testing tool[2]. Later, we connect the second mobile service provider *Destination* and run the migration framework application on both mobile phones.

[2] https://advancedrestclient.com/

In this experiment, the framework controller discovers the violations in S preference rule and tries to find an alternative service provider. The framework controller decides to migrate Service S onto*Destination* based on the reasoning process that guarantees the defined preference rules. The suggested migration of S is suggested to *Destination* based on its *CPUUsage* status is only 40% which satisfies S preference rule. On the other hand, the migration of Service S (that has *ServicePriority* of 50%) onto *Destination* does not violate but satisfies *Destination* preference that permits only services with *ServicePriority* ≥ 50 to be migrated to *Destination*.

Similarly, measurements of response times of Service S are made to convert the same AVI file while S is hosted on Destination. The measurements show that the average response time of service S is 48.6 s when hosted on Destination while it is around 134.4 s when hosted on Source.

The results show that by the migration of Service S from Source to be hosted on Destination that has CPU Usage less than 50%, the proposed migration framework provides the mechanism that achieves improvements on Service S's *QoS* measured by its response time. We configured the experiment to be repeatedly executed by the framework Controller on *Destination* for an hour in order to investigate the migration process time and the impact of running the implemented framework on device resources.

7.2.4 Experiment Results

- Migration Process Time: During this test, the controller performs the migration process 634 times. We observed that the average time to perform the migration process from Source to Destination is 4.836 s. By excluding the time to download and deploy Service S WAR file of 2.30 MB on Destination, the time spent to take the migration decision is 0.576, 0.449, and 1.309 s at its Average, Minimum and Maximum values, respectively. Based on these measurements, we see that the proposed framework enables a seamless adaptation in SOA to redistribute system components.

- CPU Usage Consumption: We collected the CPU usage samples consumed by the framework application. Figure 7.5 presents the percentage of time for the CPU usage of the framework during this experiment. The measurements show that the framework total CPU usage is 23% in average (18% in User mode and 5% in Kernel mode), while it is 8% and 50% at its minimum and maximum values respectively.

- Battery Consumption: To investigate battery consumption by the framework, we collected Battery level drop when the migration framework is used and compare it with battery level drop when the framework is not running. The result is presented in Figure 7.6. The battery level drop is by 4% higher when the framework is disabled (OFF).

7.2.5 Conclusion

Through this case-study, we demonstrate our Restful-based Web service migration framework for dynamic relocation and provisioning of Web services on mobile devices.

The experimental results show the lightweight impact of the implemented Web services migration framework on mobile device resources. Moreover, the results demonstrate the efficiency of utilizing the proposed migration approach to assure services' and devices'

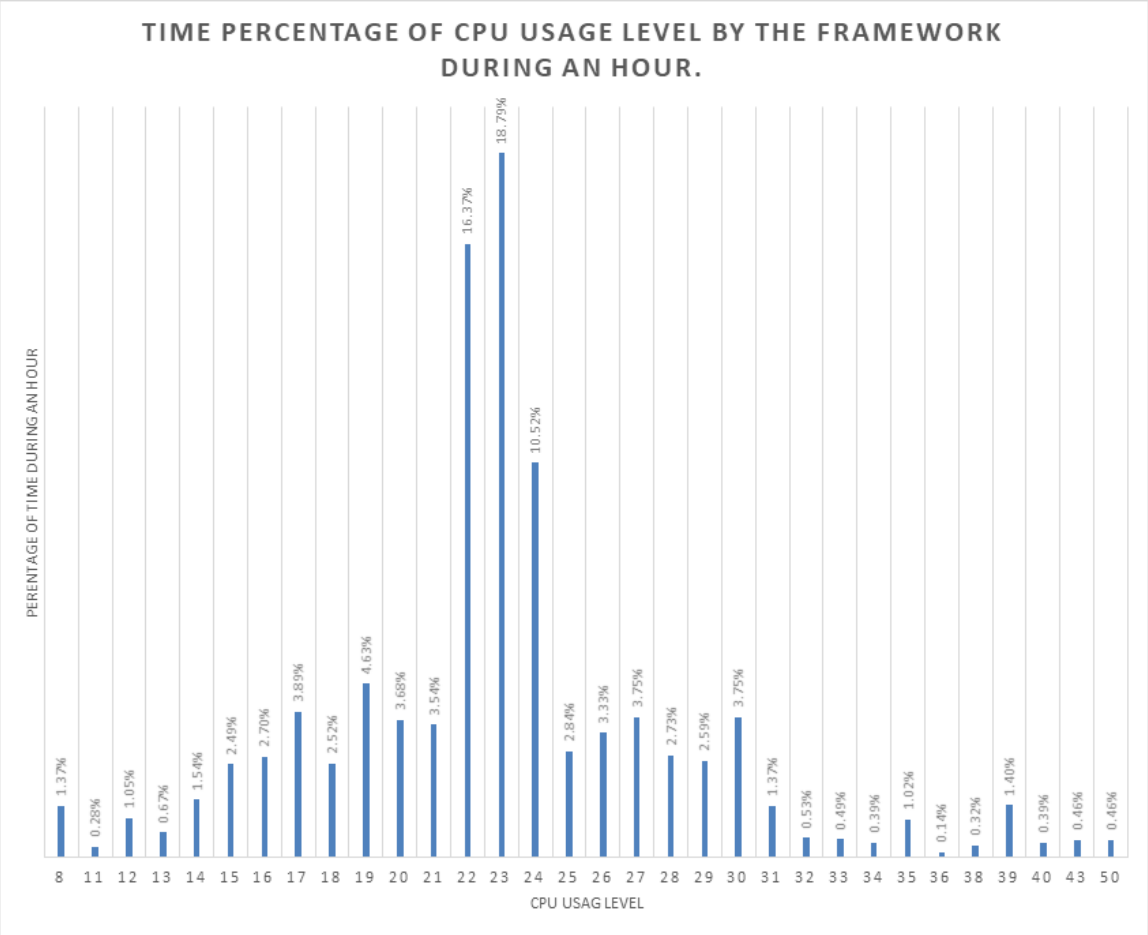

Figure 7.5: Framework Total CPU Usage by its Average Lasting Time During an Hour.

preferences and improve *QoS* in SOA by enabling self-adaptation on mobile devices in P2P network.

With regards to the aim of the thesis described in Section 1.3, the experiments show the applicability of our approach for decentralized context-aware self-adaptation in mobile hosted service-oriented architecture. The proposed migration adaptation is proposed to automatically satisfy operation conditions of system components that can be semantically described in their context models. It enables software architectures to automatically react to changes in their component status based on a designed adaptation strategy and to remove unexpected violations.

7.3 Evaluation and Conclusion

In the previous sections, we provided two case studies that use our context-aware self-adaptation architecture model to support a decentralized adaptation in SOA-based systems through service migration as example of adaptation strategy. The introduced case studies have demonstrated the application of the adaptive context-aware service-oriented architecture model, the formal component context model, and the behavioural description of the

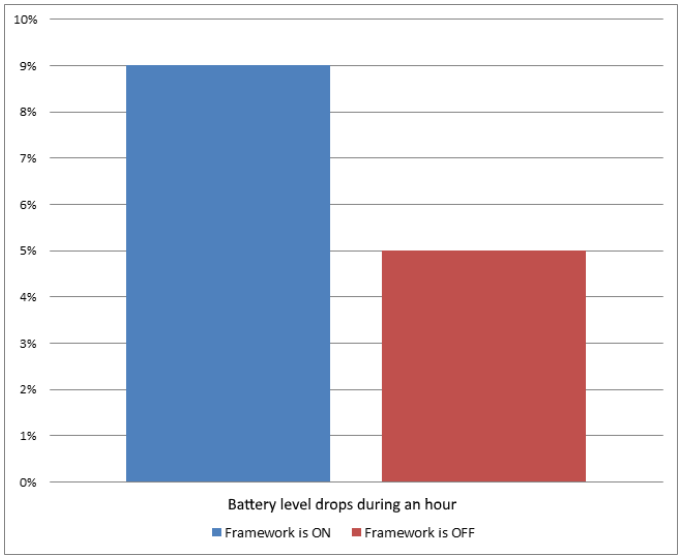

Figure 7.6: Battery Level Drop during the Experiments.

adaptation and decision-making processes in SOA, that has been proposed in Chapters 3 and 4.

In this section we describe the contributions that have been provided through this work to solve the limitations of current approaches (see Section 2.2.5) and to support adaptation in software systems. The contributions of our approach are listed as follows:

1. A SOA-based architecture meta-model has been provided in Chapter 3 to support system adaptation and context-awareness in stationary and mobile architectures.

2. To solve limitation L1, the limitation of reusability and extensibility in the system, we adopt the SOA principles of service reusability and extensibility in designing our architecture model. As presented in our case studies (see Chapter 7), service reusability has been a key factor to support customization of our architecture model to support different adaptation plan.

3. To solve limitation L2, the limitation in context information modelling in the system, an ontology has been provided (see Section 3.4 and Section 3.5) to guarantee a dynamic and sharable understanding of context information between system components, which also supports system extensibility over different domains. Moreover, a generic ontology-based component context model has been introduced to describe system components semantically and to support content information modelling. The provided component context model supports the usage of the OWL-S description of Web service as discussed in Section 3.3. It enables the utilization of system adaptation and context awareness by providing a method to semantically describe system services and devices including their properties and preferences to be used in planning the adaptation.

4. To overcome limitation L3, the limitation in adaptation strategies, a customizable adaptation approach has been provided in Chapter 3 to support different adaptation plans by extending the system ontology core model and integrating rules stating the adaptation plans. In our case studies (see Chapter 7), we customized our adaptation

model to support service migration as example of adaptation plan and demonstrated its application through two real-life scenarios.

5. To overcome limitation L4, the limitation in the adaptation making process, a dynamic multi-criteria decision-making process has been provided to choose the best adaptation to perform from the possible adaptation plans (see Section 4.1 and Section 4.2). The provided decision-making process is extensible so that the newly ontology-defined terms, metrics, properties and criteria that can be dynamically considered in the adaptation decision making process.

Part IV

Future Work and Conclusion

Chapter 8

Future Work

The thesis demonstrates a decentralized context-aware self-adaptive service-oriented architecture model proposed to cope with changes of software system environment and components context and react through an adaptation strategy designed to guarantee system availability and improve its performance. During the research to reach the thesis's goals, few points have been shifted for future work as improvements and extensions that can be considered as new research openings of this work.

- Service discovery: In Section 5.5.1, we introduced the abstract Discovery Module proposed to discover existing devices and services in the network that can join and contribute in the adaptation process. In the stage of implementing this Module, we used the SOAP messaging exchange discovery framework JMEDS to handle the discovery process in ad-hoc networks. Despite the fact that JMEDS has proven its feasibility in both standalone and mobile platforms in small-scale ad-hoc networks, we think that the discovery methodology for medium and large-scale networks should be improved by utilizing the UDDI registry for Web service with a light-weight message exchange payload such as REST framework due to the significant increment in the number of participated services.

- Service migration: In Chapter 5 (Section 5.2), Service Migration is introduced through packing and moving service internal running status with the service and resume its execution on its new service provider. The concept of saving service status can leverage the seamless migration of the service. In the conducted experiments in Chapter 7, service internal status has not been migrated with the service to leverage the performance of the framework. However, we think that it is required to conduct further investigations to consider the migration of the internal status based on the application scenario and optimization of service/system performance.

- To consider other adaptation scenarios that involve large number of services and devices. As discussed in Chapter 7, the case-studies are demonstrated as a proof-of-concept for the thesis approach and to provide a real-life scenario of possible implementation. Moreover, the general adaptive approach introduced in Chapter 3 can be customized to be used for other adaptation scenarios than service migration. Such a promising adoption and implementation of this research contribution is to include for goal-oriented cooperative robots' scenario.

Chapter 9

Conclusion

The research work of the thesis has been conducted based on the objectives presented in Section 1.4. The main objective of the thesis is reached by the proposal of a decentralized context-aware self-adaptive service-oriented architecture approach.

Following the objective O1, we proposed a formal representation of system context using ontology. The proposed ontology enables a common understanding of the semantic meaning of system components and allows to define system components models and realizes changes in these models. A context model states the description of a system component including its attributes and operation conditions to consider in system adaptation.

With regard to objective O2, we introduced a decentralized context-aware self-adaptive architecture model that enables SOA-based software system to react to context changes in the surrounding environment and/or the status of system components. We presented a formal description of service migration strategy to demonstrate the behavioural description of service migration-based adaptive SOA.

To reach objective O3, we developed a framework that supports service migration adaptation and allows stationary and mobile devices as service providers to join the adaptation process. The implanted framework enables system component context modelling, context exchange, context reasoning, decision making, service migration adaptation, service provisioning on both Android and Windows platforms.

Finally, regarding to objective O4, we presented two case studies to demonstrate the applicability and efficiency of the thesis approach. The experiment results are provided as proof-of-concept to show the validity of the decentralized context-aware adaptive architecture model. The results provide measurements to presents the efficiency of the proposed approach through showing improvements in system performance and quality of service factor by considering the thesis approach.

In comparison with the related approaches, our approach utilizes context-awareness and self-adaptivity to guarantee and improve system performance. Using the proposed context-aware self-adaptive meta-model allows to provide semantical description for both Web services and service provides. Moreover, the thesis promotes the extension of service-oriented architecture on mobile devices and enables the participation of type different types of context resources and devices independently from their computing platform.

For future work, we consider empowering the proposed approach with other discovery methodologies of service and service provider in large-scale network. Moreover, we intend to extend the implementation work to join new types of devices and services. Another part of the future work is to test the approach through other real-life examples with different adaptation scenarios.

Bibliography

[1] Abeywickrama, D. B.; Zambonelli, F.: Model checking goal-oriented requirements for self-adaptive systems. In *2012 IEEE 19th International Conference and Workshops on Engineering of Computer-Based Systems*. IEEE. 2012. pp. 33–42.

[2] Abowd, G. D.; Dey, A. K.; Brown, P. J.; et al.: Towards a better understanding of context and context-awareness. In *International symposium on handheld and ubiquitous computing*. Springer. 1999. pp. 304–307.

[3] Al-Azab, F. G. M.; Ayu, M. A.: Web based multi criteria decision making using AHP method. In *Proceeding of the 3rd International Conference on Information and Communication Technology for the Moslem World (ICT4M) 2010*. IEEE. 2010. pp. A6–A12.

[4] AlShahwan, F.; Carrez, F.; Moessner, K.: Providing and evaluating the mobile web service distribution mechanisms using fuzzy logic. *Journal of Software*. vol. 7, no. 7. 2012: pp. 1473–1487.

[5] Android, C.: Open Source Android Apps for Developers: I-Jetty (webserver for the android mobile platform).

[6] Bandara, A.; Payne, T. R.; de Roure, D.; et al.: An ontological framework for semantic description of devices. 2004.

[7] Banks, T.: Web services resource framework (wsrf)-primer v1. 2. *OASIS committee draft*. 2006: pp. 02–23.

[8] Bellifemine, F.: Jade-a white paper. *exp*. vol. 3, no. 3. 2003.

[9] Berners-Lee, T.; Hendler, J.; Lassila, O.; et al.: The semantic web. *Scientific american*. vol. 284, no. 5. 2001: pp. 28–37.

[10] Bettini, C.; Brdiczka, O.; Henricksen, K.; et al.: A survey of context modelling and reasoning techniques. *Pervasive and Mobile Computing*. vol. 6, no. 2. 2010: pp. 161–180.

[11] Bianchini, D.; De Antonellis, V.; Melchiori, M.; et al.: Lightweight ontology-based service discovery in mobile environments. In *17th International Workshop on Database and Expert Systems Applications (DEXA'06)*. IEEE. 2006. pp. 359–364.

[12] Brown, A. W.: *Large-scale, component-based development*. vol. 1. Prentice Hall PTR Englewood Cliffs. 2000.

[13] Candan, K. S.; Liu, H.; Suvarna, R.: Resource description framework: metadata and its applications. *ACM SIGKDD Explorations Newsletter*. vol. 3, no. 1. 2001: pp. 6–19.

[14] Carroll, J. J.; Dickinson, I.; Dollin, C.; et al.: Jena: implementing the semantic web recommendations. In *Proceedings of the 13th international World Wide Web conference on Alternate track papers & posters*. ACM. 2004. pp. 74–83.

[15] Chen, B.; Cheng, H. H.; Palen, J.: Mobile-C: a mobile agent platform for mobile C/C++ agents. *Software: Practice and Experience*. vol. 36, no. 15. 2006: pp. 1711–1733.

[16] Chen, Y.; Li, X.; Yi, L.; et al.: A ten-year survey of software architecture. In *2010 IEEE International Conference on Software Engineering and Service Sciences*. IEEE. 2010. pp. 729–733.

[17] Cheng, B.; De Lemos, R.; Giese, H.; et al.: A research roadmap: Software engeering for self-adaptive systems. In *Schloss Dagstuhl Seminar*, vol. 8031. 2009.

[18] Consortium, W. W. W.; et al.: Web Services Architecture, W3C Working Group Note 11 February 2004. *http://www. w3. org/TR/2004/NOTE-ws-arch-20040211/*. 2004.

[19] Dardenne, A.; Van Lamsweerde, A.; Fickas, S.: Goal-directed requirements acquisition. *Science of computer programming*. vol. 20, no. 1-2. 1993: pp. 3–50.

[20] Deng, S.; Huang, L.; Taheri, J.; et al.: Mobility-aware service composition in mobile communities. *IEEE Transactions on Systems, Man, and Cybernetics: Systems*. vol. 47, no. 3. 2016: pp. 555–568.

[21] Dustdar, S.; Schreiner, W.: A survey on web services composition. *International journal of web and grid services*. vol. 1, no. 1. 2005: pp. 1–30.

[22] Edwards, G.; Garcia, J.; Tajalli, H.; et al.: Architecture-driven self-adaptation and self-management in robotics systems. In *2009 ICSE Workshop on Software Engineering for Adaptive and Self-Managing Systems*. IEEE. 2009. pp. 142–151.

[23] Ejigu, D.; Scuturici, M.; Brunie, L.: An ontology-based approach to context modeling and reasoning in pervasive computing. In *Fifth Annual IEEE International Conference on Pervasive Computing and Communications Workshops (PerComW'07)*. IEEE. 2007. pp. 14–19.

[24] Elkhodary, A.; Esfahani, N.; Malek, S.: FUSION: a framework for engineering self-tuning self-adaptive software systems. In *Proceedings of the eighteenth ACM SIGSOFT international symposium on Foundations of software engineering*. ACM. 2010. pp. 7–16.

[25] Erl, T.: *Service-oriented architecture: concepts, technology, and design*. Pearson Education India. 1900.

[26] Erl, T.: A look ahead to the service-oriented world: Defining SOA when there's no single, official definition. 2005.

[27] Famaey, J.; Wauters, T.; De Turck, F.; et al.: Network-aware service placement and selection algorithms on large-scale overlay networks. *Computer Communications*. vol. 34, no. 15. 2011: pp. 1777–1787.

[28] Fielding, R. T.; Taylor, R. N.: *Architectural styles and the design of network-based software architectures*. vol. 7. University of California, Irvine Doctoral dissertation. 2000.

[29] Fox, M.; Long, D.: PDDL2. 1: An extension to PDDL for expressing temporal planning domains. *Journal of artificial intelligence research*. vol. 20. 2003: pp. 61–124.

[30] Garlan, D.; Cheng, S.-W.; Huang, A.-C.; et al.: Rainbow: Architecture-based self-adaptation with reusable infrastructure. *Computer*. vol. 37, no. 10. 2004: pp. 46–54.

[31] Group, T. O.: *Soa source book*. Van Haren Publishing. 2009.

[32] Gu, T.; Pung, H. K.; Zhang, D. Q.: A service-oriented middleware for building context-aware services. *Journal of Network and computer applications*. vol. 28, no. 1. 2005: pp. 1–18.

[33] Hao, W.; Gao, T.; Yen, I.-L.; et al.: An infrastructure for web services migration for real-time applications. In *2006 Second IEEE International Symposium on Service-Oriented System Engineering (SOSE'06)*. IEEE. 2006. pp. 41–48.

[34] Hao, W.; Yen, I.-L.; Thuraisingham, B.: Dynamic service and data migration in the clouds. In *2009 33rd Annual IEEE International Computer Software and Applications Conference*, vol. 2. IEEE. 2009. pp. 134–139.

[35] Henricksen, K.; Indulska, J.; Rakotonirainy, A.: Modeling context information in pervasive computing systems. In *International Conference on Pervasive Computing*. Springer. 2002. pp. 167–180.

[36] Ho, W.; Xu, X.; Dey, P. K.: Multi-criteria decision making approaches for supplier evaluation and selection: A literature review. *European Journal of operational research*. vol. 202, no. 1. 2010: pp. 16–24.

[37] Hoare, C. A. R.: Communicating sequential processes. In *The origin of concurrent programming*. Springer. 1978. pp. 413–443.

[38] Hu, X.; Li, X.; Ngai, E. C.-H.; et al.: Multidimensional context-aware social network architecture for mobile crowdsensing. *IEEE Communications Magazine*. vol. 52, no. 6. 2014: pp. 78–87.

[39] Hussein, M.; Han, J.; Colman, A.; et al.: An architecture-based approach to developing context-aware adaptive systems. In *2012 IEEE 19th International Conference and Workshops on Engineering of Computer-Based Systems*. IEEE. 2012. pp. 154–163.

[40] Hussein, M.; Han, J.; Colman, A.; et al.: An architecture-based approach to developing context-aware adaptive systems. In *2012 IEEE 19th International Conference and Workshops on Engineering of Computer-Based Systems*. IEEE. 2012. pp. 154–163.

[41] Kazzaz, M.; Rychly, M.: Ontology-based context modelling and reasoning in the Web service migration framework. *Acta Electrotechnica et Informatica*. vol. 13, no. 4. 2013: pp. 5–12.

[42] Kazzaz, M. M.: Semantic Services Migration. In *Proceedings of the 18th Conference STUDENT EEICT 2012 Volume*. Brno University of Technology. 2012. pp. 386–390.

[43] Kazzaz, M. M.; Rychlỳ, M.: A web service migration framework. In *ICIW 2013, The Eighth International Conference on Internet*. The International Academy, Research and Industry Association. 2013. pp. 58–62.

[44] Kazzaz, M. M.; Rychlỳ, M.: Web service migration with migration decisions based on ontology reasoning. In *Proceedings of the Twelfth International Conference on Informatics-Informatics*. 2013. pp. 186–191.

[45] Kazzaz, M. M.; Rychlỳ, M.: Web service migration using the analytic hierarchy process. In *2015 IEEE International Conference on Mobile Services*. IEEE. 2015. pp. 423–430.

[46] Kazzaz, M. M.; Rychlỳ, M.: Restful-based mobile Web service migration framework. In *2017 IEEE International Conference on AI & Mobile Services (AIMS)*. IEEE. 2017. pp. 70–75.

[47] Kazzaz, M. M.; Rychlỳ, M.: A Case Study: Mobile Service Migration Based Traffic Jam Detection. *International Journal of Systems and Service-Oriented Engineering (IJSSOE)*. vol. 8, no. 1. 2018: pp. 44–57.

[48] Kephart, J. O.; Chess, D. M.: The vision of autonomic computing. *Computer*. , no. 1. 2003: pp. 41–50.

[49] Kim, Y.-S.; Lee, K.-H.: An efficient policy establishment scheme for web services migration. In *2007 International Conference on Convergence Information Technology (ICCIT 2007)*. IEEE. 2007. pp. 595–600.

[50] Kim, Y.-S.; Lee, K.-H.: A lightweight framework for mobile web services. *Computer Science-Research and Development*. vol. 24, no. 4. 2009: page 199.

[51] Land, R.: A brief survey of software architecture. *Mälardalen Real-Time Research Center (MRTC) Report*. 2002.

[52] Lange, D. B.; Mitsuru, O.: *Programming and Deploying Java Mobile Agents Aglets*. Addison-Wesley Longman Publishing Co., Inc.. 1998.

[53] Lassila, O.; Swick, R. R.; et al.: Resource description framework (RDF) model and syntax specification. 1998.

[54] Lee, K.-C.; Kim, J.-H.; Lee, J.-H.; et al.: Implementation of ontology based context-awareness framework for ubiquitous environment. In *2007 International Conference on Multimedia and Ubiquitous Engineering (MUE'07)*. IEEE. 2007. pp. 278–282.

[55] Louvel, J.; Templier, T.; Boileau, T.: *Restlet in action: developing restful web apis in Java*. Manning Publications Co.. 2012.

[56] MacKenzie, C. M.; Laskey, K.; McCabe, F.; et al.: Reference model for service oriented architecture 1.0. *OASIS standard*. vol. 12. 2006: page 18.

[57] Maedche, A.; Staab, S.: Ontology learning for the semantic web. *IEEE Intelligent systems*. vol. 16, no. 2. 2001: pp. 72–79.

[58] Manna, Z.; Pnueli, A.: The temporal logic of reactive systems: Specification. 1992.

[59] Martin, D.; Burstein, M.; Hobbs, J.; et al.: OWL-S: Semantic markup for web services W3C Member Submission 22 November 2004. *W3C Member Submission, from http://www. w3. org/Submission/2004/SUBM-OWL-S-20041122*. 2004.

[60] Meehean, J.; Livny, M.: A service migration case study: Migrating the Condor schedd. In *Midwest Instruction and Computing Symposium*. 2005.

[61] Messig, M.; Goscinski, A.: Service migration in autonomic service oriented grids. In *Proceedings of the sixth Australasian workshop on Grid computing and e-research-Volume 82*. Australian Computer Society, Inc.. 2008. pp. 45–54.

[62] Morandini, M.; Penserini, L.; Perini, A.: Modelling self-adaptivity: a goal-oriented approach. In *2008 Second IEEE International Conference on Self-Adaptive and Self-Organizing Systems*. IEEE. 2008. pp. 469–470.

[63] Nadareishvili, I.; Mitra, R.; McLarty, M.; et al.: *Microservice architecture: aligning principles, practices, and culture*. " O'Reilly Media, Inc.„. 2016.

[64] O'Brien, P. D.; Nicol, R. C.: FIPA—towards a standard for software agents. *BT Technology Journal*. vol. 16, no. 3. 1998: pp. 51–59.

[65] Oreizy, P.; Gorlick, M. M.; Taylor, R. N.; et al.: An architecture-based approach to self-adaptive software. *IEEE Intelligent Systems and Their Applications*. vol. 14, no. 3. 1999: pp. 54–62.

[66] Pauty, J.; Preuveneers, D.; Rigole, P.; et al.: Research challenges in mobile and context-aware service development. In *Future Research Challenges for Software and Services Conference*. Citeseer. 2006. pp. 141–148.

[67] Perry, D. E.; Wolf, A. L.: Foundations for the study of software architecture. *ACM SIGSOFT Software engineering notes*. vol. 17, no. 4. 1992: pp. 40–52.

[68] Reich, C.; Bubendorfer, K.; Banholzer, M.; et al.: A SLA-oriented management of containers for hosting stateful web services. In *Third IEEE International Conference on e-Science and Grid Computing (e-Science 2007)*. IEEE. 2007. pp. 85–92.

[69] Riva, O.; Nadeem, T.; Borcea, C.; et al.: Context-aware migratory services in ad hoc networks. *IEEE Transactions on Mobile Computing*. vol. 6, no. 12. 2007: pp. 1313–1328.

[70] Rychly, M.: Dynamically Reconfigurable Runtime Architectures: Challenges and Service-driven Approaches.

[71] Saaty, T. L.: *Multicriteria decision making: the analytic hierarchy process: planning, priority setting resource allocation*. 1990.

[72] Saaty, T. L.: *Analytic network process*. Springer. 2013.

[73] Salehie, M.; Tahvildari, L.: Self-adaptive software: Landscape and research challenges. *ACM transactions on autonomous and adaptive systems (TAAS)*. vol. 4, no. 2. 2009: page 14.

[74] Santa, J.; Pereñíguez, F.; Moragón, A.; et al.: Experimental evaluation of CAM and DENM messaging services in vehicular communications. *Transportation Research Part C: Emerging Technologies*. vol. 46. 2014: pp. 98–120.

[75] Schilit, B. N.; Theimer, M. M.: Disseminating Active Mop Infonncition to Mobile Hosts. *IEEE network*. 1994.

[76] Schmidt, C.: Context-aware computing. *Berlin Institute of Technology*. 2011.

[77] Schmidt, H.; Kapitza, R.; Hauck, F. J.; et al.: Adaptive web service migration. In *IFIP International Conference on Distributed Applications and Interoperable Systems*. Springer. 2008. pp. 182–195.

[78] Shadbolt, N.; Berners-Lee, T.; Hall, W.: The semantic web revisited. *IEEE intelligent systems*. vol. 21, no. 3. 2006: pp. 96–101.

[79] Shaw, M.; Garlan, D.; et al.: *Software architecture*. vol. 101. prentice Hall Englewood Cliffs. 1996.

[80] Sheng, Q. Z.; Benatallah, B.: ContextUML: a UML-based modeling language for model-driven development of context-aware web services. In *International Conference on Mobile Business (ICMB'05)*. IEEE. 2005. pp. 206–212.

[81] Sheng, Q. Z.; Pohlenz, S.; Yu, J.; et al.: ContextServ: A platform for rapid and flexible development of context-aware Web services. In *Proceedings of the 31st International Conference on Software Engineering*. IEEE Computer Society. 2009. pp. 619–622.

[82] da Silva, C. E.; de Lemos, R.: A framework for automatic generation of processes for self-adaptive software systems. *Informatica*. vol. 35, no. 1. 2011.

[83] Stal, M.: Using architectural patterns and blueprints for service-oriented architecture. *IEEE software*. vol. 23, no. 2. 2006: pp. 54–61.

[84] Sun, M.; Zang, T.; Xu, X.; et al.: Consumer-centered cloud services selection using AHP. In *2013 International Conference on Service Sciences (ICSS)*. IEEE. 2013. pp. 1–6.

[85] Tang, S.; Peng, X.; Yu, Y.; et al.: Goal-directed modeling of self-adaptive software architecture. In *Enterprise, Business-Process and Information Systems Modeling*. Springer. 2009. pp. 313–325.

[86] Team, A.; et al.: Androjena-Jena Android port.

[87] Van Lamsweerde, A.: The KAOS project: Knowledge acquisition in automated specification of software. In *Proc. of the AAAI Spring Symposium Series, Design of Composite Systems, 1991*. 1991. pp. 59–62.

[88] Vinayagasundaram, B.; Srivatsa, S.: Implementation of hybrid software architecture for Artificial Intelligence System. *IJCSNS*. vol. 7, no. 1. 2007: page 35.

[89] Wagh, K.; Thool, R.: A comparative study of soap vs rest web services provisioning techniques for mobile host. *Journal of Information Engineering and Applications*. vol. 2, no. 5. 2012: pp. 12–16.

[90] Wagh, K.; Thool, R.: Mobile web service provisioning and performance evaluation of mobile host. *International Journal on Web Service Computing*. vol. 5, no. 2. 2014: page 1.

[91] Wang, X.; Zhang, D.; Gu, T.; et al.: Ontology Based Context Modeling and Reasoning using OWL. In *Percom workshops*, vol. 18. Citeseer. 2004. page 22.

[92] Weyns, D.; Iftikhar, M. U.; Malek, S.; et al.: Claims and supporting evidence for self-adaptive systems: A literature study. In *Proceedings of the 7th International Symposium on Software Engineering for Adaptive and Self-Managing Systems*. IEEE Press. 2012. pp. 89–98.

[93] Weyns, D.; Malek, S.; Andersson, J.: On decentralized self-adaptation: lessons from the trenches and challenges for the future. In *Proceedings of the 2010 ICSE Workshop on Software Engineering for Adaptive and Self-Managing Systems*. ACM. 2010. pp. 84–93.

[94] Wijngaards, N. J.; Overeinder, B. J.; van Steen, M.; et al.: Supporting internet-scale multi-agent systems. *Data & Knowledge Engineering*. vol. 41, no. 2-3. 2002: pp. 229–245.

[95] Wu, Y.; Wu, Y.; Peng, X.; et al.: Implementing self-adaptive software architecture by reflective component model and dynamic AOP: A case study. In *2010 10th International Conference on Quality Software*. IEEE. 2010. pp. 288–293.

[96] Zeeb, E.; Moritz, G.; Timmermann, D.; et al.: Ws4d: Toolkits for networked embedded systems based on the devices profile for web services. In *2010 39th International Conference on Parallel Processing Workshops*. IEEE. 2010. pp. 1–8.

[97] Zeng, L.; Benatallah, B.; Ngu, A. H.; et al.: QoS-aware middleware for web services composition. *IEEE Transactions on software engineering*. vol. 30, no. 5. 2004: pp. 311–327.

[98] Zheng, L.; Wu, S.: An infrastructure for web services migration in clouds. In *2010 International Conference on Computer Application and System Modeling (ICCASM 2010)*, vol. 10. IEEE. 2010. pp. V10–554.

[99] Zuo, Y.: Towards a Logical Framework for Migration-Based Survivability. In *Proceedings of the 7th Annual Symposium on Information Assurance/Secure Knowledge Management*. 2012. pp. 29–33.

Appendix A

Abbreviations

AHP	Analytic hierarchy process
API	Application programming interface
AVI	Audio Video Interleave
CBD	Component-based Development
CPU	Central Processing Unit
CI	Consistency Index
CR	Consistency Ratio
CSP	Communicating Sequential Processes
DPWS	Devices Profile for Web Services
FLV	Flash Video
FSM	Finite State Machine
FUSION	FeatUre-oriented Self-adaptatION
JSON	JavaScript Object Notation
HTTP	Hypertext Transfer Protocol
IP	Internet Protocol
JMEDS	WS4D.org Java Multi Edition DPWS Stack
JSON	JavaScript Object Notation
KAOS	Keep All Objectives Satisfied
LTL	Linear temporal logic
MCA	Multiple-Criteria Analysis
MCDA	Multiple-Criteria Decision Analysis
MCDM	Multiple-Criteria Decision-Making
OWL	The W3C Web Ontology Language
OWL-DL	OWL-Description Logic
OWL-S	The Web Ontology Language for Services
QoS	Quality of Service
RDF	Resource Description Framework
RDFS	Resource Description Framework Schema
REST	Representational state transfer
RFID	Radio-frequency identification
SOA	Service Oriented Architecture
SOAP	Simple Object Access Protocol
SOCAM	Service oriented context-aware middleware architecture
SWRL	Semantic Web Rule Language
UML	Unified Modeling Language

URI	Uniform Resource Identifier
WADL	Web Application Description Language
WAR	Web Application Resource or Web Application ARchive
WS	Web Service
WS4D	Web Services for Devices
WSDL	Web Services Description Language
XML	Extensible Mark-up Language

Appendix B

The Framework Applications

In this thesis, two software applications have been developed to support the implementation of the proposed adaptation approach of provision and migration of mobile services between mobile and stationary devices.

1. Application 1: Service Migration Framework Application for Mobile Devices (Android system).
2. Application 2: Service Migration Adaptation Framework Application for Stationary Devices (Windows system).

Figure B.1 and Figure B.2 demonstrate the GUI of application 1 and application 2, respectively.

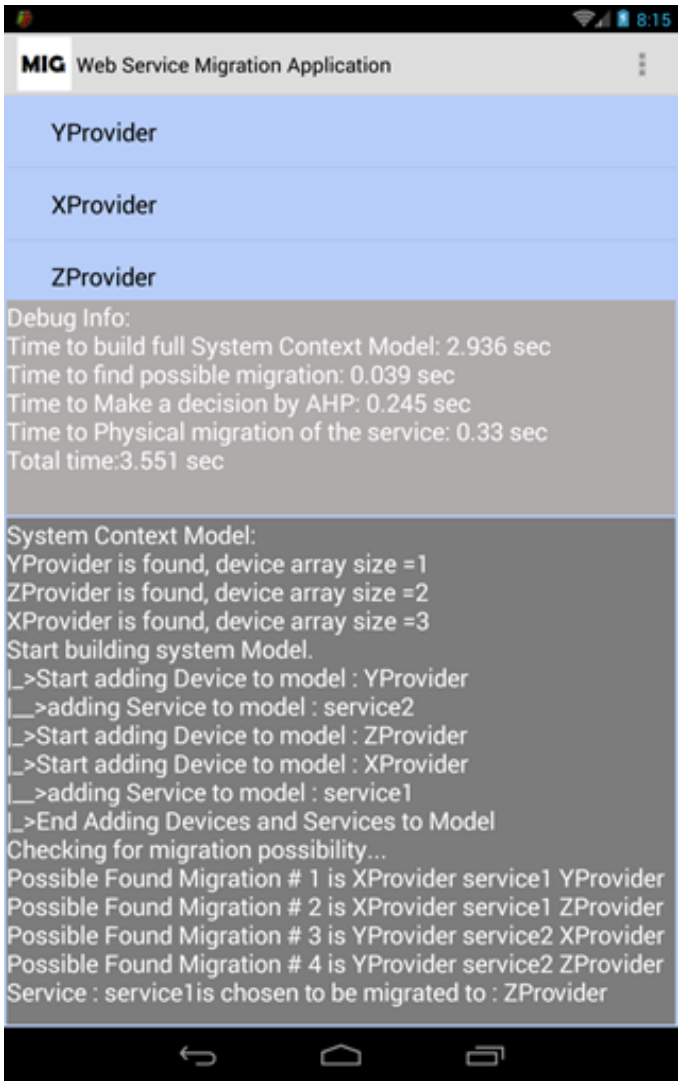

Figure B.1: Application 1 - Service Migration Android Application GUI for Mobile Devices.

Figure B.2: Application 2 - Service Migration Java-based Application GUI for Stationary Devices.

Appendix C

Author's Publications related to the Thesis

This Appendix presents the list of publications that has been contributed by this thesis. All of the publications are included in the Bibliography and referred as [42, 43, 44, 41, 45, 46, 47], respectively.

1. KAZZAZ M. Mohanned. SEMANTIC SERVICES MIGRATION. In: Proceedings of the 18th Conference STUDENT EEICT 2012 Volume 3. Brno: Brno University of Technology, 2012, pp. 386-390. ISBN 978-80-214-4462-1. In this paper we addressed the technical requirements to enable our proposal of service migration SOA. First, we presented the need to utilise a formal architecture description language to provide semantic description of system components stating the status of their properties and operation conditions. Second, we introduced our initial approach of service migration and noted the required operations to enable service migration.

2. KAZZAZ M. Mohanned and RYCHLÝ Marek. A Web Service Migration Framework. In: ICIW 2013, The Eighth International Conference on Internet and Web Applications and Services. Rome: The International Academy, Research and Industry Association, 2013, pp. 58-62. ISBN 978-1-61208-280-6. This paper presented a detailed description of service migration process and addressed the need to utilize a decision-making method to select the best migration decision. A formal modelling of the service migration process is provided in the paper using the linear temporal logic and integrated in the thesis in Section 5.3. Moreover, the paper describes a prototype design of a framework for service migration which represents the blueprints of framework implementation presented in Chapter 6 of this thesis.

3. KAZZAZ M. Mohanned and RYCHLÝ Marek. Web Service Migration with Migration Decisions Based on Ontology Reasoning. In: Proceedings of the Twelfth International Conference on Informatics - Informatics'2013. Košice: Faculty of Electrical Engineering and Informatics, University of Technology Košice, 2013, pp. 186-191. ISBN 978-80-8143-127-2.

4. KAZZAZ M. Mohanned and RYCHLÝ Marek. Ontology-based Context Modelling and Reasoning in the Web Service Migration Framework. Acta Electrotechnica et Informatica. 2013, vol. 13, no. 4, pp. 5-12. ISSN 1335-8243. The research work provided in these two publications demonstrates a customized utilization of the proposed

service-oriented architecture model described in Chapter 5 of the thesis in order to enable service migration in SOA. An extension of the thesis SOA ontology is proposed to describe formal context model of system components of services and service providers and their operation conditions. Parts of this work formulate the content of Section 5.4 of the thesis.

5. KAZZAZ M. Mohanned and RYCHLÝ Marek. Web Service Migration using the Analytic Hierarchy Process. In: 2015 IEEE International Conference on Mobile Services. New York: IEEE Computer Society, 2015, pp. 423-430. ISBN 978-1-4673-7284-8. In this paper, we present a framework for dynamic Web service migration in Service-oriented Architecture (SOA). It allows an automatic discovery of system components (i.e., service providers and Web services) and creates a full system context model by aggregating the context model of the discovered components. The framework provides a mechanism to monitor and sense violations in pre-defined operation preferences of system components through ontology reasoning process of the system context model. The publication addresses the problem of making a decision from a set of alternative adaptation decision and proposed the utilization of the Analytic Hierarchy Process method to find the best Web service migrations to be performed. In Section 4.2 we addressed the AHP algorithms proposed in this paper. Moreover, we provided an AHP-based migration example in Section 6.3 of the thesis.

6. KAZZAZ M. Mohanned and RYCHLÝ Marek. Restful-based Mobile Web Service Migration Framework. In: 2017 IEEE International Conference on AI & Mobile Services (AIMS). Honolulu: IEEE Computer Society, 2017, pp. 70-75. ISBN 978-1-5386-1999-5. This paper addresses the second case study presented in Section 7.2. It demonstrates the applicability of the thesis's proposed self-adaption approach on stationary and mobile architectures. The paper describes experiments on the implemented framework for Mobile Web service migration in P2P wireless network and shows the improvement in system performance by adopting the thesis approach.

7. KAZZAZ M. Mohanned and RYCHLÝ Marek. A Case Study: Mobile Service Migration Based Traffic Jam Detection. International Journal of Systems and Service-Oriented Engineering (IJSSOE). Hershey, PA: IGI Global, 2018, vol. 8, no. 1, pp. 44-57. ISSN 1947-3052. In this paper we described the design and implementation of a framework for service migration between cars to support traffic jam detection as an adaptation to the loss of traffic information service. The experiment result demonstrates the application of our proposed adaptation model supporting service mobility between mobile devices. Moreover, it presents the usability of service migration adaptation plan to survive service absence during route planning as real-life scenario.

Appendix D

Contents of the Enclosed CD-ROM

The case studies provided in Chapter 7 are supported with two software applications implementing the service migration framework for Windows and Android based devices (see Appendix A). The enclosed CD-ROM contains a source package of the migration framework projects and other related case studies documents in the following directories:

./sources/app – Android studio and Eclipse projects of the developed framework application for service migration.
./sources/service – Eclipse project of the framework service hosted on service provider (see Section 5.4.1).
./model – system context model of system components in JSON (see Section 3.5).
./case-studies – documentation of the services and devices specifications used in the case studies including the experiments data.

Appendix E

JENA Rules of the Web Service Migration System Context Model

1. Core rule to derive semantical statements with *possibleProvidedService* predicate.

```
[FindServiceForProvidersWithoutPreferences:
(?service rdf:type WSMF:MigratableService),
(?destination rdf:type WSMF:CandidateDestinationServiceProvider),
(?destination WSMF:noPreferenceRules 'true'^^xsd:boolean)
->
(?destination WSMF:possibleProvidedService ?service)
]
```

2. Core rule to derive semantical statements with *possibleDestinationProvider* predicate.

```
[FindProvidersForServicesWithoutPreferences:
(?service rdf:type WSMF:MigratableService),
(?destination rdf:type WSMF:CandidateDestinationServiceProvider),
(?service WSMF:noPreferenceRules 'true'^^xsd:boolean)
->
(?service WSMF:possibleDestinationProvider ?destination)
]
```

3. Core rule to derive instances of *CandidateForMigrationService* class due to a violation of service preference rule, (see Section 5.4.3).

```
[CandidateForMigrationServiceDueToServicesPreferences:
(?service rdf:type WSMF:MigratableService),
(?origin rdf:type WSMF:CandidateOriginServiceProvider),
(?destination rdf:type WSMF:CandidateDestinationServiceProvider),
(?origin rdf:type WSMF:CandidateDestinationServiceProvider),
(?origin WSMF:provides ?service),
(?service WSMF:possibleDestinationProvider ?destination),
(?service WSMF:noPreferenceRules 'false'^^xsd:boolean),
noValue(?service, WSMF:possibleDestinationProvider, ?origin)
->
(?service rdf:type WSMF:CandidateForMigrationService)
]
```

4. Core rule to derive instances of *CandidateForMigrationService* class due to a violation of provider preference rule, (see Section 5.4.3).

```
[CandidateForMigrationServiceDueToProvidersPreferences:
(?service rdf:type WSMF:MigratableService),
(?origin rdf:type WSMF:CandidateOriginServiceProvider),
(?destination rdf:type WSMF:CandidateDestinationServiceProvider),
(?origin rdf:type WSMF:CandidateDestinationServiceProvider),
(?origin WSMF:provides ?service),
(?destination WSMF:possibleProvidedService ?service),
(?destination WSMF:noPreferenceRules 'false'^^xsd:boolean),
noValue(?origin, WSMF:possibleProvidedService, ?service)
->
(?service rdf:type WSMF:CandidateForMigrationService)
]
```

5. Component rule example

```
[SamplePreference:
(?service rdf:type WSMF:MigratableService),
(?origin rdf:type WSMF:CandidateOriginServiceProvider),
(?destination rdf:type WSMF:CandidateDestinationServiceProvider),
equal(?destination, WSMF:sP1),
(?origin WSMF:provides ?service),
(?service WSMF:hasProperty ?property),
(?property rdf:type WSMF:ServicePriority),
(?property WSMF:propertyValue ?v1),
ge(?v1, "50„^^xsd:int)
->
(?destination WSMF:possibleProvidedService ?service)
]
```

Appendix F

Curriculum Vitae

Personal Information	
Name	M. Mohanned Kazzaz
Title	Ing.
Nationality	Syrian
Date of Birth	18.05.1983
Contact Information	
Address	Halap Al-jaddeda, C5, Aleppo, Syria
E-mail	mohanned.kazzaz@gmail.com
Phone	+420 774 923 358
LinkedIn	www.linkedin.com/in/mohannedkazzaz
Education	
2011 - present	Faculty of Information Technology, Brno University of Technology doctoral study (PhD.) Computer Science and Engineering http://www.fit.vutbr.cz/∼ikazzaz/
2010	Brno University of Technology, Brno. Czech Republic. Master's degree of Computer Engineering and Networks – Education Recognition.
2001-2007	Faculty of Electronic and Electrical Engineering, University of Aleppo, Aleppo, Syria. Bachelor's degree of Electronic Engineering "Computer Engineering".
Work Experience	
8/2009 - 10/2011	Software Engineer Employer: Aleppo City Council, Aleppo, Syria.