Towards Service Co-evolution in the Internet of Things

Dissertation by

M.Sc. Huu-Tam Tran

in partial fulfillment of the requirements for the degree

Doktor der Naturwissenschaften (Dr. rer. nat.)

submitted to the

Department of Electrical Engineering and Computer Science,
University of Kassel, Germany

Supervisor: Prof. Dr. Kurt Geihs
Date of defense: 3rd December 2021

December 2021

Acknowledgements

I would like to take this opportunity to thank those individuals and organizations, without whom this thesis and associated work would not have been possible. I am thankful for their aspiring guidance, invaluable constructive criticism, and friendly advice during the dissertation work.

First and foremost, I would like to thank my supervisor Prof. Dr. Kurt Geihs, for offering me the chance to work in his research group. I greatly appreciate the advice, opinions, and insight he has provided throughout my work. He also gives all his Ph.D. students the freedom to pursue their interests and research ideas. I was fortunate to work on his project named PROSECCO.

I would like to thank Dr. Nguyen Xuan Thang from Hanoi University as the second reviewer of this thesis for his insighful comments and valuable advices.

It has been a great experience to work with the Distributed System Group at the University of Kassel. My colleagues: Harun Baraki, Alexander Jahl, Tareq Razaul Haque, Daniel Saur, Nugroho Fredivianus, Stephan Opfer, Thao Nguyen, Stephan Jakob, Stefan Niemczyk, Dominik Kirchner, Marie Ossenkopf, Andreas Witsch, Christoph Evers - with whom I have shared my ideas and from whom I have received constructive feedback. The Group has been a source of friendships, helpful advice, and collaboration. Special thanks go to Harun Baraki and Alexander Jahl for their supports not only during the time of research work but also in the outdoor activities. They are also co-authors of my publications regarding Chapters 4-7. I would like to thank my students, Ramaprasad Kuppili and Srivardhan Cholkar, who assist me a lot in implementing the ideas in Chapters 6 and 7 of the dissertation. It is a great honor to be their advisor during my Ph.D. time in Germany. I will always remember their positive attitude. I also would like to express my thanks to all colleagues who supported the everyday work, such as Thomas Kleppe, Heidemarie Bleckwenn, Inken Poßner and all the others.

In the same fashion, moving from individuals to organizations, I would like to thank the Ministry of Education and Training of Vietnam and the German Academic Exchange Service for awarding me the scholarship for doing my research.

Last but not least, I must thank my parents, my wife, my brothers for their support, encouragement, and inexhaustible patience in all the past years.

Abstract

Nowadays, in the world of technology, where the new technologies are devolved and replace the old ones every single day, any software product in order to maintain its competitiveness, must be updated continually. Updating a software product results from an evolution process, including adding new features, replacing outdated features, repairing bugs, closing security gaps, and improving performance, which requires a lot of effort and knowledge. Therefore, the Internet of Things (IoT) services are no exception.

An interesting research question is how to handle service changes for service consumers and how to enable and facilitate end-user application updates in case the dependent clients provide services to other clients, especially in IoT environments. These manifold interdependencies make on-the-fly service evolution a particularly difficult and challenging problem because the evolution of one service may incur changes in other dependent services and clients. In analogy to biology, we call this service coevolution. Thus, the thesis aims to develop a comprehensive solution for the coordinated evolution of heterogeneous services in IoT.

The main contribution of this dissertation is a set of theoretical models and approaches that facilitate service coevolution. In particular, we developed a solution for coordinated service coevolution through a design technique that equips every service with an intelligent agent, called EVA (Evolution Agent), that performs the service evolution in collaboration with other EVAs. The EVA can control service versions, update local service instances. Furthermore, we proposed a notification management architecture for IoT services. Additionally, an approach to describe and detect changes in IoT services with support for a shared knowledge base is introduced. Last but not least, a method to find out changes in service behavior is presented by analyzing the data stream between the service client and service provider.

Kurzfassung

In der Welt der heutigen Technologie, in der neue Technologien täglich weiterentwickelt und die alten Technologien ersetzt werden, muss jegliche Software ständig aktualisiert werden, damit ihre Wettbewerbsfähigkeit aufrechterhalten wird. Die Aktualisierung eines Softwareproduktes ist das Ergebnis eines Evolutionsprozesses. Dieser Prozess erfordert einen größeren Aufwand und viel Wissen, es beinhaltet das Hinzufügen neuer Funktionen, das Entfernen veralteter Funktionen, das Beheben von Softwarefehlern, das Schließen von Sicherheitslücken und die Leistungssteigerung. Die Dienste des Internets der Dinge (IoT) stellen dabei keine Ausnahme dar.

Eine interessante Forschungsfrage besteht darin, wie Änderungen von Diensten für deren Nutzer gehandhabt, wie sie aktiviert und wie Aktualisierungen von Endnutzer Anwendungen ermöglicht werden. Das gilt insbesondere in IoT Umgebungen und im Falle von abhängigen Clients, die anderen Clients Dienste anbieten. Diese vielfältigen und gegenseitigen Abhängigkeiten machen die on-the-fly Evolution eines Dienstes zu einem besonders schwierigen und herausfordernden Problem, weil die Evolution eines Dienstes Änderungen in anderen abhängigen Diensten und Clients notwendig macht. In Analogie zur Biologie nennen wir diesen Dienst-Koevolution. Somit zielt diese Dissertation darauf ab, eine umfassende Lösung für die koordinierte Evolution von heterogenen Diensten in IoT zu entwickeln.

Der Hauptbeitrag dieser Dissertation besteht aus einer Reihe von theoretischen Modellen und Ansätzen, die die Dienst-Koevolution ermöglichen. Insbesondere entwickelten wir eine Lösung für die koordinierte Dienst-Koevolution durch eine Designtechnik, die jeden Dienst mit einem intelligenten Agenten, genannt EVA (Evolutions-Agent) ausstattet. Diese Designtechnik führt die Dienst-Koevolution in Zusammenarbeit mit anderen EVAs durch. Der EVA kann die Dienstversionen kontrollieren und die lokalen Dienstinstanzen aktualisieren. Außerdem schlugen wir eine Benachrichtigungsmanagementarchitektur für IoT Dienste vor. Zusätzlich wird ein Ansatz zur Beschreibung und Änderungserfassung in IoT Diensten mit Unterstützung für eine verteilte Wissensdatenbank dargestellt. Zu guter Letzt wird eine Methode zur Erkennung von Änderungen im Verhalten von Diensten durch die Analyse des Datenstroms zwischen dem Client des Dienstes und dem Dienstanbieter präsentiert.

Table of Contents

List of Figures

Abbreviations

The list below gives an overview of abbreviations used throughout the thesis.

ACM:	Association for Computing Machinery
ANN:	Artificial Neural Networks
ASP:	Answer Set Programming
BPEL:	Business Process Execution Language
CD:	Change Detection
CIA:	Change Impact Analysis
CR:	Change Reaction
CPU:	Central Processing Unit
DYMOS:	DYnamic MOdification System
DKB:	Default Knowledge Base
DKEM:	Distributed Knowledge Based Evolution Model
EVA:	Evolution Agent
IoT:	Internet of Things
HTTP:	Hypertext Transfer Protocol
JSON:	JavaScript Object Notation
RFID:	Radio-frequency identification
REST:	REpresentational State Transfer
SOA:	Service Oriented Architecture
SOAP:	Simple Object Access Protocol
XML:	Extensible Markup Language
UDDI:	Universal Description Discovery and Integration
URI:	Universal Resource Identifier

MAS: Multi-agent systems

NCEI: National Centers for Environmental Information

OSGi: Open Service Gateway initiative

OCSVM: One-Class Support Vector Machine

OWL: Web Ontology Language

PROSECCO: Provisions for Service Co-Evolution

QoS: Quality of Service

XSLT: Extensible Stylesheet Language Transformations

KB: Knowledge Base

KNN: K-Nearest Neighbour

KRR: Knowledge Representation and Reasoning

WADL: Web Application Description Language

WSDL: Web Service Description Language

W3C: World Wide Web Consortium

1 Introduction

1.1 Motivation

In the last decades, Service Oriented Architecture (SOA) has become a widely accepted paradigm that provides a flexible IT infrastructure to deal with the increasing pace of business changes and global competition [4]. The service in SOA is a software component that provides specific capabilities over a network to service consumers in a loosely coupled fashion. This service is encapsulated and offers a clearly defined service interface to service consumers.

With the advent of SOA, our private life and business activity increasingly depends on SOA applications [5]. By the time, the advent of Cloud Computing [6] and the Internet of Things [7] (IoT), or the Web of Things [8] have reinforced this trend even more. The downside of this trend is increasing the complexity of service landscapes. This complexity is due to the sheer size of these systems. The manifold interdependencies make service management as a whole a very substantial challenge for service providers [9].

In this context, a critical research problem is how to handle service changes for each consumer and how to facilitate the updates of the end-user application. Addressing the challenge of service changes means that service modifications require adaptations in participating parties to prevent outages and failures due to individual service modifications.

Besides, in the point of view of managing changes, in order to control service development, the developers need to know why a change was made, what its implications are, who did trigger, what kind of evolutionary changes occurred, and whether the resulting service version is compatible with existing consumers.

The term "service evolution" refers to deploying a new service version, which is caused by necessary changes [10]. On the one hand, service evolutions rise in order to satisfy requirements from clients. The changes are requested from clients for additional functionalities or bug reports. If the conditions are not met, the clients may switch to another offer. On the other hand, service providers adapt and evolve to the new market trends and attract more clients. Thereby, the service providers undergo necessary updates such as supporting new technologies, discharging obsolete functionalities, enhancing reliability, and improving performance. Thus, the need for evolution comes from service providers and their clients.

Service evolution in SOA systems may be triggered by the service providers or the service consumers since they may desire an arrangement with another competitor who meets their new requirements or performs better. However, in distributed networks of large-scale environments, every service depends on other services to avail of certain functionalities. Consequently, a change in one service can lead to another change in other services. It implies interdependent services need to co-evolve modifications together. This phenomenon in biology is called co-evolution, which is defined as the coordination of individual evolution to prevent outdated and failures.

Even though the successful co-evolution of services may bring advantages, it appeals very little attention from the research community. Therefore, this thesis aims to provide a solution for handling service interdependencies in large-scale service environments and minimizing service downtime. Furthermore, this thesis also aims at providing a general solution for coordinated decentralized service co-evolution within a resource-constrained environment like IoT.

In the application domain of this dissertation, we are considering a new insight into the feasibility and limitation of service evolution in large-scale service landscapes, including cloud-based and IoT services. Since the envisioned IoT foresees a future Internet incorporating smart physical objects that offer hosted functionality as IoT services, these service-based integrations of IoT will be easier to communicate with and integrate into existing application environments [11]. However, the management of IoT services and their interfaces will require new techniques due to resource constraints on IoT devices in processing capacity, communication bandwidth, battery lifetime, and memory capacity.

1.2 Research Questions

This thesis focuses on the environment of IoT systems. The primary research question that we intend to answer is: How can we facilitate coordinating service co-evolution that consists of various services that depend on each other in IoT environments?

Unquestionably, multi-agent systems have proved to be adequate for implementing complex systems with autonomous components and high communication demands. Thus, we consider them the most suited architecture to develop the IoT infrastructure. Consequently, in this research, we developed a solution for coordinating service co-evolution through a design that equips every service with EVA (Evolution Agent) that performs the service evolution in collaboration with other EVAs.

Besides, services in IoT are frequently subject to change in order to, for example, maintain their functionality, reliability, availability, and performance. Thus, detecting changes in services during the application life-cycle is essential for both change analysis and change management. The demand for defining and classifying the evolutionary changes, which may occur in service and their potential impacts on dependent clients and services, has recently arisen. Hence, one of the main tasks of EVA is to detect changes precisely; another critical responsibility is how to inform affected clients about updates.

For achieving service co-evolution successfully, specific problems should be taken into account as follows:

1. Which requirements EVAs have to meet to tackle the challenges above? This question can be divided into sub-questions such as:

    a) How to design the EVA? Or what components should EVAs constitute?

    b) How can each EVA coordinate and collaborate with other EVAs?

2. How to describe services in the IoT applications domain to support service (co) evolution?

3. How to detect and notify changes of IoT services? Or what are efficient mechanisms for change detection and communication in light of resource-constrained IoT devices?

4. How to find the clients that depend on the service and store and maintain this list of clients in IoT environments?

5. How to evaluate the solution?

1.3 Methodology

This research aims to provide a set of theoretical models and approaches that facilitate service co-evolution in IoT environments. Mainly, we decompose the research process into five steps:

Step 1: Problem definition

The primary motto of any research work is to define the problem statement for a short-term goal. The problem could be identified by defining the requirements, carrying out sufficient research, which considers some related work and corresponding literature, to categorize the issues to a specific domain. The next step is to establish a hypothesis about the research problem, perform further advancements to prove the considered hypothesis valid, and finally justify the problem statement, which is mentioned in the subsection of research questions.

Step 2: Review existing approaches When the problems are defined clearly, the next step is to explore the literature to determine the explicit approaches. The existing methods can also offer a concrete catalog for the identified problem that provides a holistic view of the research situation. A combination of different search terms was used to obtain all available research in this area, such as service co-evolution, service evolution, coordinated evolution, service changes, evolutionary changes, and co-evolve changes. These terms were defined to search relevant articles from the electronic databases listed in Table 1.1

Table 1.1: Identified databases

Identifier	Database	URL
ED1	IEEEXplore	http://ieeexplore.ieee.org
ED2	ACM	https://dl.acm.org
ED3	Science Direct	https://sciencedirect.com
ED4	Springer Link	https://link.springer.com
ED5	Wiley	https://onlinelibrary.wiley.com
ED6	Research Gate	https://www.researchgate.net
ED7	Scholar Google	https://scholar.google.com

Step 3: Define the solution

To solve a problem theoretically, the critical step before developing the approach is to build an architecture (or framework or model) for this solution. The theory architecture defines the roles and behaviors of the approach, indicates the path to solve the problems, and describes the system architecture of the solution. The architecture is an abstract of the solution to the problem that is clarified in step 1.

Step 4: Implementation and Evaluation

The research target is to deal with the service co-evolution problems. Once the proposed models are defined, it is time to work out a scenario to evaluate the approach. At the end of this step, a conclusion is made to comment on the experiment results.

1.4 Contributions

This dissertation has five contributions to solving fundamental problems for coordinated decentralized on-the-fly service co-evolution as follows:

- The first contribution concerns a solution for coordinating service co-evolution through a design that equips every service with an agent called EVA (Evolution Agent) that performs the service evolution in collaboration with other EVAs. The solution also introduces a new vision of service co-evolution in IoT by providing an evolution management model and reference architecture. The results of this contribution are presented in IoT360 conference in Rome, 2014 and published in a Springer book chapter [12] and a journal of the EAI Endorsed Transactions on Cloud Systems [13].
- The second contribution provides a notification management architecture in the context of the IoT domain since the main challenge for an IoT service provider is finding out about clients that depend on the service and storing and maintaining this list of clients who should be informed whenever a change takes place. The results were reported at the IEEE MESOCA symposium [14] in Raleigh, NC, USA, 2016.
- The third contribution is regarding a comprehensive framework DECOM to describe and detect changes in IoT services by using Answer Set Programming. The results of this work were presented at the ACM SoICT2018 symposium [15] in Da Nang, Vietnam, 2018.
- The fourth contribution proposes an approach to find out changes in the service behavior by analyzing the data stream between the service client and service

provider. The numerical results of this work were reported as a Master thesis [16] of the University of Kassel, 2018.

- The fifth contribution provides a survey of service co-evolution in SOA environments. The results of this work are published in a Springer book chapter [17] in 2020. The extended results were submitted to the Special Issue on Context-Aware Computing: Theory and Applications, Journal Concurrency and Computation, 2021.

1.5 Dissertation Outline

The dissertation is comprised of 8 chapters as following:

- In chapter 1, we introduce the motivation of our work by defining the problems and the methodology to achieve the desired solution.
- In chapter 2, we present the background knowledge, which forms the basis of solutions in this thesis work. It describes the Service-Oriented Architecture, Multi-Agent Systems, Answer Set Programming, and the Internet of Things.
- In chapter 3, we provide details of related work in service co-evolution, which includes the techniques followed to treat and represent the evolving services. We start from the overview of software evolution aspects as the root of service evolution and then investigate service evolution in detail. Finally, some practical approaches supporting service co-evolution are introduced.
- In chapter 4, we propose a multi-agent architecture for distributed service co-evolution. This chapter will answer the questions concerning the requirements that EVAs have to meet to tackle the challenges mentioned above.
- In chapter 5, we develop a notification management infrastructure that automatically detects and informs dependent third-party applications about service changes that require an adaptation. Moreover, it maintains a list of possibly affected clients without consuming further resources on the IoT device that the service is running on.
- In chapter 6, we introduce the framework DECOM to describe services for dynamic and heterogeneous IoT environments. It is based on an extended Web Application Description Language for REST services. Our framework is enabled to transform the interface description into a logic program based on Answer Set Programming.

- In chapter 7, we describe our detecting behavior changes approach based on anomaly detection that has received considerable attention in the data mining and machine learning community.
- In chapter 8, we summarize the contributions of this thesis. Finally, we outline several potential workstreams for future research efforts.

2 Foundation

In this chapter, we describe the selected foundations that represent the basics of this dissertation. Firstly, we give some insight into Service Oriented Architecture (SOA) and Web Service paradigms. Secondly, we introduce some background knowledge of the intelligent agent and multi-agent systems. Thirdly, we present some perspectives of the Answer Set Programming (ASP), a declarative problem-solving paradigm, rooted in Logic Programming. Finally, we summarize some of the general and key features of IoT.

2.1 Service Oriented Architecture

Nowadays, enterprises have to be competitive and adaptable to new business requirements. Thus, applications and services of different enterprises need to cooperate independently of their platforms. As time goes by, systems become more complex soon; consequently, flexibility is an important goal to be reached. In this context, SOA is a common paradigm that addresses these challenges mentioned above.

2.1.1 Definitions

SOA: Several SOA definitions are proposed since the term has been invented; for instance, the authors [18, 19] state that SOA is an architectural style integrating services that distribute on different servers.

Rosen et al. [20] defined SOA as follows: "*SOA is an architectural style for building enterprise solutions based on services. More specifically, SOA is concerned with the independent construction of business-aligned services that can be combined into meaningful, higher-level business processes and solutions within the context of the enterprise*". Meanwhile, CM. Mackenzie et al. [21] describe the SOA concept as "*a paradigm for* *organizing and utilizing distributed capabilities that may be under the control of different ownership domains.*"

A service is a self-contained unit of software that performs a specific task. According to M. Rosen et al. [20], services are at the core of SOA. A *service* is a software component or program that may solve a computational problem or a specific business task and is available in a network. Services are loosely coupled; they offer support for overcoming interoperability issues and distributivity challenges that enhance collaboration between services [22].

SOA participant roles: The primary participant roles in a SOA are (i) *service provider*, (ii) *service consumer*, and (iii) *service registry.* Figure 2.1 illustrates the three different roles in a SOA and their interaction. The *service provider* offers its services to potential clients, who may call the services and use their functionalities. The *service registry* is responsible for making the service interface and implementation access information available to any potential service consumers. The provider registers the service in the service registry and publishes its service description so that potential consumers may find it. A *service consumer* may be an application, such as a Web portal, or a business process that needs to integrate the service to use its functionality. The service consumer searches for services in the service registry by submitting a query that contains the criteria for the searched service. A SOA can consist of publish/subscribe mechanisms to inform service consumers about state changes [1].

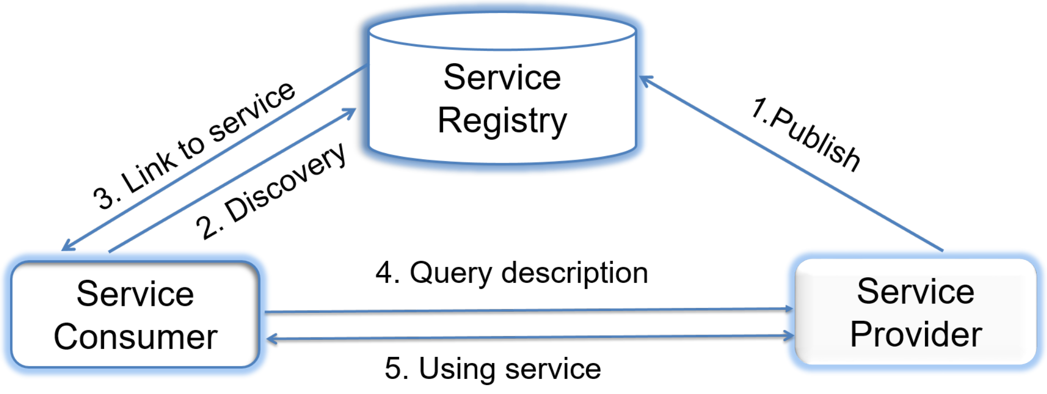


Figure 2.1: SOA participant roles and their interaction [1]

SOA principles: SOA is based on some essential principles which are described shortly as follows [18, 21]:

(i) Standardized Service Contract: A service must have some sort of description that describes what the service is about. This principle makes it more straightforward for consumers to understand what the service does.

(ii) Loose Coupling: This principle states that there should be less dependency between the services and the client invoking the service. Hence if the service functionality changes at any point in time, it should not disclose the client application or stop it from acting.

(iii) Service Abstraction: A service uses standard interfaces to expose its internal logic. In a service abstract, the implementation details are hidden, allowing them to implement the service with different technologies and platforms.

(iv) Service Autonomy: Services should have control over the logic they encapsulate. The service knows everything on what functionality it offers and therefore should have comprehensive control over the code it involves.

(v) Service Reusability: Logic is divided into services with the intent of maximizing reuse. It means developers do not want to spend time and effort building the same code repeatedly across multiple applications that require them. Hence, once the code for a service is written, it should work with various application types.

(vi) Stateless Operation: Service should be stateless to minimize resource consumption and to simplify failure recovery. It means that services should not withhold information from one state to the other. This would need to be done from either the client application.

(vii) Service Discoverability: To make the service accessible to interested consumers, it must be centrally accessible. This could either be done by publishing the service to a dedicated service registry or by simply placing this information in a shared directory.

(viii) Service Composability: Services break big problems into minor problems. One should never embed all functionality of an application into one single service but instead break the service down into modules, each with separate business functionality.

2.1.2 WS-* Paradigms

The SOA paradigm provides an approach to describe and invoke service from different platforms based on, e.g., HTTP and XML to aid service interoperability. In practice, Web Service is an example of SOA patterns with a well-defined set of implementation choices [23].

The World Wide Web Consortium (W3C)[1] defines Web Service as follows: *"A software system designed to support interoperable machine-to-machine interaction over a network. It has an interface described in a machine-processable format. 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"*.

WS-* usually refers to Web Service that use Simple Object Access Protocol (SOAP) messages with an Extensible Markup Language (XML) payload and an HTTP-based transport protocol to provide remote procedure-calls (RPCs) between clients and servers. The key technologies of WS-* involve SOAP, Web Service Description Language (WSDL), Universal Description Discovery and Integration (UDDI), and Business Process Execution Language (BPEL).

In general, Web Service is preferred as they conform to the SOA principles of loose coupling, platform independence, interoperability, and distributivity. Two cornerstone terms are commonplace to classify specifications and technologies for Web Service composition: (i) *orchestration*, an executable business process built with Web Service seen from a single-party perspective; and (ii) *choreography*, the message sequences between multiple Web Services seen from the perspective of multiple parties[23]. These terms were initially coined in the context of SOA-based service.

SOAP

SOAP[23] is a simple XML-based protocol to let applications exchange information over HTTP. A SOAP interface is typically designed with a single URL that implements several RPCs methods, which define a message architecture and format, hence providing a rudimentary processing protocol. The top-level XML element of a SOAP message is called an envelope, which includes two XML elements: header and body. The header specifies routing and Quality of Service (QoS) configuration while the body contains the payload of the message indicating the interoperations.

[1]https://www.w3.org

WSDL

The service description is a key to making the SOA loosely coupled and reducing the amount of required common understanding, custom programming, and integration between the service provider and the service client[19]. The service description contains all the necessary artifacts like service methods, bindings, communication protocols that the consumer needs to know in order to be able to call the service. The service description has to be independent from the platform of the service. The W3C published the specification that uses a standard interface definition model called WSDL. It defines the details of the abstract service information.

Others

UDDI [24] was constructed as a platform-independent, XML-based registry framework for describing and discovering worldwide Web Service. UDDI can be viewed as a directory of WSDL. Web services can be registered and located in the directory. It can be requested using SOAP messages to provide access to WSDL documents, which describe the protocol bindings and message formats required to interact with the Web Service listed in its directory.

2.1.3 RESTful Service

This section provides an introduction to REST principles and Web Application Description Language named WADL.

REST Principles

The term REpresentational State Transfer (REST) was first coined by Roy Fielding [25] in his Ph.D. thesis. According to Roy Fielding, the basic concept of REST is that everything is modeled "resource", or particularly HTTP resources, with a Universal Resource Identifier (URI). In the following section, we give an overview of the most essential principles in REST architectural style, which are described in full detail by Deze Zeng in [24] and Roy Fielding [25]:

(i) Resource: All the resources exposed by RESTful web services are identified by URIs. Through URI, the clients can identify their interaction targets. A global addressing space is provided for service and resource discovery.

(ii) Uniform interface: RESTful services treat the HTTP as an application protocol instead of a transport protocol in WS-*. Therefore, the term REST is often used in conjunction with HTTP, and the RESTful resources can be manipulated using HTTP verbs such as PUT, GET, POST and DELETE. PUT creates a new resource while DELETE deletes it. GET retrieves the current state of a resource in some representation while POST updates a resource with a new state.

(iii) Self-descriptive messages: Resources are decoupled from their representations such that it is free to use a variety of data formats to describe themselves provided that the appropriate representation formats are agreed and understandable by endpoints. For example, the data can be in any common-used formats such as HTML, XML, plain text, PDF, Metadata about the resource can be used to control caching, detect transmission errors, negotiate the representation format, and perform authentication or access control between endpoints.

(iv) Stateless operations: Every interaction with a resource itself is stateless. However, stateful interactions can be realized through hyperlinks. The state of a resource can be explicitly transferred by URI rewriting, cookies, and hidden form fields. The states can also be embedded in a response message for stateful interactions.

REST is a set of architectural constraints, not a protocol or a standard. It possess less verbose and can support several representation formats such as HTML (HyperText Markup Language), JSON (JavaScript Object Notation), XML (eXtensible Markup Language) and plain TEXT besides providing the HTTP's runtime content negotiation [25]. This ability makes REST hold less payload and aid better support for client-server communications. However, REST's hypermedia does not prescribe how to describe a RESTful standard. For this reason, a new description language is emerged, called WADL (Web Application Description Language). Therefore, the next section summarizes what WADL is and some of its elements.

WADL

A services description named WADL is a simple interface representation. WADL consists of the following elements [26–28]:

(i) Grammars: A container for definitions of any XML data exchanged during the execution.

(ii) Resource: It describes a resource provided by a Web application. Each one consists of a path URL, defining it uniquely from other existing resources in the WADL definition. A resource can have an optional ID that is used to identify the corresponding resource element.

(iii) Method: It describes the input and output set of operations performed via the HTTP protocol, including its definition or any method reference defined elsewhere. It consists of at least one or more requests, responses, or representation elements.

(iv) Representation: A representation element describes a representation of a resource's state. A representation element can either be a representation definition or a reference to a representation defined elsewhere.

In practice, WADL data is usually rich in syntax but not in semantics. Therefore, for successful service discovery and change detection, semantic annotations should be added to the WADL data. In our scenarios in this thesis, IoT services can be accessed through REST interfaces. The resources and data elements can be arbitrarily represented using various formats, thus, enabling a lightweight communication compared to Web Service.

2.2 Multi-agent System

2.2.1 Concepts

The definition of an agent is a topic of some debates in various application domains. In general, there is no universally accepted definition for the term *agent*. In this thesis, we rely on the definition of the rational agent by Russell and Norvig [29] that is also used in the context of multi-agent systems by M. Woolridge [30]. This term *agent* is defined as follows: *"An agent is a computer system situated in some environment and that is capable of autonomous action in this environment in order to meet its design objectives."*

The term agent can be realized in various application domains. Furthermore, an agent is also able to interact with other agents in order to accomplish its goals. Agents can be divided into types spanning simple to complex. The interactions between agents and their environment are shown in Figure 2.2.

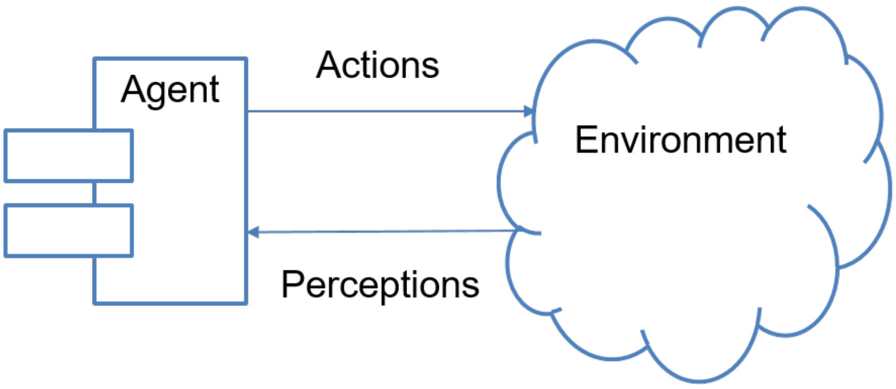


Figure 2.2: Agent and Environment Interaction [2]

Multi-agent systems (MAS) are distributed systems composed of some autonomous software entities called agents [31]. The concept of MAS has been considered as one of the most important paradigms that improves the designing and implementation of software systems [32]. The practical utility of agents has been demonstrated in a wide range of domains such as online trading [33], disaster response [34, 35], business process management [36] and information management [37].

Kubera et al. [38] view a MAS as a computerized system composed of multiple interacting intelligent agents that can solve problems that are difficult or impossible for an individual agent. Intelligence may include methodical, functional, procedural approaches, algorithmic search, or reinforcement learning. In general, software agents or multi-agent systems may encompass different attributes such as architecture, communication, coordination strategies, decision-making, and learning abilities. So far, such useful notions of intelligent agents have made them a popular choice in the domain of software design and software development.

2.2.2 Modeling Agents

According to M. Gelfond and Y. Kahl [2], a mathematical model of an intelligent agent consists typically of the following elements: (i) A language(s) for representing the agent´s knowledge. (ii) Reasoning algorithms use this knowledge to perform intelligent tasks, including planning, diagnostics and learning. (iii) An agent architecture is the structure combining different submodules of an agent in a coherent whole.

Similar to M. Gelfond and Y. Kahl, in this thesis, we consider the following typical agent model that contains knowledge about the world and that entity´s capabilities and goal. In this model, there are various challenges, such as: How can we create a knowledge base for our agents? How can we enable it to function in changing environments? Or how do we make it capable of recognizing events?

Regarding the solution, M. Gelfond and Y. Kahl recommend the logic-based approach named Answer Set Programming (ASP) as an effective solution to overcome the challenges mentioned above. This approach uses declarative language for representation, defining reasoning tasks as queries to a program and computing the result using an inference engine (i.e., a collection of reasoning algorithms).

In the following section, we will give a brief introduction to ASP, an auspicious tool for knowledge preservation and declarative problem-solving in the area of Knowledge Representation and Reasoning (KRR).

2.3 Answer Set Programming

This section begins with the introduction to ASP on how knowledge can be represented by giving insights into the ASP concepts and their advantages. The main content of this section has relied on a well-known book by M. Gelfond and Y. Kahl [2].

As we have known, the amount of computational problems seems to be limitless in both academics and industry. There is a massive demand for new insights from the vast amount of available data. To achieve this knowledge, researchers use all kinds of programming languages for creating algorithms. Nonetheless, many of the current real-world problems are complex search problems. In addition, attempts to solve the search problems by creating various algorithms requiring significant effort and time-consuming. Fortunately, the ASP approach offers a simple and powerful modeling language to solve combinatorial problems [39].

ASP is a form of declarative programming towards difficult, primarily NP-hard, search problems. It is based on the stable model (called *answer set*) semantics of logic programming. ASP mainly focuses on what we want to achieve, not statements concerning how we can achieve it. In other words, instead of writing statements describing the control flow of computation, declarative programming expresses its logic. The idea of ASP is to represent a given computational problem by a program whose answer sets correspond to solutions and then use an answer set solver to generate answer sets for the program.

Whenever a program has no answer sets (no solution can be found), it is said to be inconsistent. Otherwise, it is said to be consistent. So far ASP has been applied to various applications, among them, decision support for space shuttles at NASA [40], product configuration [41], train scheduling in Switzerland [42], Linux package configuration [43] and many more.

ASP is constructed on the basement of atoms, literals, and rules in a more general sense. Atoms or atomic statements depict the declarative information that can be either *true* or *false*. A literal is an atom a or its negation $\neg a$, a negative literal, which can be read as "a may be false". Syntactically, an ASP program is a collection of rules, where a rule is formalized as follow:

$$a_0 \leftarrow a_1 \wedge ... \wedge a_n \wedge \sim a_{n+1} \wedge ... \wedge \sim a_m.$$

A rule in ASP program has three parts, the head a_0, the positive part $a_1 \wedge ... \wedge a_n$ and the negative part $\sim a_{n+1} \wedge ... \wedge \sim a_m$ of the body. Each a_i denotes a predicate $p(t_1, ..., t_j)$ with terms $t_1, ..., t_j$ built from constants, variables, and functions. This rule states that the head is believed to be true if the body is believed to be true. This rule introduces us to a new concept called *default negation (**not**)*. This concept differs from classical negation $-a$. The concept **not** a is a statement about belief and $-a$ states the opposite of a. In ASP, a rule without a body is a fact and a rule without a head is a constraint.

As we know that the closed world assumption introduced the term *default* - a statement of natural language containing words such as "normal," "typical," or as "a rule." A default is a rule that can be used unless an exception overrides it. Since defaults are very useful to humans as in the absence of complete information, they allow us to draw conclusions based on knowledge of what is common or typical. In ASP, one of the key advantages is that the knowledge or given default can be changed and extended at runtime by adding new information without causing inconsistencies.

Let us consider a simple example from logic about *Batfish* [2] and swim. It is a well-known fact that a fish can swim. This can be encoded as an answer set rule:

$$canSwim(X) \leftarrow fish(X).$$

[2]https://blog.padi.com/the-fish-that-doesnt-swim/

This line denotes the fact that "If X is a fish, then X can swim." Now let's add more knowledge, for instance, facts that tell the unconditional truth such as a *Salmon* fish named **Harry** and a *Batfish* fish named **Sally**.

$$\begin{gathered} canSwim(X) \leftarrow fish(X). \\ fish(harry). \\ fish(sally). \\ salmon(harry). \\ batfish(sally). \end{gathered}$$

Next, we execute the above piece of knowledge representation by an ASP solver, for instance by *Clingo* [3]. The following answer set is produced and shown in Figure 2.3.

```
clingo version 5.5.0
Reading from stdin
Solving...
Answer: 1
fish(harry) fish(sally) canSwim(harry) canSwim(sally) salmon(harry) batfish(sally)
SATISFIABLE
```

Figure 2.3: Fish can swim

As a result, the answer set tells us the known facts that *Salmon* and *Batfish* are fishes and they could swim. However, the biology enthusiasts among us know that *Batfish* are unable to swim. In fact, Batfish cannot swim. This fish "walk" with their pectoral fins across the ocean floor. Thus the model is not accurate! To get acceptable results, we need to add more knowledge in the form of facts and integrity constraints to the program. Here, the model needs to know that *Batfish* is not only a fish but also this fish cannot swim. A rule can represent this new information as the following line:

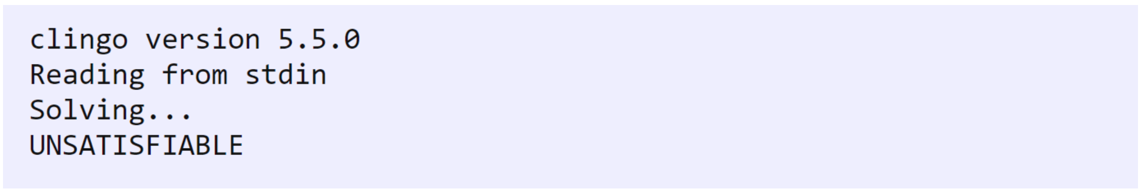
$$\leftarrow canSwim(X), fish(Batfish).$$

Now we have an ASP program as the following:

$$\begin{gathered} canSwim(X) \leftarrow fish(X). \\ fish(harry). \\ fish(sally). \end{gathered}$$

[3]https://potassco.org/clingo

$$
\begin{gathered}
salmon(harry).\\
batfish(sally).\\
\leftarrow canSwim(X), fish(Batfish).
\end{gathered}
$$



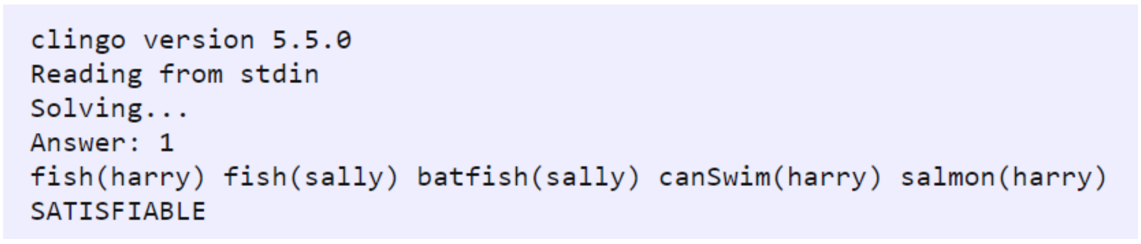
```
clingo version 5.5.0
Reading from stdin
Solving...
UNSATISFIABLE
```

Figure 2.4: Batfish cannot swim

Unfortunately, the addition of this rule to the above program is inconsistent, although this new information does not cause inconsistency. Figure 2.4 shows the result after running *Clingo*.

$$
\begin{gathered}
canSwim(X) \leftarrow fish(X), not\, batfish(X).\\
fish(harry).\\
fish(sally).\\
salmon(harry).\\
batfish(sally).\\
\leftarrow canSwim(X), fish(batfish).
\end{gathered}
$$

To avoid this, we could add that only fish that are not Batfish can swim. Please note that symbol *not* is a new logical connective called default negation. After adding this knowledge, if we run *Clingo* to acquire the answer set of the program, we get the answer set shown in Figure 2.5. Thus, the stable model consists of the expected outcome.



```
clingo version 5.5.0
Reading from stdin
Solving...
Answer: 1
fish(harry) fish(sally) batfish(sally) canSwim(harry) salmon(harry)
SATISFIABLE
```

Figure 2.5: Addition of new facts to the program

In summary, this simple example illustrates the use of ASP definitions of relations for building simple knowledge bases that allow incompleteness of information. Thus, it makes ASP suitable for the formalization of defeasible and commonsense arguments.

2.4 IoT in Nutshell

2.4.1 Concept

In recent years, the term IoT has become a dynamic research definition adopted in various areas, such as computer science and robotics. The definition of the IoT varies depending on different technologies for implementation. Kevin Ashton [44] initially proposed the concept of IoT in 1999, and he referred to the IoT as uniquely identifiable interoperable connected objects with radio-frequency identification (RFID) technology. The researchers then connected IoT to other technologies such as sensors, actuators, GPS devices, and mobile devices [45].

However, the precise definition of IoT is still in the forming process, depending on the perspectives taken. This term has been adapted and refers to a heterogeneous network of physical and virtual objects embedded with electronics, software, sensors, and connectivity [46]. This term is also considered as a part of the Internet of the future and will comprise billions of intelligent communicating "Things". "Thing" in terms of IoT, maybe a person with a heart monitor implant, a farm animal with a biochip transponder, a robot for field operations that helps with a search and rescue mission [47]. According to Cluster of European research projects [48] on the Internet of Things, "Things" are active participants in business, information, and social processes, in which they are able to communicate with each other and with the environment by exchanging data and information perceived about the environment. Regarding this concept, Adrian McEwen et al. [49] stated that IoT, in summary, can be described as the following simplistic equation:

$$Physical\ Objects + Controller,\ Sensors\ and\ Actuators + Internet = IoT$$

In order to provide smart services to end-users or applications, the technical standards for IoT should be set in terms of the specifications for data exchange, processing, and communications within the network. The success of IoT depends on the standardization that ensures interoperability, compatibility, reliability, and effectiveness of operations on a global level. Although significant research efforts for the development of IoT, there still are several significant challenges that should be taken into consideration such as lack of standardized architecture [45], lack of standardized service description [46], poor context-awareness for services [50].

2.4.2 Applications and Frameworks

IoT enables information gathering, storing, and transmitting to be available for things equipped with tags or sensors [50]. The use of IoT technologies has addressed many problems in our daily life. For example, in assisted living, a ubiquity of IoT devices and services can address the need for independent living for the growing numbers of people living with a physical disability. In the agriculture area, IoT can be used to manage production by monitoring and tracking variables that influence food production, such as weather, natural disasters, consumption, crop and animal diseases [51]. IoT has been used to follow-up on patient recovery in the healthcare domain and assesses that against several parameters unique to the patient using IoT-enabled devices [51]. These applications can be classified into the following domains: (i)*autonomous driving and logistics domain*; (ii) *healthcare domain*; (iii)*smart cities*; (iv)*smart homes and buildings*, (v)*personal and social domain.*

In the past few years, a variety of IoT frameworks have been developed in several research communities. Many of these frameworks (see at [46, 52]) have been developed for specific application domains. Some of these frameworks have been getting more and more attention in the IoT research community. This section will highlight some of them as follows:

FRASAD

Nguyen et al. [53, 54] proposed the FRASAD framework. The framework is a practical model-driven software development framework to manage the complexity of IoT applications. It aims at allowing developers to design their IoT applications using sensor node domain concepts. The framework is an extension of two layers in the existing sensor node architecture. These two additional layers are the Application Layer (APL) and the Operating System Abstraction Layer (OAL). The essence of these two layers is to magnify the level of abstraction and thus to conceal the lower levels. In order to achieve this, the framework employs the use of a designed Domain Specific Language (DSL) to model the sensor nodes and separate the operating system from the application. The OAL is then contracted to explicate the modeled application based on the specific operating system for implementation. For further detailed information about the FRASAD, we refer to the publication [53] and Nguyen's Ph.D thesis [55].

Calvin

Calvin is an IoT framework for application development, deployment, and execution in heterogeneous environments that include clouds, edge resources, and constrained resources [56]. Inside Calvin, all the distributed resources are viewed as one environment by the application. The Calvin framework model offers a distributed runtime environment and is multi-client capable since actors can share runtimes with actors from other applications. It also supports the restrictions of actors with high resource consumption [46]. For further features and capabilities of this framework, we refer to the publication [56].

Other Frameworks

The AllJoyn framework was developed by the AllSeen Alliance [4] to enable interoperability for home automation applications. It is an open-source framework that aims to connect and integrate things regardless of the communication module, operating system, and manufacturer. The framework consists of some implemented standard services and interfaces that developers use to integrate a variety of devices, things, or apps. It is optionally dependent on cloud services because it runs on a local network. In this way, devices and apps can communicate within the network with only one gateway agent, which is designed to connect to the Internet [46, 52].

Generally speaking, there are more than 30 other frameworks [46] that include many groups either targeting niche or specific aspects of the IoT or are looking to provide oversight and guidance to the development of the IoT technologies. Each of these IoT frameworks applies to a specific domain with different goals. In general, there is still a lack of an IoT framework for service co-evolution in IoT environments. Thus, the main focus of this thesis, i.e., providing an IoT framework for distributed co-evolution of services, has not been appropriately addressed in recent years. Furthermore, attempting to define service co-evolution and its vision is still immature. In the next chapter, we will present several critical related works in this area, starting from the first effort to support software evolution.

[4]AllSeen Alliance, https: //wiki.allseenalliance.org/

3 Related Work

Service co-evolution has been an appealing subject, wherein the coordination among various services is discussed. In the scope of SOA, researchers have spent significant effort investigating methods and techniques for the management of service changes. However, there many research challenges in this domain, and this chapter briefly presents few contributions from various researchers.

This chapter presents the general introduction of software evolution, the foundation of service evolution. After that, we provide further insights into the service evolution aspects, e.g., change terminology, kinds of change, the process of change, and eventually highlights some distinguished contributions related to service co-evolution (see Table 3.2). The main results are published in [17].

3.1 Software Evolution

Software evolution is of great importance in satisfying user requirements under specific changes in the environment [57]. Substantial works have been conducted in related areas promoting software evolution. In the early days, the term software evolution applied to general software maintenance and configuration. Later, this term refers to a phase that adapts application software to ever-changing user requirements and operating environments [58, 59].

Nowadays, it is widely accepted that continuous changes are a critical feature of evolution. One of the fundamental research works on software evolution was investigated by Lehman, and his colleagues [60], who presented the famous eight laws of software evolution. These laws describe a balance between forces driving new developments and forces that slow down progress. In their studies, Lehman et al. [60, 61] considered that the changing and adapting requirements from the real-world software systems drive the application to evolve with inevitable and continual feedback. The authors concluded that evolution is an intrinsic and feedback-driven property of software.

Even though Lehman's software evolution laws have not been wholly validated, they became fundamental knowledge of software engineers, and widely accepted [62]. Furthermore, most of the rules are just for solving general static maintenance or software evolution problems.

In general, there is a large body of research results available for managing software evolution. These studies fall into the following classifications: (i) analyzing the evolution trend of a software system over a long period; (ii) developing effective techniques to support software evolution [59].

At present, researchers have focused on solving two challenging aspects. The first challenge is how to evolve the software. The second one is how to react with software that has evolved. To the former question, the traditional solution is to improve software development and deployment approaches. Concerning the latter, the users of services have to deal with incompatible interfaces of modified modules and adapt to these changes.

With the advancement of software engineering, the software execution environment has become more dynamic and complex. Traditional methods of maintenance and software evolution face significant challenges related to various rapidly changing customer requirements. Therefore, some researchers have proposed new models and tools for evolving the software at runtime. For instance, DYnamic MOdification System (*DYMOS*) by Insup Lee [63] presented general principles for modifying the running software system. Later, *K-Component* by Dowling et al. [64] defined a meta-model for software architecture to provide possibilities for dynamic adaptation. In 2003, *OSGi* (Open Service Gateway initiative) was proposed by researchers at IBM and SUN, aiming to improve the practical use of the limited resources in the embedded devices. These architectures are important in the field of software evolution [65].

In general speaking, researchers described these solutions for software evolution as *static software evolution* and *dynamic software evolution*. The static software evolution refers to refactoring the software at the development stage and then install the new version by shutting down the running application. The static software evolution could make the software temporarily unavailable, and thus, it may cause delays for enterprise business. In contrast, dynamic evolution refers to adjusting the behavior of the software at runtime without breaking down the business. Dynamic evolution improves software adaptability and can be more competitive in modern, complex, and distributed environments. However, the dynamic software evolution also poses more complex challenges as users attempt to apply themselves to the large-scale distributed environment.

Besides, through an extensive survey, Feng-lin et al. [66] discovered that software evolution is closely related to the changing code, module, and architecture.

In summary, the existing approaches have provided a strong foundation in the development of software evolution. The researchers have tried hard to develop the architectures, tools, and frameworks to enable the software to be reconfigured at runtime on-the-fly. However, these approaches lack coordinated, distributed evolution where software service dependencies have to be negotiated with other services.

3.2 Service Evolution

Service evolution can be recognized as a particular case of software evolution. Various research in the literature has widely accepted that service evolution is a continuous development through a series of changes displayed in different versions of service.

3.2.1 Terminology

Let us first briefly define the basic terminology used in this thesis. This is necessary because there is no general agreement on the terminology on the broader service evolution community.

According to M. Papazoglou et al. [10, 19], **service evolution** is *"a continuous process of service development through a series of consistent and unambiguous changes"*. It is expressed through a service's different versions, and the key challenge is the forward compatibility between different versions. Similarly, a definition of service evolution is given by Wang et al. [67] who stated that service evolution is *"the process of maintaining and evolving services to carter to new requirements and technological changes"*. Both of these definitions have the same views that service evolution involves deploying a new service version, which is caused by necessary changes in interface structure, functionality, usage protocol, usage policies, business rules and regulations, and more.

In distinguished studies on service evolution, M. Papazoglou et al. [10, 19, 68] provided the fundamentals for service evolution until the present, especially when they distinguished two kinds of changes named **shallow changes** and **deep changes** based on the nature of service evolution. **Shallow changes** are *"small-scale, incremental changes that are localized to a service and/or are restricted to the consumers of that service"*. This kind of change typically could lead to mismatches between services at two levels: interface level and interaction protocol level. **Deep changes** are *"large-scale, transformational changes cascading beyond the consumers of a service possibly to consumers of an entire end-to-end service chain"* . This kind of change involves operational behavior changes or policy-induced modifications.

Furthermore, M. Papazoglou defines three following concepts [10]:

(i) Version compatibility means when we can introduce a new version of either a provider or client of service without changing the other.

(ii) Backward compatibility means when a new version of a client is introduced to the providers are unaffected. The client may introduce new features but should still be able to support all the old ones.

(iii) Forward compatibility refers to the old version of a client application that could interpret new operation(s) or message(s) introduced by a service.

Some types of changes that are both backward- and forwards-compatible involve the addition of new service operations to an existing service definition, the addition of new schema elements within a service that are not contained within previously existing types [10].

The key problem of service evolution is that the service's compatibility may change when the service evolves. One of the primary objectives of the research on service evolution is to reduce the unexpected effects caused by incompatibilities.

We adjust these definitions of service evolution in [10, 19, 67] for service co-evolution by giving a simple notion, i.e., **service co-evolution** stands for a coherent process of evolving and maintaining service *and its interdependent services* through a series of explicit changes. We further explain this domain by considering the dependency graph shown in Figure 3.1 of different service providers offering various services. It shows the service dependency graph S, which is a set of different services.

Now, consider node S_0, which is actively connected to all other nodes providing different services without any interruptions in the edges. Similarly, S_1 is connected to the nodes S_2 and S_3 and these nodes are further connected to various other nodes. Considering a scenario, in which S_2 evolves to a new version but S_3 is not affected at all proves

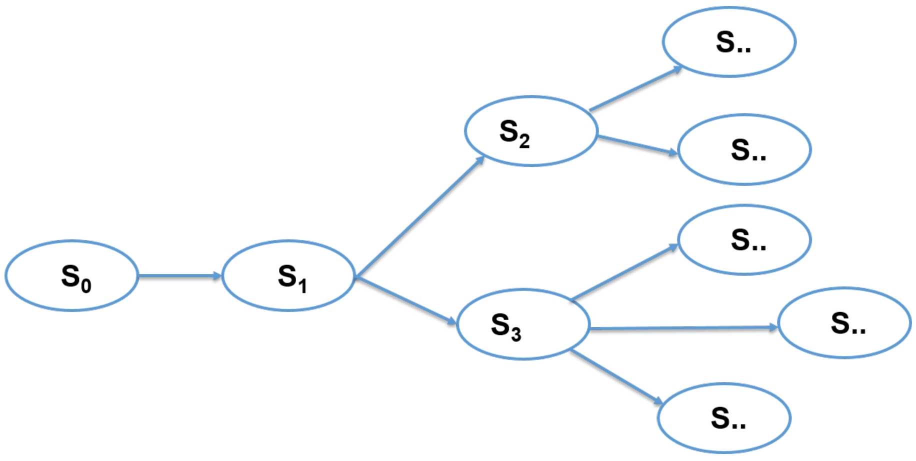


Figure 3.1: Dependency Graph in Service Co-evolution Scenario

a successful service evolution. In this case, the change is confined to the clients of S_2 only. However, in case the evolution of S_2 implies that the nodes S_0, S_1 and S_3 should also evolve in a coordinated fashion, we call this as a co-evolution of service where the evolution is required to update the interdependent services.

3.2.2 Change Taxonomy

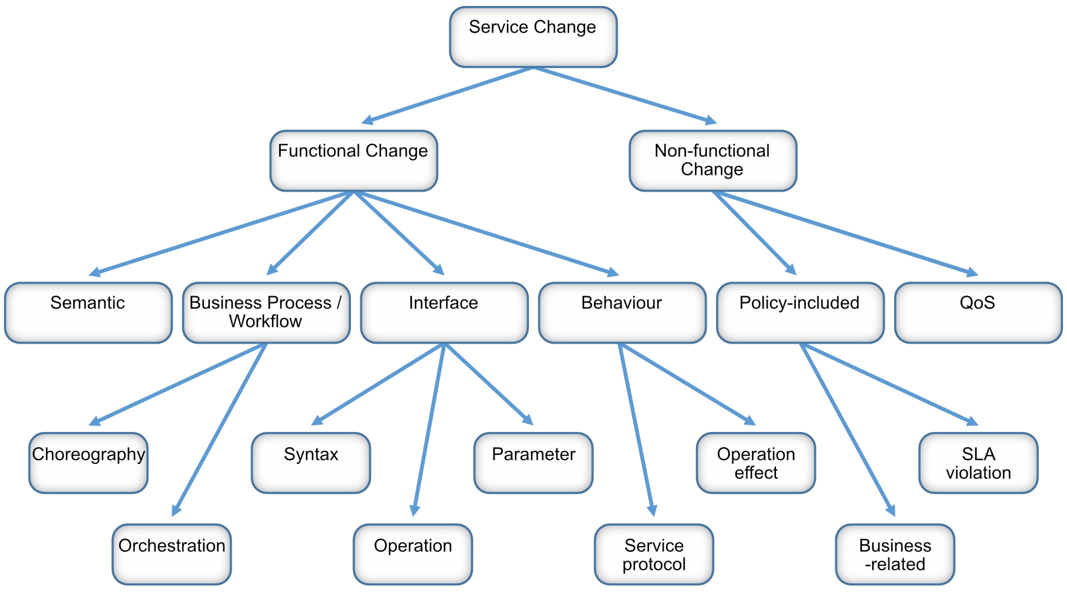


Figure 3.2: Change Taxonomy

Different change taxonomies for service evolution have been developed over the years. Various change taxonomies are proposed based on the effects or scope of changes during the interaction process. Nowadays, evolutionary changes in existing research works can be defined into various categories such as change of interface, change of semantics protocols, change of requirements, and change of business process models. These changes are described in detail in Table 3.1. We adopt a change taxonomy shown in Figure 3.2. Our terminology is consistent with the one used in [69] and shown in Table 3.1. Some other kinds of changes such as QoS, policy, parameters, optional operations are defined similarly to the previous works in [69, 70].

Table 3.1: Change Terminology

Category	**Type of Change**	**Characteristic**
Non-functional	QoS	- performance of service properties e.g., server load, concurrent users - performance of network latency, - performance of throughput
	Policy	- change in policy assertions on services, which specify business agreements
Functional	Behavior	- change in the service protocol i.e., prescribed invocation, operational behavior
	Semantic	- cover all changes that are not involved description of services, operations, parameters or return values.
	Interface	- change in the interface signature e.g., parameters, operations, message structure - addition of new functionality or the update of existing functionality - interface changes may affect the implementation, QoS, pre-condition, post-condition, usage of the service
	Business Process/ Workflow	- modify the business process model - change in choreography or orchestration model of service composition

3.3 Positioning Approaches

Existing works for supporting service co-evolution have mainly focused on the following steps: change detection, change impact analysis, and reaction.

(i) Change Detection (CD). Change detection is a critical process in service evolution management. It helps affected services to find out changes and kinds of change that can be used as input data for analyzing the level of impact. Researchers classified evolutionary changes into different types of critical changes. However, in many cases, changes in behavior usually are more complicated and need more effort to adapt by considering the actual values communicated between services and their clients. These approaches (see Table 3.2) mainly focused on service interfaces, workflow, semantic and behavior change. These approaches have provided many tools or frameworks such as: *Vtracker* [71], *WSDarwin* [72], WSDLiff [73], *DiCORE* [74].

Fokaefs and his colleagues [71] work on analyzing WSDL interfaces by building a tool called VTracker [71] to find out the differences between WSDL specifications. Specifically, the authors created an intermediate XML representation to reduce the verbosity of the WSDL specification. However, VTracker does not take into account the syntax of WSDL interfaces. Furthermore, transforming a WSDL interface into a simplified representation can lead to unprecise detection results. Similarly, D. Romano et al. [73] presented a significant contribution with the WSDLDiff tool to find out the fine-grained changes in the WSDL descriptions by comparing the subsequent versions of the WSDL. Unlike VTracker, the framework depends on the schema used to define the data types in the WSDL, besides the syntax of the description language used to extract the changes. Later in other works, M. Fokaefs et al. [72] proposed the WSDarwin tool to recognize the changes in the specification of a service. In these works, the authors did not consider service changes regarding the semantic aspects that differ our approach, which will present in Chapters 5 and 6.

(ii) Change Impact Analysis (CIA). The goal of this process is to understand the relationship between the service and the change. The service users should know which parts of the system will be affected by the change and examine them for additional impacts since the modification in one part of the service may have subsequent effects on other related services. Significant literature works in this area can be classified into two categories techniques: dependency analysis and trace-ability analysis [75]. It is possible to evaluate

the change effects and procedure evolution strategies to reduce risks and maintenance costs through an impact assessment.

The output of existing approaches for this phase, usually are models, techniques or design patterns, such as Trust Dependency Graph [76], Versioning Model [77], Dependency Model [78], Change Pattern [67], DiCORE-CIA [74]. Furthermore, we found that there has been special attention paid to the topic of change impact analysis from the research community.

(iii) Change Reaction (CR). This process involves other steps such as decision-making, propagation of changes, and an optional broader changes context to support other affected services (in case of coordinated co-evolve services), eventually giving a set of prioritized actions to adapt to new changes. Propagation of changes addresses how the impact of a change can be effectively propagated to other entities with a minimal ripple or what additional changes are required for a service to maintain consistency. As new functionality is added or changed in services, developers must ensure that other system entities are updated and consistent in response.

Existing approaches have mainly focused on changes in business process models (workflow). Some important frameworks are DYCHOR [79] and C3Editor [80]. The DYCHOR framework evaluates the change propagation of a process in choreography based on an extended automated model. Simultaneously, the C3Editor visualizes the different models and enables the definition and application of changes to the private models. The C3Editor determines and visualizes the partners affected by a change and the updates required for change propagation. These approaches could effectively support the coordination of change processes. Additionally, these approaches showed that changes in a business process model usually require manual intervention by developers.

3.3.1 Support Service Evolution

In the scope of evolution services, researchers have spent significant effort investigating methods and techniques for the management of service changes. This research can be seen as a precursor and established a critical foundation for research on service co-evolution.

Table 3.2 lists critical approaches related to service evolution aspects such as authors, key contributions, available software (denote Yes (Y) or No (N)), and support processes (e.g., change detection (CD), change impact analysis (CIA) or change reaction (CR)).

In our view, all of these approaches have high-value results in the field of service (co) evolution in recent years.

These approaches could fall into one of the following categories: (I) Tool/Model-based, (II) Versioning-based, (III) Pattern/Adaptor-based, and (IV) Analysis of Change Impact-based. Naturally, some approaches might belong to more than one category; for instance, the work by Khebizi et al. [81] provided a framework, a software tool, and patterns to support dynamic change management of business protocols.

(I) **Tool/Model-based**

One of the first works handling service evolution is developed by Treiber et al. [82]. Their work addressed two main problems related to services: (a) what type of information is required for a particular perspective, how all types of information are integrated into a single model, and (b) how these types of information are managed. The first problem concerns the development of an aggregated, flexible and extensible information model for services. The latter is linked to the development of a management framework that can track historical information.

Romano et al. [73] proposed the WSDLDiff tool that can be used to derive the set of delta changes applied to a service. This tool considers the syntax of the WSDL file and the schema file XSD that is used to define the data types of the WSDL interface. Similarly, Li et al. [83], presented critical empirical studies about the most common types of service changes.

Another important framework comes from V. Andrikopoulos et al. [69], who introduced a service specification reference model and introduced the concept of service evolution management. Based on the type and set theory and the service specification model, the authors developed an approach to recognize the conditions (i.e., a set of changes) under which services can evolve while preserving compatibility. However, their works mainly focus on the preventive evolution model and do not pay much attention to the impact and adaptation aspects [59].

Fokaefs et al. [71] also published empirical results of evolution scenarios and presented the Vtracker. Specifically, the authors created an intermediate XML representation to reduce the verbosity of the WSDL specification. However, Vtracker does not take into account the syntax of WSDL interfaces. An upgraded Vtracker named WSDarwin [72, 84], which can be used to automatically identify changes between different versions of service by comparing interface description documents. WSDarwin tool provided a solution to answer how to support a client application in adapting to changed services. However, WSDarwin does not indicate how the Web Service provider could perform the adaptation assistance or deal with the dependencies when generating and compiling the client stub.

Latter, Zou et al. [59] proposed a change-centric model in which necessary changes are identified, planned, implemented, tested, and then notified to all necessary stakeholders. In the model, the delta is a set of changes from one version to its next version of the service.

Recently, Jahl et al. [74] developed the DiCORE framework that determines the kinds of change, categorizes the changes, and shares them with dependent clients. The framework helps the developers to find out the structural changes in workflow. In summary, these tools and models provided practical ways to address the evolutionary challenges, such as detecting changes and analyzing the change impact, eventually supporting related services during an evolution process.

(II) **Versioning-based**

Versioning is a traditional and practical way to address the incompatibility issue [66, 85]. A robust versioning strategy allows for service upgrades and improvements while continuously supporting previous versions. Leitner et al. [70] presented a comprehensive versioning approach specifically for handling compatibility issues, based on a service version graph and version selection strategies. The proposed framework is used to dynamically and transparently invoke different versions of service through service proxies. In this path, Kaminski et al. [86] outlined various requirements for versioning and demonstrated why common versioning strategies are inappropriate in the context of services. The authors proposed the Chain of Adapters pattern [86] for developing evolving services. However, the adapter does not support the parallel execution of different service implementations. Also, adapter implementations may be faulty and break old clients [85].

In order to fix this faulty and do not break old clients, Weinreich et al. [85] proposed a versioning model for supporting the evolution of service-oriented architectures. The model involves a set of services into a subsystem and assigns them the same version identifier. Even if only one service is changed, all services within the same subsystem will be tagged with a new version number. Consequently, multiple versions of the same subsystem may co-exist. Becker et al. [87] proposed an approach to automatically determine compatibility that could be applied with the compatibility pattern. Similarly, Yamashita et al. [77] introduced a novel feature-based versioning approach for assessing service compatibility and proposed a different versioning strategy, following the W3C standards.

In fact, various versioning approaches are proposed to address the challenges of the service version. At the technical level, these approaches relied heavily on the SOA [66]. In general, they are used together with the design pattern and related tools that will be presented in detail in the next section.

(III) **Pattern/Adaptor-based**

Design patterns and adapters have been widely used for software development and structuring solutions. For instance, Wang et al. [67] focused on a common evolution scenario in which a single provider provides a single service. In particular, the authors proposed four patterns involving compatibility, transition, split-map and merge-map. These patterns provide generic and reusable strategies for service evolution.

It is worth considering the actual work on service compatibility, which aims to assist service consumers in seamlessly transferring their programs to newer versions [87]. Becker et al. [87] proposed an approach to automatically determine compatibility that could be applied with the compatibility pattern.

In case the change is not compatible, the work of Kaminski et al. [86] introduced an adapter-based approach to maintaining multiple versions of service simultaneously. The novel idea of this approach is to use a proxy that enables dynamic binding and invocation for client applications to maintain multiple versions of service on the server-side and [88]. At the same time, Frank et al. [89] distinguished between a service interface (public) and its implementation (private).

To address possible interface mismatches, Dumas et al. [90] suggested an algebra over interfaces and a visual language that allows pairs of provided-required interfaces to be linked through algebraic expressions. Benatallah et al. [91] proposed adapters approach using mismatch patterns, which may capture the possible differences between two interaction protocols.

Similarly, H. R. Motahari Nezhad et al. [92] provided semi-automatic support for adapter generation to resolve interface mismatch and deadlock-free interaction incompatibility. Following previous works of Benatallah et al. [91], Ryu et al. [93] studied the protocol compatibility using path coverage algorithms based on finite state machines (FSM) service model and suggested adapter/ad-hoc protocol as solutions. Using the FSM service model is an easy way to understand inexperienced users and is suitable for representing reactive behaviors.

In research work [94], the authors tried to keep client applications being synchronized with evolved services through semi-automatic client updates. Their proposed tool first analyzes the delta between different versions of service and exports them into a well-formatted document. After that, it draws on its client's usage history. Next, they employ a consumer code customized component to highlight the client code fragments that need to be updated. In the same path, Ouederni et al. [95] introduced a framework to resolve interface and behavior mismatches by automatically updating the clients based on compatibility measuring. The update process can be parameterized with some user requirements to prevent the designer's behavior from appearing in the client interface.

(IV) **Analysis of Change Impact-based** Basu et al. [96] proposed a technique to extract dependencies from log files. The technique could be adapted to infer the number of dependent service consumers. Once the dependencies are transparent, it is also essential to infer the impact of service changes on the applications.

Later, Wang et al. [97] proposed his dependency model in analyzing the impact of service evolution. The authors considered the dependency model to analyze the dependency links among services that work in collaborations. This model extracts the degree of dependency for each link between the elements in one service or between services. It is a foundation for most of the later studies in this field. The dependency model proposes a matrix to describe the dependency relations between services and elements in one or more services. However, the disadvantages are also obvious: (a) The model assumes that the dependencies are known at design time, which is invalid in the dynamic environment; (b) The model does not distinguish the change types add, remove and modify. It considers that each type of change results in the same impact. Additionally, the dependency model does not explain how and where to obtain the changes, which is vital to service evolution.

Yamashita [77] presented an impact analysis based on usage profiles. This method helps service providers to estimate the impact on consumers as well as giving an evolution decision later. In the same fashion, Liao et al. [76] proposed a trust dependency graph that is introduced to analyze the impact on the trust of component services. Jahl et al. [74] also provided the framework DiCORE to analyze the impact of changes based on the patterns. This framework allows an intuitive and graphical formulation of patterns while other existing tools completely ignore user-defined change patterns.

Table 3.2: Existing Approaches Support Service Evolution

ID	Author/Reference	Key Contribution	Focus on /Available Software	Process
1	Papazoglou & Andrikopoulos [68, 98]	Classify and analyze shallow and deep changes A set of theories and models that unify different aspects of services (description, versioning and compatibility)	All kind of changes/ N	CD,CIA, CR
2	Treibe et al. [99]	SEMF framework to manage the changes on interface, interaction patterns or QoS aspects	Requirement change, Interface change, Implementation change and QoS change / N	CD, CR
3	Romano et al. [73]	A tool to extract changes automatically	WSDL document change / Y	CD
4	Forkaefs et al. [71]	Vtracker tool identifies changes between different version of a service	WSDL Interface change /N	CD
5	Forkaefs et al. [72, 84]	WSDarwin tool supports the clients to coevolve with the provider service	WSDL, WADL interface change / Y	CD
6	Wang et al. [67]	A service evolution model A method to analyze service dependencies Proposed four patterns: compatibility, transition, split-map, merge-map	Operation and data type change / N	CIA
7	Basu et al. [96]	A tool can extract dependencies from log files	Protocol change / Y	CIA
8	Kaminski et al. [86]	A chain of adapters technique approach for deploying multiple service versions	WSDL Interface change /N	CD

Table 3.2: Existing Approaches Support Service Evolution

ID	Author/Reference	Key Contribution	Focus on /Available Software	Process
9	Zou et al. [59]	A change-centric model and a method for the change impact analysis	WSDL interface change / N	CD, CIA
10	Ryu et al. [93]	A framework supports business protocol	Protocol change / N	CD,CIA
11	Leiner et al. [70]	End-to-end versioning support based on service history	Interface change /N	CD
12	Frank et al. [89]	A service interface proxy for dealing with the incompatibility	WSDL Interface change / N	CD
13	Dumas et al. [90]	An interface adaptation method, in which each interface is represented as an algebra expression that could be transformed and linked accordingly	Interface change / N	
14	Benatallah et al. [91]	Adapters as an approach based on mismatch patterns which captures the possible differences between two interaction protocols	Protocol change / N	CD
15	Jahl et al. [74]	Framework DiCORE to determine the kind of changes and the affected components in a business process	Business process change / Y	CD
16	Tran et al. [14, 15]	An approach detects and extracts interface changes	WADL file change / N	CD

Table 3.2: Existing Approaches Support Service Evolution

ID	Author/Reference	Key Contribution	Focus on /Available Software	Process
17	Wang et al. [78, 97]	A dependency impact analysis the kind of changes and the affected causes and effects of changes	Operation and data type change / N	CIA
18	Jahl et al. [100]	An architecture captures and assesses the behavior dimension of services	Behavior changes / N	CD
19	Becker et al. [87]	A versioning framework determines compatibility based on patterns	Interface change /N	CD, CIA
20	Motahari et al. [92]	A semi-automatic method for adapter generation for the mismatches at the interface-level, protocol level	Interface change, Protocol change / N	CD, CIA
21	Yamashita et al. [77]	A change impact analysis approach based on usage profile	Operation and data type change /Y	CIA
22	Li et al. [83]	A method for change impact analysis Proposed 16 API changed patterns	Operation and API changes / N	CIA
23	Weinreich et al. [85]	A versioning model	Implementation change, Interface change / N	CD
24	Khebizi et al. [81]	A framework with migration patterns	Protocol change / N	CD
25	Liao et al. [76]	A trust analysis model	Interface change, Binding change / N	CIA
26	Ouederni et al. [95]	A framework to resolve the compatibility; An interface model	Internal behavior change Message parameters change / N	CD
27	Le et al. [94]	A framework and model	WSDL interface /N	CD

Table 3.2: Existing Approaches Support Service Evolution

ID	Author/Reference	Key Contribution	Focus on /Available Software	Process
28	Olga et al. [101]	A framework for managing semantic, syntactic and protocol changes	semantic, syntactic, protocol /N	CD
29	Fang et al. [88]	A version-aware service description model; A version-aware service directory model	Implementation and interface change Binding change /N	CD
30	Thanos et al. [102]	A framework for semantic drift	Semantic change /Y	CD
31	Juric et al. [103]	A model to support versioning interfaces	WSDL interface, Message parameters change /N	CD
32	Labbac et al. [104]	An approach for handling services evolution.	Semantic change /N	CD

3.3.2 Support Service Co-evolution

The need for advancements in service co-evolution support is undisputed, just as for all software. One of the most challenging aspects of a service co-evolution is the ability to co-evolve services together in order to retain compatibility and bindings. Obviously, support service co-evolution also means support service evolution since it is a particular case of service evolution. In practice, a service co-evolution process requires further co-evolve steps, e.g., coordination among inter-dependency parties. Even though the coordinated distributed service evolution plays an essential part in SOA environments, there is currently a lack of attention being paid to this kind of service evolution.

This section highlights some valuable research works that investigated service co-evolution as the following aspects:

One of the first works handling co-evolve changes, closed to service co-evolution, was developed by M. Papazoglou [68]. The author proposed a fundamental classification of evolutionary changes in the service-based system. Their work also introduces two types of service changes, one of them called deep-changes. In our thesis, we concentrate on coordinated deep-changes in large-scale service computing scenarios. However, M. Papazoglou did not consider the special scenario of service evolution, where several services have to co-evolve together to retain compatibility.

• *Requirements for service co-evolution:* It is worth mentioning the research by De Sanctis et al. [9] who first proposed eight requirements for service co-evolution. However, the authors mainly focus on general requirements. Even though the authors do not analyze these requirements in-depth, they induce a small step toward service co-evolution. Furthermore, the authors presented a solution for service co-evolution based on the Domain Object concept, which supports deep-changes across a service dependency graph through the decentralized collaboration of evolution agents.

• *A distributed knowledge-based evolution model:* Wang et al. [105] proposed a distributed knowledge-based evolution model named *DKEM* to promote competition and collaboration between services from different vendors. The model regards stability as a key factor in competition, and a stability evaluation model is intended to calculate the stability of services, vendors, and service-based processes. Based on the evaluation model, two evolution patterns are specified with which new, and more stable cooperation among services can be examined automatically by utilizing a runtime self-adaption mechanism. The authors reported that the model *DKEM* is effective for competition and cooperation among services with distributed knowledge, and evolved processes have higher stability and reaction efficiency. Nonetheless, the model is designed to eliminate semantics conflicts between different vendors with different ontologies.

- *Process of co-evolution using meta-models in model-driven engineering:* Cicchetti et al. [106], presented atomic changes and defined the process of co-evolution. The authors also created a differential meta-model with the identified changes. However, in this approach, a number of open questions remain, including issues of systematic validation of the dependency detection, change impact analysis and resolution technique, which necessarily encompasses larger population models and meta-models.

In summary, we have explored the related contributions in the field of service co-evolution and service evolution. Furthermore, we highlighted some practical approaches that closely focus on service co-evolution, which led us to conclude an urgent need for coordinated service evolution. In the following chapters, we will present the main contributions of this thesis.

4 Service Co-evolution Architecture

This chapter describes a vision of service co-evolution in IoT. Moreover, the chapter proposes a novel agent architecture that supports the evolution by controlling service versions, updating local service instances, and enabling the collaboration of agents. In this way, the service co-evolution can make systems more adaptive, efficient and reduce maintenance costs. The content of this chapter is presented at the First International Summit, IoT360 2014, Rome, Italy, October 27-28, 2014. The main results are published in [12, 13].

4.1 Introduction

The envisioned Internet of Things (IoT) foresees a future Internet incorporating smart physical objects that offer hosted functionality as IoT services. This service-based integration of IoT will be easier to communicate with and more valuable for enriching our environment. However, the interfaces and services can be modified due to updates and amendments. Such modifications require adaptations in all participating parties. Therefore, this chapter aims to present a vision of service co-evolution in IoT.

In this chapter, we address the challenge of coordinated services in IoT by employing an agent-based approach. Service providers may depend on third-party services to deliver quality products to customers and other service providers. A centralized solution would not be realizable due to administrative and technical reasons. It would not be scalable, in particular, in the area of IoT, and security issues would complicate the whole approach. Consequently, service providers have to be responsible for the evolution of their own services. The required actions have to be coordinated with other providers in the IoT environment. The objective is to automate the coordinated evolution as much as possible.

Service co-evolution in IoT has received barely attention so far. Thus, there are some needs for detailing the vision of service co-evolution and solutions to provide benefits for IoT users. However, there are many challenges and requirements to tackle to meet an overall trade-off between aspects like clients' satisfaction, the resource consumption of provided interface versions, and the efforts to update them. Consequently, this chapter will analyze the roles of this evolution regarding management requirements, and the solution.

This chapter aims at promoting the idea of coordination of service co-evolution in IoT environments by highlighting a novel agent architecture in the IoT environment. Furthermore, it explains how these agents can be used in service co-evolution environments. Thus, the main contribution of this chapter is to make software engineers aware of the power of service co-evolution and make systems more adaptive, efficient, and reliable.

The rest of the chapter is structured as follows. Section 4.2 illustrates an overview of our approach and its key components. Section 4.3 analyzes the coordination of services. Finally, Section 4.4 draws conclusions on our results.

4.2 Approach Overview

Services running on heterogeneous systems and offered by different providers have decoupled lifecycles, in particular, in IoT. Single services will be updated due to amendments or refinements or to provide further functionalities. Other providers may cut back the functionalities without taking notice of remaining clients that try to apply the removed functions. Business processes and applications that depend on services require appropriate coordination and adaptation by the participating parties. The solution we worked out equips every service with an agent, called EVA (Evolution Agent) that is capable to undertake these tasks. The internal structure and the rough composition of an EVA are depicted in Figure 4.1. The next sections introduce the main components of an EVA and their interactions.

4.2.1 Analysis

The flow of information interaction within our model is as follows. When an EVA receives first an Evolution Request, it is analyzed by the Analysis module. An Evolution Request demands adaptation to be able to take part in future interactions. In case a service

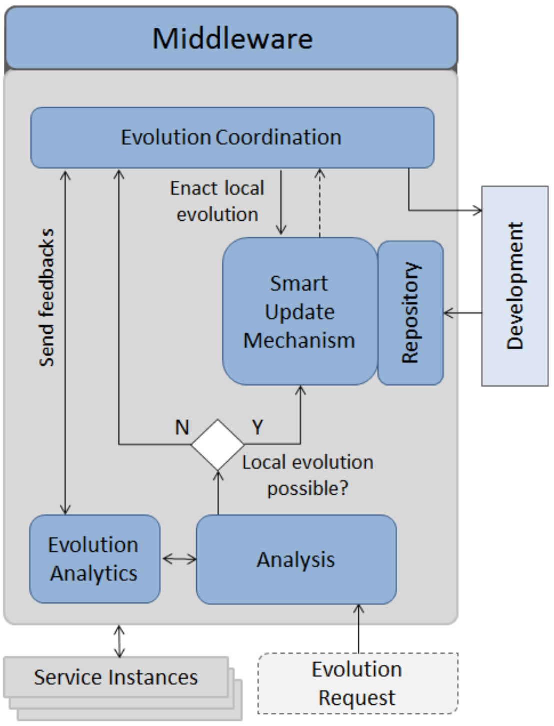


Figure 4.1: Architecture of the EVA

provider wants to update his service, its EVA can send the Evolution Request to the clients' EVAs. A further scenario is that the EVA of a service composed of other services and depends on them demands one of his service providers to evolve to his update service.

In the latter case, the Analysis module has to decide whether evolution should occur and, if so, whether a local evolution is possible or whether the evolution has to be coordinated with other EVAs. For this reason, it assesses firstly the significance of the Evolution Request by evaluating the importance, the reputation, and the number of partners who sent the request. The importance of a partner will increase the more clients are affected by him. The significance will rise too if the local service strongly depends on the other service and no alternative services. If either resource is becoming scarce or takes high efforts to satisfy the request, lowly rated Evolution Requests may be rejected. Service instances not requested for a long-time can be switched off to free resources for crucial service instances.

To estimate the efforts required for adaptation, the Analysis module initially considers local knowledge that includes information about locally available update mechanisms, the different service instances realizing different versions of the service, and the service versions' dependencies towards other services. In case the Analysis module accepts the Evolution Request and a local update would satisfy the request, it will instruct the Smart Update Mechanism module, as presented below, to execute the local update and provide a new service instance eventually. Suppose a pure local update is not available or not sufficient due to the interplay between several services. In that case, the Evolution Coordination module has to deal with a coordinating evolution and possibly ask software developers for further configurations.

4.2.2 Evolution Analytics

As time passes, the Analysis and the Evolution Coordination module can take more sophisticated decisions. The Evolution Analytics module collects runtime data about successful and unsuccessful evolution procedures. These data include information about local and coordinated evolutions since both modules feed the Evolution Analytics module. The goal is to discover promising evolution patterns by fostering successful and proven evolution procedures and preventing unsuccessful ones. Success not only depends on smooth running in a technical sense but also considers the cost-performance ratio, revenue, reputation, and QoS (Quality of Service) parameters. Costs comprise, for instance, hardware and human resources, which can be estimated hardly in the very beginning. If a new configuration has been implemented, the developer specifies the total person-hours spent. Using Evolution Analytics, EVA will learn to predict worthwhile evolutions while minimizing costs and time and maximizing the own revenue and reputation. The reputation of an EVA may decrease if it regularly denies Evolution Requests. Here, Evolution Analytics has to weigh the reputation against other factors like the costs for updates and the future revenue. To estimate reputation, costs, and QoS, we will use our two prediction algorithms presented in [107].

For reasons of bootstrapping, EVAs are allowed to share parts of their knowledge with other EVAs. Particular know-how that affects only the service supervised by the EVA has to be left out.

4.2.3 Evolution Coordination

In the event that a pure local evolution is not applicable, the Evolution Coordination module will co-operate with other EVAs and possibly interact with software developers.

For example, the service is providing a method that depends on data delivered by a third party. To customize the interface for the client sending the Evolution Request, the Evolution Coordination will first determine the involved third parties and send them an Evolution Request. Continuous feedback between the EVAs is required to keep all parties up-to-date and to recognize future developments early. If a third party rejects the Evolution Request or is not available anymore, the Evolution Coordination can search for suitable services. To this end, we will adopt our service selection algorithms proposed in [108].

If the latter fails due to a lack of matching services, the Evolution Coordination will instruct the service provider or a responsible software developer to adapt the service. For this purpose, the developer may implement a configuration that is subsequently executed by the Smart Update Mechanism.

4.2.4 Smart Update Mechanism

The Smart Update Mechanism encompasses mainly two types of evolution capabilities. Firstly, it is aware of the different versions of the services running as service instances on the local machine and the versions used in the past. If one of them is fulfilling the conditions required, then it will be assigned to the requesting party. The second approach is a specification of the evolution rules and constraints representing the possible service re-configurations and adaptations.

An EVA maintains up-to-date evolution models of its services. The models expose the possible configuration and adaptation paths. The EVA may govern multiple instances and versions of the same service simultaneously to accommodate different applications with different needs concerning the service. Eventually, outdated alternatives will be slowly retired.

The Analysis and Evolution Coordination modules introduced in the previous sections decide which configuration or version will be used for a specific client. In this connection, they consider the possibilities offered by the Smart Update Mechanism and take into account the Evolution Analytics to optimize criteria like revenue, reputation, response time, and own operability.

4.2.5 Repository

The Smart Update Mechanism makes use of a repository where several configurations were made available by developers. Developers can add new configurations to the repository during the lifetime of a service, for instance, if the Smart Update Mechanism did not find appropriate ones to update the service.

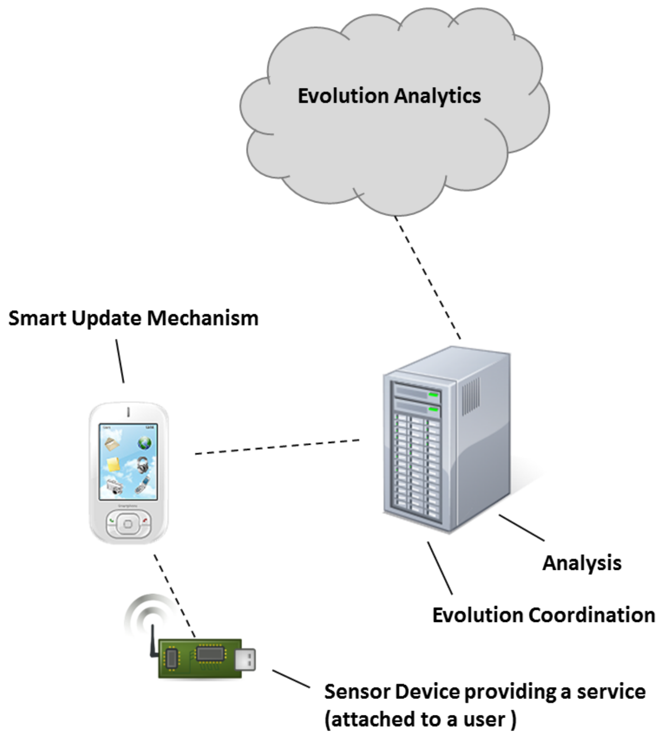


Figure 4.2: Deployment of an EVA's conceptual components

4.2.6 Middleware

Since objects or mobile devices are free to enter or leave the system, the middleware enables EVAs to communicate with each other in an asynchronous and loosely coupled manner. Besides that, the EVA itself can be divided into modules so that each module may run on another device. This allows EVA to use powerful runtime environments while energy-constrained IoT devices that deliver the data offered by the service are spared. Small services, such as an object in IoT, will not have the processing power or storage needed to implement a full EVA. Figure 4.2 shows a low-level object that has outsourced

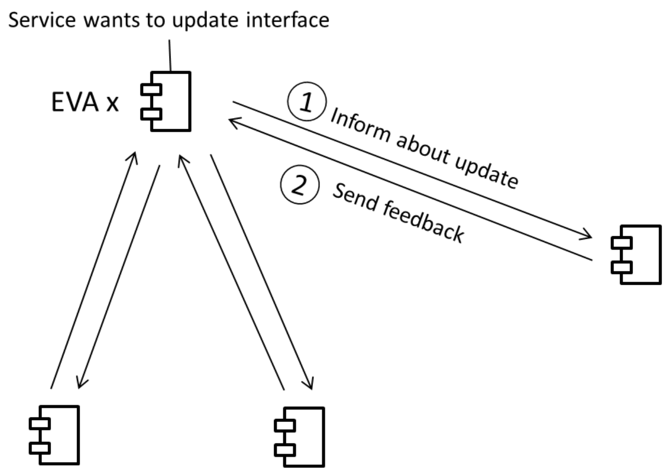


Figure 4.3: Coordination of EVAs based on client's feedback

its EVA components. This is a conservative deployment scenario since we assume that the on-site server and the cloud are always available.

However, more opportunistic approaches are also conceivable. For example, an object might rely on the availability of mobile devices that can enter or leave the system freely to provide the resources it needs to communicate with the cloud.

4.3 Coordination of EVAs

Coordinating the evolution of services is a significant challenge since it is a complex process that requires multiple interactions and continuous feedback to understand whether the distributed evolution is proceeding as desired. To prevent never-ending negotiations between service providers about which service has to adapt first or to change, we introduce an algorithm that gives a clear path for the evolution. Therefore, we include the number of clients of each concerned service and their overall reputation. Figure 4.3 shows the process of taking into account the feedback received in response to an evolution request, particularly from EVA x, which sends the requests to its clients.

To tackle the challenge mentioned above, we define the following terms to coordinate and adapt EVAs.

Weight of a service (w):

The weight of a service is defined as the product of the reputation r of the service (range [0,1]) and its number of clients n_c (scaled into the range [0,1] by incorporating the max and min values of all services considered). The reputation r of the service and the weight w of service can be defined as follows:

$$r = \frac{\sum_1^n rating_i}{n} \tag{4.1}$$

$$w = r \times n_c \tag{4.2}$$

Vote of a service (v): The EVA that is managing an affected service is either interested in an adaptation or rejects it. For this reason, an EVA can vote for or against the evolution of a used service. Hence, we adopt the values of votes:

- Vote $(v_i) = +1$ votes for accepting the new service version.
- Vote $(v_i) = -1$ votes for not updating the interface or do update but keep the old version.

Feedback (f): The higher the reputation of a service and the higher its number of clients, the higher the vote of the EVA that is managing the service is weighted. Thus, the overall feedback is comprised of the multiplication of the vote and the weight that consists of the reputation and the number of clients. This means that services that satisfy and affect more clients have a higher impact.

Feedback of one client (f_i):

$$f_i(w_i, v_i) = w_i \times v_i \tag{4.3}$$

Feedback of all n clients (f_{agg}):

$$f_{agg} = \sum_i^n w_i \times v_i \tag{4.4}$$

4.3.1 Coordination Algorithm

The co-evolution will be executed if

$$f_{agg} \geq \epsilon(threshold\ value) \tag{4.5}$$

A step-wise structure of the proposed algorithm that encompasses the equations from (1) to (5), is given in the following:

Input: Evolution request of EVA x to EVAs $c \in C$; number of clients and the reputation of the EVAs $c \in C$.

Step 1: The service managed by EVA x will be updated by the provider or EVA x received an evolution request from another EVA y.

Step 2: x is asking the EVAs $c \in C$ of its clients whether they would accept or reject the required adaptation.

Step 3: x is summing up the feedbacks of $c \in C$ by considering their vote and their reputation and number of clients that are both scaled into the range [0,1].

Step 4: x is dividing the summed-up feedbacks by the number of clients to obtain f_{agg} and compares f_{agg} with a predefined threshold value ϵ.

Step 5: The co-evolution will be executed if $f_{agg} \geq \epsilon$. In this case, the update mechanisms will be executed.

Step 6: Otherwise, the evolution requests will be rejected.

Output: Accept or reject the evolution requests

4.3.2 Optimization Problem

Hence, a service may be composed of several other services and, hence, stay in contact with different EVAs. This scenario is similar to that of business processes in SOA, where a central process orchestrates Web services to realize a particular functionality.

In service co-evolution, we are searching for the optimal interface. Service may be a composition of multiple EVAs where each EVA represents a Web service. The orchestration itself is monitored and managed by an EVA. It tries to optimize the own revenue and the rate of satisfaction of clients by providing a suitable interface for a given functionality. This interface is built by a composition of the interfaces of other EVAs (services). A selection algorithm aims to find first an optimal choice of interfaces that realize the most preferred interface for the orchestration. This is done by the aforementioned coordinating algorithm. With EVA x, for instance, we define the set I that contains the set of requested service interfaces. The set I depends on the clients who vote whether they would accept or reject the interface of a composition of certain interfaces.

After finding the set I of matching interfaces, the challenge is to find those services that optimize the orchestration's overall quality and the revenue for the provider. Finding the optimal solution means now maximizing the overall satisfaction (S) of the own clients and selecting those EVAs that will provide the required interfaces and maximize the own revenue (R).

We consider the following objective function for realizing the EVA orchestration:

$$Maximize F_{obj}(I) = F_{obj1}(R) \times F_{obj2}(S) \tag{4.6}$$

$F_{obj1}(R)$ and $F_{obj2}(S)$ are explained below. $F_{obj1}(R)$ is an objective function for maximizing the provider's revenue, $F_{obj2}(S)$ has the goal to maximize the clients' satisfaction.

We incorporate the following quality dimensions to compute the overall satisfaction for the clients:

- *throughput*(t) : number of service invocations per time unit.
- *reliability*(l) : the probability that the service executes successfully.
- *execution time*(e) : the time it takes to execute the service.
- *availability*(a) : the percentage of time during which the service is available.

The vector Q (t, l, e, a) contains the quality of service (QoS) dimensions.

The value of functions F_{obj1} and F_{obj2} are weighted and combined to make the final decision. To maximize $F_{obj}(I)$ in (6) both objective functions below has to be maximized:

$$F_{obj1}(R) = \sum_{i=1}^{n} r_i, r_i \in R \tag{4.7}$$

$$F_{obj2}(S) = k_1 \times t + k_2 \times l + k_3 \times \frac{1}{e} + k_4 \times a \tag{4.8}$$

Therefore:

$$F_{obj}(I) = (\sum_{i=1}^{n} r_i) \times (k_1 \times t + k_2 \times l + k_3 \times \frac{1}{e} + k_4 \times a) \tag{4.9}$$

To keep the description as simple and clear as possible, the normalization steps to the range of [0,1] are not included in the formula. The factors $k_i, i = 1...4$ represent the weights for the quality dimensions depending on the preferences of the EVA. Let us assume that the current interface of EVA x can provide a revenue of R_0 and an average satisfaction of (t_0, l_0, e_0, a_0). The goal of interface selection would be to find a pair (R_i, S_i) better than the pair (R_0, S_0).

In the event of pure Web service selection for business processes, this issue can be solved by our approach in the work [108] or by other popular approaches like integer linear programming [109] and genetic algorithms [110].

4.4 Conclusions

This chapter introduces a new vision of service co-evolution in IoT. In particular, it adopts a novel conceptual agent as a solution for service co-evolution. In addition, the chapter also proposes an approach for coordinating EVAs in service co-evolution. In this way, system can be made more adaptive, efficient and reduce costs to manage maintenance.

However, the coordination of EVAs needs a notification management architecture that can enable notifying all participating parties in the network in case of updates or amendments of services, especially in IoT environments. Therefore, we will present a solution for this issue in the next chapter.

5 Notification Management Architecture

This chapter proposes a novel notification management architecture that detects service description changes and notifies all affected participating parties in the network. The main content of this chapter is reported at the IEEE MESOCA symposium [14] in Raleigh, NC, USA, 2016.

5.1 Introduction

In the IoT area, services can be modified during application lifespan due to updates and amendments. Hence, the third-party applications and other client services, dependent on changed services, need to take appropriate coordination and adaptation actions. An effective way to tackle such scenarios is to provide a notification management mechanism to inform third-party applications about a service change. For this to come to fruition, there should be an appropriate way to describe the service changes within limited resource capabilities. With their limited resources, a significant challenge in IoT systems is to store the list of clients and find the target clients to send the service change information.

Existing solutions focus on how to adapt to these changes without considering coordinated co-evolution services within resource constraints. Our solution overcomes this limitation by proposing a notification management mechanism to support the communication of updates between service providers and service consumers on-the-fly through the evolution of dependent services. To achieve this, we employ a service registry. Furthermore, we implement the notification management on a simple scenario to examine our approach. The test results show that our proposed notification management architecture can be considered as a promising solution for notifying changes in IoT environments.

This chapter provides a solution to overtake research questions mentioned in Section 1.2. In particular, the chapter focuses on a notification management architecture for co-evolution of IoT services. Furthermore, a case study is implemented to show the feasibility of the proposed solution. The rest of the chapter is organized as follows. In Section 5.2, we depict a motivating example for IoT services. In Section 5.3, we analyze service description to detect the kind of changes. In Section 5.4 we present our proposed notification management. Section 5.5 provides a case study for evaluating our approach. Finally, some concluding remarks are given in Section 5.6.

5.2 Motivating Example

In the following scenario, we want to emphasize a notification management architecture's vital role that IoT service providers need to inform their clients about changes.

The example in Figure 6.1 illustrates the problem in the context of an IoT scenario. A sensor node located at A (S1) is providing the temperature and humidity via a REST service. Its clients may be servers, laptops, android phones, and other devices. These different clients can access this service to use the data directly, e.g., C3 and C4, or process it and provide it to other clients, e.g., client S2 (C1, C2, ..Cx).

Someday, the temperature provider updates the service so that it does not deliver temperatures in Celsius but in Fahrenheit, which could be categorized as change in semantics. Hence, it is necessary to check the effect of updates to the clients in the network. The clients cannot replace the service by integrating other services since the other sensor nodes are deployed in other rooms and consequently measuring other locations. A solution could be given by agents, called EVA, that is presented detail in Chapter 4. The EVAs emulate the old version of the service by combining the updated service S1 with another service S2 that converts Fahrenheit to Celsius. The client applications continue to work with the old version provided by the agent. This step circumvents interruptions and gains time for manual adaptions by developers.

Obviously, service providers also do not want to lose their clients. Hence, service providers will try to inform their clients about changes. The first idea would be to keep the list of clients on the IoT device exposing the changed service. Then, the clients will be informed by the device whenever a change takes place. This means service providers need more storage and bandwidth, especially in case of a growing number of clients in the network as their addresses have to be stored and updated information has to be sent to all of them. However, with the resource limitations of typical IoT devices, it is not feasible to maintain such a growing list. Thus, service providers' major challenge is to store the list of the targeted clients so that the service change information could be shared with them.

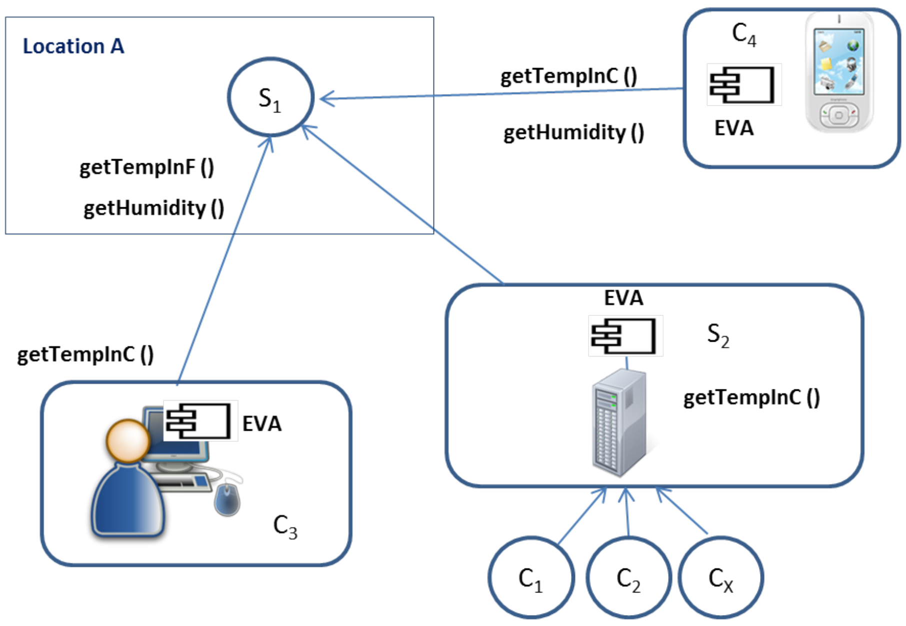


Figure 5.1: A motivating scenario in IoT environment

5.3 Change Detection based on Service Description

5.3.1 Analyzing Service Description

We assume that the IoT services have a standard of WADL interface specifications for REST services in this work. To inform the changes, service notification management must detect changes based on comparing IoT service descriptions described in WADL files. WADL files are XML-based and specify the complete interface of a web service. They describe data as resources and all the operations that can be invoked on these resources in HTTP methods (e.g., GET, POST, PUT, DELETE). In case it finds out the differences between WADL files, it will inform the change to its clients. The comparison will help the developer infer the changes in service from one version to another. However, it will make it easier to immediately and accurately understand the change impact on service clients since the WADL reflects how a client may use the service.

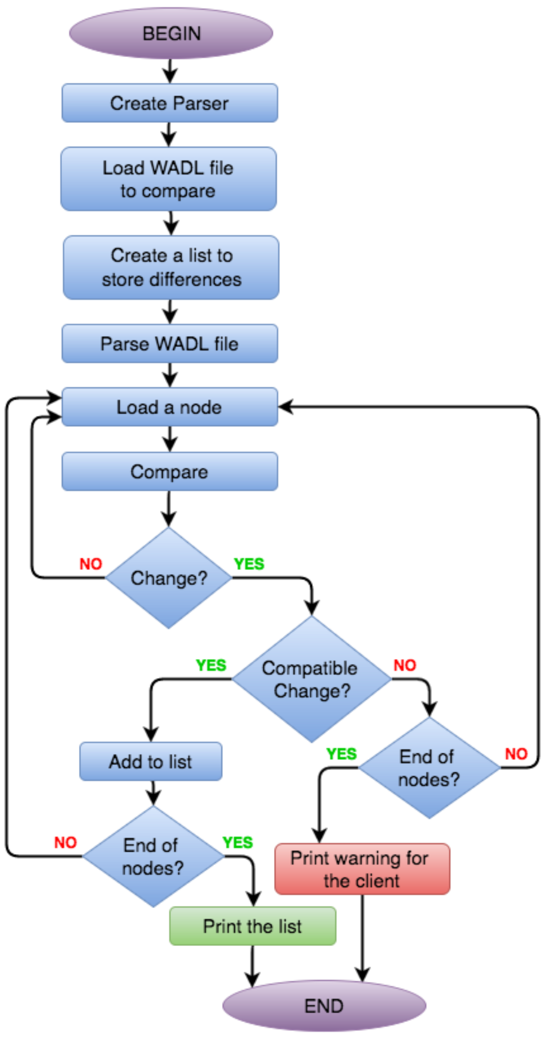


Figure 5.2: Detection of compatible changes based on service description

5.3.2 Detecting Changes

Our idea is to develop an algorithm that is integrated with our notification management component. We adopt this algorithm from an open-source toolkit called Membrane SOA Model [1] - a Java API for WSDL and XML Schema to compare and analyze WSDL files. This toolkit automatically loads WSDL data of the services and saves it on a local drive. After a certain period, the current WSDL file is loaded and compared to the previous WSDL file. Then this tool can detect if a change happens or not. However, this tool works only with WSDL formats.

[1]http://membrane-soa.org/soa-model/index.htm

In our case, we work on WADL files. Figure 5.2 illustrates the flowchart of the comparing process. In the scope of this work, the algorithm may indicate which elements are added, removed, or changed in e.g., WADL operations, HTTP method, XML schema types. These types of change are considered as compatible changes. [10].

Figure 5.2 is a flowchart explaining how the changes in two different WADL data of a particular web service can be detected by parsing them. The work flow is as follows: after an intended parser file is created, two WADL files with changes in their description after service evolution are loaded to compare the differences. The way to create parser files similar to Membrane SOA Model. Hereby, each entity in the description (for e.g., data type) is called *node*. To store the differences, a *list* is created beforehand to add the changes at the end. Two WADL files are parsed to detect the desired nodes.

Service changes that occur at the peer's side and their interdependencies can be classified into shallow changes and deep changes [10]. Confining our work to the shallow changes, it is of utmost importance to classify whether the service evolution that occurred is *compatible* or not. Some service providers obey the *compatibility* issue by adding or changing the evolution processes, like interface, operation or implementation methods etc., and also by adding a new WADL operations and/or new XML schema types to an existing WADL description, guaranteeing the interoperability with prior notification to their clients. Other providers may reduce either parameters, functionalities, or semantic changes without notifying the dependent clients, in which case it is *incompatible*.

5.4 Proposed Architecture

In this section we present a notification management architecture and its components propagating the update information to affected clients.

5.4.1 Deploying the EVA

In IoT environments, many typical sensor devices, such as Z1 mote [2], Arduino BT [3], or Waspmote [4] possess severe memory constraints, featuring less than 10KB of RAM and 130 KB of ROM, resulting in limited computing power [53]. Thus, change management

[2]https://zolertia.io/
[3]https://www.arduino.cc/en/Main/ArduinoBoardBluetoothNew
[4]https://www.libelium.com/iot-products/waspmote/

in IoT environments must be endowed with resource-efficient mechanisms that relieve service-providing IoT devices. For this reason, the idea of IoT devices being managed by an externally hosted EVA is considered.

The EVA can be deployed to a powerful gateway node or in the Cloud. An EVA needs memory and storage to create, distribute, and process incoming update messages. Therefore, it consists of components for Analysis, Evolution Analytics, and Evolution Coordination. Only the so-called Smart Update component [13] runs on the IoT devices and ensures that the external EVA can access and manage them. Hence, in the context of service co-evolution, each IoT system will base on the Cloud or other local computers to be able to cater to a growing number of clients in the network. One EVA may monitor and manage several IoT services belonging to one provider. For further detail regarding EVA architecture, readers can refer to Chapter 4.

5.4.2 Service Registry

In the SOA area, a service registry plays a pivotal role as any service can be dynamically added, removed, or modified. A service registry is often trivial database, which makes it a tedious process to update every time a new service provider wants to register itself into it. The service registry contains a list of services provided by different IoT devices in the network. As the IoT devices shall be accessed by clients, they should be registered in the registry. When a client wants to use a service, it inquires the registry about the service it needs (e.g., temperature). The registry checks its database for all the service providers that provide the requested service. The registry then sends the relevant list of all service providers to the client. Upon receiving the list of providers, the client chooses one from the list and requests the service. On the other hand, after the client sends a request to the service, the sensor node has the information regarding the client and binds with it. In case of any service updates, the provider can inform the clients about the relevant service changes from now on, as it has the required information about its clients.

5.4.3 Communication Mechanism

As can be seen in Figure 5.3, an EVA is deployed in a gateway node and responsible for a group of IoT devices on the left-hand side. We assume in our scenario, the services are provided by these IoT devices. However, the notification system is not implemented

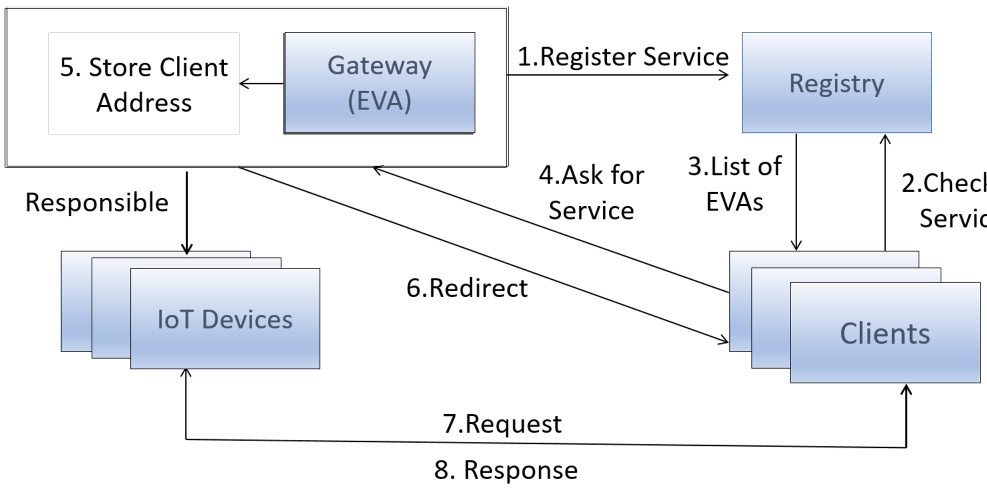


Figure 5.3: Notification Management for Service Co-evolution

by the resource-constrained IoT devices, but the EVA is located remotely. This EVA is responsible for the services and service updates by the sensor nodes it is managing.

Since EVA is not constrained in terms of memory or processing power, thus, it can handle the problem of a growing number of clients. The EVA has the information of the IoT devices' services and registers these services at the service registry (step 1). This enables potential clients to search for concrete services and get a list of suitable IoT services (step 2).

However, in our case, the service registry is not returning the address of the actual IoT service but the address of the EVA responsible for that IoT service (step 3). This enables the responsible EVA to take note of the client and register in the list of clients of the requested IoT service (steps 4 and 5).

In step 6, the EVA on the server will redirect the client to the real IoT service so that it can be used directly (steps 7 and 8). In this way, the IoT device does not need to maintain the list of clients.

The provider's EVA only informs those clients about updates on IoT services that are indeed affected. Furthermore, clients may also have clients which are using their services. As soon as a client is indirectly affected by a service update, it will be informed through this mechanism as well. Update notifications will encompass the whole chain of affected parties, so that in case of cycles, the initial causing service provider recognizes that.

5.5 Implementation

We have implemented the notification management using EVAs on Apache Tomcat [5] on Raspberry Pi 3, Apache Fuseki [6] on a server, REST services on Z1 motes and Android devices as clients. This section will explain the details of our implementation.

In our scenario, both the service providers and the clients implement a REST interface in order to communicate with each other. The service is registered at the registry and hence it can be accessed by the clients. Our scenario is depicted in Figure 5.4, consisting of Raspberry Pi 3 and several Z1 motes. These Raspberry Pi 3 work as border routers but have also EVAs installed. Z1 motes play as different IoT services for measuring, e.g., temperature, humidity, and brightness.

Whenever a client (e.g., Android device in Figure 5.4) requests a service by specifying, e.g., the sensor type and region, the registry gives a list of providers offering a suitable service. Upon reception of the response, the Android client requests the provider for the service. But, as the current provider is an EVA and not the real IoT device, the EVA adds the client to its list of clients for notifying it about any changes of the requested IoT service. The provider redirects the client to the real IoT service with a redirection message which is nothing but the HTTP redirection. Through HTTP redirection, the client is notified about the address of the real IoT service, and the client can finally use the service it has asked for. In this way, the built-in constraints of IoT devices are faced by grouping several devices under one external EVA. During the experiment, we changed the return type of the temperature reading operation from Float to String and passed the new service implementation to the responsible EVA of the IoT device. The EVA took as additional inputs the updated descriptions and executed the algorithm presented in section 5.3. The change detection identified the new return type as an incompatible change and noted down the update's operation and resource. Then, the client on the Android device was informed about the incompatible change. Finally, the EVA deployed the new service implementation on the Z1 motes.

The implementation has shown that our approach is a resource-efficient solution for IoT environments. The list of clients and finding the target clients to send the service change information to were executed externally. This notification management architecture plays an essential part in coordinating services in IoT environments.

[5]http://tomcat.apache.org//
[6]https://jena.apache.org/documentation/fuseki2/

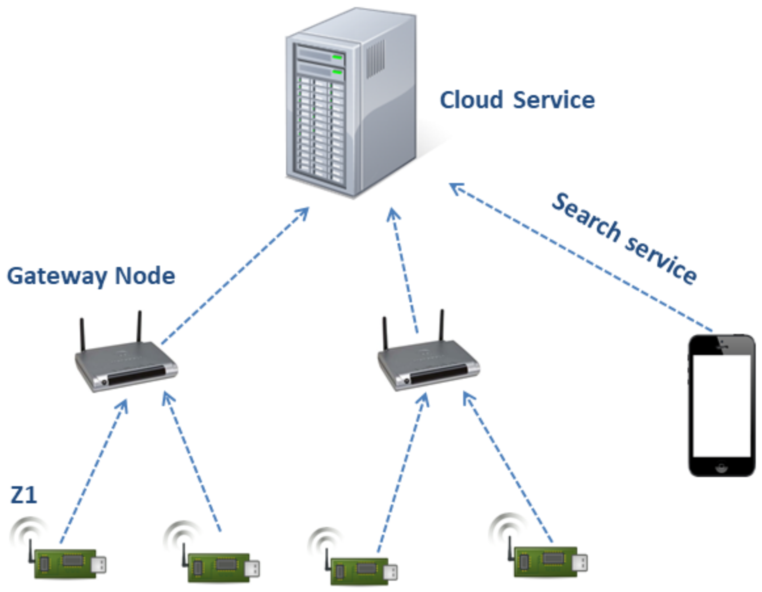


Figure 5.4: Implementation Scenario

5.6 Conclusions

This chapter presents a novel notification management architecture that can enable notifying all participating parties in the network in case of updates or amendments of services in IoT service co-evolution. This is the first step to support affected peers to adapt to changes towards a coordinated services environment.

The notification management architecture is implemented to cater to the growing, dynamic networks in IoT, particularly in resource-constrained devices providing services. However, this chapter mainly emphasizes the need for a notification mechanism and does not dive into change detection aspects. In the next chapter, we will present a comprehensive framework named DECOM that allows detecting syntactic as well as semantic changes in IoT services.

6 DECOM: A Framework to Support Evolution of IoT services

This chapter presents a comprehensive framework called DECOM for automatic detection and communication changes. The content of this chapter is reported at the ACM SoICT conference in Da Nang, Vietnam, December 06-07, 2018. The main results are published in the ACM Proceedings [15].

6.1 Introduction

As all software, IoT services evolve over time to include enhancements and to increase their value to meet the requirements of their service consumers. Often service providers undergo necessary updates like supporting new technologies, discharging obsolete functionalities, bug fixes, and improvement in the quality of the provided service [70, 98, 111]. When a service changes, the client that depends on this service also needs to adapt. However, a service may be modified without notification and updates may lead to service interruptions [14].

Awareness of service changes is an essential ingredient of a reliable IoT environment. As changes are unavoidable, appropriate mechanisms are required to detect and communicate them precisely and automatically. Both aspects have received limited attention so far and remained a critical challenge in heterogeneous IoT domains. Our effort in this work focuse on enhancing the reliability of IoT services through comprehensive change detection. Particularly, we aims to detect semantic changes in IoT services by employing ASP [2] in stead of Web Ontology Language (OWL) [112].

As mentioned before in the foundation chapter, one of the significant advantages of ASP is that knowledge or given default can be changed and extended by new information without causing inconsistencies [2].

Existing approaches like OWL have some limitations. For example, OWL does not support general-purpose rules, which are seen as an essential paradigm in knowledge representation. Furthermore, OWL is monotonic. This means that other parties can adjust a referenced ontology, but the OWL ontologies cannot be used as a common language. Additionally, creating and processing semantic service descriptions is time-consuming and requires a deep knowledge of the applied logic, explanations, and tools. Therefore, this chapter presents a comprehensive method to describe and detect services for dynamic and heterogeneous IoT environments.

Our evaluation examines the run-time performance and proves the suitability for highly dynamic environments. Thus, the main contributions of this work are: (1) an approach to describe IoT services semantically and syntactically with support for a shared knowledge base; (2) a comprehensive change detection to enhance the reliability of the heterogeneous and dynamic IoT service environments; (3) an evaluation with realistic scenarios to measure the run-time performance and to prove the practicability of our approach.

The rest of the chapter is organized as follows. Section 6.2 introduces an example scenario with service changes. Section 6.3 explains the key building blocks of our approach. In Section 6.4, we describe our implementation as well as its evaluation. Finally, some concluding remarks are given in Section 6.5.

6.2 Motivating Example

It is considering a health care application named FitService that is implemented as a RESTful Web service. The aim of this application is to provide many health-related services to its customers. The application involves many indicators such as heart rate, basal body temperature, and respiratory rate besides some flourishing value-added services like nutrition-related information, health specialists, training plans provided by third-party services.

Figure 6.1 shows a flowchart diagram for the FitService application. The green nodes represent the invocation of third-party services. This scenario can be well established by deploying some smart devices providing the functionalities to consumers. In case there is any disturbance in the normal health readings, warnings are displayed to the user.

Let us assume that in order to improve the functionality provided by the third parties, any of these REST services could evolve independently without notifying the customer

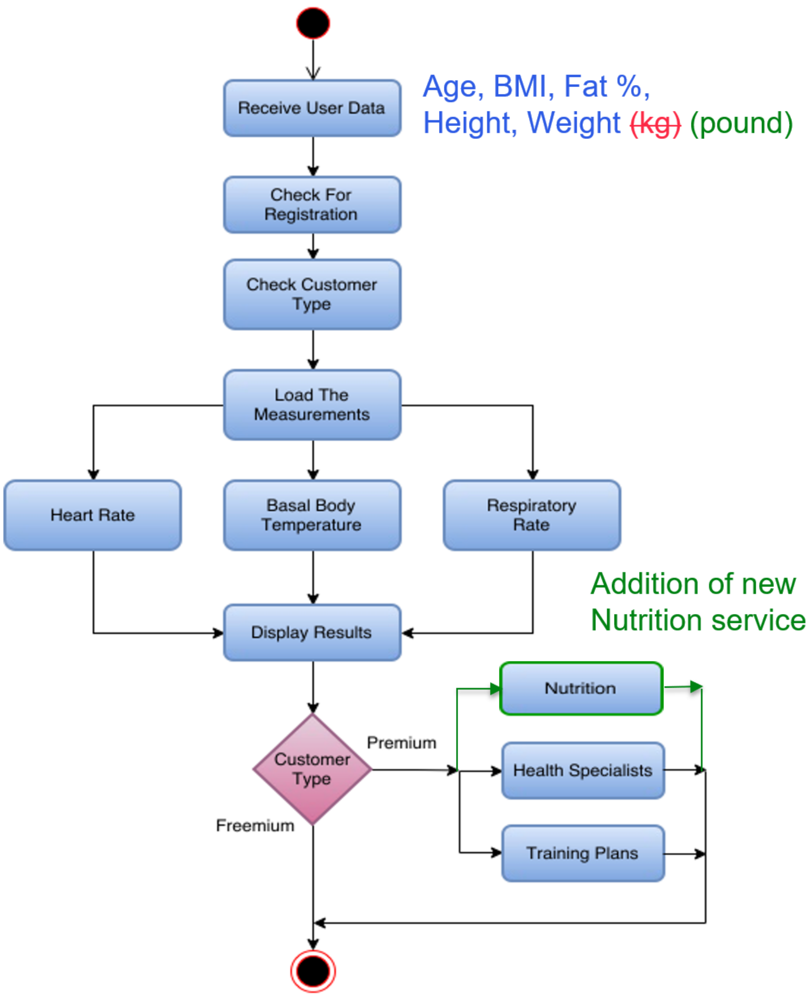


Figure 6.1: A motivation scenario in IoT environment

applications. In this case, the entire application may disrupt, making the application unusable. Hence, changes would indirectly affect the customer applications, making the FitService not trustworthy anymore. Thus, a service provider should notify its clients in time and in a precise way about the change from the existing version to the updated one so that the clients can have a grace period to switch over to the new service.

The following section explains by means of the *FitService* workflow the different types of changes that we will consider. Such changes can also appear in other direct and indirect service layers and affect the customer.

Listing 6.1: Change in the WeightType semantic

```
...
<xs: complexType name= "WeightType">
<xs: sequence>
<xs: element name= "pound" type="xs:float"/>
</xs: sequence>
</xs:complexType>
_________________________________________________________
<xs: complexType name= "WeightType">
<xs: sequence>
<xs: element name= "kilo" type="xs:float"/>
</xs: sequence>
</xs:complexType>
...
```

- *Change the semantic of WeightType*: let us consider that the *FitService* extends its services in an international market and thus it has undergone a few changes in its functionality. Without any prior notification, the *FitService* evolves into its new version. However, the changes do not interrupt the clients but it provides unexpected results. Let us consider that the FitService asks its users to input the weight value in kilo and not in pounds anymore. This change is presented in the related WADL file as seen on Listing 6.1).

- Addition of new service: FitService has decided to include a new service called "Nutrition" to its already existing service without any other changes to it. This is to satisfy its clients who could also be health-conscious, wanting to know the kind of diet they have to maintain. For the new service, the FitService manager also increases the cost of subscriptions for its premium customers.

These scenarios are used in the following to show how a service can undergo changes and subsequently how these changes are described, detected and propagated to the dependent clients.

6.3 Detecting Changes

6.3.1 Framework Overview

In the scope of this work, a precise detection means that not only structural changes like the addition or removal of a function are detected but also semantic changes which are not reflected by the structure. For example, a return value may keep its data type Float but change its interpretation from the imperial system to the metric system. This section presents a framework to detect syntactic and semantic service changes by describing these services before and after the evolution.

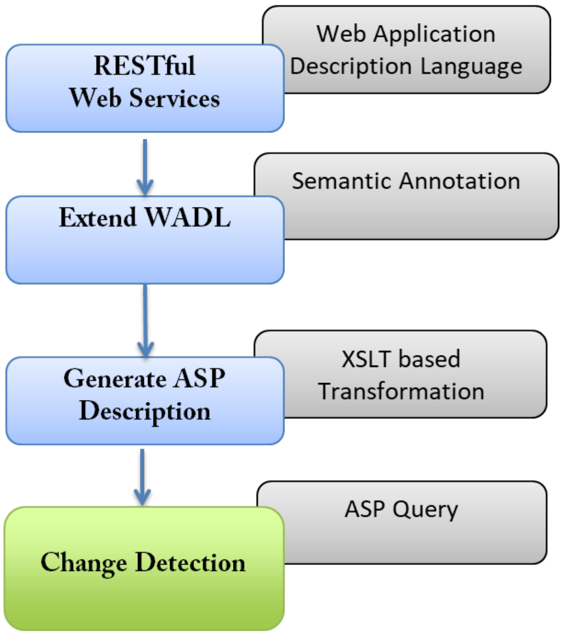


Figure 6.2: Change Detection Work-flow of Framework

Figure 6.2 shows how to detect changes in our framework involving four stages. The first stage describes RESTful services by using WADL. The second stage annotates semantically RESTful services. This stage adds change information of services into WADL descriptions. The third stage generates an ASP program based on the extended WADL description by XSLT[1]. The last stage detects changes by means of ASP Query.

[1]https://www.w3.org/TR/xslt/, visited last on 16th February 2021

6.3.2 Semantic Annotation

Semantic changes in the service interfaces can be referenced, extended, and implemented by application providers. Thus, for successful service discovery and change management, semantic annotations should be added to the description of service interfaces. They will enrich the interface of services with various semantic information. Coming back to our example, we assume that the FitService is implemented by using RESTful Web services. In our scenario, the WADL description before the service evolution of FitService. Extending WADL with semantic annotations to the service description is performed by adding *asp:conceptReference* as seen in Listing 6.2. The customer resource is annotated because this resource later undergoes a change in WeightType. The prefix *asp* refers to the ASP description of the service.

Listing 6.2: An example of Semantic annotation

```
...
<resource path="user">
<method id="createUser" name="POST">
<request>
<ns:representation asp:conceptReference
="http://.../asp/
Fitness#Customer" element
="Customer" mediaType="application/json">
</request>
</method>
</resource>
...
```

The second change scenario can be observed in Listing 6.3 below, which is an addition of a new service called "Nutrition" after the service evolution.

Listing 6.3: Semantic Annotation for Nutrition Service

```
...
<resource path="nutrition">
<method id="getNutrition" name="GET">
<response>
<representation asp:conceptReference=
"http://.../asp/Fitness#Macro" element=
"Macro" mediaType="application/json">
<response>
</method>
</resource>
...
```

It can also be observed in its corresponding XML Schema Definition (XSD). The element Macro is semantically annotated (shown in Listing 6.4).

Listing 6.4: Semantic Annotation for Nutrition Service

```
...
<xs:element name="Customer" type="tns:CustomerType"/>
<xs:element name="CustomerResults" asp:conceptReference=
"http://.../asp/Fitness#CustomerResults"
type="tns:CustomerResultsType"/>
<xs:element name="Macro" asp:conceptReference=
"http://.../asp/Fitness#Macro" type="tns:MacroType"/>
.. .
<xs:complexType name="MacroType">
<xs:sequence>
<xs:element name="carbs" type="xs:float"/>
<xs:element name="fat" type="xs:float"/>
<xs:element name="protein" type="xs:float"/>
</xs:sequence>
</xs:complexType>
</xs:schema>
...
```

After the addition of semantic annotations to both versions of the FitService's WADL descriptions, the next stage is to generate the ASP description from the extended WADL, as discussed in the following section.

Listing 6.5: WADL resource

```
...
<resource path="/auth">
<resource path="/login">
<method id="login" name="POST">
<request>
<param xmlns:xs="http://www.w3.org/2001/XMLSchema"
name="login" style="query"
type="xs:string"/>
<param xmlns:xs="http://www.w3.org/2001/XMLSchema"
name="password" style="query" type="xs:string"/>
</request>
<response>
<representation mediaType="text/plain"/>
</response>
</method>
</resource>
...
```

6.3.3 Generate ASP Description

In our approach, the main reason for using ASP instead of OWL is that ASP is non-monotonic and supports so-called defaults [113], which allow service providers to refer to the same default knowledge. On the other hand, it can also override parts and adapt to details without breaching consistency. Other frameworks, for example, Web service execution environment [114] combine OWL with rule-based formalisms grounded in logic programming. However, logical conclusions that an OWL reasoner would draw from an ontology differ from those that would be obtained when using a logic program engine [2]. In fact, the root of ASP is logic programming providing defaults for expressing standard representations. This feature of ASP provides various benefits for service vendors and participating parties in dynamic and heterogeneous environments [2]. A short introduction of ASP is described in Section 2.3. Further details related to ASP and representing programs are presented in the book [2].

In practice, RESTful services are syntactically described using the XML-based WADL that defines the complete interface of a service. Its service interface presents resources and all the operations that can be invoked on these resources through HTTP methods, e.g., GET, POST, DELETE, and PUT. In our scenario, IoT services can be accessed through REST interfaces. As mentioned earlier in Section 2.1.3, WADL is XML-based and can be considered a tree structure with nodes representing the 'elements' and edges representing the 'relations'. Besides, we use the XSLT as one of the transformation languages to extract the element values. Furthermore, we employ ASP that provided many advantages, such as updating existing rules or overwriting initial default assumptions. The following listings show the fundamental XSL Transformations applied on the target XML data, besides providing the corresponding output in 'text' format. This output file can be stored and processed later using ASP. For generating ASP from the WADL data, it has to be noted that all datatype attribute values are taken as *defaults* in the Default Knowledge Base (DKB) [115], that is created by the developer.

Listing 6.6: Example *defaults* for DKB-Predefined ASP Rules

```
....
is_a(X,Y) :- property(X), not d(X,Y), not -is_a(X,Y).
d(X,Y) :- property(X), X=float, Y=type(xs_float).
...
```

Listing 6.6 consists of various default statements for some of the property values. The developers create this DKB beforehand. The rest of the WADL data is realized by using XSLT as discussed in Listing 6.5 and 6.7. The result in Listing 6.8 of the XSLT transformation of the WADL description presents the declaration of *facts* [115] in ASP.

Listing 6.7: Example for transformation rules in XSLT

```
..
<xsl:stylesheet version="2.0" xmlns:ns= "https://www.w3.org/TR/xslt>
<xsl:output encoding="UTF-8" indent="no" method= "text" omit-xmldeclaration="
   yes" />
<xsl:template match="/">
<xsl:for-each select= "ns:application/ns:resources/ns:resource">
property(<xsl:value-ofselect= "substring-after(@path,'/')"/>).
object(<xsl:value-of select="name()" />).
is_a(<xsl:value-of select="@path" />,
<xsl:value-of select="name(@path)"/>).
...
```

Listing 6.8: An example of generating ASP

```
...
object(resource).
is_a(auth,path(P1)).
property(login).
object(resource).
is_a(login,path(P2)).
...
```

Consider that the FitService has upgraded its service, which includes scenarios described in Section 6.2. This section focuses on semantic change that kinds of change play a vital role in service evolution. As can be seen in Listing 6.9, it shows a snippet of logical description, realized using XSL Transformation. It consists of describe a service change definition of WeightType. The arguments of every predicate are well described as facts.

Listing 6.9: Change in WeightType as kilo

```
...
 is_a(weight,name(N)).
 is_a(WeightType,type(T)).
 has(sequence(S),element(E)).
has(element(E),name(B)).
 has(element(E),type(T)).
...
```

By representing the service description using ASP, KB consisting of unconditional facts cannot be overwritten. In order to change a fact representing a specific element with its attribute value in the knowledge base, defaults are used to add some additional facts to the KB as shown in Listing 6.9. In the scope of service evolution, a *default statement* for name(Y) value is defined as X in the DKB. Furthermore, the knowledge representation for the WADL data before the service change can import the DKB. This can be represented as shown in Listing 6.10.

Listing 6.10: Defautl statement and change in WeighType -X

```
...
// Default statement
is_a(X,name(Y)) :- property(X), not d(X,name(Y)),
not -is_a(X,name(Y)).
d(X,name(Y)) :- property(X), X=kilo.
...
// Change in WeighType -X
# include "Fitsness.lp".
# include "Defaults.lp".
is_a(pound,X(Y).
...
```

Consider that a service provider has updated the service, changing the WeightType from pound to kilo, it is then simpler to add this fact to the new ASP program, and by using #include statements to import all the rules to the new ASP program and also the DKB, besides adding the new facts about the service.

6.4 Implementation and Evaluation

Section 6.4, relies on a Master thesis [116] [2].

This section presents solutions for detecting the changes to the affected clients. It is furnished with an evaluation setup in Section 6.4.1 required to implement the approach. In Section 6.4.2, the evaluation results are provided that show the service evolution successfully performed on the WADL data by using the ASP Queries to figure out the differences between the two versions of the WADL before and after the change of services. Section 6.4.3 provides a performance analysis of the ASP Queries.

[2]Huu Tam Tran wrote the Master thesis description with the support from project PROSECCO

6.4.1 Evaluation Setup

The following evaluations were derived on a PC, with 2.5 GHz Intel Core i7 processor and 8 GB 1600 MHz DDR3 RAM by utilizing Clingo, an ASP system to ground and solve logic programs [113].

6.4.2 ASP Queries and Results

The setup discussed above forms the basis for the implementation of the prototype for our service change detection. For this evaluation, the following steps are considered to find out the solution. It involves:

1. The logical WADL description of the Fitness service (before the change) saved under the *filename Fitness.lp* is taken into account, having the imported DKB Defaults.lp that consists of all the default statements for the data objects and properties. *Fitness.lp* also consists of *FitnessX.lp*, the logical description for FitService XSD.

2. The new logical description *FitnessNew.lp* of the web service description (after the change) has to import all the facts declared in *Fitness.lp* along with the DKB *Defaults.lp* and the new FitService XSD saved under the filename *FitnessNewX.lp.*

3. In order to write an ASP Query to find out the service changes, an important step is now to change all the predicates of FitnessNew.lp and FitnessNewX.lp to considering the predicates in *Fitness.lp* and *FitnessX.lp.*

4. By using the keyword #include which sets all the atoms of *Fitness.lp*, *FitnessX.lp*, *Defaults.lp*, *FitnessNew.lp* and *FitnessNewX.lp.*

5. If required, update the *DKB Defaults.lp* with some additional rules and defaults, which could be the result of newly added atoms in *FitnessNew.lp* and *Fitness-NewX.lp.*

Before proceeding further with the ASP Queries, the predicate changes in FitnessNew.lp and FitnessNewX.lp are provided. It is achieved with a piece of Java program, which takes every text line in ASP consisting of a predicate as a string.

Listing 6.11: An example of ASP Query for multiple changes

```
...
%----FitnessX.lp
property(kilo;liter). object(name(X);name(Y)).
is_a(float,type(X)).
-is_a(P,X) :- is_a(P0,X), property(P), P0 != P.
%----Defaults.lp
is_a(X,name(X)) :- property(X), not d(X,name(X)),
not -is_a(X,name(X)).
d(X,name(X)) :- property(X), X=kilo.
is_a(X,name(Y)) :- property(X), not d(X,name(Y)),
not -is_a(X,name(Y)).
d(X,name(Y)) :- property(X), X=liter.
%----FitnessNewX.lp
property(pound;milliL).
is_a_n(pound,name(Z)).
is_a_n(milliL,name(W)).
%----ASP Query
delta(A,B) :- is_a_n(A,B), not d(A,B).
delta(A,B) :- not is_a_n(A,B), d(A,B).
#show delta/2.
...
```

After creating new predicates of *FitnessNew.lp* and *FitnessNewX.lp*, we compare all the atoms of the respective logical descriptions with their corresponding peers of *Fitness.lp* and *FitnessX.lp* for both the scenarios. Thereby the Clingo solver is used to execute these queries. The DKB Defaults.lp can consist of 'd' number of defaults corresponding to predicates of either the property or object. This ASP query is more advantageous if it has more defaults. These queries can be performed by changing multiple numbers of property values in our scenario, and find out exactly the differences between the atoms of both FitnessX.lp and FitnessNewX.lp. To depict this, we randomly added two new properties and objects to the logical descriptions and also a new default statement in Defaults.lp. The results can be observed in Listing 6.11.

6.4.3 Evaluation

This section will present the test analysis that is performed for a proposed scenario by using Cling solver [113]. In our scenario, we focus on semantics change, a crucial change for the change detection process. A simple example of this changes is *WeightType* presented in Section 6.2.

We conducted the scenario by adding several changes and then examining the runtime during our experiments. With each time of an increasing number of changes, we count five times and take the average results to improve the reliability of our test. For example, we add 16 new changes to our scenario. After that, we measuring the average runtime is 11.56 ms.

Regarding increasing the number of changes from 1 to 128, we examined runtime and found that the average time increased from 10.52 ms to 12.92 ms. These results can be seen in Figure 6.3.

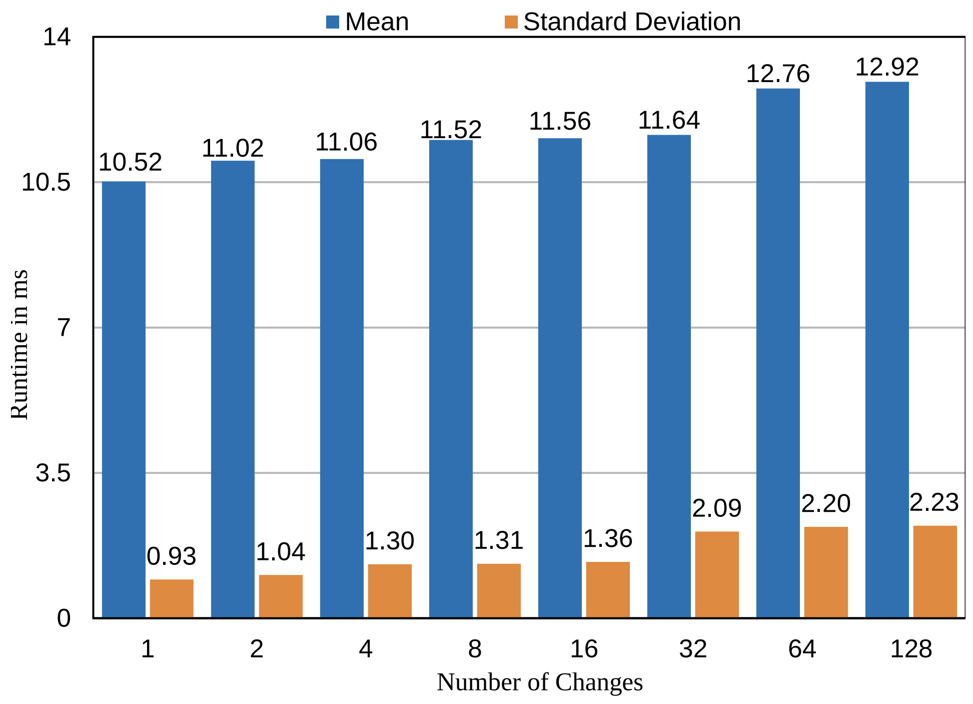


Figure 6.3: Runtime execution in the scenario

Compared to any simulation that consumes a reasonable amount of time, the Clingo solver still performs well in executing the queries in the range of some milliseconds. Meanwhile, the average standard deviation is considered, which measures the spread of the data that shows how dispersed the data is around the mean. It increases gradually, resulting in the reliability of the data points considered for the average runtime at each step.

To summarize the performances of the above scenario, the increment of changes shows an impact on the performance of the Clingo solver.

6.5 Conclusions

This chapter presents a comprehensive framework DECOM to describe and detect changes in IoT services by using Answer Set Programming. Additionally, our evaluation examines the run-time performance and proves the suitability for highly dynamic environments.

However, this framework is designed for detecting changes based on the service interface. Because we know that changes due to updates may affect functional and non-functional properties, the former can be recognized by analyzing the service interface in many cases. However, the detection could be a failure if updates merely address behavioral aspects. Therefore, we will present an approach to determine service behavior changes by inspecting input and output values in the next chapter.

7 Detection of Service Behavior Changes

This chapter extends current change detection approaches by considering behavioral changes of services. The behavior addresses the correlation and distribution of input and output values of service. This differs from structural and semantic change detection, which examines interface descriptions only. Structural changes modify the signature of the service interface, while semantic changes affect interface annotations defined in referenced ontologies. The main content of this chapter has relied on a Master thesis [16][1] and a part of the publication [100][2].

7.1 Introduction

Current approaches mainly focus on the interface changes without considering the actual service behavior by inspecting input and output values [100]. This can lead to a selection of unsuitable service during an autonomous service replacement. Although service interface matching could be successful, the behavior of the selected service could deviate from the replaced service significantly [100].

Our goal is to detect any kinds of service change that is relevant for a client application by analyzing the streamed sensor data. These changes are detected in an anomaly detection fashion by introducing a smart agent where anomaly detection models are deployed at each client depending on their service subscription. The models are developed based on machine learning techniques. With the use of machine learning, the models can be learned and updated automatically, detecting anomalies efficiently, which reduces a part of a manual effort in the continuous provision of services to the end-users.

Since there are many machine learning techniques available, thus the performance of each of the techniques should carefully be assessed through its benchmarks and evaluation.

[1]Huu Tam Tran wrote the Master thesis description with the support from project PROSECCO
[2]Alexander Jahl is the first author of this publication.

The performance is measured to feasibly support the models on resource-constrained IoT devices in terms of processing power, communication bandwidth, battery lifetime, and memory capacity. In our approach, the performance of several algorithms is evaluated through benchmarks, training, and test results. The benchmark was run on a Raspberry Pi minicomputer. Memory usage, number of threads used, number of classes loaded, CPU usage, and execution time for different algorithms are assessed in order to check suitability for resource-constrained IoT devices, assuming a REST-based service architecture.

This chapter presents the following points: (i) How to detect the service behavior changes in an IoT domain to support service evolution? (ii) What is the efficient mechanism to be implemented to detect the changes in such a large-scale environment? Thus, the main contributions of this work are: (1) propose an approach to detect the service behavior changes in an IoT domain to support service evolution; (2) realize an implementation of REST service architecture providing appropriate models for IoT resource-constrained devices.

The rest of the chapter is organized as follows. Section 7.2 introduces a motivation scenario of service behavior changes. Section 7.3 provides some background information related to anomaly detection and machine learning techniques. Section 7.4 explains the key building blocks of our approach. In Section 7.5 we describes our implementation. Section 7.6 evaluates our approach. Finally, some concluding remarks are given in Section 7.7.

7.2 Motivating Example

Let us consider an IoT scenario that is depicted in Figure 7.1. In this scenario, service provider C is a client of service providers A and B, providing the information of *atmospheric pressure* and *temperature*, respectively. Service provider C also has client 1, client 2 and client 3 since they subscribed to service C. Here we assume that service provider C is delivering the Temperature service in Fahrenheit to clients 2 and 3 when they invoked a Temperature service in Fahrenheit by a *getTempInF()* method. Someday the service provider B updates its service and provides the Temperature service in Celsius instead of Fahrenheit. However, service C does not get any notice from service B. Hence, clients 2 and 3 could be interrupted or not. In the latter case, clients 2 and 3 may get the wrong temperature information. If this case occurs, we may consider it as a behavioral change in service.

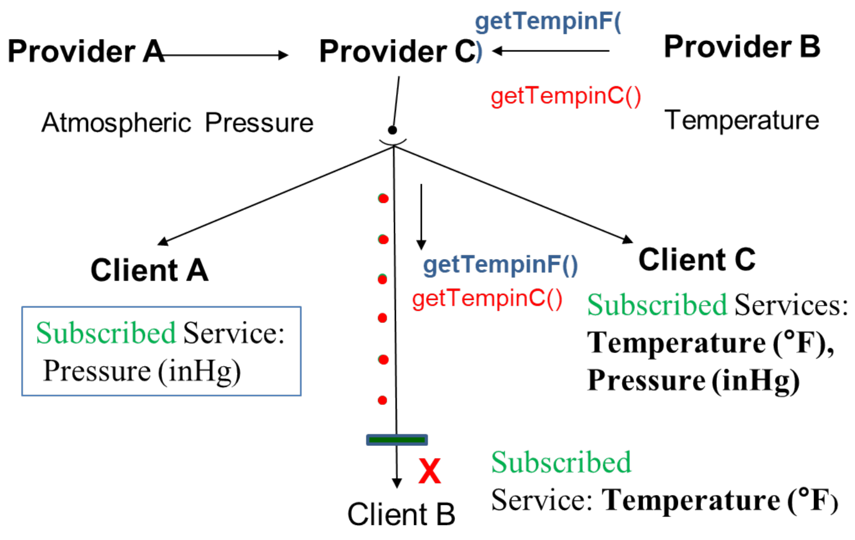


Figure 7.1: Change Scenario

In this scenario, the Fahrenheit values are considered normal behavior of the data while Celsius values are considered the anomalies (when the client subscribed to Temperature in Fahrenheit). The problem of detecting such anomalies is called anomaly detection that will be considered in the next sections.

7.3 Background

This section introduces a couple of different concepts of anomaly detection and machine learning techniques. The main content of this section is extracted from two well-known publications, including a survey of anomaly detection by Varun Chandola [3] and a technical paper by K. Singh [117].

7.3.1 Anomaly Detection

Anomalies are the unexpected changes of the behavior in the data that do not conform to a well-defined notion of normal behavior [3]. Thus, anomaly detection refers to the problem of finding patterns in data that do not conform to expected behavior. These non-conforming patterns are often called anomalies or outliers, or exceptions in a wide variety of application domains. Of these terms, the terms anomalies and outliers are used most commonly in the context of anomaly detection.

Besides, detecting anomalies in data has been studied in the statistics community for many decades. Over time, a variety of anomaly detection techniques have been developed in several research communities. Typically, anomalous data can be connected to some problem or rare events such as bank fraud, structural defects, image processing, malfunctioning equipment, network intrusion detection, or novel topic detection in Text mining [117].

An important aspect of anomaly detection is the nature of the desired anomaly. In general, anomalies are classified into three categories: Point Anomalies, Contextual Anomalies and Collective Anomalies. These categories are described as the followings:

- *Point Anomalies:* An individual data instance is considered as an anomaly when it is far from most of the points in the normal region. For example, in Figure 7.2, points O_1 and O_2 and points in O_3 are considered as point anomalies as they lie outside and far away from the boundary of normal data points [117].

- *Contextual Anomalies:* If a data instance is anomalous in a specific context, but not otherwise, then it is termed as a contextual anomaly. The notion of a context is induced by the structure in the dataset and must be specified as a part of the problem formulation. Each data instance is defined using two sets of attributes, contextual attributes and behavioral attributes. The contextual attribute is used to determine the context, for instance. In time-series data, time is a contextual attribute that determines the position of an instance on the entire sequence, while the behavior attribute defines the non-contextual characteristics of an instance [3].

- *Collective Anomalies:* If a collection of related data instances is anomalous with respect to the entire dataset, it is termed as collective anomaly[3].

Figure 7.2 illustrates anomalies in a simple two-dimensional dataset. The data has two normal regions, N_1 and N_2 since most of the data points lie in these two regions. Points that are adequately far away from these regions, for instance, points O_1 and O_2 and the points in the region O_3, are anomalies.

The following section provides a summary of potential machine learning techniques for the analysis of numerical values.

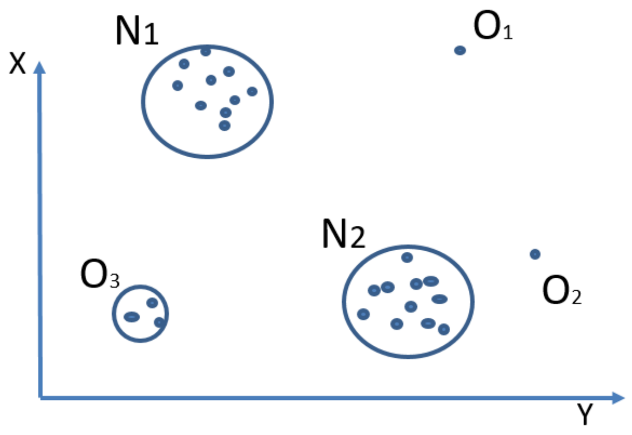


Figure 7.2: An example of anomalies in dataset [3]

7.3.2 Machine Learning

Machine learning has been introduced to detect anomalies in the data as efficient and promising solutions. There are many machine learning methods such as supervised, unsupervised and semi-supervised.

Supervised methods or classification methods required a labeled training set containing both normal and anomalous samples to construct the predictive model. Theoretically, supervised methods provide better detection rates than semi-supervised and unsupervised methods since they have access to more information [118]. In the scope of this work, most of the algorithms are used from supervised learning for anomaly detection purposes. However, some technical issues exist, making these methods seem not accurate as they are supposed to be. The first issue is the shortage of a training data set is a challenge, and the training data sets usually contain some noises that result in higher false alarm rates. For these reasons, we will mainly focus on supervised learning algorithms for our implementations.

The supervised learning algorithms such as Decision Tree [119], Naive Bayes [120], K-Nearest Neighbour [121] are used to classify anomalies. Artificial Neural Networks (ANN) like Feed-forward neural network [122], Elman and Jordan recurrent neural networks[123] are used to predict the data and classify the anomalies based on the Mean Square Error [124]. Unsupervised learning like K-Means [125] and semi-supervised learning like One-Class SVM (OC-SVM) [125] is used for clustering and classification of anomalies,

Table 7.1: Selection of existing anomaly detection techniques (source [118])

.

Techniques	**Characteristic**
Decision Tree	- require little data preparation - able to handle both numerical data - need to select a good kernel function -possible to validate a model using statistical tests
KNN	- easy to understand with few predictor variables - large storage requirements - perform well with large data in a short time - useful for building models involving non-standard data
Naive Bayes	- suitable for text categorization - highly scalable, requires features and predictors
K-means	- low complexity - necessity specifying k - sensitive to noise and outlier data points
One-Class Support Vector Machine	- highly specialized support vector machine, - optimized for outlier detection requires less parameters and training data than SVMs
Feedforward	- the neural network needs the training to operate - contain at least three layers of neurons - feed the data to the network in a forward manner, that is, from the input layer to the output layer - they do not contain any loops
Elman	- requires high processing time for large neural network - the neural network needs the training to operate - context layer always has the same number of neurons present in the hidden layer - a layer of context neurons is combined to the input layer which together inputs the data to the hidden layer - can perform tasks that a linear program cannot, - the neural needs training to operate
Jordan	- requires high processing time for large neural network - the neural network needs the training to operate - number of context neurons are always equal to the number of output neurons

respectively. The significant characteristics of these algorithms can be summarized in Table 7.1).

7.4 Approach

In our approach, each IoT service is equipped with a smart agent like EVA (see Section 4.2) that can communicate with others. These agents automatically gather data from dozens of sensors for parameters such as temperature, humidity. Due to the dynamic streaming of sensor data measurements, the model representing the expected behavior of data is prepared and distributed to different agents. To archieve this, we apply machine learning techniques for agents to discover service behavior changes. In the scope of our motivation scenario, we ask for machine learning algorithms suitable for resource-constrained devices. In addition, they should operate unsupervised, classify quickly, detect precisely with a low error rate, should not tend to overfit, and should be optimized for one cluster detection.

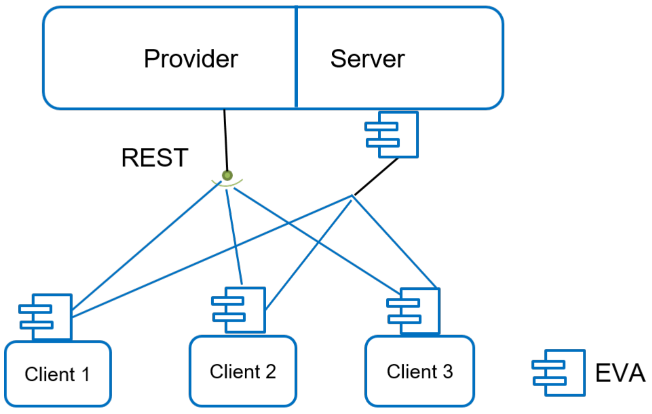


Figure 7.3: Service Architecture

In our scenario, a REST service architecture is established between the service provider and its clients. Figure 7.3 shows a service provider delivers temperature service of three different cities to three different clients 1, 2 and 3 respectively, depending on their subscription of services, for example. The temperature services are provided via REST service in Fahrenheit values. The agent EVAs are deployed to both clients and servers. In our scenario, EVAs are the only module the developer has to integrate. Assuming an application is running on different devices, like, e.g., a Raspberry, personal laptop, each of them is equipped with its own agent. The agents are collecting the in- and output sent between the application and the invoked services. Depending on the number of agents and the message sizes, the responsible agent will ask its agent intermittently for a subset of their collected data. This enables the agent to analyze the typical interaction between the application and its connected services and generate a representative model through machine learning algorithms.

When clients subscribed to the server to get Temperature service in Fahrenheit, the models are prepared based on Fahrenheit's normal data using the most common machine learning algorithms. In the scope of this work, we consider several algorithms that are introduced in Table 7.1.

In order to support the models on constraint devices, the performance of each algorithm is analyzed with benchmarking and accuracy. The efficient algorithms are chosen and then deployed on agents at the clients based on their service subscriptions. When the deployed models detect normal values as anomalies, the value or specific data instance is sent to the server where other efficient models are employed. This is usually done to avoid false alarms. The model on the server checks the data instance for an anomaly. If it detects an anomaly, the model deployed at the clients is performing well. If it detects as a normal value instead of an anomaly, then it replies back to the concerned agents to update the model. The models are then updated offline with new data and deployed back with updated models.

These changes are detected in an anomaly detection fashion by introducing a smart agent or a proxy node where anomaly detection models are deployed at each client depending on their service subscription. With the use of machine learning, the models can be learned and updated automatically. Its process reduces a part of manual effort in the continuous provision of services to the end-users. The model is usually prepared offline for the normal behavior of the data. When the model provided encounters new data online, it detects any anomalies. Anomaly detection is used to either adapt or replace the services. Using state-of-the-art techniques, several models are created to classify or detect anomalies in the sensor data. Because many machine learning techniques are available, each of these techniques' performance is carefully assessed based on their benchmarks and ratings. Performance is measured to support models on resource-constrained IoT devices in terms of processing power, communication bandwidth, battery life and storage capacity. The models are optimized to avoid false alarms due to updates. In order to validate the models provided on the clients, the instance of the service data is sent to other efficient models that are used on the server. We will describe the implementation in detail in the following sections.

7.5 Implementation

In this section, we present the implementation[3] of various models using machine learning algorithms, which were described briefly in Section 7.3. The classical machine learning algorithms such as Decision Tree, Naive Bayes, K-Nearest Neighbour, K-Means, and One-Class Support Vector Machine are developed using a WEKA [4]. WEKA is a data mining software that has collections of machine learning algorithms for data mining tasks. The models based on Neural Networks are developed using ENCOG in Java [5]. ENCOG is an advanced machine learning framework that supports a variety of advanced algorithms. It is also a tool to normalize and process data.

7.5.1 Data Acquisition

In this scope of thesis work, we focus on anomaly detection based on the temperature sensor data. A publicly available raw temperature sensor data is collected from National Centers for Environmental Information (NCEI), which is responsible for hosting and providing access to comprehensive oceanic, atmospheric, and geophysical data. The weather data that is considered. The temperature dataset in Fahrenheit is used to make it exactly resemble the service behavior change problem.

7.5.2 Data Pre-Processing

The time-series data, such as temperature data with missing values, is often incomplete due to sensor errors, transmission errors. The temperature-related characteristics had several missing values about 0.7 percent of the total and randomly dispersed over 30 years (1960-1990). The missing values are filled with the traditional imputation method. The data should also be prepared in a format that the algorithms understand.

In order to have service behavior changes, the data is introduced with 1 percent of anomalies randomly by converting the Fahrenheit values into Celsius. The data is provided to the algorithms in a supervised manner by including a target variable into the temperature data specifying the class type labeled as “anomaly” or “normal” for each instance containing the dataset. The “normal” label for a specific data instance indicates

[3]The experiments of this work conducted by Srivardhan Cholkar in his Master thesis.
[4]Data Mining Software in Java. https://www.cs.waikato.ac.nz/ml/weka/index.html
[5]Encog is a machine learning framework. https://www.heatonresearch.com/encog/

that the data values in that instance are in the units, “Fahrenheit,” and the “anomaly” label indicates that the data values in that instance are in the units, “Celsius.” Except for the semi-supervised algorithm like One-Class SVM, the anomaly class is labeled as “1” and the normal class is labeled as “0”.

The data is normalized in the range [0, 1] to learn the data pattern and understand the features’ correlation clearly. The more data provided for the model, the better the model; for instance, the accuracy of the model is higher. The entire data set is divided into training data sets and test data sets. The model is created based on this training data set, and the test data serve to validate the trained model.

The neural network is created with five input neurons, one output neuron, and a hidden layer with two hidden neurons. The number of hidden layers and hidden neurons is chosen using the pruning method. The best architecture of the neural network is the optimized number of hidden layers and hidden neurons, which can be determined with a trial and error process.

7.5.3 Training Algorithms

The training starts with initializing the weights using the Nguyen-Widrow method [126] which generates random weights distributed roughly evenly over the input space. With the use of this method, the training works faster. The hyperbolic tangent activation function is chosen for each of the hidden neurons and the linear activation function for the output neuron with a trial and error process. The best selection of the activation function is based on the training that the loss function gives the lowest error rate and less execution time.

The Resilient training technique [127] is chosen with a trial and error process to train or adjust the weights of the neural network. In order to improve the accuracy of the network, several strategies were used, such as Greedy strategy and Hybrid strategy [128].

Greedy strategy [128] always checks the network error for each iteration. It allows only those iterations to update the weights and other training parameters for which there is an improvement in the network error. If there is no improvement, the trainer discards that iteration effect. This type of strategy is typically used in conjunction with a hybrid strategy. The training results and test results are presented in the following section.

7.6 Evaluation

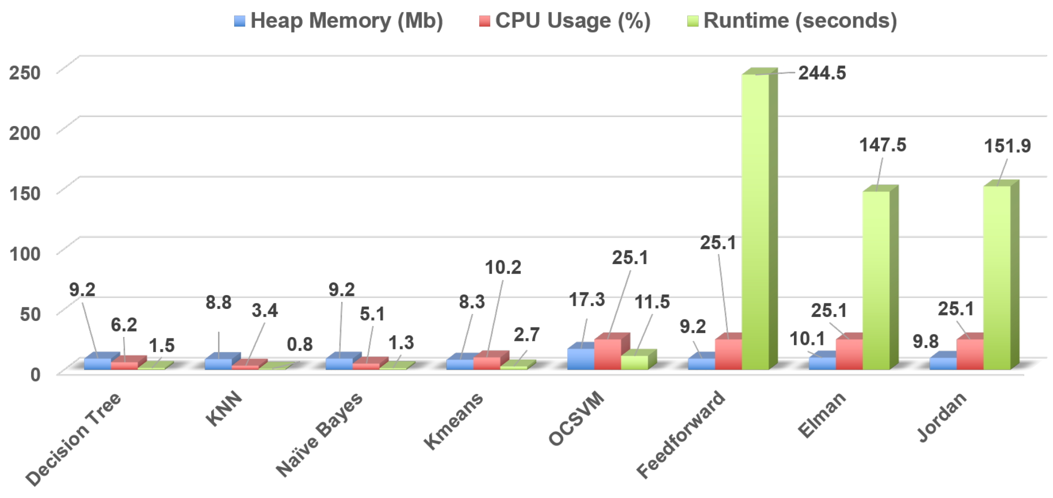


Figure 7.4: Benchmark - Raspberry Pi, 32-bit ARM Cortex- A52 Quadcore 1.2 GHz

To deploy efficient models on a number of devices, including memory-constrained devices, it is essential to check the performance of each algorithm through benchmarking and compare them to see which models are best suited to support different IoT devices. We use Raspberry Pi that plays as IoT clients in our experiments.

The benchmarks are taken on Java Virtual Machine on Raspberry Pi, evaluating the model's complexity for the given data. Furthermore, we can observe using heap memory usage, committed memory usage, number of threads used, the total number of classes loaded, CPU usage, and runtime while training different algorithms. Figure 7.4 [16] shows the critical performance characteristics like heap memory usage, CPU usage, and runtime are only considered with different to compare the algorithms with each other as the others are more or less the same for all the algorithms.

7.6.1 Performance of Algorithms

The general overview of the performance of all algorithms is based on the frameworks for machine learning and the considered temperature data record with three characteristics maximum, minimum, and observed temperature with more than 15,000 instances.

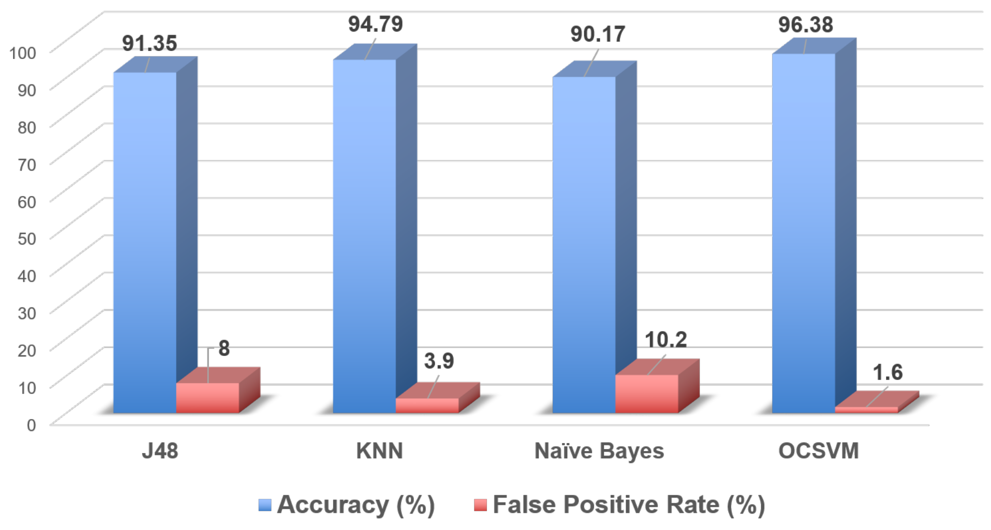


Figure 7.5: Performance of Algorithms: J48, KNN, Naive Bayes and OCSVM

Figure 7.5 [16] shows that classic machine learning algorithms such as Decision Tree, Naive Bayes, KNN, and K-Means are easy to implement, understand, and have high training and prediction speeds with low memory usage, except OCSVM. Among them, Decision Tree, Naïve Bayes, KNN, and OCSVM are highly accurate. The accuracy of K-means cannot simply be calculated with the other algorithms. Although highly accurate, they were unable to see the contextual and collective anomalies analyzed in practice. However, OCSVM partially discovered contextual anomalies that have the highest accuracy of 96.38 percent. Detecting contextual anomalies is essential for time series and periodic data such as temperature.

Figure 7.6 [16] shows the performance of our selected neural network algorithms, including Feedforward, Elman and Jordan. Although the neural networks have a low training and prediction speed compared to classical machine learning algorithms, they use less memory. Their complexity increases with the number of features and data instances.

The static memory of the Feedforward neural networks could not predict the contextual anomalies well. The Elman and Jordan recurrent neural networks are specialized with their contextual neurons using which they construct a dynamic memory or short-term memory. With this dynamic memory, they were able to recognize the contextual anomalies well. However, the accuracy of anomaly detection is high for Jordan, and the training speed is similarly high for other neural networks because they converged to a considerably small error with a small number of iterations.

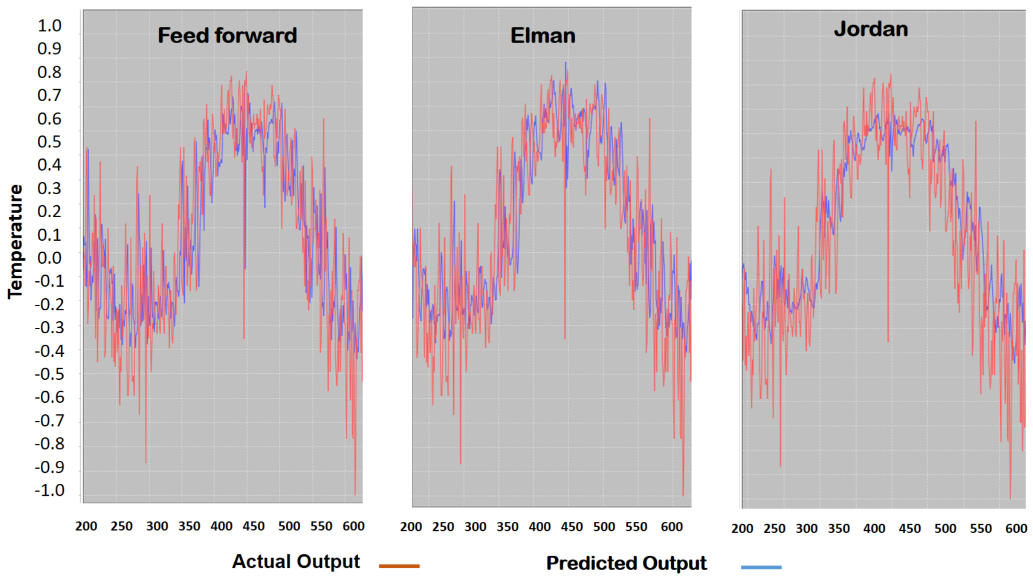


Figure 7.6: Performance of Algorithms: Feedforward, Elman and Jordan

The Decision Tree, Naive Bayes, KNN algorithms are therefore suitable for IoT-restricted devices to detect point anomalies if high accuracy and low resource consumption are taken into account, for which KNN is best suited. OCSVM is considered when there is a trade-off between accuracy and memory usage. Jordan and Elman are suitable for devices with high memory.

7.7 Conclusions

In this chapter, we presented an approach to detect anomalies efficiently in the streamed sensor data by deploying agents that automatically take care of these changes. To detect abnormalities in the sensor data, we have to use anomaly detection methods based on machine learning algorithms. The performances of several algorithms are analyzed through benchmarks, training, and test results. The performance is measured to satisfy the requirements of IoT-constrained devices. Thus, the main contribution of this chapter is realized by the verifying agents that are responsible for the behavioral analysis of the client-server communication. This represents a new dimension not included by other existing approaches. In the future, we will adopt the proposed approach with other machine learning algorithms for the realization of full service-oriented architecture in real-time for business applications.

8 Conclusion

This chapter summarises the research problems and their corresponding solutions. It also presents some potential extensional directions to the current work.

8.1 Summary

In this thesis, we are primarily interested in foundational research aiming at the distributed coordination of service co-evolution in the context of IoT services. Mainly, we have presented the following contributions:

- We have developed a solution for coordinating service co-evolution through a design that equips every service with an EVA that performs the service evolution in collaboration with other EVAs. The solution also introduced a new vision of service co-evolution in IoT by providing an evolution management model. We have illustrated how a service co-evolution is carried out, what should be involved, why it is essential, and what should be prepared to meet the co-evolution requirements. In this model, evolution tasks like assessing and coordinating evolution requests, updating the interfaces, and selecting matching services can be performed automatically or semi-automatically by EVAs.
- We have proposed a novel notification management architecture that detects service description changes and notifies all affected participating parties in the network. The design of this architecture has taken into account the combination of EVAs, a service registry, and the detecting algorithm. The notification management architecture is implemented to cater to the increasing dynamic networks in IoT, particularly for services exposed by resource-constrained devices.
- We have developed a comprehensive framework DECOM to describe and detect service changes in IoT services using Answer Set Programming. Here, we assume that the capabilities and interfaces of IoT devices are described and provided

through REST services. To detect syntactic and semantic changes, we transform an extended version of the interface description into a logic program and apply a sequence of analysis steps to detect changes. The feasibility and applicability of the approach are demonstrated through a running example.

- We have proposed an approach to find out changes in the service behavior by analyzing the data stream between the service client and the service provider. In our approach, an intelligent agent is deployed at each client that monitors the data stream. Anomaly detection is based on machine learning techniques. The performance of several algorithms is evaluated through benchmarks, training, and test results. The benchmarks were run on a Raspberry Pi minicomputer. Memory usage, number of threads used, number of classes loaded, CPU usage, and execution time for different algorithms are assessed in order to check the suitability for resource-constrained IoT devices, assuming a REST-based service architecture.

- Last but not least, we have presented a short overview of existing approaches that support service evolution and, in particular, of service co-evolution. It provides a technical document to researchers and practitioners building industrial-strength adaptive applications related to service co-evolution.

8.2 Limitation and Future Work

The thesis presented a set of theoretical models and approaches that facilitate service co-evolution in IoT environments. Although our contribution is a significant step towards on-the-fly service evolution in IoT circumstances, we intend to address some open challenges in our future work. This work calls for the following extensions:

- **Extending the EVA framework architecture:**

  The first direction is to extend the EVA framework architecture to realize fully autonomous agent collaboration. The EVA framework should be extended with the following components:

  (i) A component of decision making and planning of service co-evolution, which is designed for EVAs in order to compute a joint executable plan, for instance, an updated plan for a group of EVAs in a co-evolution scenario based on logic programming. This component may use the game-theoretic modeling that is proposed in a distinguished work by M. Fokaefs, and E. Stroulia [129].

(ii) A component of monitoring and run-time testing aims to validate the functional and non-functional properties of co-evolved services. This component could support capturing information about possible violations or changes in the quality of service parameters. This component aids in verifying the compliance of functional and non-functional requirements of a co-evolved service configuration. It aims to avoid changes in a worst-case scenario. Hence, this component aims to ensure that service evolution does not break the correctness condition and other specified requirements. In conclusion, these integrated components with EVAs could support joint service co-evolution activities.

- **Extending the current notification management architecture:**

  The second direction is to extend our notification management architecture. The current notification management architecture mainly emphasizes the need for a notification mechanism and does not dive into the impact of changes. Thus, further works related to the impact of changes should be taken into account. The notification management architecture should particularly determine what kind of service changes have to be considered in the service evolution support, e.g., interface changes, quality of services, or behavior changes. Furthermore, to handle such kinds of changes, it is to assure that the services involved have communication protocols to propagate the requested changes.

- **Extending the framework DECOM:**

  This framework should support testing the service version compatibility after the successful detecting of changes. Since the compatibility version plays a vital role in service co-evolution, any change during the service evolution should be tested for its compatibility. Of course, only changes that ensure no interruption in the bond between services by upgrading to a new version are considered. Besides, further efforts are needed to detect the impact of complex changes.

- **Extending a suitable model of detection of service changes in complex scenarios:**

  The current defined model is suitable for numerical and text output data. Thus, we need to expand models and algorithms for monitoring and analyzing service behaviors with complex data structures. A possible direction is to combine several machine learning techniques, which could lead to a hybrid classifier. Additionally, after the detection, a specific notification management architecture is needed to notify the clients about the service behavior changes. Finally, an evaluation in a detailed scenario with typical sensors, actors, service dependencies with complex data structures should be performed.

- **Others:** The security aspects that can be used to check the safety of service evolution activities before being commissioned have not been addressed yet. A further option that is taken into consideration for future work, is the transfer and application of the solution for distributed service co-evolution in the area of Smart Cities. The application scenarios must reflect a broad range of change requests and IoT service dependencies.

Publications as (Co-) Author

This thesis has a single author; however, it is the result of years of collaboration. Parts of the work conducted for this thesis have already been reported in the following publications:

1. H.T. Tran, V.T. Nguyen, C.V. Phan, "Towards Service Co-evolution in SOA Environments: A Survey", in the proceedings of Context-Aware Systems and Applications, and Nature of Computation and Communication, pp. 233-254, Springer 2020, (Best Paper Awards).
2. H.T. Tran, V.T. Nguyen, X. T. Nguyen, C.V. Phan, "Service Co-evolution in SOA Environments: A Survey and Outlook", in the special issue on Context-Aware Computing: Theory and Applications, Concurrency and Computation- Practice and Experience, 2021 (submitted).
3. H.T. Tran, A. Jahl, K. Geihs, R. Kuppili, X.T. Nguyen, and T. T. B Huynh, "DECOM: A framework to support evolution of IoT services", in the proceedings of the Ninth International Symposium on Information and Communication Technology, pp. 389–396, ACM 2018.
4. A. Jahl, H. T. Tran, H. Baraki, and K. Geihs, "Wip: Behavior-based service change detection," in the proceedings of IEEE International Conference on Smart Computing (SMARTCOMP), pp. 267–269, IEEE 2018.
5. H.T. Tran, H. Baraki, R. Kuppili, A. Taherkordi and K. Geihs, "A notification management architecture for service co-evolution in the Internet of Things", in the proceedings of Maintenance and Evolution of Service-Oriented and Cloud-Based Environments, IEEE 10th International Symposium, pp 9-15, 2016.
6. H.T. Tran, H. Baraki, and K. Geihs, "An approach towards a service co-evolution in the internet of things," in the proceedings of Internet of Things. User-Centric IoT, pp. 273–280, Springer 2015.
7. H.T. Tran, H. Baraki, and K. Geihs,"Service co-evolution in the internet of things," in the Endorsed Transactions on Cloud Systems, pp. 1-15, EAI 2015.
8. A. Jahl, H. Baraki, H.T Tran, R. Kuppili, and K. Geihs, "Lifting low-level workflow changes through user-defined graph rule-based patterns", in the proceedings of

International Conference on Distributed Application and Interoperable Systems, pp. 115-128, Springer 2017.

9. X.T. Nguyen, H.T. Tran, H. Baraki, and K. Geihs, "Optimization of non-functional properties in internet of things applications," in Journal of Network and Computer Applications, vol. 89, pp. 120–129, Elsevier, 2017.
10. X. T. Nguyen, H. T. Tran, H. Baraki, and K. Geihs, "FRASAD: A framework for model-driven iot application development," in the proceedings of the Internet of Things (WF-IoT), 2015 IEEE 2nd World Forum, pp. 387–392, IEEE 2015.
11. T. V. Nguyen, N. Fredivianus, H. T. Tran, K. Geihs and T. T. B. Huynh, "Formal verification of alica multi-agent plans using model checking," in the proceedings of the Ninth International Symposium on Information and Communication Technology, pp. 351–358, ACM 2018.

Bibliography

[1] M. Zajac, B. Oesterdiekhoff, C. Loeser, and H. Bohn, "Bilateral service mapping approaches between the different service oriented architectures web services and upnp," in *21st International Conference on Advanced Information Networking and Applications Workshops (AINAW'07)*, vol. 1, pp. 964–969, IEEE, 2007.

[2] M. Gelfond and Y. Kahl, *Knowledge representation, reasoning, and the design of intelligent agents: The answer-set programming approach.* Cambridge University Press, 2014.

[3] V. Chandola, A. Banerjee, and V. Kumar, "Anomaly detection: A survey," *ACM computing surveys (CSUR)*, vol. 41, no. 3, p. 15, 2009.

[4] M. N. Huhns and M. P. Singh, "Service-oriented computing: Key concepts and principles," *IEEE Internet computing*, vol. 9, no. 1, pp. 75–81, 2005.

[5] K. Geihs, "Provisions for servicce co-evolution, dfg proposal 2015.".

[6] D. C. Marinescu, *Cloud computing: theory and practice.* Morgan Kaufmann, 2017.

[7] J. Gubbi, R. Buyya, S. Marusic, and M. Palaniswami, "Internet of things (iot): A vision, architectural elements, and future directions," *Future generation computer systems*, vol. 29, no. 7, pp. 1645–1660, 2013.

[8] D. Pfisterer, K. Römer, D. Bimschas, O. Kleine, R. Mietz, C. Truong, H. Hasemann, A. Kröller, M. Pagel, M. Hauswirth, *et al.*, "Spitfire: toward a semantic web of things.," *IEEE Communications Magazine*, vol. 49, no. 11, pp. 40–48, 2011.

[9] M. De Sanctis, K. Geihs, A. Bucchiarone, G. Valetto, A. Marconi, and M. Pistore, "Distributed service co-evolution based on domain objects," in *International Conference on Service-Oriented Computing*, pp. 48–63, Springer, 2015.

[10] M. P. Papazoglou, V. Andrikopoulos, and S. Benbernou, “Managing evolving services,” *Software, IEEE*, vol. 28, no. 3, pp. 49–55, 2011.

[11] K. Dar, A. Taherkordi, H. Baraki, F. Eliassen, and K. Geihs, “A resource oriented integration architecture for the internet of things: A business process perspective,” *Pervasive and Mobile Computing*, vol. 20, pp. 145–159, 2015.

[12] H. T. Tran, H. Baraki, and K. Geihs, “An approach towards a service co-evolution in the internet of things,” in *Internet of Things. User-Centric IoT*, pp. 273–280, Springer, 2014.

[13] H. T. Tran, H. Baraki, and K. Geihs, “Service co-evolution in the internet of things,” *EAI Endorsed Transactions on Cloud Systems*, vol. 15, 2 2015.

[14] H. T. Tran, H. Baraki, R. Kuppili, A. Taherkordi, and K. Geihs, “A notification management architecture for service co-evolution in the internet of things,” in *Maintenance and Evolution of Service-Oriented and Cloud-Based Environments (MESOCA), 2016 IEEE 10th International Symposium on the*, pp. 9–15, IEEE, 2016.

[15] H. T. Tran, A. Jahl, K. Geihs, R. Kuppili, X. T. Nguyen, and T. T. B. Huynh, “Decom: A framework to support evolution of iot services,” in *Proceedings of the Ninth International Symposium on Information and Communication Technology*, pp. 389–396, 2018.

[16] S. Cholkar, “An efficient anomaly detection mechanism for streamed sensor data in iot environments.,” Master’s thesis, University of Kassel, 2018.

[17] H. T. Tran, C. V. Phan, *et al.*, “Towards service co-evolution in soa environments: A survey,” in *Context-Aware Systems and Applications, and Nature of Computation and Communication*, pp. 233–254, Springer, 2020.

[18] T. Erl, *SOA principles of service design (the Prentice Hall service-oriented computing series from Thomas Erl).*
Prentice Hall PTR, 2007.

[19] M. P. Papazoglou, “Web services and soa: principles and technology 2nd,” *Harlow, Essex: Pearson Education Limited*, 2012.

[20] M. Rosen, B. Lublinsky, K. T. Smith, and M. J. Balcer, *Applied SOA: service-oriented architecture and design strategies.*
John Wiley & Sons, 2012.

[21] C. M. MacKenzie, K. Laskey, F. McCabe, P. F. Brown, R. Metz, and B. A. Hamilton, "Reference model for service oriented architecture 1.0," *OASIS standard*, vol. 12, p. 18, 2006.

[22] W. Roshen, *SOA-based enterprise integration: A step-by-step guide to services-based application.*
McGraw-Hill, Inc., 2009.

[23] C. Peltz, "Web services orchestration and choreography," *Computer*, vol. 36, no. 10, pp. 46–52, 2003.

[24] D. Zeng, S. Guo, and Z. Cheng, "The web of things: A survey," *JCM*, vol. 6, no. 6, pp. 424–438, 2011.

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

[26] R. Daigneau, *Service Design Patterns: fundamental design solutions for SOAP/WSDL and restful Web Services.*
Addison-Wesley, 2012.

[27] C. Pautasso, "Restful web services: principles, patterns, emerging technologies," in *Web Services Foundations*, pp. 31–51, Springer, 2014.

[28] A. Bouguettaya, Q. Z. Sheng, and F. Daniel, *Web services foundations.*
Springer, 2016.

[29] S. J. Russell and P. Norvig, *Artificial intelligence: a modern approach.*
Malaysia; Pearson Education Limited,, 2016.

[30] M. Wooldridge, *An introduction to multiagent systems.*
John Wiley & Sons, 2009.

[31] S. Poslad, "Specifying protocols for multi-agent systems interaction," *ACM Transactions on Autonomous and Adaptive Systems (TAAS)*, vol. 2, no. 4, p. 15, 2007.

[32] F. L. Bellifemine, G. Caire, and D. Greenwood, *Developing multi-agent systems with JADE*, vol. 7.
John Wiley & Sons, 2007.

[33] A. Rogers, E. David, N. R. Jennings, and J. Schiff, “The effects of proxy bidding and minimum bid increments within ebay auctions,” *ACM Transactions on the Web (TWEB)*, vol. 1, no. 2, p. 9, 2007.

[34] Z. Genc, F. Heidari, M. A. Oey, S. van Splunter, and F. M. Brazier, “Agent-based information infrastructure for disaster management,” in *Intelligent Systems for Crisis Management*, pp. 349–355, Springer, 2013.

[35] N. Schurr, J. Marecki, M. Tambe, P. Scerri, N. Kasinadhuni, and J. P. Lewis, “The future of disaster response: Humans working with multiagent teams using defacto.,” in *AAAI spring symposium: AI technologies for homeland security*, pp. 9–16, 2005.

[36] B. Burmeister, M. Arnold, F. Copaciu, and G. Rimassa, “Bdi-agents for agile goal-oriented business processes,” in *Proceedings of the 7th international joint conference on Autonomous agents and multiagent systems: industrial track*, pp. 37–44, International Foundation for Autonomous Agents and Multiagent Systems, 2008.

[37] M. Pěchouček and V. Mařík, “Industrial deployment of multi-agent technologies: review and selected case studies,” *Autonomous agents and multi-agent systems*, vol. 17, no. 3, pp. 397–431, 2008.

[38] Y. Kubera, P. Mathieu, and S. Picault, “Everything can be agent!,” in *Proceedings of the 9th International Conference on Autonomous Agents and Multiagent Systems: volume 1-Volume 1*, pp. 1547–1548, International Foundation for Autonomous Agents and Multiagent Systems, 2010.

[39] M. Gebser, R. Kaminski, B. Kaufmann, and T. Schaub, “Answer set solving in practice,” *Synthesis lectures on artificial intelligence and machine learning*, vol. 6, no. 3, pp. 1–238, 2012.

[40] M. Nogueira, M. Balduccini, M. Gelfond, R. Watson, and M. Barry, “An a-prolog decision support system for the space shuttle,” in *International symposium on practical aspects of declarative languages*, pp. 169–183, Springer, 2001.

[41] T. Soininen and I. Niemelä, “Developing a declarative rule language for applications in product configuration,” in *International Symposium on Practical Aspects of Declarative Languages*, pp. 305–319, Springer, 1999.

[42] D. Abels, J. Jordi, M. Ostrowski, T. Schaub, A. Toletti, and P. Wanko, “Train scheduling with hybrid answer set programming,” *Theory and Practice of Logic Programming*, vol. 21, no. 3, pp. 317–347, 2021.

[43] M. Gebser, R. Kaminski, and T. Schaub, “aspcud: A linux package configuration tool based on answer set programming,” *arXiv preprint arXiv:1109.0113*, 2011.

[44] K. Ashton *et al.*, “That ‘internet of things’ thing,” *RFID journal*, vol. 22, no. 7, pp. 97–114, 2009.

[45] P. P. Ray, “A survey on internet of things architectures,” *Journal of King Saud University-Computer and Information Sciences*, vol. 30, no. 3, pp. 291–319, 2018.

[46] O. Uviase and G. Kotonya, “Iot architectural framework: connection and integration framework for iot systems,” *arXiv preprint arXiv:1803.04780*, 2018.

[47] J. A. Stankovic, “Research directions for the internet of things,” *IEEE Internet of Things Journal*, vol. 1, no. 1, pp. 3–9, 2014.

[48] H. Sundmaeker, P. Guillemin, P. Friess, and S. Woelfflé, “Vision and challenges for realising the internet of things,” *Cluster of European research projects on the internet of things, European Commision*, vol. 3, no. 3, pp. 34–36, 2010.

[49] A. McEwen and H. Cassimally, *Designing the internet of things.* John Wiley & Sons, 2013.

[50] S. Li, L. Da Xu, and S. Zhao, “The internet of things: a survey,” *Information Systems Frontiers*, vol. 17, no. 2, pp. 243–259, 2015.

[51] Y. Liu, H. Wang, J. Wang, K. Qian, N. Kong, K. Wang, L. Zheng, Y. Shi, and D. W. Engels, “Enterprise-oriented iot name service for agricultural product supply chain management,” *International Journal of Distributed Sensor Networks*, vol. 11, no. 8, p. 308165, 2015.

[52] P. P. Ray, “A survey of iot cloud platforms,” *Future Computing and Informatics Journal*, vol. 1, no. 1-2, pp. 35–46, 2016.

[53] X. T. Nguyen, H. T. Tran, H. Baraki, and K. Geihs, “Frasad: A framework for model-driven iot application development,” in *Internet of Things (WF-IoT), 2015 IEEE 2nd World Forum on*, pp. 387–392, IEEE, 2015.

[54] X. T. Nguyen, H. T. Tran, H. Baraki, and K. Geihs, "Optimization of non-functional properties in internet of things applications," *Journal of Network and Computer Applications*, vol. 89, pp. 120–129, 2017.

[55] X. T. Nguyen, *Model-driven development of sensor network applications with optimization of non-functional constraints.*
PhD thesis, University of Kassel, 2016.

[56] P. Persson and O. Angelsmark, "Calvin–merging cloud and iot," *Procedia Computer Science*, vol. 52, pp. 210–217, 2015.

[57] H. Naji and M. Mikki, "A survey of service oriented architecture systems maintenance approaches," *International Journal of Computer Science and Information Technology*, vol. 8, pp. 21–29, 2016.

[58] K. H. Bennett and V. T. Rajlich, "Software maintenance and evolution: a roadmap," in *Proceedings of the Conference on the Future of Software Engineering*, pp. 73–87, ACM, 2000.

[59] W. Zuo *et al.*, *Managing and modeling web service evolution in SOA architecture.*
PhD thesis, Université de Lyon, 2016.

[60] M. M. Lehman and J. F. Ramil, "Software evolution—background, theory, practice," *Information Processing Letters*, vol. 88, no. 1-2, pp. 33–44, 2003.

[61] M. M. Lehman and L. A. Belady, *Program evolution: processes of software change.*
Academic Press Professional, Inc., 1985.

[62] L. Yu and A. Mishra, "An empirical study of lehman's law on software quality evolution.," *Int. J. Software and Informatics*, vol. 7, no. 3, pp. 469–481, 2013.

[63] R. P. Cook and I. Lee, "Dymos: A dynamic modification system," *ACM SIGPLAN Notices*, vol. 18, no. 8, pp. 201–202, 1983.

[64] J. Dowling and V. Cahill, "The k-component architecture meta-model for self-adaptive software," in *International Conference on Metalevel Architectures and Reflection*, pp. 81–88, Springer, 2001.

[65] O. Alliance, *Osgi service platform, release 3.*
IOS Press, Inc., 2003.

[66] F.-L. Li, L. Liu, and J. Mylopoulos, “Software service evolution: A requirements perspective,” in *Computer Software and Applications Conference Workshops (COMPSACW), 2012 IEEE 36th Annual*, pp. 353–358, IEEE, 2012.

[67] S. Wang, W. A. Higashino, M. Hayes, and M. A. Capretz, “Service evolution patterns,” in *Web Services (ICWS), 2014 IEEE International Conference on*, pp. 201–208, IEEE, 2014.

[68] M. P. Papazoglou, “The challenges of service evolution,” in *Advanced Information Systems Engineering*, pp. 1–15, Springer, 2008.

[69] V. Andrikopoulos *et al.*, “A theory and model for the evolution of software services,” tech. rep., Tilburg University, School of Economics and Management, 2010.

[70] P. Leitner, A. Michlmayr, F. Rosenberg, and S. Dustdar, “End-to-end versioning support for web services,” in *Services Computing, 2008. SCC’08. IEEE International Conference on*, vol. 1, pp. 59–66, IEEE, 2008.

[71] M. Fokaefs, R. Mikhaiel, N. Tsantalis, E. Stroulia, and A. Lau, “An empirical study on web service evolution,” in *Web Services (ICWS), 2011 IEEE International Conference on*, pp. 49–56, IEEE, 2011.

[72] M. Fokaefs and E. Stroulia, “Wsdarwin: Studying the evolution of web service systems,” in *Advanced Web Services*, pp. 199–223, Springer, 2014.

[73] D. Romano and M. Pinzger, “Analyzing the evolution of web services using fine-grained changes,” in *Web Services (ICWS), 2012 IEEE 19th International Conference on*, pp. 392–399, IEEE, 2012.

[74] A. Jahl, H. Baraki, H. T. Tran, R. Kuppili, and K. Geihs, “Lifting low-level workflow changes through user-defined graph-rule-based patterns,” in *IFIP International Conference on Distributed Applications and Interoperable Systems*, pp. 115–128, Springer, 2017.

[75] H. K. Dam and A. Ghose, “Supporting change impact analysis for intelligent agent systems,” *Science of Computer Programming*, vol. 78, no. 9, pp. 1728–1750, 2013.

[76] L. Liao, S. Qi, and B. Li, “Trust analysis of composite service evolution,” in *2016 IEEE 14th International Conference on Software Engineering Research, Management and Applications (SERA)*, pp. 15–22, IEEE, 2016.

[77] M. Yamashita, B. Vollino, K. Becker, and R. Galante, "Measuring change impact based on usage profiles," in *Web Services (ICWS), 2012 IEEE 19th International Conference on*, pp. 226–233, IEEE, 2012.

[78] S. Wang and M. A. Capretz, "Dependency and entropy based impact analysis for service-oriented system evolution," in *Proceedings of the 2011 IEEE/WIC/ACM International Conferences on Web Intelligence and Intelligent Agent Technology-Volume 01*, pp. 412–417, IEEE Computer Society, 2011.

[79] W. Song, G. Zhang, Y. Zou, Q. Yang, and X. Ma, "Towards dynamic evolution of service choreographies," in *Services Computing Conference (APSCC), 2012 IEEE Asia-Pacific*, pp. 225–232, IEEE, 2012.

[80] W. Fdhila, C. Indiono, S. Rinderle-Ma, and M. Reichert, "Dealing with change in process choreographies: Design and implementation of propagation algorithms," *Information systems*, vol. 49, pp. 1–24, 2015.

[81] A. Khebizi, H. Seridi-Bouchelaghem, B. Benatallah, and F. Toumani, "A declarative language to support dynamic evolution of web service business protocols," *Service Oriented Computing and Applications*, vol. 11, no. 2, pp. 163–181, 2017.

[82] M. Treiber, H.-L. Truong, and S. Dustdar, "Semf - service evolution management framework," in *Software Engineering and Advanced Applications, 2008. SEAA'08. 34th Euromicro Conference*, pp. 329–336, IEEE, 2008.

[83] J. Li, Y. Xiong, X. Liu, and L. Zhang, "How does web service api evolution affect clients?," in *2013 IEEE 20th International Conference on Web Services*, pp. 300–307, IEEE, 2013.

[84] M. Fokaefs and E. Stroulia, "Using WADL specifications to develop and maintain REST client applications," in *Web Services (ICWS), 2015 IEEE International Conference on*, pp. 81–88, IEEE, 2015.

[85] R. Weinreich, T. Ziebermayr, and D. Draheim, "A versioning model for enterprise services," in *Advanced Information Networking and Applications Workshops, 2007, AINAW'07. 21st International Conference on*, vol. 2, pp. 570–575, IEEE, 2007.

[86] P. Kaminski, H. Müller, and M. Litoiu, "A design for adaptive web service evolution," in *Proceedings of the 2006 international workshop on Self-adaptation and self-managing systems*, pp. 86–92, ACM, 2006.

[87] K. Becker, A. Lopes, D. S. Milojicic, J. Pruyne, and S. Singhal, "Automatically determining compatibility of evolving services," in *Web Services, 2008. ICWS'08. IEEE International Conference on*, pp. 161–168, IEEE, 2008.

[88] R. Fang, L. Lam, L. Fong, D. Frank, C. Vignola, Y. Chen, and N. Du, "A version-aware approach for web service directory," in *Web Services, 2007. ICWS 2007. IEEE International Conference on*, pp. 406–413, IEEE, 2007.

[89] D. Frank, L. Lam, L. Fong, R. Fang, and M. Khangaonkar, "Using an interface proxy to host versioned web services," in *2008 IEEE International Conference on Services Computing*, pp. 325–332, IEEE, 2008.

[90] M. Dumas, M. Spork, and K. Wang, "Adapt or perish: Algebra and visual notation for service interface adaptation," in *International Conference on Business Process Management*, pp. 65–80, Springer, 2006.

[91] B. Benatallah, F. Casati, D. Grigori, H. R. M. Nezhad, and F. Toumani, "Developing adapters for web services integration," in *International Conference on Advanced Information Systems Engineering*, pp. 415–429, Springer, 2005.

[92] W. Kongdenfha, H. R. Motahari-Nezhad, B. Benatallah, F. Casati, and R. Saint-Paul, "Mismatch patterns and adaptation aspects: A foundation for rapid development of web service adapters," *IEEE Transactions on Services Computing*, vol. 2, no. 2, pp. 94–107, 2009.

[93] S. H. Ryu, F. Casati, H. Skogsrud, B. Benatallah, and R. Saint-Paul, "Supporting the dynamic evolution of web service protocols in service-oriented architectures," *ACM Transactions on the Web (TWEB)*, vol. 2, no. 2, p. 13, 2008.

[94] Z. Le Zou, R. Fang, L. Liu, Q. B. Wang, and H. Wang, "On synchronizing with web service evolution," in *2008 IEEE International Conference on Web Services*, pp. 329–336, IEEE, 2008.

[95] M. Ouederni, G. Salaün, and E. Pimentel, "Client update: A solution for service evolution," in *Services Computing (SCC), 2011 IEEE International Conference on*, pp. 394–401, IEEE, 2011.

[96] S. Basu, F. Casati, and F. Daniel, "Toward web service dependency discovery for soa management," in *Services Computing, 2008. SCC'08. IEEE International Conference on*, vol. 2, pp. 422–429, IEEE, 2008.

[97] S. Wang and M. A. Capretz, "A dependency impact analysis model for web services evolution," in *Web Services, 2009. ICWS 2009. IEEE International Conference on*, pp. 359–365, IEEE, 2009.

[98] V. Andrikopoulos, S. Benbernou, and M. P. Papazoglou, "On the evolution of services," *IEEE Transactions on Software Engineering*, vol. 38, no. 3, pp. 609–628, 2012.

[99] M. Treiber, H.-L. Truong, and S. Dustdar, "On analyzing evolutionary changes of web services," in *International Conference on Service-Oriented Computing*, pp. 284–297, Springer, 2008.

[100] A. Jahl, H. T. Tran, H. Baraki, and K. Geihs, "Wip: Behavior-based service change detection," in *2018 IEEE International Conference on Smart Computing (SMARTCOMP)*, pp. 267–269, IEEE, 2018.

[101] O. Groh, H. Baraki, A. Jahl, and K. Geihs, "Coop-automatic validation of evolving microservice compositions.," in *SATToSE*, 2019.

[102] T. G. Stavropoulos, S. Andreadis, M. Riga, E. Kontopoulos, P. Mitzias, and I. Kompatsiaris, "A framework for measuring semantic drift in ontologies," in *1st Int. Workshop on Semantic Change & Evolving Semantics (SuCCESS'16). CEUR Workshop Proceedings, Leipzig, Germany*, 2016.

[103] M. B. Juric, A. Sasa, B. Brumen, and I. Rozman, "Wsdl and uddi extensions for version support in web services," *Journal of Systems and Software*, vol. 82, no. 8, pp. 1326–1343, 2009.

[104] H. Labbaci, N. Cheniki, Y. Sam, N. Messai, B. Medjahed, and Y. Aklouf, "A linked open data approach for web service evolution," in *OTM Confederated International Conferences" On the Move to Meaningful Internet Systems"*, pp. 265–281, Springer, 2019.

[105] X. Wang, Z. Feng, S. Chen, and K. Huang, "Dkem: A distributed knowledge based evolution model for service ecosystem," in *2018 IEEE International Conference on Web Services (ICWS)*, pp. 1–8, IEEE, 2018.

[106] A. Cicchetti, D. Di Ruscio, and A. Pierantonio, "Managing dependent changes in coupled evolution," in *Theory and Practice of Model Transformations*, pp. 35–51, Springer, 2009.

[107] H. Baraki, D. Comes, and K. Geihs, "Context-aware prediction of qos and qoe properties for web services," in *Networked Systems (NetSys), 2013 Conference on*, pp. 102–109, IEEE, 2013.

[108] D. Comes, H. Baraki, R. Reichle, M. Zapf, and K. Geihs, "Heuristic approaches for qos-based service selection," in *Service-Oriented Computing*, pp. 441–455, Springer, 2010.

[109] L. Zeng, B. Benatallah, A. H. Ngu, M. Dumas, J. Kalagnanam, and H. Chang, "Qos-aware middleware for web services composition," *IEEE Transactions on software engineering*, vol. 30, no. 5, pp. 311–327, 2004.

[110] G. Canfora, M. Di Penta, R. Esposito, and M. L. Villani, "An approach for qos-aware service composition based on genetic algorithms," in *Proceedings of the 7th annual conference on Genetic and evolutionary computation*, pp. 1069–1075, ACM, 2005.

[111] D. Webster, P. Townend, and J. Xu, "Restructuring web service interfaces to support evolution," in *Service Oriented System Engineering (SOSE), 2014 IEEE 8th International Symposium on*, pp. 158–159, IEEE, 2014.

[112] S. Bechhofer, "Owl: Web ontology language," in *Encyclopedia of Database Systems*, pp. 2008–2009, Springer, 2009.

[113] M. Gebser, R. Kaminski, B. Kaufmann, and T. Schaub, "Multi-shot asp solving with clingo," *arXiv preprint arXiv:1705.09811*, 2017.

[114] A. Haller, E. Cimpian, A. Mocan, E. Oren, and C. Bussler, "WSMX - A semantic service-oriented architecture," in *Web Services, 2005. ICWS 2005. Proceedings. 2005 IEEE International Conference on*, pp. 321–328, IEEE, 2005.

[115] G. Brewka, T. Eiter, and M. Truszczynski, "Answer set programming: An introduction to the special issue," *AI Magazine*, vol. 37, no. 3, pp. 5–7, 2016.

[116] R. Kuppili, "Towards a communication protocol for service co-evolution in iot systems.," Master's thesis, University of Kassel, 2017.

[117] K. Singh and S. Upadhyaya, "Outlier detection: applications and techniques," *International Journal of Computer Science Issues (IJCSI)*, vol. 9, no. 1, p. 307, 2012.

[118] S. Omar, A. Ngadi, and H. H. Jebur, “Machine learning techniques for anomaly detection: an overview,” *International Journal of Computer Applications*, vol. 79, no. 2, 2013.

[119] M. Brijain, R. Patel, M. Kushik, and K. Rana, “A survey on decision tree algorithm for classification,” 2014.

[120] S. Chen, G. I. Webb, L. Liu, and X. Ma, “A novel selective naïve bayes algorithm,” *Knowledge-Based Systems*, vol. 192, 2020.

[121] G. E. Batista, M. C. Monard, *et al.*, “A study of k-nearest neighbour as an imputation method.,” *His*, vol. 87, no. 251-260, p. 48, 2002.

[122] G. Bebis and M. Georgiopoulos, “Feed-forward neural networks,” *IEEE Potentials*, vol. 13, no. 4, pp. 27–31, 1994.

[123] D. T. Pham and D. Karaboga, “Training elman and jordan networks for system identification using genetic algorithms,” *Artificial Intelligence in Engineering*, vol. 13, no. 2, pp. 107–117, 1999.

[124] D. Fumo, “Types of machine learning algorithms you should know,” *Towards Data Science, Towards Data Science*, vol. 15, 2017.

[125] J. Muñoz-Marí, F. Bovolo, L. Gómez-Chova, L. Bruzzone, and G. Camp-Valls, “Semisupervised one-class support vector machines for classification of remote sensing data,” *IEEE transactions on geoscience and remote sensing*, vol. 48, no. 8, pp. 3188–3197, 2010.

[126] D. Nguyen and B. Widrow, “The truck backer-upper: An example of self-learning in neural networks,” in *Advanced neural computers*, pp. 11–19, Elsevier, 1990.

[127] V. K. Chippa, S. T. Chakradhar, K. Roy, and A. Raghunathan, “Analysis and characterization of inherent application resilience for approximate computing,” in *Proceedings of the 50th Annual Design Automation Conference*, pp. 1–9, 2013.

[128] S. Chatterjee, B. Datta, S. Sen, N. Dey, and N. C. Debnath, “Rainfall prediction using hybrid neural network approach,” in *2018 2nd International Conference on Recent Advances in Signal Processing, Telecommunications & Computing (SigTelCom)*, pp. 67–72, IEEE, 2018.

[129] M. Fokaefs and E. Stroulia, “Wsdarwin: A decision-support tool for web-service evolution,” in *2013 IEEE International Conference on Software Maintenance*, pp. 444–447, IEEE, 2013.